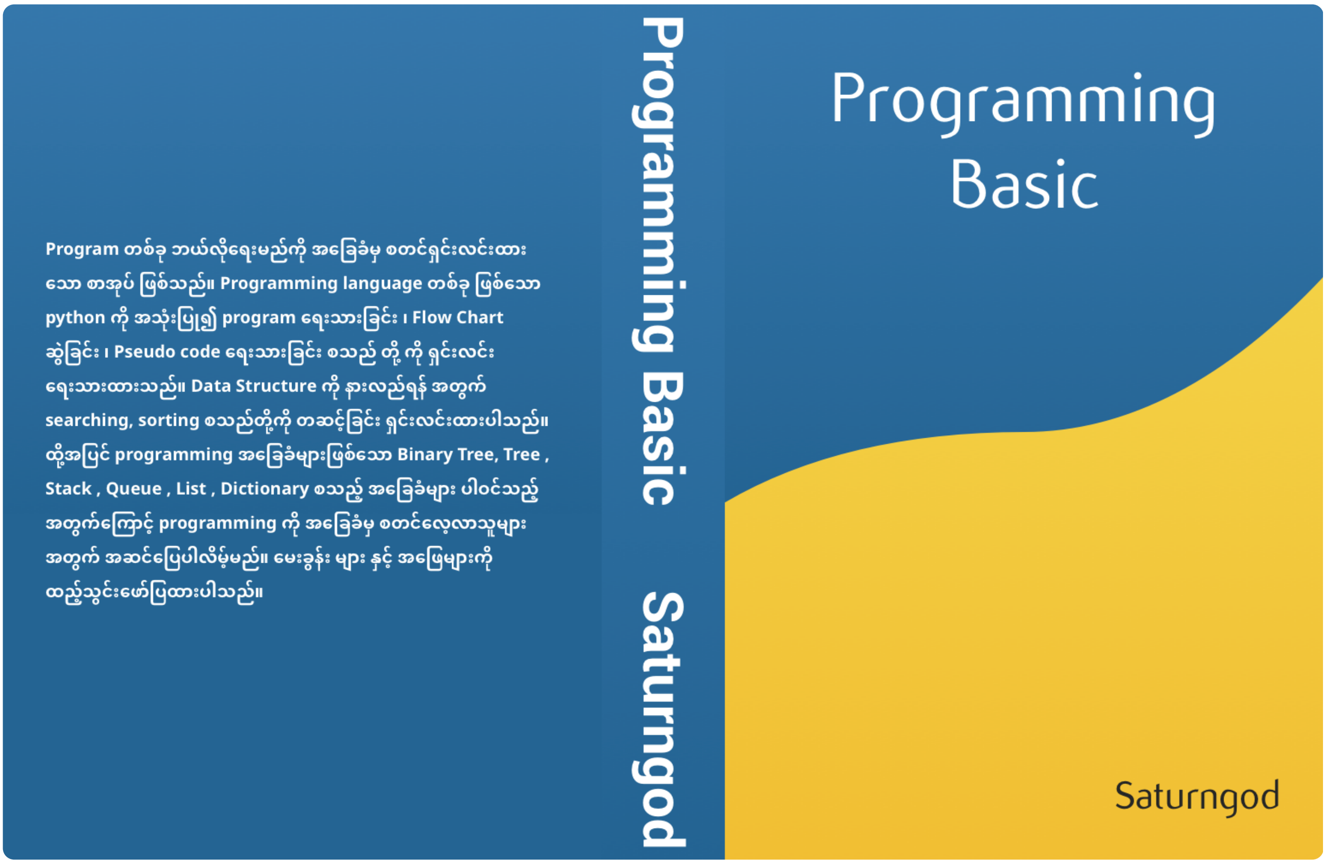

Program တစ်ခု ဘယ်လိုရေးမည်ကို အခြေခံမှ စတင်ရှင်းလင်းထား
သော စာအုပ် ဖြစ်သည်။ Programming language တစ်ခု ဖြစ်သော
python ကို အသုံးပြု၍ program ရေးသားခြင်း ၊ Flow Chart
ဆွဲခြင်း ၊ Pseudo code ရေးသားခြင်း စသည် တို့ ကို ရှင်းလင်း
ရေးသားထားသည်။ Data Structure ကို နားလည်ရန် အတွက်
searching, sorting စသည်တို့ကို တဆင့်ခြင်း ရှင်းလင်းထားပါသည်။
ထို့အပြင် programming အခြေခံများဖြစ်သော Binary Tree, Tree ,
Stack , Queue , List , Dictionary စသည့် အခြေခံများ ပါဝင်သည့်
အတွက်ကြောင့် programming ကို အခြေခံမှ စတင်လေ့လာသူများ
အတွက် အဆင်ပြေပါလိမ့်မည်။ မေးခွန်း များ နှင့် အဖြေများကို
ထည့်သွင်းဖော်ပြထားပါသည်။
Programming Basic
Saturngod
Programming
Basic
Saturngod

အမှာစာ

Computational Thinking ဆိုတာ ကလေးဘဝကတည်းက လူတိုင်း သင်ထားတတ်ထားသင့်တဲ့ အရာတစ်ခုပါ။ တစ်ခြားနိုင်ငံတွေမှာဆိုရင် Computation Thinking နဲ့ ပတ်သက်ပြီး ငယ်စဉ်ကတည်းက ကျောင်းသင်ခန်းစာတွေမှာ ထည့်သွင်းသင်ကြား နေတာကြာပါပြီ။

Computational Thinking တွေထဲမှာပါတဲ့ Problem Solving Skill တို့ Debugging Skill တို့ဆိုတာ တကယ့် လက်တွေ့ဘဝမှာလဲ အသုံးဝင်ပါတယ်။ ကွန်ပျူတာထဲက ပြဿနာတွေကို ဖြေရှင်းရင် အပြင်လောကက ပြဿနာတွေကိုလဲ တွေးတတ် ဖြေရှင်းတတ်လာပါတယ်။ ကွန်ပျူတာထဲက Program တွေကို အမှားရှာရင်း အပြင်လောက ပြဿနာတွေမှာလဲ အမှားတွေ ရှာတတ်လာပါတယ်။ ကွန်ပျူတာထဲက လက်တွေ့အသုံးဝင်တဲ့ Program တွေ ရေးသားရင်း အပြင်လောကမှာ လက်တွေ့အသုံးဝင်တဲ့ အစီအစဉ်တွေ ရေးဆွဲတတ်လာပါတယ်။

အဲဒီလိုပဲ Computer Programming ဆိုတာ ကလေးတိုင်း တတ်ထားသင့်တဲ့ အရာဖြစ်ပါတယ်။ ကျွန်တော်တို့ Computer စသင်ခါစက Assembly ဆိုတဲ့ Language ကနေ စလွတ်ခါစအချိန်ဆိုတော့ Programming Language တွေဟာ DOS Mode လို့ခေါ်တဲ့ အပေါ်မှာ တင်ပြီး ရေးကြရတာဖြစ်ပါတယ်။ အဲဒီအပြင် အခုခေတ်လို 4GL အဆင့်အထိလဲ မရောက်သေးတော့ Programming Thinking နဲ့ ပတ်သက်ရင် အခြေခံအထိ နီးစပ်ခဲ့တယ်လို့ ဆိုရပါမယ်။ ကျွန်တော်တို့ ကွန်ပျူတာ ပရိုဂရမ်ရေးသားနည်း သင်ပေးတဲ့ ဆရာတွေက Punch Card လို့ခေါ်တဲ့ အရာတွေနဲ့ ကုဒ်တွေ ရေးခဲ့ကြရပါတယ်။ ဆိုလိုတာက ပရိုဂရမ်တွေက Monitor မပါပဲ ရေးခဲ့ တွေးခဲ့ကြတာပါ။

ကျွန်တော်တို့ကို အဲဒီလို ဆရာတွေက သင်တော့ Dry Run တွေ Run ခဲ့ရတာတွေ၊ FlowChart တွေ ဆွဲခဲ့ရတာတွေ Pseudo Code တွေ ရေးခဲ့ကြရတာ မှတ်မိပါသေးတယ်။ ဆရာတွေက သူတို့ရေးထားတဲ့ Program တွေကို ကူးရေးခိုင်းပြီး စက်ထဲထည့် Run ခိုင်းတာ မဟုတ်ပါဘူး။ တစ်ဆင့်ခြင်းစီ ရှင်းပြ၊ ပြီးရင် ပြဿနာတွေပေးပြီး ဖြေရှင်းခိုင်းတာပါ။ အဲဒီလို သင်ခဲ့ရတဲ့အတွက် ကျွန်တော် Computer Programming ကို ယခုတိုင် ခုံမင်နှစ်သက်ဆဲ အသစ်အသစ်တွေ ထွက်သမျှ လေ့လာနေဆဲ ဖြစ်ပါတယ်။

အခု ကိုစေတန် ရေးထားတဲ့ စာအုပ်ကို ဖတ်လိုက်တော့ အဲဒီလို ငယ်ငယ်က ဆရာတွေ သင်ပေးခဲ့တဲ့ အချိန်တွေကို ပြန်ရောက်သွားပါတယ်။ စာတစ်မျက်နှာလှန်လိုက်တိုင်း ကျွန်တော့်ရဲ့ Computer လက်ဦးဆရာတစ်ယောက်ခြင်းစီကို ပြန်သတိရလာပါတယ်။ ဆရာတွေ ဘယ်လို ရှင်းပြခဲ့တယ်ကအစ ဘယ်လို ပြဿနာတွေ ရှင်းခိုင်းခဲ့တယ်အဆုံး အများကြီး ခေါင်းထဲမှာ ပေါ်လာပါတယ်။

အခုနောက်ပိုင်း Computer Programming Basic Knowledge နဲ့ ပတ်သက်ရင် အခုလိုမျိုး ရှင်းပြနိုင်တာတွေ ရှားလာတယ်ဆိုတော့ ကိုစေတန်စာအုပ်ထွက်လာတာ အတိုင်းထက်အလွန်ပါပဲ။ အစကတည်းက Online Version ပါ ရှိနေတာဆိုတော့ အွန်လိုင်းမှာလဲ လေ့လာလို့ရပါတယ်။ ဒါပေမယ့် ဒီလို အပင်ပန်းခံ ပြုစုပေးထားတဲ့ စာအုပ်ကိုလဲ မဖြစ်မနေ ဝယ်ဖတ်ပြီး လေ့ကျင့်ကြဖို့ တိုက်တွန်းလိုက်ရပါတယ်။

--

ဦးစိုးသီဟနောင်
Myanmar Link

မိတ်ဆက်

မြန်မာလို ရေးသားထားတဲ့ programming ကို စတင်လေ့လာတဲ့ စာအုပ်ကို မတွေ့ဖြစ်တာနဲ့ ဒီ စာအုပ်ကို ရေးမယ်လို့ ဆုံးဖြတ်ဖြစ်တာပါ။ ဒီ စာအုပ်ဟာ programming ဆိုတာ ဘာမှန်း မသိသေးတဲ့ သူများ အတွက် ရည်ရွယ်ပါတယ်။ ဒီ စာအုပ်ဖတ်ပြီးရင် program တွေ ရေးလို့ ရမလား ဆိုတော့ ရတယ်လည်း ဆိုလို့ ရသလို မရဘူးလည်း ဆိုလို့ရပါတယ်။ ဒီ စာအုပ်ဟာ အခြေခံ ဖြစ်တဲ့ အတွက် အခြေခံ သဘောတရားကို အဓိက ထား ရေးသားထားပါတယ်။ Programming အတွက် python language ကို အသုံးပြုပြီး ရေးသားထားပါတယ်။ သို့ပေမယ့် windows form တွေ ui button တွေ စသည့် UI ပိုင်းဆိုင်ရာတွေ မပါဝင်ပါဘူး။ Database ပိုင်းဆိုင်ရာတွေ လည်း ပါဝင်မှာ မဟုတ်ပါဘူး။ python အခြား language တွေ ထက် နားလည်လွယ်အောင် သင်ကြားပေးနိုင်သည့် အတွက် အခြေခံ အဆင့်မှာ python ကို အဓိကထားပြီး ရေးသားထားခြင်း ဖြစ်ပါသည်။

စာအုပ်ဟာ အခြေခံ ပိုင်းဆိုင်ရာ ကို အဓိက ထားတဲ့အတွက် Python language သင်ကြားပေးတဲ့ စာအုပ်မဟုတ်တာကို သတိပြုစေလိုပါတယ်။ Programming ဆိုတာ ဘာလဲဆိုတာ သိချင် စမ်းချင်သူတွေ ၊ နောက်ပြီး Programming ဆိုတာ ကို လေ့လာချင်သူတွေ အတွက် အခြေခံ ပိုင်းဆိုင်ရာတွေ ရေးသားထားပါတယ်။ အခြေခံတွေ ဖြစ်တဲ့ အတွက်ကြောင့် ဒီ စာအုပ် ဖတ်ပြီးတာနဲ့ လုပ်ငန်းခွင် ဝင်လို့ မရပါဘူး။ တခြား နှစ်သက်ရာ language တွေကို စပြီး လေ့လာနိုင်အောင်တော့ အထောက် အကူပြုမယ်လို့ မျှော်လင့်ပါတယ်။

Programming အနေနဲ့ လုပ်ငန်းခွင် ဝင်ဖို့အတွက် အနည်းဆုံး ၁ နှစ်လောက် လေ့လာဖို့ လိုပါတယ်။ ဒီစာအုပ်ဟာ programming ကို လေ့လာလိုသူတွေ အတွက် ပထမဆုံး လှေကားထစ် တစ်ခုမျှသာ ဖြစ်ပါတယ်။ Programming ကို ရေးသားရာမှာ စဉ်းစား တွေးခေါ်တတ်ဖို့ ပြဿနာတွေ ဖြေရှင်းတတ်ဖို့ အတွက် အခြေခံ အဆင့် အဖြစ်သာ ရှိပါတယ်။ ဒီ စာအုပ်ကို ပြီးအောင် ဖတ်ဖြစ်ခဲ့ရင်တော့ နောက် အဆင့်တွေကို လွယ်လင့် တကူ လေ့လာနိုင်မယ်လို့ မျှော်လင့် ပါတယ်။

ဒီစာအုပ်ဟာ Brad Miller, David Ranum တို့ ရေးသည့် Problem Solving with Algorithms and Data Structures စာအုပ်ကို မှီငြမ်း ကိုးကားထားသည်များ ပါဝင်ပါတယ်။ သို့ပေမယ့် စာအုပ်တစ်အုပ်လုံး ဘာသာပြန်ထားခြင်း မဟုတ်သည်ကိုတော့ သတိပြုစေလိုပါတယ်။

ဘယ်သူတွေ ဖတ်သင့်သလဲ

ဤစာအုပ် မဖတ်ခင်မှာ ဘယ်သူတွေ ဖတ်သင့်သလဲ ဆိုတာက အရေးကြီးပါတယ်။ Programming စာအုပ်ဖြစ်သလို အခြေခံပိုင်းတွေ ရေးသားထားသည့် အတွက် အခြေခံ အဆင့် ကျော်လွန်ပြီးသူတွေ လက်တွေ့ လုပ်ငန်းခွင်ထဲမှာ ရောက်နေသည့် သူတွေ အတွက် မလိုအပ် လောက်ပါဘူး။

Python ကို အခြေခံပြီး ရေးသားထားသော်လည်း python programming language လေ့လာသူတွေ အနေဖြင့်လည်း ဤ စာအုပ်ဟာ အခြေခံကျလွန်းနေပါလိမ့်မယ်။

Programming ကို စတင်လေ့လာသူတွေ ကွန်ပျူတာ တက္ကသိုလ်တက်ရောက်နေသည့် ကျောင်းသား ကျောင်းသူတွေ အတွက် အကျိုး ရှိပါလိမ့်မယ်။ စာအုပ် ၏ အခန်းတိုင်းတွင် စဉ်းစားပုံ အခြေခံ အဆင့် Flow chart , pseudo တွေ ထည့်သွင်း ရေးသားထားပါတယ်။ Programming ဆိုတာ ဘာလဲ။ ဘာတွေ လုပ်လို့ရသလဲ။ program တစ်ခု ဘယ်လို ရေးလို့ရသလဲ ဆိုတာကို လေ့လာချင်သူတွေ အတွက်လည်း သင့်လျော်ပါတယ်။

အခန်း ၁ ။ Programming ဆိုတာ

Programming ဆိုတာကတော့ process တွေ ဖြစ်ပြီးတော့ အလုပ်ပြီးမြောက်အောင် computer ကို ခိုင်းစေခြင်း ဖြစ်ပါတယ်။ ကျွန်တော်တို့ အသုံးပြုနေတဲ့ OS , Microsoft Word , Viber Messenger စတာတွေက programming ကို အသုံးပြုပြီးတော့ ရေးသားထားခြင်း ဖြစ်ပါတယ်။

ဒီစာအုပ်မှာတော့ Python 3 ကို အသုံးပြုပြီး သင်ကြားပါမယ်။ Python 3 ကို အဓိက သင်ရတဲ့ ရည်ရွယ်ချက်ကတော့ ရိုးရှင်း လွယ်ကူသည့်အတွက် programming မသိတဲ့ သူတွေ အနေနဲ့ လွယ်ကူစွာ လေ့လာနိုင်ပါတယ်။ ဒီစာအုပ်မှာ python 3 ကို run time ထည့်ထားပေးတဲ့ အတွက် python 3 ကို စက်ထဲမှာ မထည့်ထားပဲ စမ်းလို့ ရပါတယ်။

အခု Chapter မှာတော့ အခြေခံ အဆင့်တွေ ရေးသား သွားမှာ ဖြစ်တဲ့ အတွက် နားလည် သဘောပေါက်ဖို့ အရမ်း အရေးကြီးပါတယ်။ နားမလည်တာတွေကို အဆင်မပြေသည်များကို Telegram Group (https://t.me/programmingbasicbook) သို့မဟုတ် github issues (https://github.com/saturngod/programming_basic_qa/issues) မှာ မေးမြန်းနိုင်ပါသည်။

Programming ဆိုတာ

Programming ဆိုတာ ဘာလဲ ဆိုတဲ့ မေးခွန်းက စတင်လေ့လာမယ့် သူတွေ အတွက် မေးနေကျ မေးခွန်းပါပဲ။

ကျွန်တော်တို့ Computer မသုံးရင်တောင် Phone တွေကို နေ့စဉ် အသုံးပြုဖူးမှာပါ။ ကျွန်တော်တို့ phone တွေ အသုံးပြုရင် App တွေကိုလည်း အသုံးပြုမိကြမှာပါ။ App တွေက ကျွန်တော်တို့ အတွက် မျက်လှည့် ပစ္စည်းလိုပါပဲ။ လိုချင်တာတွေကို ထွက်လာဖို့ screen ပေါ်မှာ လက်နဲ့ နှိပ် လိုက်ရုံပါပဲ။

Programmer တွေ က App တွေ Program တွေကို ရေးစွဲထားပြီးတော့ အသုံးပြုတဲ့ အခါမှာ လွယ်ကူအောင် ဖန်တီးထားကြပါတယ်။ Programmer တွေဟာ programming language တစ်ခုခု ကို အသုံးပြုပြီး app တွေကို ဖန်တီးကြပါတယ်။ Programming language ကို အသုံးပြုပြီး program တွေကို ရေးသားပြီး နောက်ဆုံး App အနေနဲ့ ထွက်လာတာပါ။

Game တွေဟာလည်း programming language နဲ့ ရေးသားထားပါတယ်။ ဒါကြောင့် App တွေ Game တွေကို ဖန်တီးချင်တယ်ဆိုရင် Programming ကို သိဖို့ လိုအပ်ပါတယ်။

ဘယ်လို အလုပ်လုပ်လဲ ?

Computer ဟာ အလိုအလျောက် အလုပ်မလုပ်နိုင်ပါဘူး။ Computer နားလည်တဲ့ ဘာသာစကား နဲ့ computer ကို ခိုင်းစေရပါတယ်။ ဥပမာ။။ Viber မှာ call ကို နှိပ်လိုက်ရင် ဒီလူ ရဲ့ ဖုန်းကို သွား ခေါ်ဆိုပြီး ရေးသားထားရပါတယ်။ ဒါမှ သုံးစွဲ သူက Call ဆိုတဲ့ ခလုတ်ကို နှိပ်လိုက်တဲ့ အခါမှာ ဖုန်း သွားခေါ်ပေးပါတယ်။

Microsoft Words မှာလည်း ထိုနည်းတူပါပဲ။ Print ဆိုတာကို နှိပ်လိုက်ရင် printer ကနေ စာထွက် လာအောင် ဆိုပြီး programming နဲ့ ရေးသားထားရပါတယ်။ သုံးစွဲ သူတွေ အနေနဲ့ကတော့ print ဆိုတာကို နှိပ်လိုက်တာနဲ့ printer ကနေ print ထုတ်ပေးပါတယ်။

Computer ဟာ 0 နဲ့ 1 ကို သာ သိပါတယ်။ ကျွန်တော်တို့ အနေနဲ့ 0 နဲ့ 1 နဲ့ ရေးဖို့ အရာမှာ မလွယ် ကူလှတဲ့ အတွက် high level language တွေကို အသုံးပြုပြီး computer ကို ခိုင်းစေအောင် ရေးသား ကြပါတယ်။ Computer ကို ခိုင်းစေတတ်တဲ့သူဟာ programmer ဖြစ်လာပါတယ်။

Progrommer ဟာ သုံးစွဲ သူ နဲ့ computer ကြားမှာ ကြားခံ အနေနဲ့ သုံးစွဲ သူ ခိုင်းစေလိုတာတွေကို computer နားလည်အောင် ရေးသားပေးရတဲ့ သူပါ။ Programming language ကတော့ ဘာသာစကား တစ်ခုပါပဲ။ computer နဲ့ programmer ကြားမှာ ဆက်သွယ်ပေးတဲ့ ဘာသာစကား ပါ။ Computer ဟာ အလိုအလျောက် ဘာမှ မလုပ်နိုင်ပါဘူး။ Programmer ဟာ computer ကို ဒါ လုပ် ဒါလုပ် စသည် ဖြင့် ခိုင်းစေရပါတယ်။

ဥပမာ။။ အောက်ပါ code လေးဟာ computer screen ပေါ်မှာ စာပေါ်လာအောင် ဖန်တီးပေးပါ တယ်။

```
print("Hello World!")
```

ကျွန်တော်တို့က computer ကို ဒါလုပ်လိုက် ဆိုပြီး ခိုင်းလိုက်တဲ့ အတွက်ကြောင့် computer က လုပ်ပေးပါတယ်။ ကျွန်တော်တို့တွေ computer ကို မခိုင်းပဲနဲ့ ကျွန်တော်တို့ ဖြစ်ချင်တာတွေကို computer က အလိုအလျောက် မသိနိုင်ပါဘူး။

Programming Language

Programming ကို ရေးသားရာမှာ သက်ဆိုင် ရာ ဘာသာ စကားနဲ့ ရေးသားရပါတယ်။ Computer ဟာ 0 နဲ့ 1 ကိုပဲ သိပါတယ်။ 0 နဲ့ 1 ကို နားလည်အောင် ကြားခံ ဘာသာစကား တစ်ခု ကို အသုံးပြု ပေးရပါတယ်။ ထို့မှသာ computer က နားလည်ပြီး မိမိ လိုအပ်တာတွေကို ဖန်တီးနိုင်ပါလိမ့်မယ်။

Generation

programming language generation နဲ့ ပတ်သက်ပြီးတော့ programming ကို စတင် သင်တဲ့ သူတွေ တော်တော်များများ သိထားသင့်ပါတယ်။ မသိလို့ ဘာဖြစ်လည်း ဆိုတော့ ဘာမှတော့ မဖြစ်ပါဘူး။ သိထားတော့ လက်ရှိ ကိုယ် သုံးနေတာ ဘယ် generation ရောက်နေပြီလဲ။ ဒီ generation မတိုင်ခင်က ဘယ် language တွေ ရှိခဲ့လဲ။ အခု ကိုယ်လေ့လာနေတာက ဘယ် generation လဲ။ စတာတွေကို သိရှိနိုင်ပါတယ်။

First Generation Language (1GL)

1950 မတိုင်ခင်က UNIVAC I နဲ့ IBM 701 တို့ဟာ ပထမဆုံး machine language program လို့ ဆိုလို့ရပါတယ်။ သို့ပေမယ့် 1GL ဟာ လျင်မြန်စွာ ကုန်ဆုံးသွားပြီး 2GL ကို ကူးပြောင်းလာခဲ့ပါတယ်။

Second Generation Language (2GL)

2GL ကတော့ လူသိများတဲ့ assembly language သို့မဟုတ် assembler ပေါ့။ assembler ကတော့ အခုထက်ထိတော့ အချို့နေရာတွေမှာ အသုံးချနေဆဲပါပဲ။

Third Generation Language (3GL)

အဲဒီနောက်ပိုင်းမှာတော့ 3GL တွေ ဖြစ်တဲ့ FORTRAN , LISP, COBOL တွေ ထွက်ခဲ့ပါတယ်။ 3GL ဟာ ပိုမို ရေးသားရမှာ လွယ်ကူလာပြီး အရင်တုန်းက machine code တွေနဲ့ မတူညီတော့ပါဘူး။ 3GL ဟာ gerneral use အနေနဲ့ အသုံးချလာနိုင်ခဲ့ပါတယ်။ 3GL နဲ့ အတူတူ general purpos language တွေကိုလည်း ပေါ်ထွက်လာခဲ့ပါတယ်။

C language ကို 1969 နဲ့ 1973 ကြားမှာ developed လုပ်ခဲ့ပြီးတော့ အခုအချိန်ထိ popular ဖြစ်နေသေးတဲ့ langauge တစ်ခုပါ။ C ကို ထပ်ပြီးတော့ version အသစ်တိုးကာ 1980 မှာ C++ ကို ထုတ်ခဲ့ပါတယ်။ C++ က object-oriented နဲ့system programming တွေ ပါဝင်လာပါတယ်။

Third Generation နဲ့ အတူ လက်ရှိ အသုံးပြုနေတဲ့ general purpose programming language တွေကတော့ PHP, ASP, C, C++, Java, Javascript, Perl, Python, Pascal, Fortran တို့ ဖြစ်ပြီး သူတို့ဟာလည်း Third generation Language တွေပါပဲ။

Fourth Generation Language (4GL)

Fourth generation language ကိုတော့ စီးပွားရေးဆိုင်ရာ business software တွေအတွက် ရည်ရွယ်ပြီး ဖန်တီးခဲ့ကြပါတယ်။ အချို့ 3GL ဟာ 4GL ထဲမှာ General Use အနေနဲ့ ပါဝင်လာပါတယ်။

အောက်မှာ ဥပမာ အချို့ ဖော်ပြပေးထားပါတယ်။

- General Use
  - Perl
  - Python
  - Ruby

- Database
  - SQL
- Report generators
  - Oracle Report
- Data manipulation, analysis, and reporting languages
  - SQL PL
  - SPSS
- GUI creators
  - XUL
  - OpenROAD
- Mathematical optimisation
  - AIMMS
  - GAMS
- Database-driven GUI application development
  - Action Request System
  - C/AL
- Screen painters and generators
  - SB+/SystemBuilder
  - Oracle Forms
- Web development languages
  - CFML

Fifth Generation Language (5GL)

5GL ကတော့ အဓိကအားဖြင့် programmer မလိုပဲနဲ့ program တွေကို တည်ဆောက်ဖို့အတွက် ရည်ရွယ်ထားတာပါ။ 5GL တွေကို အဓိကအားဖြင့် Artificial Intelligence research တွေ မှာ အဓိက အသုံးပြုပါတယ်။ Prolog , OPS5, Mercury တို့က 5GL example တွေပေါ့။

Installing Python 3

ကျွန်တော် ဒီ စာအုပ်မှာ သင်ကြားမှာက programming အကြောင်းပါ။ Python programming language ကို သင်ကြားတာ မဟုတ်ပါဘူး ။ Programming language တစ်ခု နဲ့ တစ်ခုက အများ အားဖြင့် စဉ်းစားတွေးတောရသည့် အခြေခံက အတူတူပါပဲ။ ဒါကြောင့် တစ်ခုကို တတ်မြောက်ထားရင် နောက်ထပ် တစ်ခုကိုလည်း လွယ်လင့်တကူ လေ့လာနိုင်ပါတယ်။

Python 3 သွင်းသည့် အဆင့်ဟာ အရေးကြီးပါတယ်။ သွင်းသည့် အခါ အဆင်မပြေသည်များကို Telegram Group (https://t.me/programmingbasicbook) သို့မဟုတ် github issues (https://github.com/saturngod/programming_basic_qa/issues) မှာ မေးမြန်းနိုင်ပါသည်။

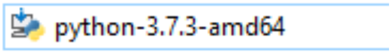
Download

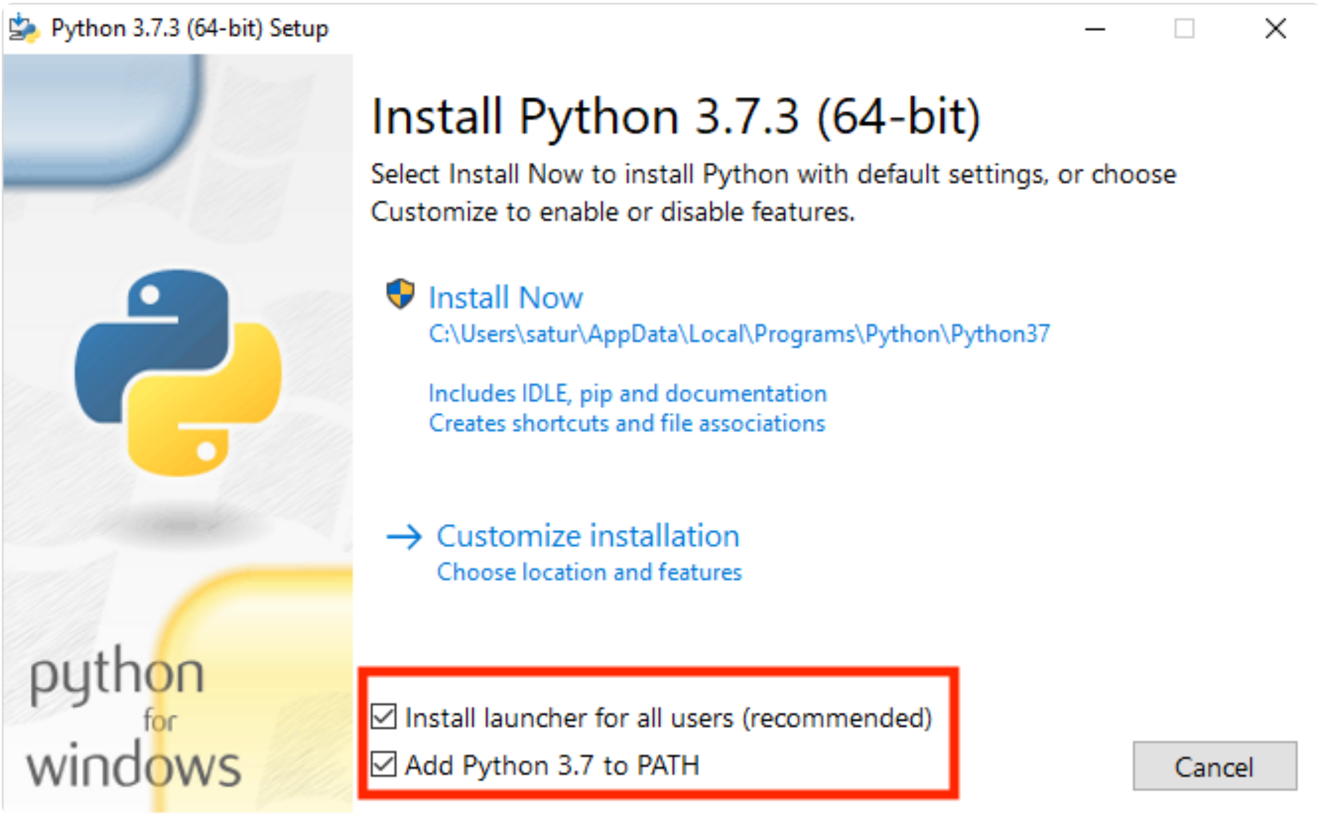
Python ကို https://www.python.org မှာ download ချယူနိုင်ပါတယ်။ Python 3 သို့မဟုတ် နောက် အသစ် version ကို download ချပါ။ လက်ရှိ စာအုပ် code တွေကို python 3 ဖြင့် ရေးသားထား သောကြောင့်ဖြစ်ပါသည်။ အခု စာအုပ်ရေးသည့် အချိန်မှာ Python version 3.7.3 ဖြင့် ရေးသား ထားခြင်းဖြစ်သည်။

Windows

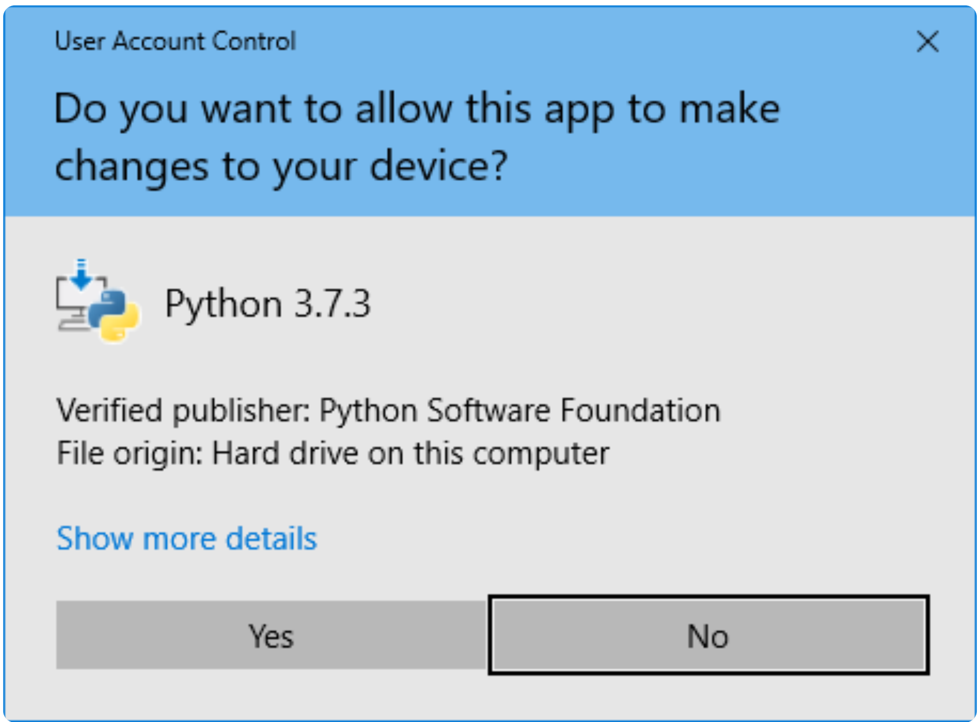
Windows အတွက် https://www.python.org/downloads/windows/ ကို သွားပါ။ အခုနောက်ပိုင်း windows တွေဟာ 64 bits ဖြစ်သည့် အတွက်ကြောင့် 64 bit version ကို download ချပါ။

Python ကို download ချပြီးတော့ ရလာသည့် installe rကို double click ပြီး install သွင်းပါ။

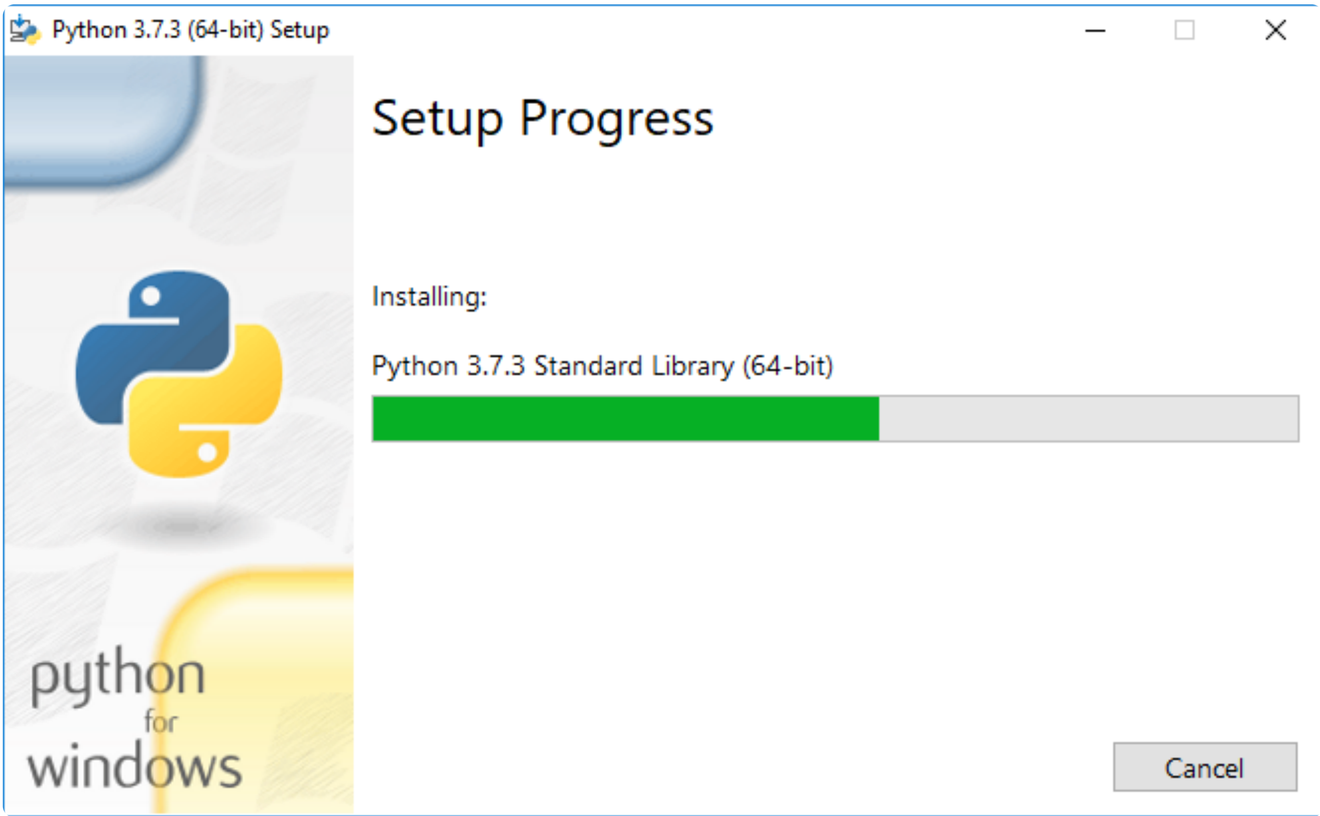


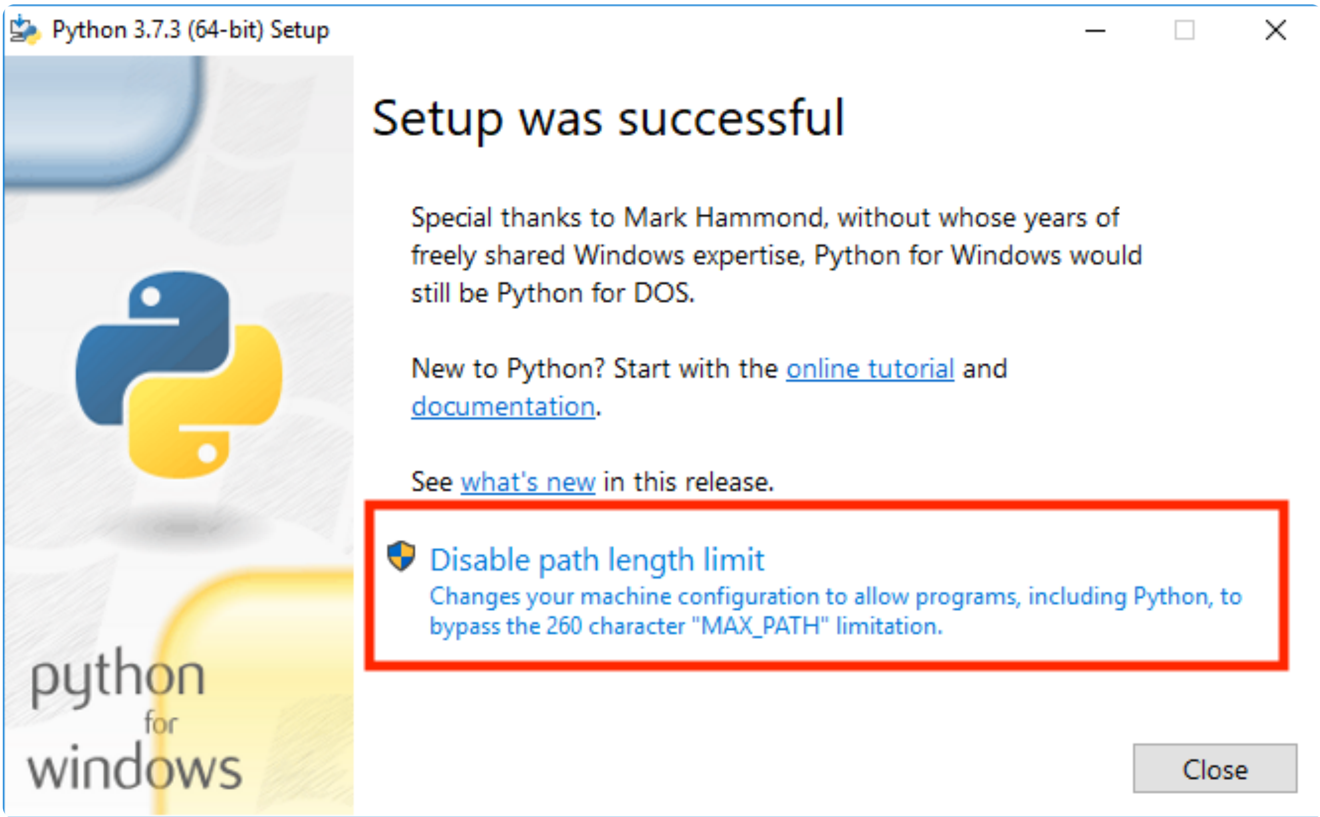


နောက်ဆုံး checkbox ဖြစ်သည့် **Add Python 3.7 to Path** ကို check လုပ်ဖို့ လိုပါသည်။

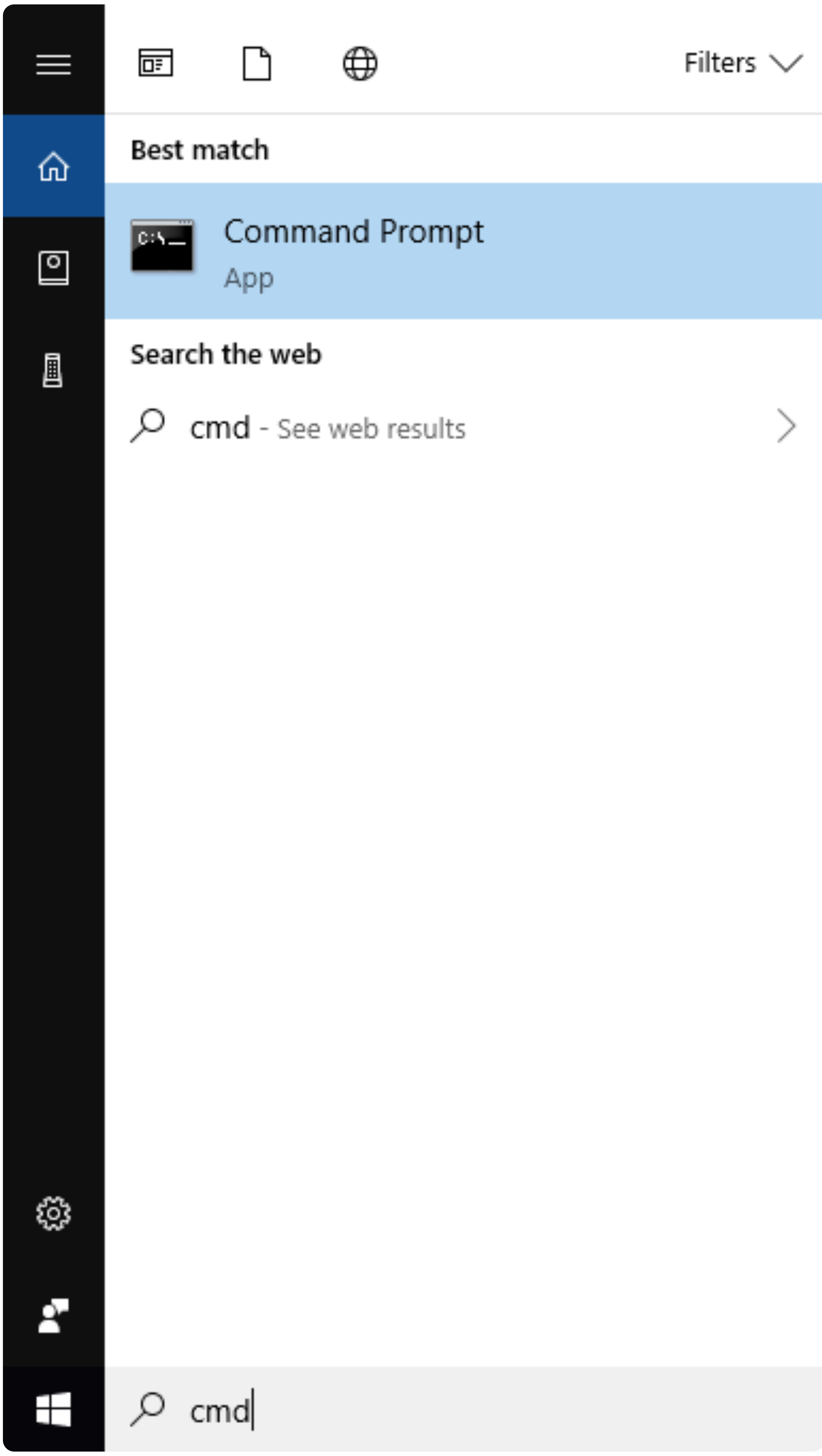


အခုလို box တက်လာရင် Yes ကို နှိပ်လိုက်ပါ။

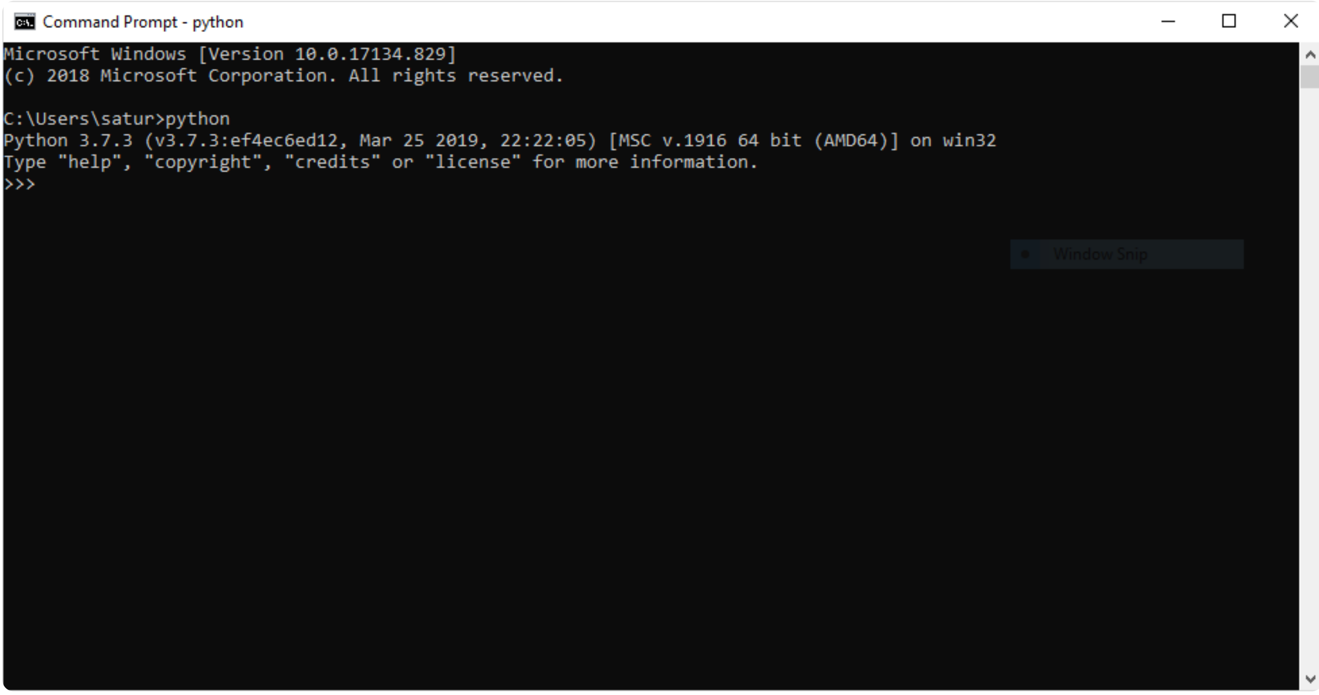




Install ပြီးသွားရင် Disable path length limit ကို နှိပ်ပေးပါ။



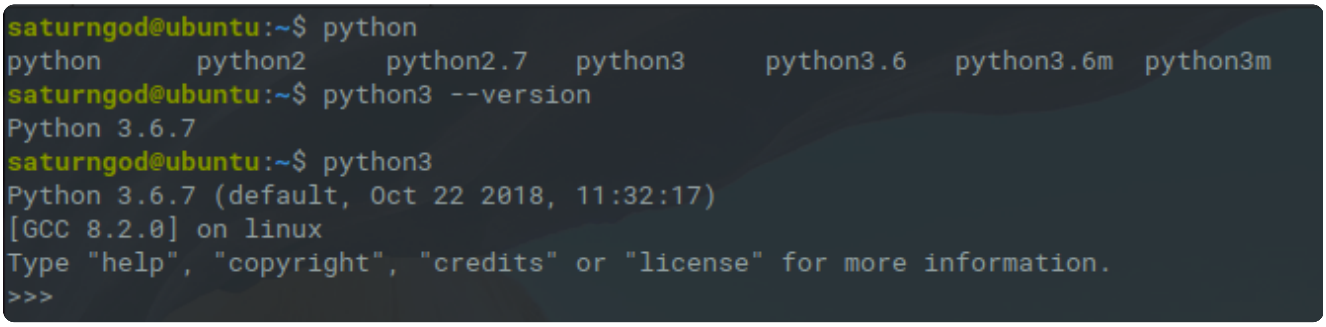
Search မှာ cmd ရိုက်လိုက်ပါ။ Command Promopt ကို နှိပ်ပါ။



command line မှာ python ရိုက်ပါ။ အခုဆိုရင် python ကို စသုံးလို့ရပါပြီ။

Linux

Ubuntu , Debian စသည့် Linux တွေမှာ Python 3 က default အနေနဲ့ သွင်းထားပြီးသား ဖြစ်ပါသည်။ Terminal မှာ python3 -V လို့ ရိုက်ကြည့်ပါ။ Python 3.6 သို့မဟုတ် အခြား version ကို တွေ့ရပါလိမ့်မည်။

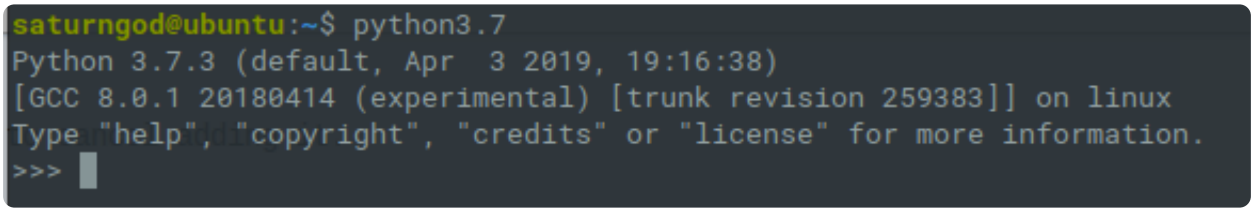


Python 3.7 နောက်ဆုံး version ဖြစ်ချင်ရင်တော့ update ပြုလုပ်ပေးဖို့ လိုအပ်ပါသည်။ terminal တွေ အောက်ပါအတိုင်း ဆောင်ရွက်ရန် လိုအပ်ပါသည်။

```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.7
```

Install သွင်းပြီးပါက python 3.7 ကို စတင် အသုံးပြုနိုင်ပါပြီ။

```
python3.7 --version
```

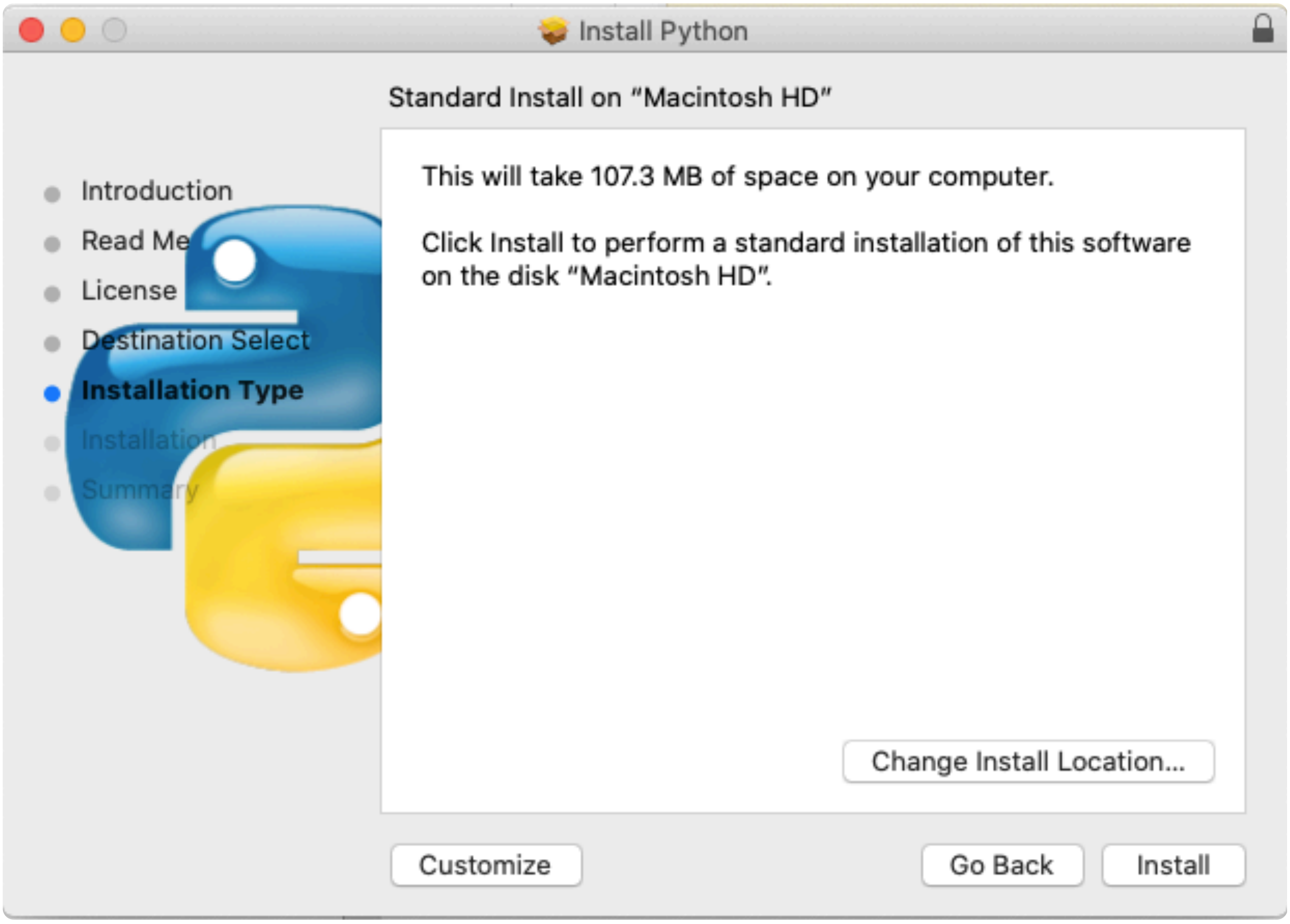


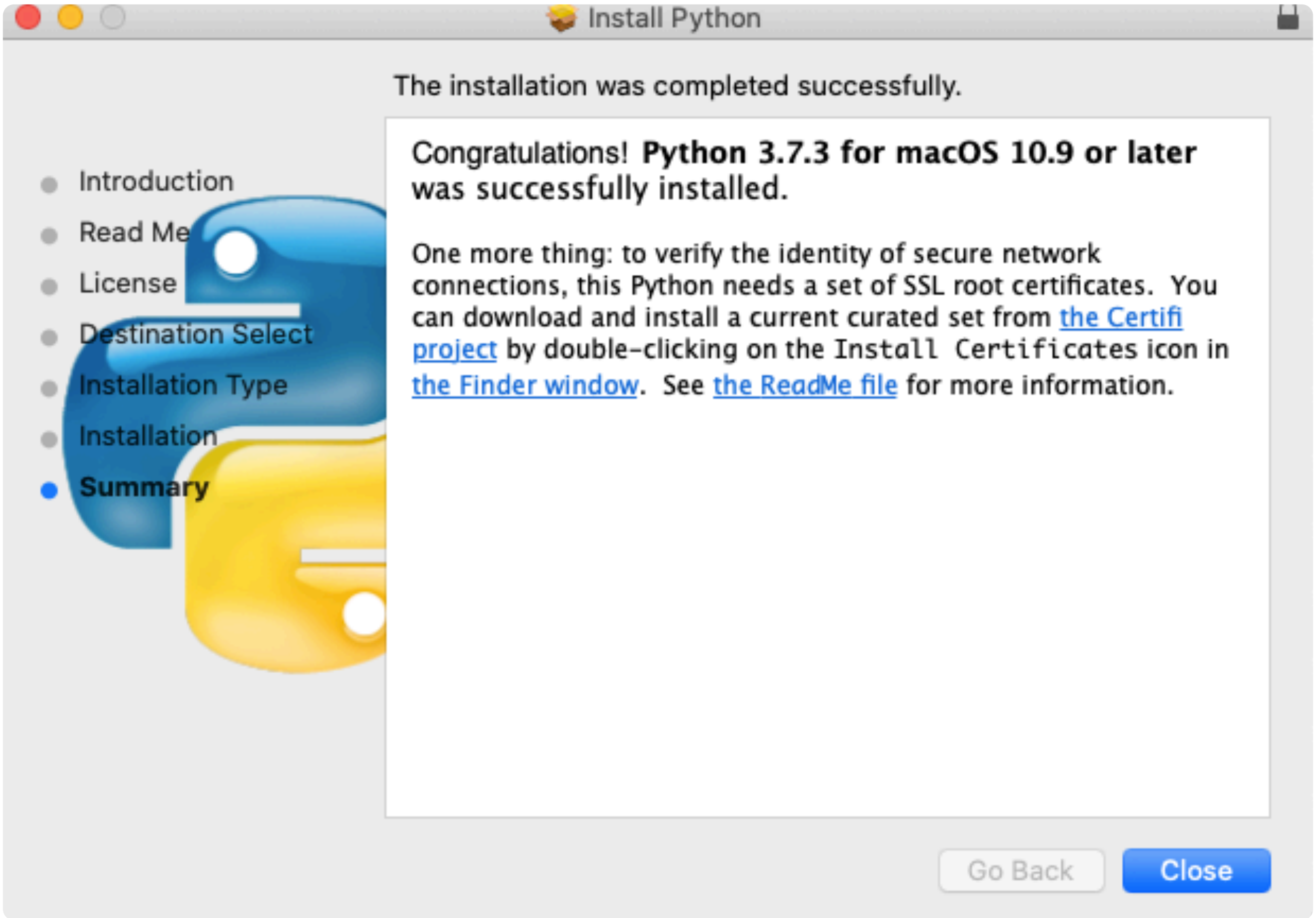
Mac

Mac အတွက် https://www.python.org/download/mac-osx/ မှာ download ချနိုင်ပါသည်။

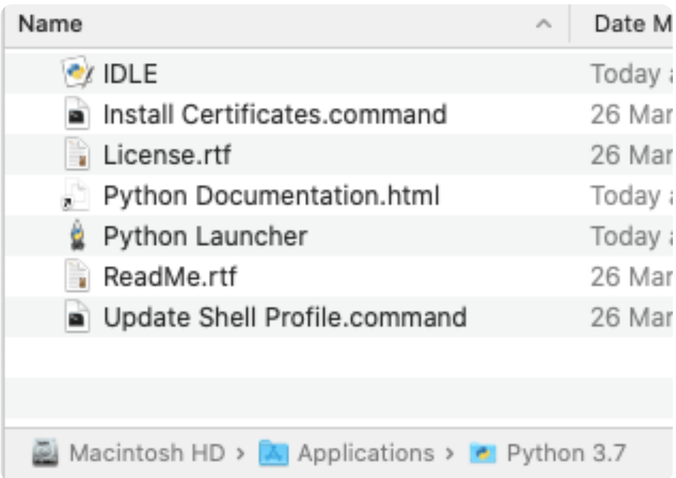
pkg file ရလာပါက double click လုပ်ပြီး install သွင်းနိုင်ပါသည်။

Install Python
Welcome to the Python Installer
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
This package will install Python 3.7.3 for macOS 10.9 or later.
Python for macOS consists of the Python programming language interpreter, plus a set of programs to allow easy access to it for macOS users including an integrated development environment IDLE.
At the end of this install, click on Install Certificates for SSL root certificates
NEW in 3.7.0: two installer variants (10.9+ 64-bit-only, 10.6+ 64-/32-bit), built-in Tcl/Tk 8.6 support (no additional third-party downloads!), OpenSSL 1.1.0, and more!
Go Back
Continue


Install Python
Standard Install on "Macintosh HD"
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
This will take 107.3 MB of space on your computer.
Click Install to perform a standard installation of this software on the disk "Macintosh HD".
Change Install Location...
Customize
Go Back
Install



Install သွင်းပြီးပါက **Application** > **Python 3.7** ကို ဖွင့်ပါ။



Install Certificates.command ကို double click နှိပ်ပြီး သွင်းပါ။

```
 -- pip install --upgrade certifi
Looking in indexes: https://pypi.python.org/simple, https://pypi.apple.com/simpl
e
Collecting certifi
  Downloading https://files.pythonhosted.org/packages/69/1b/b853c7a9d4f6a6d00749
e94eb6f3a041e342a885b87340b79c1ef73e3a78/certifi-2019.6.16-py2.py3-none-any.whl
(157kB)
    100% |████████████████████████████████| 163kB 1.7MB/s
Installing collected packages: certifi
Successfully installed certifi-2019.6.16
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
 -- removing any existing file or link
 -- creating symlink to certifi certificate bundle
 -- setting permissions
 -- update complete
logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

Update Shell Profile.command ကို သွင်းပါ။

```
Last login: Fri Jun 21 01:14:32 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Htains-MacBook-Pro:~ htainlinshwe$ /Applications/Python\ 3.7/Update\ Shell\ Prof
ile.command ; exit;
This script will update your shell profile when the 'bin' directory
of python is not early enough of the PATH of your shell.
These changes will be effective only in shell windows that you open
after running this script.
All right, you're a python lover already
logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...55 completed.

[Process completed]
```

အခုဆိုရင် python3 ကို စတင်အသုံးပြုနိုင်ပါပြီ။ Terminal မှာ python3 လို့ ရိုက်လိုက်ပါ။

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:52:21)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Testing Python

command prompt (Windows) သို့မဟုတ် Terminal (Mac , Linux) ကို ဖွင့်ပြီး python3 , (Windows တွင် python) ရိုက်လိုက်ပါ။

Python verison number နဲ့ python ရိုက်ဖို့ နေရာ တက်လာပါမယ်။

```
print("Hello World")
```

ရိုက်လိုက်ပါ။ ပြီးရင် Enter ခေါက်ရင် Hello World ဆိုတာ ထုတ်ပြတာကို မြင်ရပါမယ်။

```
bash-3.2# python3
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:52:21)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

ပြန်ထွက်ဖို့ အတွက်

```
exit()
```

```
bash-3.2# python3
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:52:21)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>> exit()
bash-3.2#
```

ရိုက်ပြီး enter ခေါက်လိုက်ပါ။

notepad သို့မဟုတ် text editor တစ်ခုခုမှာ helloworld.py ကို ဖန်တီးပါ။ Text Editor ကိုတော့ ၂၀၁၉ ခုနှစ်မှာ ခေတ်စားနေသည့် VS Code ကို အသုံးပြုစေချင်ပါသည်။

VS Code ကို Windows , Mac , Linux စသည့် platform သုံးခု လုံးမှာ အသုံးပြုနိုင်ပါသည်။

VS Code မှာ New File ကို ရွေးပါ။ ပြီးလျှင်

```
print("Hello World")
```

ရိုက်ပြီး `helloworld.py` အမည်ဖြင့် save လုပ်ထားပါ။

ပြီးလျှင် terminal ကို ဖွင့်ပါ။ File save လုပ်ထားသည့် path ကို သွားပါ။

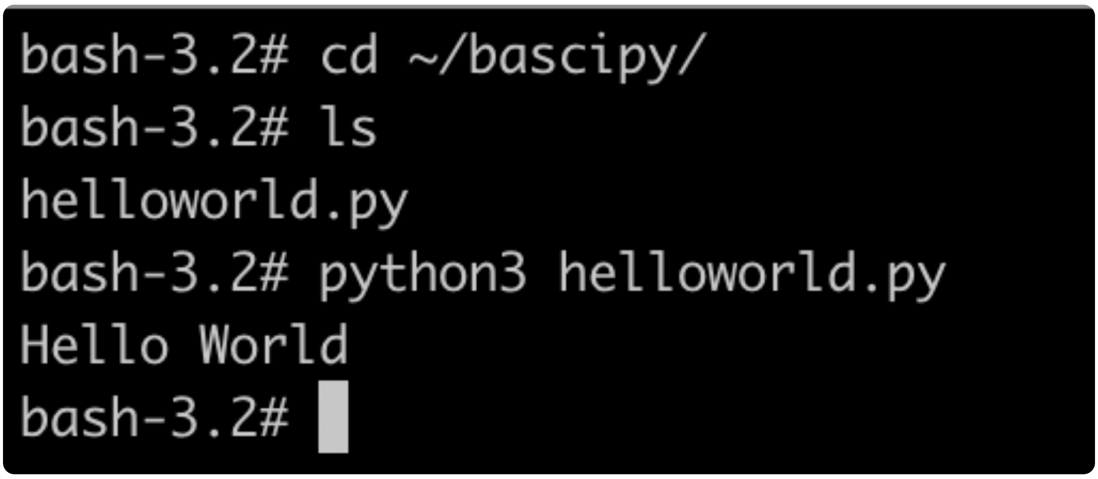
ဥပမာ အနေဖြင့် basicpy ဆိုသည့် folder ထဲမှာ helloworld.py ထည့်ထားပါသည်။ ထို့ကြောင့် basicpy folder ကို ဦးစွား သွားရန် လိုအပ်ပါသည်။

Linux နှင့် Mac OS မှာ ဆိုလျှင်

```
cd ~/basicpy
python3 helloworld.py
```

Windows မှာဆိုလျှင်

```
cd C:\basicpy
python helloworld.py
```



Hello World ထွက်လာသည် ကို တွေ့ရပါမည်။

Python မှာ သတိထားရမှာကတော့ spacing မှာပါ။ Programming language တွေဟာ တစ်ခု နှင့် တစ်ခု ရေးသားသည့် syntax မတူညီကြပါဘူး။ Python မှာကတော့

```
k = 5
data = true
if(data==true):
        k = 10
```

```
print(k)
```

ဒီ code လေးမှာ ကျွန်တော် tab သုံးထားတာကို တွေ့ရပါလိမ့်မယ်။

```
if(data==true):
<tab>k = 10
```

Python ကို ရေးသည့်အခါမှာ tab သုံးမလား space သုံးမလား တစ်ခုပဲ အသုံးပြုရပါတယ်။ Tab သုံးလိုက် Space သုံးလိုက် ဆိုရင် မရပါဘူး။ ဥပမာ

```
if(data==true):
        k = 10
    k = 20
```

အပေါ်က code ကတော့ tab တစ်ကြိမ် အသုံးပြုထားပြီး ဒုတိယ လိုင်းကတော့ space ၄ ခါ အသုံးပြုထားပါတယ်။ အဲဒီ လို မျိုး အသုံးမပြုနိုင်ပါဘူး။ မြင်အောင် ရေးပြရရင်

```
if(data==true):
<tab>k = 10
<space><space><space><space>k = 20
```

tab တစ်ကြိမ်ဟာ space ၄ ခု နဲ့ တူညီပေမယ့် code မှာ space သုံးလိုက် tab သုံးလိုက် ရောထွေးသုံးခြင်းကို python က ခွင့်မပြုထားပါဘူး။ နောက်ပိုင်း အခန်းတွေမှာ ကျွန်တော် space အစား tab ပဲ အသုံးပြုသွားတာကို တွေ့ရပါလိမ့်မယ်။

```
if(data==true):
<tab>k = 10
<tab>if(k == 10):
<tab><tab>k = 20
```

```
if(data==true):
        k = 10
        if(k == 10):
                k = 20
```

တစ်ခါတစ်လေ အထက်ပါ code ကို tab နှစ်ခါ သုံးရတာတွေ တွေ့ရပါလိမ့်မယ်။ နောက်ပိုင်း အခန်းမှာ ရှင်းပြသွားပါမယ်။

အခုဆိုလျှင် ကျွန်တော်တို့ computer ထဲမှာ python သွင်းပြီးပါပြီ။ နောက်တဆင့် အနေဖြင့် programming အကြောင်း ဆက်ပါမည်။

Sequential

Programming မှာ code တွေက တစ်ကြောင်းခြင်းစီ အလုပ်လုပ်ပါတယ်။ တစ်ခုပြီးမှ နောက်တစ်ခုက အလုပ်လုပ်တယ်။

ဥပမာ အောက်က code လေးကို တချက်ကြည့်လိုက်ပါ။

```
i = 5 + 4
i = i  + 6
```

5 နဲ့ 4 ကို ပေါင်းပြီးတော့ i ထဲကို ထည့်တယ်။ ပြီးမှ ရလာတဲ့ အဖြေကို ၆ နဲ့ ပေါင်းတယ်။ အဲလို တစ်ကြောင်းစီ အလုပ်လုပ်ပေးပါတယ်။ 5+4 ကို ပေါင်းတာ မပြီးသေးပဲ 6 နဲ့ သွားပေါင်းသည့် အဆင့်ကို မကျော်သွားပါဘူး။

ဒါကြောင့် programming အတွက် စဉ်းစားသည့် အခါမှာ တဆင့်ပြီး တဆင့် စဉ်းစားပြီးတော့ ရေး ရပါတယ်။ ကျွန်တော်တို့ ခိုင်းလိုတဲ့ အရာတွေကို တဆင့်ပြီးတဆင့် ရေးသားပြီးတော့ ခိုင်းစေရ ပါတယ်။ ထို့မှသာ ကျွန်တော်တို့ လိုချင်တဲ့ ရလဒ် ကို ရရှိမှာ ဖြစ်ပါတယ်။

Variable

Programming ကို စလေ့လာတော့မယ်ဆိုရင် ပထမဆုံး သိဖို့လိုတာကတော့ variable ပါပဲ။ variable ဆိုတာကတော့ data တွေကို ခဏတာ memory ပေါ်မှာ သိမ်းထားပေးတဲ့ နေရာပေါ့။ variable ကို နာမည်ပေးဖို့ လိုပါတယ်။ ဒါ့အပြင် variable အမျိုးအစား သတ်မှတ်ပေးဖို့လည်း လိုအပ်ပါတယ်။

```
print("Hello World!")
```

အထက်ပါ code မှာ ဘာ variable မှ မပါပါဘူး။

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
boolean = True         # Boolean Value True and False only

print (counter)
print (miles)
print (name)
print(boolean)
```

```
100
1000.0
John
True
```

ဒီ code လေးမှာ ဆိုရင်တော့ variable ၃ ခု ပါတာကို တွေ့ပါလိမ့်မယ်။

counter ကတော့ integer ပါ။ Integer ဆိုတာကတော့ ဒသမ မပါတဲ့ ကိန်းပြည့် တန်ဖိုး တွေကို ဆိုတာပါ။

miles ကတော့ floating ပါ။ ဒသမ တန်ဖိုး တွေပေါ့။

name ကတော့ String ပါ။ စာလုံး စာကြောင်းတွေ အတွက်ပါ။

boolean ကတော့ Boolean ပါ။ True နဲ့ False value ပဲ ရှိပါတယ်။ True , False တွေကို နောက်ပိုင်းမှာ condition တွေ နဲ့ တွဲသုံးတာကို တွေ့ရပါလိမ့်မယ်။

print ကတော့ value ကို ပြန်ပြီး ထုတ်ထားပေးတာပါ။

variable တွေကို နာမည်ပေးရမှာ သက်ဆိုင်ရာ နာမည်တွေ ပေးရပါတယ်။ x , y ,z ဆိုပြီး ပေးမည့်အစား ဒီ တန်ဖိုးကတော့ counter ဖြစ်ပါတယ်။ ဒီ တန်ဖိုးကတော့ miles ဖြစ်ပါတယ်။ ဒီ စာကတော့ name ဖြစ်ပါတယ် ဆိုပြီး variable name ကို ပေးလိုက်တဲ့ အတွက် ဖတ်လိုက်တာနဲ့ သဘောပေါက်လွယ်သွားပါတယ်။

Bit and Storage Data on Memory

Variable က memory ပေါ်မှာ နေရာ ယူပြီး ခဏ သိမ်းထားပါတယ်။ program ပိတ်လိုက်တဲ့ အခါမှာ memory ပေါ်ကနေလည်း ရှင်းလင်းလိုက်ပါတယ်။

Computer မှာ 0 နဲ့ 1 ပဲ ရှိပါတယ်။ 0 မဟုတ် ရင် 1 ပေါ့။ အဲဒါကို bit လို့ ခေါ်ပါတယ်။

8 Bit ကို 1 Byte လို့ ခေါ်ပါတယ်။ 8 Bit ဆိုတာကတော့ binary system အရ 00000000 ကနေ ပြီးတော့ 11111111 ထိ ရှိပါတယ်။

binary value 11111111 ကို decimal ပြောင်း 255 ရလာပါတယ်။

ဒါဟာ 8 Bit မှာ သိမ်းနိုင်တဲ့ အများဆုံး တန်ဖိုးပါ။

Integer ဟာ 32 Bit ရှိပါတယ်။ Integer တန်ဖိုးမှာ unsigned နဲ့ signed ဆိုပြီး ရှိပါတယ်။ Unsign ဟာ အပေါင်း ကိန်းတွေ ပဲ ဖြစ်တဲ့ အတွက်ကြောင့် 32 bit အပြည့် အသုံးပြုနိုင်ပါတယ်။ Signed ကတော့ +/- စတာပါလာတဲ့ အတွက်ကြောင့် 31 Bit ပဲ အသုံးပြုနိုင်ပါတယ်။ 1 bit ကိုတော့ အပေါင်း လား အနှုတ်လား ဆုံးဖြတ်ဖို့ အတွက်ပေးလိုက်ရပါတယ်။

Bit ပေါ်မှာ မူတည်ပြီး Integer တန်ဖိုးကို တွက်တဲ့ ပုံသေးနည်း ရှိပါတယ်။

```
(2 ^ [Total Bit]) - 1
```

`^` သည် power ဖြစ်သည်။

ဒါကြောင့် 8 Bit အတွက်ဆိုရင်တော့

```
(2 ^ 8) - 1
= 256 - 1
= 255
```

Sign Integer အမြင့်ဆုံး တန်ဖိုး တွက်ကြည့်ရင်တော့

31 Bit ကို အများဆုံးထားတယ်။ ဒါကြောင့်

```
(2 ^ 31) - 1
=  2147483648 - 1
= 2,147,483,647
```

ဆိုပြီး ရလာပါတယ်။

ကျွန်တော်တို့ အနှုတ် ရဲ့ အနည်းဆုံး တန်ဖိုး မတွက် ခင် အောက်က ဇယားလေးကို တချက် ကြည့်လိုက်ပါ။

Binary Value	**Two's complement interpretation**	**Unsigned interpretation**
00000000	0	0
00000001	1	1
⋮	⋮	⋮
01111111	127	127
10000000	-128	128
10000001	-127	129
10000010	-126	130
⋮ ⋮	⋮	
11111110	−2	254
11111111	−1	255

ဒီ table မှာ ဘယ်ဘက် ဆုံးကတော့ Binary တန်ဖိုး နဲ့ အလယ်က Singed တန်ဖိုး ၊ နောက်ဆုံး ကတော့ Unsigned တန်ဖိုးပါ။

အနှုတ် ဖြစ်ပြီဆိုတာနဲ့ ရှေ့ဆုံး binary ကို 1 ပြောင်းလိုက်ပါတယ်။ Sign မှာ 0 အတွက် က အပေါင်း အနေနဲ့ တည်ရှိနေပေမယ့် အနှုတ် 0 ဆိုတာ မရှိပါဘူး။ ဒါကြောင့် ကျွန်တော်တို့ အနေနဲ့ အနှုတ် တန်ဖိုး တစ်ခု ပို ပြီး သိမ်းလို့ရပါတယ်။

ဒါကြောင့် အောက်က equation နဲ့ Integer ရဲ့ Max range ကို တွက်လို့ ရပါတယ်။

```
- (2 ^ [Total Bit]) to (2 ^ [Total Bit]) - 1
```

ဒါကြောင့် 32 Bit Integer Signed ကို တွက်ကြည့်ရင်တော့

```
- (2 ^ 31) to (2 ^31) -1
=  -2,147,483,648 to 2,147,483,647
```

Unsigned ကို တွက်ရင်တော့ 32 Bit အပြည့် နဲ့ တွက်ရပါမယ်။

```
0 to (2 ^ 32) - 1
= 0 to 4,294,967,295
```

ရ လာပါမယ်။

Float ကတော့ 32 Bit ရှိပါတယ်။

32 bit မှာ

- sign 1 bit
- exponent 8 bit
- Mantissa 23 bit

binary value `0 01111100 01100000000000000000000`

ကို ၃ ပိုင်း ခွဲထုတ်ပါမယ်။

```
Sign 0
exponent 01111100
fraction က 01100000000000000000000
```

sign 0 ဖြစ်တဲ့ အတွက်ကြောင့် +

```
1 + SUM ( i=1 To 23) of b(23-i) 2 ^ -i
```

ဒါကြောင့်

```
1 + 2 ^ -2 + 2 ^ -3 = 1.375
```

exponent က 01111100

```
2 ^ (e - 127) = 2 ^ 124-127 = 2 ^ -3
value = 1.375 x 2 ^ -3 = 0.171875
```

ဒါကြောင့် 0.171875 ကို binary 0 01111100 01100000000000000000000 အနေနဲ့ သိမ်းဆည်းပါတယ်။

Float ဟာ ဒဿမ ၇ နေရာထိ သိမ်းနိုင်ပါတယ်။

နောက်ထပ် ဒဿမ တန်ဖိုးကတော့ Double ပါ။

Double ကတော့ 64 Bit ရှိပါတယ်။ Double ကတော့ ဒဿမ 16 နေရာထိ သိမ်းနိုင်ပါတယ်။

String တန်ဖိုးကတော့ character storage ပေါ်မှာ မူတည်ပါတယ်။

- ASCII ဆိုရင် 1 Character 8 Bit
- UTF-8 ဆိုရင် 8 Bit ကနေ 32 Bit (4 Bytes)
- UTF-16 ဆိုရင် 16 Bit ကနေ 32 Bit (4 Bytes)

အထိ နေရာ ယူပါတယ်။

ကျွန်တော်တို့ အနေနဲ့ storage တွေ အကြောင်း အနည်းငယ် သိထားခြင်းအားဖြင့် variable တွေ အများကြီး ဘာကြောင့် မသုံးသင့်တယ်ဆိုတာကို သဘောပေါက်စေဖို့ပါ။ memory အသုံးပြုပုံ အနည်းဆုံး ဖြစ်အောင် ဘယ်လို ရေးရမလဲ ဆိုတာကို စဉ်းစားစေနိုင်ဖို့ ရည်ရွယ်ပါတယ်။ တခြား အသေးစိတ်ကိုတော့ Computer Science ပိုင်းနဲ့ သက်ဆိုင်သွားပါပြီ။ ကျွန်တော့် အနေနဲ့ Programming Basic ပိုင်းမှာ တော့ ဒီလောက် ပဲ သင်ကြားပြီးတော့ programming နဲ့ သက်ဆိုင်ရာ တွေကို ဆက်လက် ရေးသားသွားပါမယ်။

Operators

Operators ဆိုတာကတော့ ပေါင်းနှုတ်မြှောက်စား ပါ။ Programming မှာ

- အပေါင်း +
- အနှုတ် -
- အမြှောက် *
- အစား /
- အကြွင်း %

ဆိုပြီး သုံးပါတယ်။ ကျွန်တော်တို့ သင်္ချာ မှာ အသုံးပြုသည့် × နှင့် ÷ အစားကို အသုံးမပြုပါဘူး။

အပေါင်း

အပေါင်း အတွက် ဥပမာ လေး အောက်မှာ ကြည့်ကြည့်ပါ။

```
k = 5 + 4
print(k)
```

9

ကိန်း ၂ ခု ကို ပေါင်းထားပြီးတော့ ရလဒ် ကို k ထဲကို ထည့်ထားတာပါ။
programming မှာ data တွေကို ထည့်သွင်းမယ်ဆိုရင် ဘယ်ဘက်မှာ ရေးပါတယ်။

```
k = 5
```

အဲဒီ အဓိပ္ပာယ်ကတော့ `k` ထဲကို `5` ထည့်လိုက်လို့ ဆိုလိုတာပါ။

သင်္ချာမှာကတော့

```
5 + 1 = 6
```

ဆိုပြီး ရပါတယ်။ Programming မှာတော့

```
6  = 5 + 1
```

ဆိုပြီး ရေးရပါတယ်။ 6 က ရလဒ်ပါ။ ရလာတဲ့ အဖြေကို k ဆိုတဲ့ variable ထဲ အစား သွင်းဖို့ အတွက်

```
k = 5 + 1
```

ဆိုပြီး ရေးပါတယ်။အဲဒါဆိုရင် k ထဲမှာ 6 ဝင်သွားပါပြီ။ တနည်းပြောရရင် ဘယ်ဘက် မှာ process လုပ်ပြီးတော့ ညာဘက် က variable ထဲကို process လုပ်ထားသည့် data ထည့်ပေးလိုက်ပါတယ်။ အပေါင်း မှ မဟုတ်ပါဘူး။ အနှုတ် ၊ အမြှောက် ၊ အစားတွေမှာလည်း variable assign က ဒီပုံစံ ပါပဲ။ နားလည်အောင် ဥပမာလေး ကြည့်ကြည့်ပါ။

```
a = 3
b = 4
c = a + b
print (c)
```

```
7
```

a ထဲကို 3 ထည့်။ b ထဲ ကို 4 ထည့်။ ပြီးလျှင် a နဲ့ b ကို ပေါင်း။ ရလာတဲ့ အဖြေကို c ထဲ ထည့်ပြီး တော့ ရလဒ် ပြန်ထုတ်ပြထားပါတယ်။

အနှုတ်

အပေါင်း အတိုင်းပါပဲ။ အနှုတ် အတွက် - ကို အသုံးပြုပါတယ်။

```
a = 10
b = 4
c = a - b
print (c)
```

```
6
```

အမြှောက်

အမြှောက်အတွက် * ကို အသုံးပြုပါတယ်။

```
a = 3
b = 4
c = a * b
print (c)
```

```
12
```

အစား

အစား အတွက် / ကို အသုံးပြုပါတယ်။

```
a = 10
b = 2
c = a / b
print (c)
```

```
5.0
```

အကြွင်း

အကြွင်းကို % ကို အသုံးပြုပါတယ်။

```
a = 13
b = 8
c = a % b
print (c)
```

```
5
```

Problem Solving

Programming ကို ရေးသားရာမှာ သင်္ချာ ကဲ့သို့ပင် ပြဿနာတွေ ကို ဖြေရှင်း ရတာတွေ ပါဝင်ပါတယ်။ အသုံးပြုသူတွေ ဖြစ်နေတဲ့ ပြဿနာတွေကို လွယ်လင့်တကူ ဖြေရှင်းပေးဖို့ program တွေကို စဉ်းစား တွေးခေါ် ရေးရပါတယ်။

ဥပမာ။။ ကိန်း ၂ လုံးကို လက်ခံပါ။ ပြီးရင် ၂ ခု ပေါင်းလဒ်ကို ထုတ်ပြပါ။

လွယ်လွယ်လေးပါ။ ကျွန်တော် တို့ အနေနဲ့ ကိန်း ၂ လုံး လက်ခံမယ်။ ပြီးရင် ပေါင်း ပြီး ရတဲ့ အဖြေကို ထုတ်ပေးလိုက်ရုံပါပဲ။

အသုံးပြုသူကို input ထည့်ပေးဖို့ အတွက် python3 မှာတော့ input ကို အသုံးပြုပါတယ်။

user_input = input("Please enter something: ")

print ("you entered", user_input)

အဲဒီ code လေးကို python 3 မှာ run လိုက်ရင်

```
Please enter something: hi
you entered hi
```

ဆိုပြီး ပြပါလိမ့်မယ်။

ကျွန်တော်တို့ user input လက်ခံ တတ်ပြီ ဆိုရင် ကိန်း ၂ လုံး လက်ခံရအောင်။ ပြီးတော့ ပေါင်း ပြီးတော့ ရလဒ်ကို ထုတ်ပေးရုံပါပဲ။

```
input1 = int(input("Please enter first number: "))
input2 = int(input("Please enter second number: "))
result = input1 + input2

print (input1,"+",input2,"=", result)
```

```
Please enter first number: 5
Please enter second number: 89
5 + 89 = 94
```

ကျွန်တော်တို့ user ဆီကနေ data ကို လက်ခံတဲ့ အခါ string value အနေနဲ့ ရလာပါတယ်။ integer အနေနဲ့ လိုချင်တဲ့ အတွက်ကြောင့် int() ကို အသုံးပြုထားပါတယ်။

```
input1 = int(input("Please enter first number: “))
```

input ကနေ user အနေနဲ့ နံပတ်ကို ရိုက်ထည့်ပေးလိုက်ပေမယ့် string အနေနဲ့ ဝင်လာပါတယ်။ int() နဲ့ ပြောင်းလိုက်တဲ့ အတွက်ကြောင့် နံပတ်ရပါတယ်။

```
a = "5"
b = "6"
print(a+b)
```

```
56
```

string ၂ ကို ပေါင်းသည့် အခါမှာ 11 အစား 56 ဖြစ်သွားတာကို တွေ့ရမှာပါ။

String နံပတ်ကို int ပြောင်းချင်တာကြောင့် int() ကို အသုံးပြုရပါတယ်။

```
a = "5"
b = "6"
print(int(a)+int(b))
```

```
11
```

အခု ဟာ ဥပမာ အသေးလေး တစ်ခုပါ။

နောက်ပြီး စဉ်းစား ရမှာ က အသုံးပြုသူက ဂဏန်းတွေ မထည့်ပဲ စာတွေလည်း ရိုက်ထည့် နိုင်တယ်။ ဂဏန်းတွေ မဟုတ်ရင် ဂဏန်းသာ ထည့်ပါဆိုပြီး message ပြဖို့ လိုလာတယ်။ ဒီလိုမျိုး ဖြစ်နိုင်ခြေ ရှိတာတွေကို programming ရေးတဲ့ အခါ ထည့်စဉ်းစားရပါတယ်။

အဲဒီလိုမျိုး စစ်ဖို့ အတွက် နောက် အခန်းမှာမှ looping တွေ condition တွေ အကြောင်း ရေးသွားပါမယ်။

လေ့ကျင့်ခန်း ၁ - ၁

မေးခွန်း ၁။

အောက်ပါ program မှာ k ရဲ့ value က ဘာဖြစ်ပါသလဲ။

```
i = 7 + 3
k = i + 2
```

အဖြေက

A. 7

B. 10

C. 12

—

မေးခွန်း ၂။

လူတစ်ယောက်၏ မွေးဖွားသည့် နှစ်ကို လက်ခံပါ။ ထို့နောက် ၂၀၁၈ တွင် ရောက်ရှိနေသည့် အသက်ကို ဖော်ပြပါ။

အခန်း ၂ ။ Programming

ဒီအခန်းမှာတော့ Programming နဲ့ သက်ဆိုင်ရာ စဉ်းစားတွေးခေါ်ပုံတွေ ရေးပုံတွေ ကို ရေးသား သွားမှာပါ။ နောက်ပြီး ကိုယ်တိုင် စဉ်းစားပြီး ရေးရမယ့် အပိုင်းတွေ ပါပါတယ်။ Programming က သချ်ာလို လက်တွေ့ လေ့ကျင့် စဉ်းစား ရပါတယ်။ စာဖတ်ရုံနဲ့ မရပါဘူး။ ကျွန်တော် ဒီ အခန်မှာ Pseudo code အကြောင်း ရေးထားပေးပြီးတော့ နောက်ပိုင်းမှာ Pseudo code က ကိုယ်ပိုင် program ကို python3 နဲ့ ပြန်ပြီး ရေးကြည့်ဖို့ လေ့ကျင့်ခန်းတွေ ပါဝင်ပါမယ်။

Pseudo

Pseudo code ဆိုတာကတော့ အတုအယောင် code ပေါ့။ programming မှာ language အမျိုးမျိုး ရှိပြီးတော့ language တစ်ခု နဲ့ တစ်ခုမှာ ပါဝင်တဲ့ function တွေ မတူပါဘူး။ ဒါကြောင့် ကျွန်တော်တို့တွေဟာ Pseudo code ကို အသုံးပြုပြီးတော့ တစ်ယောက် နဲ့ တစ်ယောက် နားလည်အောင် ရေးသားပေးကြပါတယ်။ Pseudo code ဆိုတာက ဘယ်သူ မဆို နားလည်အောင် ရေးသားထားတဲ့ language တစ်မျိုး ဖြစ်တဲ့ အတွက် ဘယ်လိုမျိုး ရေးရမယ် ဆိုတာကို အတိအကျ သတ်မှတ်ပြီး ပြောလို့ မရပါဘူး။ တချို့ကလည်း C++ style အသုံးပြုသလို တချို့ကလည်း javascript style အသုံးပြုပါတယ်။ ဒါပေမယ့် pseudo code က တကယ် run ကြည့်လို့ မရဘူး။ အသုံးပြုလို့ မရဘူး။ pseudo code ကို ပြန်ကြည့်ပြီးတော့ မိမိ နှစ်သက်ရာ language နဲ့ ပြန်ရေးပြီး run မှ သာ ရပါလိမ့်မယ်။

pseudo code ဥပမာ လေးကို ကြည့်ရအောင်

```
If student's grade is greater than or equal to 40

  Print "passed"

else

  Print “failed"
```

ဒီ code လေးမှာ ဆိုရင် ရေးထားတာက english လိုပါပဲ။ ကျောင်းသား ရဲ့ အမှတ်က ၄၀ ကျော်ရင် အောင် တယ်လို့ ပြမယ်။ မဟုတ်ခဲ့ရင် ကျတယ်လို့ ပြောမယ်။ ရှင်းရှင်းလေးပါပဲ။

Pseudo code မှာ ကျွန်တော်တို့

- SEQUENCE လုပ်မယ့် အလုပ် အဆင့်ဆင့် ကို ရေးသားခြင်း

- WHILE ကတော့ loop အတွက်ပါ။ ထပ်ခါ ထပ်ခါ အကြိမ်ကြိမ် လုပ်ဖို့ အတွက်ပါ။ ဘယ် အထိ လုပ်ဖို့ ဆိုတာကို စစ်ဆေးထားပြီး စစ်ဆေးတဲ့ အဆင့် မဟုတ်တော့ဘူးဆိုမှသာ looping ထဲက ထွက်ပါလိမ့်မယ်။ ဥပမာ။ ထပ်ခါ ထပ်ခါ လုပ်မယ်။ စုစုပေါင်း ရမှတ် ၁၀၀ မ ပြည့်မချင်း လုပ်မယ် ဆိုတာ မျိုးပေါ့။
- IF-THEN-ELSE စစ်ဆေးပြီးတော့ ဖြစ်ခဲ့ရင် ဒါလုပ် မဖြစ်ခဲ့ရင်တော့ ဒါကို လုပ်ပါ ဆိုတဲ့ condition တွေ အတွက်ပါ။
- CASE ကတော့ condition အတွဲလိုက် စစ်ဖို့ပါ။ 1 ဖြစ်ခဲ့ရင် ဒါလုပ်။ 2 ဖြစ်ခဲ့ရင် ဒါလုပ်။ ၃ ဖြစ်ခဲ့ရင် ဒါလုပ် စတာတွေ အတွက်ပါ။
- FOR ကတော့ while နဲ့ အတူတူပါပဲ။ သို့ပေမယ့် FOR ကတော့ ဘယ်ကနေ ဘယ်အတွင်း ဆိုတာ ရှိပါတယ်။ ဥပမာ ။ ထပ်ခါ ထပ်ခါ လုပ်မယ်။ ဒါပေမယ့် ၁ ကနေ ၅ အတွင်း လုပ်မယ် ဆိုတာ မျိုးပေါ့။

SEQUENCE

Programing ဆိုတာက sequential ဆိုတာကို ကျွန်တော် အခန်း ၁ မှာ ပြောခဲ့ပါတယ်။ တစ်ခုပြီးမှ တစ်ခုလုပ်မယ်။ ဒါကြောင့် Pseudo code က programming အတွက် ဖြစ်တဲ့ အတွက်ကြောင့် တစ်ဆင့်ပြီး တစ်ဆင့် သွားရပါတယ်။

ဥပမာ

```
READ height of rectangle

READ width of rectangle

COMPUTE area as height times width
```

ဒီ code လေးကို ကြည့်လိုက်တာနဲ့ ဒါဟာ area တွက်ထားတဲ့ code လေး ဆိုတာ နားလည် သွားတယ်။ height ကို လက်ခံမယ်။ width ကို လက်ခံမယ်။ ပြီးရင် height နဲ့ width ကို မြှောက်ပြီး တော့ area ရလာမယ်။

ဒါကို python နဲ့ ပြန်ရေးကြည့်ရအောင်။

```python
height = input("Enter Height Of Rectangle: ")
width = input("Enter Width Of Rectangle: ")
area = int(height) * int(width)
print("Area is ",area)
```

```
Enter Height Of Rectangle: 40
Enter Width Of Rectangle: 80
Area is  3200
```

ကျွန်တော် ရေးထားတဲ့ python code ဟာ programming မတတ်တဲ့ သူ တစ်ယောက်အတွက် ဖတ်လိုက်ရင် နားလည်ဖို့ ခက်ခဲတယ်။ Pseudo code ကတော့ ဘယ်သူ မဆို နားလည်နိုင်အောင် ရေးသားထားပါတယ်။

Input, output, processing တွေ အတွက် အောက်ပါ keyword တွေကို ကျွန်တော်တို့ အသုံးပြုပါတယ်။

- Input: READ, OBTAIN, GET
- Output: PRINT, DISPLAY, SHOW
- Compute: COMPUTE, CALCULATE, DETERMINE
- Initialize: SET, INIT
- Add one: INCREMENT, INCREASE, DECREMENT , DECREASE

စတာတွေကို အသုံးပြုနိုင်ပါတယ်။

Flow Chart

Programming ကို လေ့လာရာမှာ အခြေခံ အနေနဲ့ Pseudo code အပြင် Flow chart ကို ပါ သိထားသင့်တယ်။ အခုအချိန်ထိ coding အကြောင်းကို ကျွန်တော် မရေးသေးပါဘူး။ အခြေခံ အဆင့်တွေ ဖြစ်တဲ့ Flow Chart , Pseudo စတာတွေ ကို နားလည် သွားတဲ့ အခါမှာ programming ကို လွယ်လင့် တကူ စဉ်းစားနိုင်အောင် အထောက် အကူပြုနိုင်ပါတယ်။

Flow Chart ဆိုတာကိုတော့ ကျွန်တော်တို့ တွေ ဘာပြီး ရင် ဘာလုပ်မယ် ဆိုတာကို အဆင့်ဆင့် ပုံတွေနဲ့ ဆွဲပြထားပါတယ်။ Flow Chart ဆွဲရာမှာ သက်ဆိုင်ရာ သတ်မှတ် ချက်တွေ ရှိပါတယ်။ အရင်ဆုံး ဘယ်ပုံတွေက ဘာကို ကိုယ်စားပြုတယ်ဆိုတာကို အောက်မှာ ဖော်ပြထားပါတယ်။

Terminal

Flowchart အစ သို့မဟုတ် အဆုံး စသည့် နေရာတွေ မှာ အသုံးပြုပါတယ်။

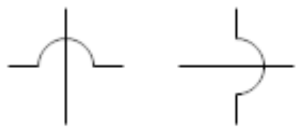
အစကို Start , Begin စသည်ဖြင့် အသုံးပြုပါတယ်။

အဆုံးကို တော့ End , Exit, Return တွေကို အသုံးပြုပါတယ်။

Lines with Arrows

တစ်ခုကနေ နောက်တစ်ခုကို သွားဖို့ ညွှန်ပြထားတာပါ။ ဒါအဆင့် ပြီးရင် ဘယ်ကို သွားမလဲ ဆိုတာကို ညွှန်ပြထားပါတယ်။

Line တွေ ဆွဲတဲ့ အခါမှာ cross ဖြစ်နေရင် မျဉ်းကို မဖြတ်သွားပဲ အခုလို ဂငယ် ပုံလေးနဲ့ ဆွဲပါတယ်။

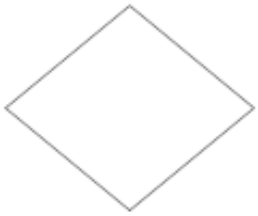


Rectangle

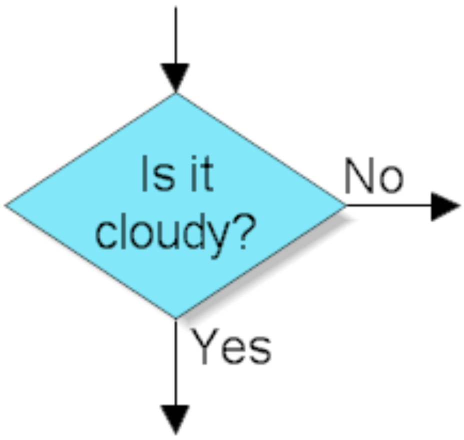
Flowchart မှာ စတုဂံတွေကို process, task, action, operation စသည့်အတွက် အသုံးပြုပါတယ်။ စတုဂံပုံဟာ action တစ်ခုခုလုပ်ဖို့ တစ်ခုခုပြီးမြောက်ဖို့အတွက် ညွှန်ပြထားပါတယ်။

Flowchart မှာ စတုဂံတွေကို process, task, action, operation စသည့်အတွက် အသုံးပြုပါတယ်။ စတုဂံပုံဟာ action တစ်ခုခုလုပ်ဖို့ တစ်ခုခုပြီးမြောက်ဖို့အတွက် ညွှန်ပြထားပါတယ်။

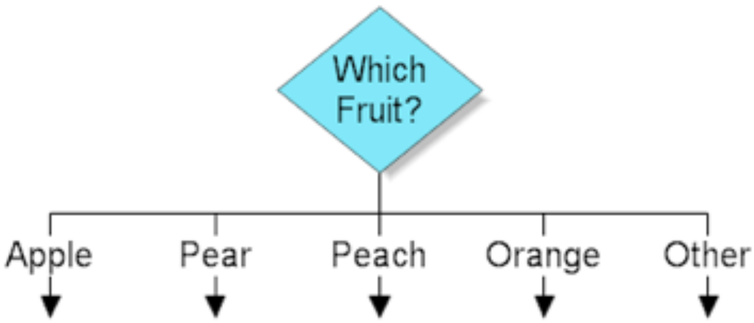
Decision

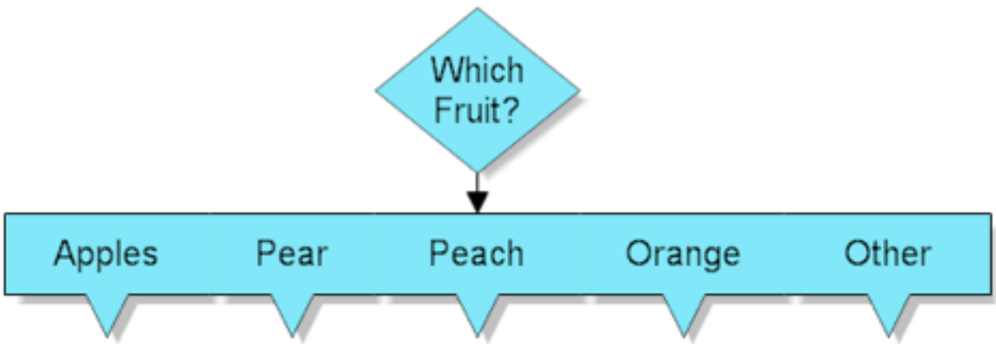


အသုံးပြုသူကို မေးခွန်းမေးပြီးတော့ အဖြေပေါ်မှာ မူတည်ပြီး အလုပ်လုပ်စေချင်တဲ့အခါမှာ decision ကို အသုံးပြုရပါတယ်။ input တစ်ခု ဝင်ပြီးတော့ output မှာ YES,NO ဖြစ်ပါတယ်။ ဒါ ဖြစ်ခဲ့ရင် ဒါလုပ်။ မဟုတ်ခဲ့ရင် ဘာလုပ် စသည်အတွက် အသုံးပြုနိုင်ပါတယ်။



တစ်ခါတစ်လေ တစ်ခုထက် မက ဖြစ်နိုင်တဲ့ အဖြေတွေ အတွက်လည်း အသုံးပြုပါတယ်။





Circle

Flow chat က အရမ်းရှည်သွားရင် သီးသန့် ဆွဲဖို့အတွက် circle ကို အသုံးပြုပါတယ်။ flow chart တစ်ခုနဲ့ တစ်ခုကို connect လုပ်ထားတယ်ဆိုတာကို ဖော်ပြထားသည့် သဘောပါ။ Circle အတွင်းမှာ နာမည်ပါဝင်ပါတယ်။

B မှာ ဆုံးသွားပြီးတော့ တဖက်မှာ B နဲ့ပြန်စထားပါတယ်။

Input/Output

User ဆီကနေ Data ကို လက်ခံတော့မယ်ဆိုရင်တော့ အနားပြိုင် စတုဂံ ကို အသုံးပြုပါတယ်။

ပုံမှန် အခြေခံ အားဖြင့် Flow chart အတွက် ဒါလေးတွေ သိထားရင် လုံလောက်ပါတယ်။ ကျွန်တော်တို့ Example တွေ နဲ့ တချက်ကြည့်ရအောင်။

Flowchart with codes

ကျွန်တော်တို့ အနေနဲ့ အထက်ပါ symbols တွေကို နားလည် ထားဖို့ လိုအပ်ပါတယ်။ သို့မှသာ flowchart ကို ကြည့်ရုံနှင့် ဘယ်အဆင့်တွေပါသလဲ ၊ ဘာတွေလုပ်ရမလဲ ၊ code ဘယ်လိုရေးရမလဲ စတာတွေကို နားလည်ပါလိမ့်မယ်။ Flowchart ကို code မရေးခင် ဦးစွာ ရေးဆွဲပြီး မိမိ ရေးမယ့် system အကြောင်းကို နားလည် စေပါလိမ့်မယ်။

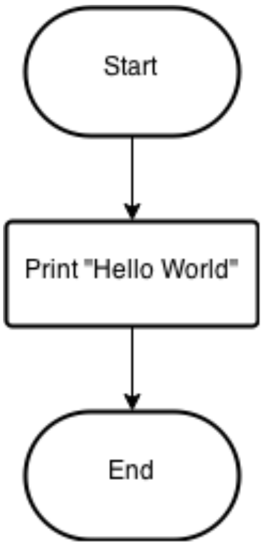
Hello World

Programming ကို စလိုက်တိုင်း ပထမဆုံး ရေးသားကြတဲ့ code ကတော့ Hello World ပါပဲ ။ Screen ပေါ်မှာ ဘယ်လိုပေါ်အောင် ဖန်တီးရမယ် ဆိုတာရဲ့ အစပါပဲ။

Pseudo code နဲ့ ဆိုရင်တော့

```
print("Hello World”)
```

Flow chart နဲ့ ဆိုရင်တော့



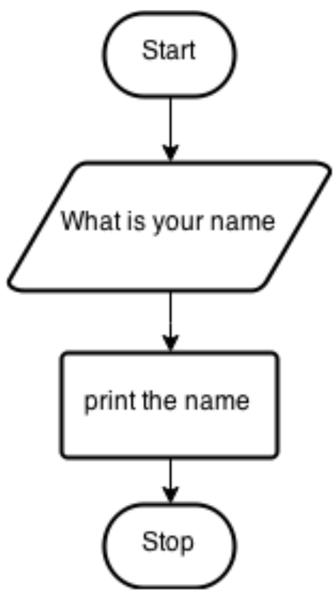
Python နဲ့ဆိုရင်တော့

```
print("Hello World”)
```

What is your name

အခု ကျွန်တော်တို့ user ကို နာမည် ဘယ်လို ခေါ်လဲ မေးမယ်။ ပြီးရင် user ဆီကနေ ပြီးတော့ လက်ခံမယ်။ နာမည်ကို ရိုက်ထည့်ပြီးတာနဲ့ Your name is ဆိုပြီး နာမည်ကို ထုတ်ပြမယ်။

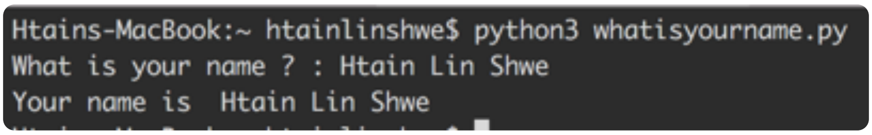
အရင်ဆုံး flow chart ကို ကြည့်ပါ။



ပြီးရင် flow chart အတိုင်း code ကို ရေးထားပါတယ်။

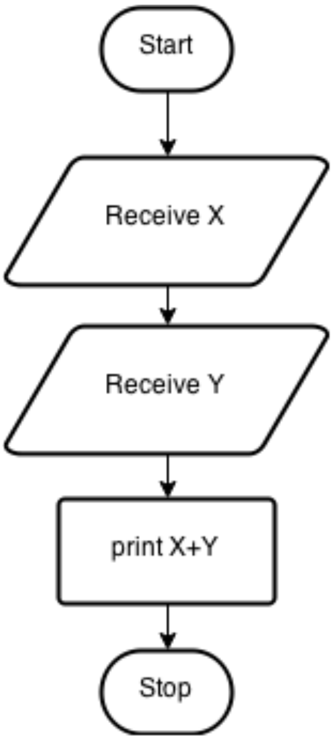
```
username = input("What is your name ? : ")
print("Your name is ",username)
```

အဲဒီ code ကို run လိုက်ရင် နာမည်အရင်မေးပါတယ် ပြီးတော့ ထည့်လိုက်သည့် စာ ကို ပြန်ပြီး တော့ ဖော်ပြပေးပါတယ်။



Sum

အခု ကျွန်တော်တို့တွေ user ဆီကနေ ကိန်း ၂ ခု ကို လက်ခံမယ်။ ပြီးရင် ပေါင်းမယ်။ ပေါင်းပြီး ရ လာတဲ့ အဖြေကို ပြပေးမယ်။ Flowchart ကို ဦးစွာကြည့်လိုက်ပါ။



အခု ကျွန်တော်တို့တွေ user ဆီကနေ ကိန်း ၂ ခု ကို လက်ခံမယ်။ ပြီးရင် ပေါင်းမယ်။ ပေါင်းပြီး ရ လာတဲ့ အဖြေကို ပြပေးမယ်။

Flow chat ထဲမှာ ပြထားသလို လွယ်လွယ်လေးပါပဲ။

```
x = input("Enter first value : ")
y = input("Enter second value : ")
print("X + Y = ", x + y)
```

```
Enter first value : 5
Enter second value : 6
X + Y =  56
```

ဒီ code မှာ ဆိုရင် ကိန်း ၂ လုံးကို ပေါင်းပေမယ့် စာလုံး ၂ လုံး ပေါင်းတာဖြစ်နေတာ ကို တွေ့ရပါလိမ့်မယ်။ ဥပမာ ၅ နဲ့ ၆ ထည့်လိုက်ရင် ၅၆ ထွက်လာပါလိမ့်မယ်။

ကျွန်တော်တို့တွေ အနေနဲ့ နံပတ်ကို လက်ခံမယ်ဆိုတဲ့ အတွက်ကြောင့် အောက်ကလို ပြောင်းရေးပါတယ်။

```
x = input("Enter first value : ")
y = input("Enter second value : ")

try :
    x = int(x)
    y = int(y)

    print("X + Y = ",x + y)

except ValueError:
    print("Please enter number only")
```

```
Enter first value : A
Enter second value : 7
Please enter number only
```

```
Enter first value : 5
Enter second value : 6
X + Y =  11
```

အခု code မှာ **try , except** ဆိုတာကို တွေ့ပါလိမ့်မယ်။ **try** ထဲမှာ Error တစ်ခုခု ဖြစ်တာနဲ့ **except** ကို ရောက်လာပါမယ်။ အခု code ထဲမှာ int ပြောင်းထားတဲ့ နေရာမှာပဲ ပြဿနာ ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့် **except ValueError:** ကို အသုံးပြုထားတာပါ။

အခု ကုဒ်မှာ သတိထားရမှာကတော့ tab တွေပါပဲ။ python ရေးသည့် အခါမှာ block structure ဖြစ်အောင် ရေးသားထားတာကို တွေ့နိုင်ပါတယ်။ **try :** ပြီးတော့ နောက်လိုင်းမှာ tab တစ်ဆင့် ခံထားတာကို တွေ့နိုင်ပါလိမ့်မယ်။ try block အောက်မှာ ဆက်ပြီး ကုဒ်က အလုပ်လုပ်မယ် ဆိုသည့် သဘောပါ။

```
try :
x = int(x)
y = int(y)
```

အထက်ပါ ကုဒ် က အလုပ်မလုပ် ပါဘူး။

Tab ထည့်ထားသည့် ကုဒ်မှ သာ အလုပ်လုပ် မယ်ဆိုတာကို သတိပြုဖို့ လိုပါတယ်။

```
try :
    x = int(x)
    y = int(y)
```

Tab ထည့်ထားသည့် အထက်ပါ ကုဒ်မှ သာ run သည့် အခါမှာ error မတက်ပဲ အလုပ်လုပ်ပါလိမ့်မယ်။

Condition

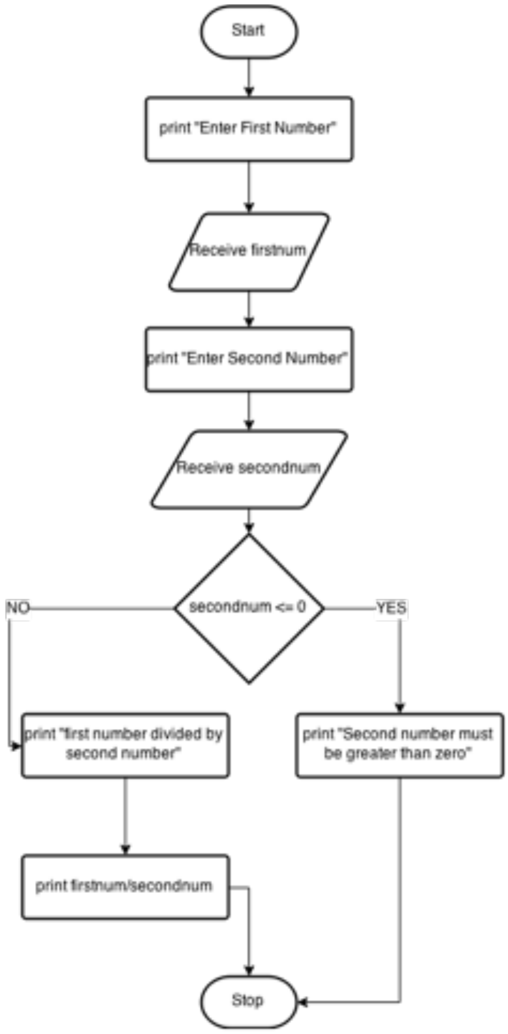
ဒါမှမဟုတ်ခဲ့ရင် ဒါလုပ်မယ် စသည်ဖြင့် အခြေအနေကို စစ်သည့်အရာတွေ အတွက် ကျွန်တော်တို့တွေ if , switch စတဲ့ syntax ကို အသုံးပြုရပါတယ်။ သို့ပေမယ့် Python မှာ switch ကို support မလုပ်ပါဘူး။

အခု ကျွန်တော်တို့ သုည ဆီက ဂဏန်း ၂ ခု ကို လက်ခံမယ်။ ကိန်း ၂ ခု ကို စားမယ်။ ဒုတိယ ကိန်း က သုညဖြစ်နေရင် အသုံးပြုသူကို သုည ထည့်လို့ မရဘူးဆိုရင် error message ပြမယ်။

Pseudo code အရ ဆိုရင်တော့

```
Print "Enter First Number"
READ firstnum

Print "Enter Second Number"
READ secondnum

if secondnum is less than or equal zero
  Print "Second number must be greater than zero"
else
  Print firstnum + "divied by " + secondnum
  Print firstnum/secondnum
```

flowchart ကို ကြည့်ရအောင်။



Python ကို အောက်မှာ ကြည့်ရအောင်

```
firstnum = input("Enter First Number ? : ")
secondnum = input("Enter Second Number ? : ")

try :
    firstnum = int(firstnum)
    secondnum = int(secondnum)

    if secondnum <= 0 :
        print ("Second number must be greater than 0")
    else:
        print (firstnum, " divied by ",secondnum)
        print (firstnum/secondnum)

except ValueError:
    print ("Please enter number only")
```

```
Enter First Number ? : 8
Enter Second Number ? : 0
Second number must be greater than 0
```

အခု code မှာ ဒါမဟုတ်ခဲ့ရင် ဒါလုပ်ဆိုတာပဲ ရှိပါသေးတယ်။
ကျွန်တော်တို့တွေ တစ်ခု ထက် မက condition တွေကို စစ်နိုင်ပါတယ်။ အဲဒီ အတွက် pseudo code နဲ့ flow chart ကိုတော့ မဆွဲပြတော့ပါဘူး။ အောက်က python code လေးကို တချက် လောက် လေ့လာကြည့်ပါ။

```
firstnum = input("Enter First Number ? : ")
secondnum = input("Enter Second Number (between 1-10) ? : ")

try :
    firstnum = int(firstnum)
    secondnum = int(secondnum)

    if secondnum <= 0 :
        print ("Second number must be greater than 0")
    elif secondnum < 1 or secondnum > 10 :
        print ("Second number must be between 1-10")
    else:
        print (firstnum, " divied by ",secondnum)
        print (firstnum/secondnum)

except ValueError:
    print ("Please enter number only")
```

```
Enter First Number ? : 1000
Enter Second Number (between 1-10) ? : 1000000000
Second number must be between 1-10
```

Calculator

အခု ကျွန်တော်တို့တွေ အပေါင်း အနှုတ် အမြှောက် အစား လုပ်တဲ့ calculator လေး တစ်ခု ရေး ရအောင်။

အရင်ဆုံး Pseudo code စရေးပါမယ်။

```
Print "Enter First Number"
READ firstnum

Print "Enter Operator (+,-,*,/)"
READ operator

Print "Enter Second Number"
READ secondnum
output = true
if operator is + then
    result = firstnum + secondnum
else if opeartor is - then
    result = firstnum - secondnum
else if opeartor is * then
    result = firstnum * secondnum
else if operator is / then
    result = firstnum/secondnum
else
    Print "Wrong Operator"
```

```
        output = false

    if output == true
        Print "Result is " , result
```

Code က တော့ ရှင်းရှင်းလေးပါ။ ကျွန်တော်တို့တွေ နံပတ် ၂ ခု လက်ခံမယ်။ ပြီးရင် Operator ကို လက်ခံမယ်။ operator ပေါ်မှာ မူတည်ပြီးတော့ result ကို ထုတ်ပြမယ်။

Operator က + - * / ထဲက မဟုတ်ရင် မထုတ်ပြပါဘူး။ အဲဒီ အတွက် ကျွန်တော်တို့တွေ boolean variable ကို အသုံးပြုပါတယ်။ output ကို ထုတ်ပြမယ် ဆိုပြီး output = true ဆိုပြီး ရေးထားတာပါ။ ဒါပေမယ့် Operator မှားနေရင် result ကို ထုတ်ပြလို့ မရတော့ပါဘူး။ ဒါကြောင့် false ပြောင်းလိုက်တာ ကို တွေ့ရပါလိမ့်မယ်။

Python နဲ့ရေးကြည့်ရအောင်။

```
    x = input("Enter first value : ")
    y = input("Enter second value : ")
    op = input("Operator (+ - * /) : ")
    try :
        x = int(x)
        y = int(y)

        output = True
        if op == "+" :
            result = x+y
        elif op == "-" :
            result = x-y
        elif op == "*" :
            result = x*y
        elif op == "/" :
            result = x/y
        else :
            output = False
            print("Wrong Operator")

        if output :
            print("Result is ",result)


    except ValueError:
        print("Please enter number only")
        print(ValueError);
```

Pseudo code အတိုင်းပါပဲ။ boolean value ကို true ဖြစ်မဖြစ် ကို output == True နဲ့ စစ်မနေတော့ပါဘူး။ if condition က true နဲ့ false အတွက်ပါပဲ။ output က true value ဆိုရင် အလုပ်လုပ်မယ်။ false value ဆိုရင် အလုပ်မလုပ်ပါဘူး။

လေ့ကျင့်ခန်း ၂ - ၁

မေးခွန်း ၁။

```
x = input("Enter first value : ")
y = input("Enter second value : ")
op = input("Operator (+ - * /) : ")
try :
    x = int(x)
    y = int(y)

    output = True
    if op == "+" :
        result = x+y
    elif op == "-" :
        result = x-y
    elif op == "*" :
        result = x*y
    elif op == "/" :
        result = x/y
    else :
        output = False
        print("Wrong Operator")

    if output :
        print("Result is ",result)


except ValueError:
    print("Please enter number only")
    print(ValueError);
```

အထက်တွင် ဖော်ပြထားသော Calculator code ကို flow chart ဆွဲပါ။

Looping

Looping ဆိုတာကတော့ ထပ်ခါ ထပ်ခါ လုပ်တဲ့ အခါတွေ မှာ အသုံးပြုပါတယ်။

ဥပမာ ၀ ကနေ ၉ အထိ ကို ထုတ်ပြချင်တယ်။ ဒါဆိုရင်တော့ print ၁၀ ကြောင်းရေးရပါမယ်။ ဒါမှမဟုတ် ၅ ကနေ ၉ အထိ ထုတ်ပြချင်ရင်တော့ ၄ ခါ ရေးရမယ်။

တစ်ခါတစ်လေ user ဆီကနေ နောက်ဆုံး ဂဏန်းကို လက်ခံပြီး အဲဒီ အကြိမ် အရေ အတွက် လိုက် ထုတ်ပြရမယ်။ အဲဒီ အခါမှာ ကျွန်တော်တို့တွေ print ဘယ်နှစ်ကြိမ် ရိုက်ထုတ်ပြရမယ်ဆိုတာကို မသိတော့ဘူး။ အဲဒီလို အခြေအနေတွေ အတွက် Looping ကို အသုံးပြုနိုင်ပါတယ်။

Looping အမျိုးစားက ပုံမှန်အားဖြင့်

- For Loop
- While Loop
- Do While Loop

ဆိုပြီး ၃ မျိုး ရှိပါတယ်။ Python မှာတော့ Do While Loop ကို support မလုပ်ထားပါဘူး။

For Loop

အကြိမ် အရေ အတွက် အတိအကျ ရှိတယ်ဆိုရင်တော့ For Loop ကို အသုံးပြုနိုင်ပါတယ်။ ဘယ်ကနေ ဘယ်ကနေ စပြီး ဘယ် အထိ သိတယ်ဆိုရင်တော့ For Loop ကို အသုံးပြုနိုင်ပါတယ်။

```
# will print 0 to 9
for i in range(10):
    print(i)

print("----")

# will print 5 to 9
for i in range(5,10):
    print (i)
```

အဲဒီ code လေးကို run လိုက်ရင်တော့

```
0
1
2
3
4
5
6
7
8
9
----
5
6
7
8
9
```

ဆိုပြီး ထွက်လာပါမယ်။

range(10) ဆိုတာကတော့ 0 ကနေ 9 အထိ လုပ်မယ် လို့ ဆိုပါတယ်။ range(5,10) ကတော့ 5 ကနေ 9 အထိ အလုပ်လုပ်မယ် လို့ဆိုတာပါ။

အထက်ပါ code အတိုင်းက for နဲ့ loop ပတ်ပြီးတော့ value တွေက i ထဲကို ဝင်သွားတာကို တွေ့နိုင်ပါတယ်။

While Loop

အကြိမ်အရေအတွက်ကို ကျွန်တော်တို့ မသိဘူး ဒါမှမဟုတ် condition တစ်ခုခု ကို ထားပြီးတော့ loop လုပ်မလား မလုပ်ဘူးလား ဆိုတာကို သိချင်တဲ့ အခါမှာတော့ While Loop ကို အသုံးပြုလို့ရပါတယ်။

```
count = 0
```

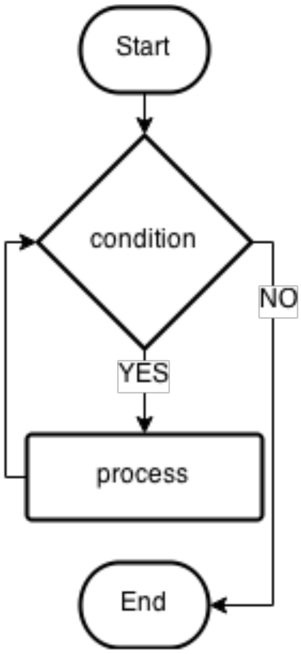
```
while (count < 9):
   print('The count is:', count)
   count = count + 1
```

ထွက်လာသည့် ရလာဒ်ကတော့

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
```

အထက်ပါ code လေး အတိုင်းဆိုရင်တော့ while (count < 9): ဖြစ်သည့် အတွက်ကြောင့် 0 ကနေ 8 အထိ ကို ထုတ်ပြပါလိမ့်မယ်။

While Loop ကို ပိုရှင်းသွားအောင် Flow Chart လေးကို ကြည့်ကြည့်ပါ။



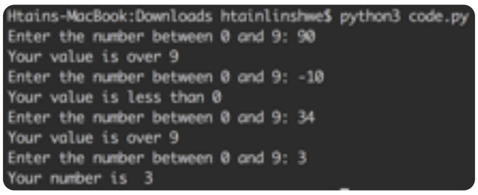
condition က မှန်နေသ၍ ဒါကို ထပ်ခါ ထပ်ခါ လုပ်နေမယ်လို့ ဆိုတာပါ။

ဥပမာ။။ ကျွန်တော်တို့ User ဆီကနေ 0 ကနေ 9 အတွင်း နံပတ် တောင်းတယ်။ 0 ကနေ 9 အတွင်း ဂဏန်း မဟုတ်သ၍ ကျွန်တော်တို့တွေ user ဆီကနေ တောင်းနေမှာပဲ။ အဲဒီလိုမျိုး အကြိမ် အရေအတွက် အတိအကျ မရှိတဲ့ Looping တွေအတွက် while loop ကို အသုံးပြုပါတယ်။

While Loop အသုံးပြုပုံ ဥပမာလေး အောက်က code ကို ကြည့်ပါ။

```
x = False
```

```
while x == False :
    value = input("Enter the number between 0 and 9: ")

    try:
        value = int(value)

        if value > 9:
            print("Your value is over 9")
        elif value < 0:
            print("Your value is less than 0")
        else:
            print("Your number is ",value)
            x = True
    except ValueError:
        print("Please enter the number between 0 and 9")
```

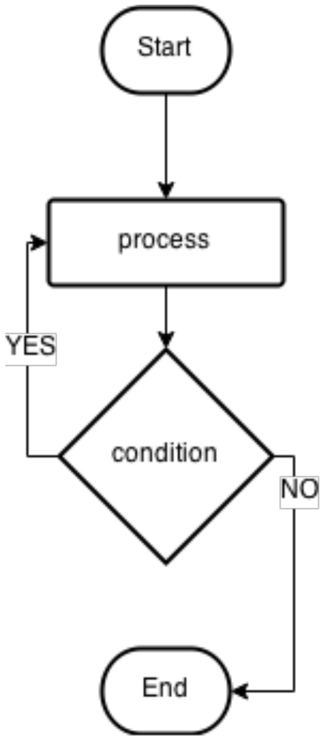


```
Htains-MacBook:Downloads htainlinshwe$ python3 code.py
Enter the number between 0 and 9: 90
Your value is over 9
Enter the number between 0 and 9: -10
Your value is less than 0
Enter the number between 0 and 9: 34
Your value is over 9
Enter the number between 0 and 9: 3
Your number is  3
```

ဒီ code လေးကို စစ်ကြည့်လိုက်ရင် while loop က ဘယ်လို နေရာတွေ မှာ အသုံးဝင် သလဲ ဆို တာကို တွေ့နိုင်ပါတယ်။

Do While Loop

Do while loop က while loop လိုပါပဲ။ ဒါပေမယ့် အနည်းဆုံး တစ်ကြိမ် အလုပ်လုပ်ပါတယ်။ Flowchart ကို ကြည့်ရင် ရှင်းပါလိမ့်မယ်။



Python မှာကတော့ do while loop အတွက် သီးသန့် looping မရှိပါဘူး။

ဒါကြောင့် Java code လေး ကို ကြည့်ကြည့်ပါ။

```
int count = 1;
do {
  System.out.println("Count is: " + count);
  count = count + 1;
} while (count < 11);
```

`System.out.println` က python က print နဲ့တူပါတယ်။

အဲဒီ code မှာဆိုရင်တော့ Count is 1 ကို အရင်ဆုံး ထုတ်ပြပါတယ်။ ပြီးတော့ count ကို ၁ တိုးတယ်။ တိုးပြီးမှ count က ၁၁ ထက် ငယ်နေလား ဆိုပြီး စစ်ပါတယ်။

Python နဲ့ အနီးစပ် ဆုံး ရေးပြရရင်တော့

```
count = 1
print('The count is:', count)
count = count + 1
while (count < 11):
   print('The count is:', count)
   count = count + 1
```

code ကို run ကြည့်ရင်တော့

```
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
The count is: 9
The count is: 10
```

ဆိုပြီး ထွက်လာပါမယ်။

အခု ကျွန်တော်တို့တွေ အောက်ကလိုမျိုး ပုံလေး ထုတ်ကြည့်ရအောင်

```
*
**
***
****
*****
```

ကျွန်တော်တို့ တစ်ကြောင်းဆီ ရိုက်ထုတ်မယ် ဆိုရင် ရပါတယ်။ ဒါပေမယ့် Looping နဲ့ ပတ်ပြီး ထုတ်ကြည့်ရအောင်။

ပထမဆုံးအဆင့်က Looping ဘယ်နှစ်ကြိမ် ပတ်ရမလဲဆိုတာကို စဉ်းစားဖို့ လိုတယ်။

```
1. *
2. *
3. *
4. *
5. *
```

ဒေါင်လိုက်က ငါးခါ ပတ်ရမယ်။ ဒီတော့

```
for x in range(5):
    print("*")
```

```
*
*
*
*
*
```

Loop ၅ ခါ ပတ်ပြီးတော့ print ထုတ်လိုက်တယ်။ * ငါးလိုင်းတော့ ရပြီ။

အလျားလိုက်က တစ်ခါ။ ဒေါင်လိုက်က တစ်ခါ ဆိုတော့ looping ၂ ခါ ပတ်ရမယ်။

```
for x in range(5):
    for k in range(5):
        print("*", end=“")
```

```
*************************
```

ဒီ code လေးမှာ `print("*", end="")` ဆိုပြီး ပါပါတယ်။ ပုံမှန် `print("*")` ဆိုရင် တစ်ခါ ထုတ်ပြီးတိုင်း တစ်လိုင်း ဆင်းသွားတယ်။ နောက်တစ်လိုင်းကို မဆင်းသွားစေချင်သည့် အတွက်ကြောင့် `print("*", end="")` ဆိုပြီး အသုံးပြုတာပါ။

`print("*")` က `print("*", end="\n")` နဲ့ တူပါတယ်။ `print("*", end="")` ဖြစ်သည့် အတွက်ကြောင့် print ထုတ်ပြီးတော့ နောက်တစ်လိုင်း မဆင်းသွားတော့ပါဘူး။

```
for x in range(5):
    for k in range(5):
        print("*", end="")
    print("")
```

```
*****
*****
*****
*****
*****
```

အတွင်းဘက်မှာ ရှိသည့် looping ပြီးသွားတိုင်း လိုင်းတစ်ကြောင်း ဆင်းစေချင်သည့် အတွက်ကြောင့် `print("")` ကို ရေးထားတာပါ။

အခု ကျွန်တော်တို့ လေးထောင့်ပုံစံတော့ ရနေပြီ။ တြိဂံ ပုံစံ ရအောင် စဉ်းစားဖို့လိုပါတယ်။

ပထမ အကြိမ်မှာ ၁ ခါ။ ဒုတိယ အကြိမ်မှာ ၂ ခါ ဖြစ်နေတယ်။ ဒါကြောင့် အတွင်း loop ကို ပြင်ရမယ်။ x က 1 ဖြစ်ရင် *ကို တစ်ခါပဲ ပြ။ 2 ဖြစ်နေရင်* ကို နှစ်ခါပြ။ * ပြရမှာက x နဲ့ တိုက်ရိုက် ဆိုင်နေပါတယ်။ ဒါကြောင့် code ကို ထပ်ပြင်ပါမယ်။

```python
for x in range(5):
    for k in range(x):
        print("*", end="")
    print("")
```

```

*
**
***
****
```

အခုတော့ တြိဂံ ပုံပြနေပြီ။ သို့ပေမယ့် ထိပ်ဆုံး အကြိမ်မှာ empty ဖြစ်နေတာကို တွေ့မိမှာပါ။

ဘာဖြစ်လို့လည်းဆိုတော့ အတွင်း loop က range(0) ဖြစ်နေသည့် အတွက်ကြောင့် ပထမ အကြိမ်မှာ အတွင်း loop ကို မပတ်ပါဘူး။ ဒါကြောင့် 0 အစား 1 ကနေ စပြီး ပတ်ပါမယ်။

```python
for x in range(1,6):
    for k in range(x):
        print("*", end="")
    print("")
```

```
*
**
***
****
*****
```

အခု ဆိုရင် ကျွန်တော်တို့လိုချင်သည့် တြိဂံ ပုံ ရသွားပါပြီ။

လေ့ကျင့်ခန်း ၂ - ၂

မေးခွန်း ၁။

```
total = 0;
for x in range(10):
    total = total + x
print(total)
```

program ကို run လိုက်ရင် Total က

A. 0

B. 10

C. 45

—

မေးခွန်း ၂။

```
total = 0;
for x in range(10):
    total = total + 1
print(total)
```

program ကို run လိုက်ရင် Total က

A. 0

B. 10

C. 45

—

မေးခွန်း ၃။

range(5) ဆိုရင်

A. 0 ကနေ 5 ထိ

B. 0 ကနေ 4 ထိ

C. 1 ကနေ 5 ထိ

D. 1 ကနေ 4 ထိ

—

မေးခွန်း ၄။

While loop က အနည်းဆုံး တစ်ကြိမ် အလုပ်လုပ်တယ်။

A. မှန်
B. မှား

—

မေးခွန်း ၅။

While loop အသုံးပြုဖို့ အခေါက် အရေအတွက် အတိအကျ ရှိရမယ်။

A. မှန်
B. မှား

—

မေးခွန်း ၆။

Fibonacci program လို့ ခေါ်ရအောင်။ ကျွန်တော်တို့တွေ user ဆီက နံပတ် လက်ခံမယ်။ နံပတ် က 5 ဖြစ်ရင် Fibonacci sequence အရ နံပတ် ၅ ခု ထုတ်ပြမယ်။

```
1 1 2 3 5
```

7 ဖြစ်ခဲ့ရင်တော့

```
1 1 2 3 5 8 13
```

လို့ ထုတ်ပြမယ်။

Fibonacci sequence ဆိုတာကတော့ ရှေ့က နံပတ် ၂ ခု ကို ပေါင်းပြီးတော့ နောက်ထပ် ဂဏန်း တစ်ခု ရပါတယ်။

—

မေးခွန်း ၇။

Even/odd စစ်ထုတ်တဲ့ program ပါ။ user ဆီကနေ ဂဏန်း လက်ခံမယ်။ ပြီးရင် 1 ကနေ စပြီး တော့ even ဖြစ်လား odd ဖြစ်လား ဆိုပြီး ထုတ်ပြရမယ်။

ဥပမာ user က 3 လို့ ရိုက်လိုက်ရင်

```
1 is Odd
2 is Even
3 is Odd
```

ဆိုပြီး ထုတ်ပြမယ်။

တကယ်လို့ 5 လို့ ရိုက်လိုက်ရင်

```
1 is Odd
2 is Even
3 is Odd
4 is Even
5 is Odd
```

ဆိုပြီး ထုတ်ပြရပါမယ်။

Array ဆိုတာ ဘာလဲ ?

Array ဆိုတာကတော့ variable တွေ အများကြီးကို သိမ်းထားတဲ့ variable တစ်ခုပါပဲ။ Array က variable တွေကို အခန်းနဲ့ သိမ်းပါတယ်။ နားလည်အောင် ပြောရရင်တော့ variable ဆိုတာက ရေခွက် တစ်ခု ဆိုပါတော့။ အဲဒီ ရေခွက်ကို ခဲလိုက်ရင် ရေခဲ တစ်ခုပဲ ရမယ်။ Array ဆိုတာ ကတော့ ရေခဲ ခဲတဲ့ ခွက် (ice cube trays) နဲ့တူပါတယ်။ ရေခဲ ခဲ ဖို့ အတွက် အကန့်လေးတွေ ပါ တယ်။ ရေခဲခဲ လိုက်ရင် တစ်ခု ထက် မက ၊ အခန်း ရှိသလောက် ရေခဲ ရနိုင်ပါတယ်။

Array က အဲဒီလိုပါပဲ။ ကျွန်တော်တို့ computer memory ပေါ်မှာ သိမ်းဖို့ အတွက် အခန်းလေးတွေ ယူလိုက်တယ်။ ပြီးတော့ အခန်းတွေ ထဲမှာ variable တွေ ထည့်ပြီး သိမ်းတယ်။ String array ဆို ရင်တော့ String value တွေ ပဲ ထည့်တဲ့ အခန်းပေါ့။ Integer array ဆိုရင်တော့ Integer value တွေ ပဲ ထည့်တဲ့ အခန်းတွေပေါ့။

အခန်းတွေကို ရေတွက်တဲ့ အခါမှာတော့ သုည ကနေ စပါတယ်။ အခန်း ၃ ခန်း ရှိရင်တော့ 0,1,2 ဆိုပြီး ရေတွက်ပါတယ်။

ဥပမာ code လေးကို ကြည့်ရအောင်

```
list = [1,5,2,7,8,9,200,155]

# First Room
print(list[0])

# 9 will show
print(list[5])

# Last Room
print(list[7])
```

```
1
9
155
```

ဒီ code လေးမှာ ကျွန်တော်တို့တွေ # ဆိုတာလေး တွေ့နိုင်ပါတယ်။ Comment အနေနဲ့ အသုံးပြုပါတယ်။ Comment ဆိုတာကတော့ မိမိ နှစ်သက်ရာ စာရေးပြီးတော့ မှတ်သားထားသည့် သဘောမျိုးပါ။ code အနေနဲ့ ပါဝင်ခြင်း မဟုတ်ပဲ remark သဘောမျိုးအနေနဲ့ ရေးသားခြင်းပါ။ အခု code မှာ ဆိုရင်တော့ `# First Room` ဆိုတာကတော့ အောက်က code က ပထမ အခန်းကို ခေါ်တယ်ဆိုတာ သိအောင် ပြန်ရှင်းပြထားပေးသည့် သဘောပါ။

Comment ကို ရေးသားထားခြင်းဖြင့် နောင်တချိန် code တွေ ပြန်ဖတ်သည့် အခါမှာ ဘာလို့ ဒီလို ရေးသားထားသည် ကို နားလည် စေပါတယ်။ code ကို အခြား လူတစ်ယောက် က ဖတ်ကြည့်သည့် အခါမှာ ဘာအတွက် ရေးသားထားသည် ဆိုတာကို နားလည်စေချင်သည့် အခါမှာလည်း ထည့်သွင်းရေးသားပါသည်။ အခု code လေးကို ပြန်ပြီး ရှင်းပြပါမယ်။

အထက်ပါ code မှာဆိုရင် အခန်းပေါင်း 8 ခန်း ရှိပါတယ်။ အခန်း တစ်ခုခြင်းဆီမှာ နံပတ်တွေ ထည့်ထားပါတယ်။

ပထမ အခန်းကို လိုချင်တဲ့ အခါမှာ list[0] လို့ ဆိုပြီး ခေါ်သုံးထားပါတယ်။ အခြား အခန်းတွေ ကိုတော့ နံပတ်လိုက် ခေါ်ထားတာကို တွေ့နိုင်ပါတယ်။

```
list = [1,5,2,7,8,9,200,155]

#Total Room
print("Total room in array is",len(list))
```

```
Total room in array is 8
```

ဒီ code မှာဆိုရင်တော့ Array တစ်ခုမှာ အခန်း ဘယ်လောက်ရှိလဲဆိုတာကို ထုတ်ပြထားတာပါ။

ကျွန်တော်တို့တွေ ထပ်ပြီးတော့ အခန်းထဲမှာ ရှိတဲ့ data တွေကို looping ပတ်ပြီး ထုတ်ကြည့်ရအောင်။

```
list = [1,5,2,7,8,9,200,155]

for i in range(len(list)):
    print(list[i])
```

```
1
5
2
7
8
9
200
155
```

နောက်ထပ် တဆင့် အနေနဲ့ အခန်းထဲမှာရှိတဲ့ နံပတ်တွေ အားလုံးပေါင်း ရလဒ်ကို ထုတ်ကြည့်ရအောင်

```
list = [1,5,2,7,8,9,200,155]

x = 0
for i in range(len(list)):
    x = x + list[i]

print("Total:",x)
```

```
Total: 387
```

ကျွန်တော်တို့တွေ array အတွက် loop ကို အောက်က code လိုမျိုး python မှာ loop ပတ်လို့ရပါတယ်။ တခြား langauge တွေမှာ ဆိုရင်တော့ for each loop လို့ခေါ်ပါတယ်။

```
list = [1,5,2,7,8,9,200,155]

x = 0
for i in list:
    x = x + i

print("Total:",x)
```

```
Total: 387
```

အဓိပ္ပာယ်ကတော့ array အခန်းတွေက အစကနေ အဆုံး ထိ loop ပတ်မယ်။ ရောက်နေတဲ့ index ထဲက data ကို i ထဲကို ထည့်မယ်။ အဲဒီ code မှာ ကျွန်တော်တို့တွေ လက်ရှိ index ကို မသိနိုင်ပါဘူး။ for each loop ပတ်တဲ့ အခါမှာ လက်ရှိ index ပါ သိအောင် အောက်က code လိုမျိုး ရေးလို့ရပါတယ်။

```
list = [1,5,2,7,8,9,200,155]

for (i,item) in enumerate(list):
    print("Index :",i,"And Value :",item)
```

```
Index : 0 And Value : 1
Index : 1 And Value : 5
Index : 2 And Value : 2
Index : 3 And Value : 7
Index : 4 And Value : 8
Index : 5 And Value : 9
Index : 6 And Value : 200
Index : 7 And Value : 155
```

Immutable And Mutable

Array မှာ ၂ မျိုး ရှိတယ်။ မပြောင်းလို့ရတဲ့ Array နဲ့ ပြောင်းလဲလို့ရတဲ့ Array Type ၂ ခု ရှိပါတယ်။

Python မှာကတော့ tuple နဲ့ list ဆိုပြီး ၂ မျိုး ရှိတယ်။

tuple ကို လက်သည်းကွင်း နဲ့ ရေးပြီးတော့ list ကိုတော့ လေးထောင့် ကွင်းနဲ့ ရေးပါတယ်။

```
t = (1,2,3) # immutable
l = [1,2,3] # mutable
```

Tuple နဲ့ ရေးထားရင်တော့ အခန်း တွေကို ပြောင်းလဲလို့ မရပါဘူး။ ဒါပေမယ့် list ကတော့ အခန်းထဲက data တွေကို ပြောင်းလို့ ရသလို အခန်း အသစ်တွေ ထည့်တာ ဖျက်တာ စတာတွေကို လုပ်လို့ရပါတယ်။

Finding Max Number

အခု ကျွန်တော်တို့တွေ array အခန်းထဲက အကြီးဆုံး ဂဏန်းကို ရှာတဲ့ code လေး ရေးကြည့်ရအောင်။

```
list = [1048,1255,2125,1050,2506,1236,2010,1055]

maxnumber = list[0]

for x in list:
    if maxnumber < x :
        maxnumber = x

print("MAX number in array is",maxnumber)
```

ဒီ code လေးကို ကြည့်ကြည့်ပါ ရှင်းရှင်းလေးပါ။ ပထမဆုံး အခန်းကို အကြီးဆုံးလို့ သတ်မှတ်လိုက်တယ်။ ပြီးရင် အခြား အခန်းတွေနဲ့ တိုက်စစ်တယ်။ ကြီးတဲ့ ကောင်ထဲကို maxnumber ဆိုပြီး ထည့်ထည့်သွင်းထားတယ်။

နောက်ဆုံး looping ပြီးသွားရင် အကြီးဆုံး ဂဏန်းကို ကျွန်တော်တို့တွေ သိရပြီပေါ့။

လေ့ကျင့်ခန်း ၂-၃

မေးခွန်း ၁။

Max Number လိုမျိုး အငယ်ဆုံး ဂဏန်းကို ရှာတဲ့ code ရေးကြည့်ပါ။

မေးခွန်း ၂။

```
list = [1,5,2,7,8,9,200,155]
print(len(list))
```

အဖြေသည်

A. ၉
B. ၈
C. ၇

မေးခွန်း ၃။

Array [3,4,1,2,9,7] ဆိုပြီး ရှိပါသည်။ user ဆီက နံပတ်ကို လက်ခံပြီး array အခန်းထဲတွေ တွေ့မတွေ့ user ကို print ထုတ်ပြပါမည်။ တွေ့ခဲ့ပါက အခန်း ဘယ်လောက်မှာ တွေ့ခဲ့သည်ကို print ထုတ်ပြပါမည်။

မေးခွန်း ၄။

Max number ကို ရှာပါ။ ဘယ်နံပတ်က အကြီးဆုံးလဲ။ ဘယ်အခန်းမှာ ရှိတာလဲ ဆိုတာကို print ရိုက် ပြပါ။

Function

ကျွန်တော်တို့တွေ programming နဲ့ပတ်သက်ပြီးတော့ အတော်လေးကို သိပြီးပါပြီ။ အခု အပိုင်းမှာတော့ function အကြောင်းကို ပြောပြပါမယ်။ ကျွန်တော်တို့ ထပ်ခါထပ်ခါ ခေါ်လုပ်နေရတဲ့ ကိစ္စတွေမှာ ကျွန်တော်တို့တွေ looping သုံးခဲ့ပါတယ်။ အဲလိုပါပဲ code တွေ ထပ်နေရင် ဒါမှမဟုတ် ပိုပြီးတော့ အဓိပ္ပာယ် ပြည့်စုံအောင် ကျွန်တော်တို့တွေ function ခွဲရေးပါတယ်။

```
def printHello():
    print("HELLO")

printHello()
```

ဒီ code လေးမှာ ဆိုရင် ကျွန်တော်တို့တွေ printHello ဆိုတဲ့ function လေး ရေးထားတာကို တွေ့နိုင်ပါတယ်။ Hello ကို ခဏခဏ print ရိုက်နေမယ့် အစား printHello ဆိုတဲ့ funtion လေးကို ခေါ်လိုက်တာနဲ့ HELLO ဆိုပြီး ထုတ်ပြပေးနေမှာပါ။

Python မှာ function ကို ရေးတဲ့ အခါမှာတော့ def နဲ့ စတယ်။ ပြီးတော့ function နာမည်။ အခု ဥပမာမှာ printHello က function နာမည်ပါ။

လက်သည်းကွင်းစ နဲ့ ကွင်းပိတ်ကို တွေ့မှာပါ။ အဲဒါကတော့ function စီကို data တွေ ပို့ဖို့အတွက် အသုံးပြုပါတယ်။ ဘာ data မှ မထည့်ပေးလိုက်ချင်ဘူးဆိုရင်တော့ () နဲ့ အသုံးပြုနိုင်ပါတယ်။

```
def printHello(val):
    print("HELLO",val)

printHello("WORLD")
printHello("Python")
```

ဒီ ဥပမာမှာတော့ World ဆိုပြီး value လေးကို function ဆီ ပို့ပေးလိုက်ပါတယ်။ function ကနေ Hello ကို ရှေ့မှာ ထားပြီးတော့ HELLO World ဆိုပြီး ထုတ်ပေးပါတယ်။ နောက်တစ်ခေါက်မှာတော့ Python ဆိုတာကို ပို့ပေးလိုက်တဲ့ အတွက HELLO Python ဆိုပြီး ထပ်ထွက်လာပါတယ်။ တူညီနေတဲ့ code ၂ ခေါက်ရေးနေမယ့် အစား function နဲ့ ခွဲထုတ်လိုက်တာပါ။

```
def sum(val1,val2) :
    return val1+val2

print("SUM : ", sum(1,4))
```

ဒီ code လေးကို ကြည့်ကြည့်ပါ။ ပုံမှန် အပေါင်းကို ကျွန်တော်တို့တွေ function ခွဲထုတ်ပြီးတော့ ရေးထားတာပါ။ ကျွန်တော်တို့တွေ 1+4 ဆိုပြီး လွယ်လင့်တကူ ရေးလို့ရပါတယ်။ သို့ပေမယ့် ပေါင်းတယ်ဆိုတဲ့ အဓိပ္ပာယ်သက်ရောက်အောင် sum ဆိုပြီး function သီးသန့် ခွဲထုတ်လိုက်ပါတယ်။ ကိန်း ၂ လုံး ကို ပေါင်းပြီးတော့ ရလဒ်ကို ပြန်ပေးထားပါတယ်။ ကျွန်တော်တို့တွေ function မှာ parameter တစ်ခုမှ မပို့ပဲ နေလို့ရသလို တစ်ခု သို့မဟုတ် တစ်ခု ထက် မက ပို့လို့ရပါတယ်။

အခု ဆိုရင်တော့ function ကို နည်းနည်း သဘောပေါက်လောက်ပါပြီ။

နောက်ထပ် ဥပမာ ကြည့်ရအောင်။ Array တုန်းက max number ကို ကျွန်တော်တို့တွေ ရေးခဲ့ဖူးပါတယ်။

```
list = [1048,1255,2125,1050,2506,1236,2010,1055]

maxnumber = list[0]

for x in list:
    if maxnumber < x :
        maxnumber = x

print("MAX number in array is”,maxnumber)
```

အဲဒီမှာ list ကသာ ၂ ခု ရှိမယ်ဆိုပါစို့။ ကျွန်တော်တို့တွေ max number ရ ဖို့အတွက် ဒီ code ကို ပဲ ၂ ခေါက်ထပ်ရေးရမယ်။

```
list = [1048,1255,2125,1050,2506,1236,2010,1055]

maxnumber = list[0]

for x in list:
```

```
    if maxnumber < x :
        maxnumber = x

print("MAX number in array list is",maxnumber)

list2 = [1,2,5,6,9,3,2]

maxnumber = list2[0]

for x in list2:
    if maxnumber < x :
        maxnumber = x

print("MAX number in array list 2 is",maxnumber)
```

တကယ်လို့ Array ၃ ခု အတွက် ဆိုရင် ဒီ code ကို ပဲ ၃ ခေါက်ထပ်ရေးနေရမယ်။ အဲလို ထပ်ခါ ထပ်ခါ မရေးရအောင် ကျွန်တော်တို့တွေ function ခွဲပြီး ရေးလို့ရပါတယ်။

```
def max(lst):
    maxnumber = lst[0]

    for x in lst:
        if maxnumber < x :
            maxnumber = x
    return maxnumber


list = [1048,1255,2125,1050,2506,1236,2010,1055]
list2 = [1,2,5,6,9,3,2]

print("MAX number in array list is",max(list))
print("MAX number in array list2 is",max(list2))
```

အဲဒီမှာ code က ပိုပြီး ရှင်းသွားတာကို တွေ့နိုင်ပါတယ်။ Array ဘယ်နှစ်ခုပဲ ဖြစ်ဖြစ် ဒီ function ကို ခေါ်ရုံပါပဲ။ function ကို သုံးချင်းအားဖြင့် ထပ်ခါထပ်ခါ ခေါ်နေတာတွေကို သက်သာသွားစေပါတယ်။

လေ့ကျင့်ခန်း ၂-၄

မေးခွန်း ၁။

minus ဆိုသည့် function ရေးပြပါ။ ဂဏန်း ၂ လုံး ပို့လိုက်ပြီး ရလာဒ်ကို return ပြန်ပေးရမည်။

မေးခွန်း ၂။

triangle_star ဆိုတဲ့ function ကို ရေးပါ။ user ဆီက နံပတ်တောင်းပါ။ 3 လို့ရိုက်ရင် triangle_star(3) ဆိုပြီး ပို့ပေးပါ။ triangle_star မှ အောက်ပါ အတိုင်း ရိုက်ထုတ်ပြပါ။

```
*
**
```

***

အကယ်၍ 5 လို့ ရိုက်ထည့်လျှင် ၅ လိုင်း ထုတ်ပြပါမည်။

အခန်း ၃ ။ Object Oriented

Python ဟာ Object Oriented Programming Language တစ်ခုပါ။ နောက်လာမယ့် အခန်းမှာ Stack , Queue စတာတွေကို python ကို အသုံးပြုပြီး ရေးသားမှာ ဖြစ်သည့်အတွက်ကြောင့် OOP ကို အနည်းအကျဉ်းတော့ သိထားဖို့ လိုပါတယ်။ OOP အကြောင်းကို ပြောမယ်ဆိုရင်တော့ ဒီစာအုပ်မှာ မလောက်ပါဘူး။ ဒီစာအုပ်က programming အခြေခံအတွက် ဖြစ်တဲ့ အတွက်ကြောင့် နောက်အခန်းတွေ အတွက် OOP အကြောင်း အနည်းငယ်မျှသာ ဖော်ပြသွားပါမယ်။

Object Oriented ဆိုတာကတော့ programming ကို ရေးသားရာမှာ သက်ဆိုင်ရာ အစုလိုက် ခွဲထုတ်ပြီး ရေးသားထားတာပါ။ ဥပမာ။။ လူတစ်ယောက် ဆိုပါဆို့။ လူ ဟာ Object တစ်ခုပါ။ လူတစ်ယောက်မှာ ကိုယ်ပိုင် function တွေ ရှိမယ်။ ပိုင်ဆိုင်တဲ့ properties တွေ ရှိမယ်။ properties တွေကတော့ မျက်လုံး ၊ ပါးစပ် စတာတွေ ဖြစ်ပြီးတော့ function တွေကတော့ စကားပြောတာ အိပ်တာ စတာတွေပါ။

နောက်ထပ် နားလည်အောင် ထပ်ပြီး ရှင်းပြရရင်တော့ ကားတစ်စီးကို object လို့ သတ်မှတ်လိုက်ပါ။ သူ့မှာ ဘာ properties တွေ ရှိမလဲ။ ဘာ function တွေ ရှိမလဲ။ ရှိနိုင်တဲ့ properties တွေကတော့ ဘီး ၄ ခု ရှိမယ်။ တံခါး ရှိမယ်။ function တွေကတော့ ရှေ့သွားတာပါမယ်။ နောက်သွားတာပါမယ်။ ဘယ်ကွေ့ ညာကွေ့တွေ ပါမယ်။

Programming မှာ ဖန်တီးတဲ့ အခါမှာလည်း Object ကို ဖန်တီးတယ်။ ပြီးရင် properties တွေ function တွေကို သက်ဆိုင်ရာ Object မှာ ထည့်သွင်းပါတယ်။

Classes

Class ဆိုတာကတော့ object တစ်ခု ဖန်တီးဖို့ အတွက် user-defined လုပ်ထားတာပါ ။ class ထဲမှာ attributes , class varilables , method စတာတွေ ပါဝင်ပါတယ်။

Defining a Class

Python မှာ class ကို ဖန်တီးတော့မယ်ဆိုရင်

```
class ClassName:
```

ClassName ကတော့ နှစ်သက်ရာ class နာမည်ပါ။ ဥပမာ

```
class animal:
```

အဲဒါဆိုရင်တော့ animal ဆိုတဲ့ class တည်ဆောက်ပြီးပါပြီ။ ပြီးရင် class ထဲမှာ ပါဝင်မယ် variable ကို သတ်မှတ်ပါမယ်။ animal ဖြစ်တဲ့ အတွက်ကြောင့် ခြေထောက် ၄ ချောင်းဖြစ်နိုင် သလို ၂ ချောင်းတည်းရှိတဲ့ တိရစ္ဆာန်လည်း ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့်

```
class animal:
    number_of_legs = 0
```

Instances

Class ကြီး သက်သက်ဆိုရင်တော့ class တစ်ခု ကို ဖန်တီးထားတာပဲ ရှိပါတယ်။ class ကို အသုံးပြုချင်ရင်တော့ instance တစ်ခုကို တည်ဆောက်ရပါတယ်။ တည်ဆောက်ပြီးသား instance ကို variable ထဲမှာ သိမ်းရပါတယ်။ class ထဲက variable တွေကို ခေါ်ယူ အသုံးပြုလိုရင် တော့ instance ဆောက်ထားတဲ့ varilable ထဲက နေ တဆင့် ခေါ်ယူ အသုံးပြုနိုင်ပါတယ်။

```
class animal:
    number_of_legs = 0

dog = animal()
```

အခု ဆိုရင် dog variable က animal object တစ်ခု ဖြစ်သွားပါပြီ။ animal ထဲက variable ကို ခေါ် ယူ အသုံးပြုလို ရင်တော့

```
dog.number_of_legs
```

အခု ကျွန်တော်တို့တွေ variable ကို အသုံးပြုကြည့်ရအောင်။

```
class animal:
    number_of_legs = 0

dog = animal()
dog.number_of_legs = 4

print ("Dog has {} legs".format(dog.number_of_legs))

'''
you can also write like that

print("Dog has " + str(dog.number_of_legs) + " legs")
'''
```

အထက်ပါ code မှာ ''' နဲ့ စပြီးတော့ ''' နဲ့ ဆုံးထားတာကို တွေ့ရပါလိမ့်မယ်။ အဲဒီ နှစ်ခု ကြားမှာတော့ comment တွေ ရေးသားပါတယ်။ Multi line comment အတွက် ကျွန်တော်တို့တွေ ''' နဲ့ စပြီးတော့ ''' နဲ့ ဆုံးသည့် ပုံစံ ဖြင့်ရေးသားနိုင်ပါသည်။

နောက်ထပ် ဥပမာ တစ်ခု ထပ်စမ်း ကြည့်ရအောင်။

```
class animal:
    number_of_legs = 0

dog = animal()
dog.number_of_legs = 4

print ("Dog has {} legs".format(dog.number_of_legs))

chicken = animal()
chicken.number_of_legs = 2

print ("Chicken has {} legs".format(chicken.number_of_legs))
```

ကျွန်တော်တို့တွေ dog နဲ့ chicken object ၂ ခုကို တည်ဆောက်ပြီးတော့ ခြေထောက် ဘယ်နှစ်ချောင်း ရှိတယ်ဆိုတာကို print ထုတ်ပြပေးထားပါတယ်။

Function in Class

Function တွေကို class ထဲမှာ ကြေငြာလို့ ရပါတယ်။

```
class animal:
    number_of_legs = 0
    def sleep(slef) :
        print("zzz")

dog = animal()
dog.sleep()
```

class ထဲမှာ function ကို ဖန်တီးတဲ့ အခါမှာ (self) ဆိုပြီး ထည့်ထားတာကို တွေ့ရပါမယ်။ အဲဒီလို ထည့်ထားမှသာ class ထဲမှာ ရှိတဲ့ variable ကို လှမ်းခေါ်လို့ ရပါလိမ့်မယ်။

Constructor

class တစ်ခု စပြီး ဆောက်သည့် အခါမှာ တန်ဖိုးတွေကို ထည့်လိုက်ချင်တယ်။ ဒါမှမဟုတ် တန်ဖိုးတစ်ခုခု ကို ကြိုထည့်ထား မယ်ဆိုရင် def **init**(self) ကို အသုံးပြုနိုင်ပါတယ်။ ကျွန်တော်တို့က constructor လို့ ခေါ်ပါတယ်။

```
class animal:
    number_of_legs = 0

    def __init__(self):
```

```
        self.number_of_legs = 4

    def sleep(slef) :
        print("zzz")

dog = animal()
print ("Dog has {} legs".format(dog.number_of_legs))
```

ဒီ code မှာ ဆိုရင်တော့ Dog has 4 legs ဆိုပြီး ရပါလိမ့်မယ်။

အကယ်၍ ကျွန်တော်တို့က အခြား value ကို assign လုပ်ချင်တယ်ဆိုရင် အောက်ကလို ရေးနိုင်ပါတယ်။

```
class animal:
    number_of_legs = 0

    def __init__(self,legs = 4):
        self.number_of_legs = legs

    def sleep(slef) :
        print("zzz")

dog = animal()
print ("Dog has {} legs".format(dog.number_of_legs))
spider = animal(8)
print ("Spider has {} legs".format(spider.number_of_legs))
```

```
def __init__(self,legs = 4):
```

ဆိုတာကတော့ legs ကို default value အနေနဲ့ 4 ဆိုပြီး ပေးထားပါတယ်။ ဒါကြောင့်

```
dog = animal()
```

ခေါ်သည့် အခါမှာတော့ default value 4 နှင့် ဝင်သွားတာပါ။

```
spider = animal(8)
```

ဆိုသည့် အခါမှာတော့ legs က 8 ဆိုပြီး ဝင်သွားပါတယ်။

Inheritance

ပြီးခဲ့တဲ့ အခန်းကတော့ ကျွန်တော်တို့တွေ class အကြောင်း အနည်းငယ် သိပြီးပါပြီ။ အခု အခန်းမှာတော့ Inheritance အကြောင်း အနည်းငယ် ဖော်ပြပေးပါမယ်။

Inheritance ဆိုတာကတော့ အမွေဆက်ခံခြင်း တနည်းအားဖြင့် ပင်မ class ရဲ့ child class ဖန်တီးခြင်းပါပဲ။ ပြီးခဲ့တဲ့ အခန်းက animal class ကို ကျွန်တော်တို့ ဖန်တီးပြီးတော့ dog object တွေ ဆောက်ခဲ့ကြတယ်။ အခု ကျွန်တော်တို့ dog class ဖန်တီးပါမယ်။ dog ဆိုတာက animal ဆိုတဲ့ class ရဲ့ child ပါပဲ။

```
class animal:
    number_of_legs = 0

    def sleep(slef) :
        print("zzz")

    def count_legs(self) :
        print("I have {} legs".format(self.number_of_legs))

class dog(animal):
    def __init__(self):
        self.number_of_legs = 4

    def bark(self):
        print("Woof")

mydog = dog()
mydog.bark();
mydog.sleep();
```

ဒီ code မှာ ဆိုရင်တော့ Woff နဲ့ zzz ကို တွေ့နိုင်ပါတယ်။

dog class ဟာ သူ့ parent မှာ လုပ်လို့ရတဲ့ function တွေကို ခေါ်ပြီး အသုံးပြုနိုင်တာကို တွေ့နိုင်ပါလိမ့်မယ်။
ဒီ code မှာတော့ constructor မှာ self.number_of_legs = 4 ဆိုပြီး ထည့်ထားပါတယ်။ ဒါကြောင့် ခွေးဟာ ခြေ ၄ ချောင်း ဖြစ်တယ် ဆိုတာကို class ဆောက်ကတည်းက ပြောထားလိုက်တာပါ။

အခြား language တွေမှာတော့ class ရဲ့ function တွေကို private , public , protected ဆိုပြီး ပေးထားလို့ ရပေမယ့် python language မှာတော့ အဲဒီ feature မပါဝင်ပါဘူး။

Object Oriented နဲ့ ပတ်သက်ပြီးတော့ ဒီစာအုပ်မှာတော့ ဒီလောက်ပါပဲ။ နောက်ထပ် အခန်းတွေမှာ လက်တွေ့တည်ဆောက်ရင်းနဲ့ OOP ကို ပိုပြီး နားလည်လာပါလိမ့်မယ်။

လေ့ကျင့်ခန်း ၃

မေးခွန်း ၁။

ပေါင်းနှုတ်မြှောက်စား ပါဝင်သည့် MyMath class တစ်ခုကို တည်ဆောက်ပါ။

မေးခွန်း ၂။

MyMath class ကို base ယူပြီး MyMathExt ဆိုသည့် class ကို ဆောက်ပါ။ ၂ ထပ် ၊ ၃ ထပ် တွက်ရန် ထည့်သွင်းပါ။

အခန်း ၄ ။ Stack

ဒီ အခန်း မှာ Stack နဲ့ ပတ်သက်ပြီးတော့ stack ကို အဓိက ရေးသားထားပါတယ်။ နောက်ထပ် အခန်းတွေမှာ Queue , List, Recursion , Sorting , Searching စသည်တို့ကို ဖော်ပြပေးသွားပါမယ်။

Data Strucutre ရဲ့ အဓိက ရည်ရွယ်ချက်ကတော့ stack, queue , deque, list စတာတွေ ကို သိရှိနားလည် စေဖို့ပါ။ ဒီ အခန်းမှာ အဓိက အားဖြင့် Stack ဆိုတာဘာလဲ။ Queue ဆိုတာဘာလဲ စတာတွေကို မိတ်ဆက်ပေးသွားပြီးတော့ အဓိက Array ကို နားလည်ပြီး အသုံးချတတ်ဖို့ အတွက်ပါ။ အဓိကတော့ code တွေကို ဖတ်ရုံမကပဲ ကိုယ်တိုင် ရေးကြည့်ပါ။ အသစ်တွေ ထပ်ဖြည့်ပြီး စမ်းကြည့်ပါ။ လေ့ကျင့်ခန်းတွေ ကို ကြိုးစားဖြေကြည့်စေချင်ပါတယ်။ Data Structure ဟာ အဓိက programming ကို စလေ့လာကာစ သူတွေ အနေနဲ့ တွေးတောတတ်အောင် သင်ကြားလေ့ကျင့်ပေး သည့် အရာဆိုလည်း မမှားပါဘူး။

What is a Stack ?

stack ဆိုတာကတော့ အစီအစဉ်ကျ စီထားထားတော့ items collection လို့ ဆိုရပါမယ်။ အသစ်အသစ်တွေက ကျန်နေတဲ့ data ပေါ်မှာ ထပ်ဖြည့်သွားပါတယ်။ ပြန်ထုတ်မယ်ဆိုရင် နောက်ဆုံး ထည့်ထားတဲ့ data ကနေ ပြန်ထုတ်ရပါတယ်။ LIFO (last-in-first-out) လို့ ဆိုပါတယ်။ ဥပမာ။။ ကျွန်တော်တို့ စာအုပ် ပုံနဲ့ တူပါတယ်။ စာအုပ် ပုံမှာ အောက်ကလို ရှိပါတယ်။

- python
- javascript
- css
- html

နောက်ထပ် စာအုပ် တစ်အုပ်ဖြစ်တဲ့ Data Structure ဆိုတဲ့ စာအုပ်ကို စာအုပ်ပုံမှာ ထပ် တင်လိုက်ရင်တော့

- Data Structure
- python
- javascript
- css
- html

ဆိုပြီး ဖြစ်သွားပါမယ်။ စာအုပ်ပုံကနေ စာအုပ်ကို ထုတ်မယ်ဆို အပေါ်ဘက်ကနေ ပြန်ထုတ်မှ ရပါမယ်။ ဥပမာ javascript စာအုပ်ကို လိုချင်ရင် Data Structure နှင့် Python ဆိုတဲ့ စာအုပ် ၂ အုပ် ဖယ်ပြီးမှ Javascript စာအုပ်ကို ဆွဲထုတ်လို့ ရပါလိမ့်မယ်။ အဲဒီ အခါ stack က

- javascript
- css
- html

ဆိုပြီး ဖြစ်သွားပါပြီ။

Javascript စာအုပ်ကို ယူလိုက်ရင် stack က

- css
- html

ဆိုပြီး ဖြစ်သွားပါလိမ့်မယ်။

Python စာအုပ်ကို ထပ်ဖြည့်လိုက်ရင်တော့

- python
- css
- html

ဆိုပြီး stack က ဖြစ်သွားပါလိမ့်မယ်။

First In Last Out သဘောတရားပါ။

အခု ဆိုရင် stack ဆိုတာကို စာသဘော အားဖြင့် နားလည်လောက်ပါပြီ။ ကျွန်တော်တို့ stack ကို python နဲ့ ဖန်တီးကြည့်ရအောင်။

Stack Abstract Data Type

Stack မှာ ပါဝင်မယ့် data type တွေ လုပ်ဆောင်မယ့် အရာတွေ အကို အရင် ဆုံး ကျွန်တော်တို့ စဥ်းစားကြပါမယ်။ stack က LIFO ဖြစ်တဲ့ အတွက် List လိုမျိုး ကြိုက်တဲ့ နေရာကနေ ဆွဲထုတ်လို့ မရပါဘူး။ အပေါ်က data ကို ပဲ ဆွဲထုတ်ခွင့်ရှိပါတယ်။

`Stack()` Stack ဆိုတဲ့ class ကို ကျွန်တော်တို့ တွေ ဖန်တီးပါမယ်။ constructor တွေ မလိုအပ်ပါဘူး။

`push(item)` ကတော့ item အသစ်ကို ဖန်တီးထားတဲ့ stack ထဲကို ထည့်ဖို့ အတွက်ပါ။

`pop()` ကတော့ stack ထဲကနေ အပေါ်ဆုံး item ကို ထုတ်ဖို့ အတွက်ပါ။ ထုတ်လိုက်တဲ့ value ကို တော့ return မလုပ်ပါဘူး။

`peek()` ကတော့ stack ထဲက အပေါ်ဆုံး item ကို ထုတ်မယ်။ ပြီးတော့ return ပြန်ပေးပါမယ်။

`is_empty()` ကတော့ stack က empty ဖြစ်လား မဖြစ်ဘူးလား ဆိုတာကို စစ်ပြီးတော့ boolean value ကို return ပြန်ပေးပါမယ်။

`size()` ကတော့ stack ထဲမှာ စုစုပေါင်း data ဘယ်လောက် ရှိသလဲ ဆိုတာကို return ပြန်ပေးမယ်။ ပြီးတော့ 4 နဲ့ dog ကို ထည့်တယ်။ peek လုပ်တယ်။ နောက်ဆုံး ထည့်ထားတဲ့ dog ကို ရတယ်။ နောက်ပြီးတော့ True ထည့်တယ်။ အခန်းက ၃ ခု ဖြစ်သွားတာ ဟုတ်မဟုတ် စစ်ကြည့်တယ်။ ပြီးတော့ 8.4 ထည့်ပြီးတော့ pop ၂ ခု လုပ်လိုက်တယ်။ ဒါကြောင့် `4,dog,True,8.4 stack` ကနေ pop ၂ ခု လုပ်လိုက်တော့ `4,dog` ဆိုတဲ့ stack ဖြစ်သွားပါတယ်။ Size က 2 ပြပါလိမ့်မယ်။

Implementing A Stack

အခု ကျွန်တော်တို့တွေ stack ထဲမှာ ဘာတွေ ပါမယ်ဆိုတာကို သိပြီးပါပြီ။ လက်တွေ့ Stack class တစ်ခုကို တည်ဆောက်ကြည့်ရအောင်။

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)
```

အဲဒီ code ထဲမှာ ပါတဲ့ `__init__(self)` ဆိုတာကတော့ constructor ပါ။ Object တစ်ခုကို စပြီး တည်ဆောက်တာ နဲ့ ဦးစွာ constructor ကို ခေါ်ပါတယ်။ stack မှာတော့ Object စ ဆောက်တာနဲ့ object ရဲ့ items variable ကို empty ထည့်လိုက်ပါတယ်။

`len(self.items)` ဆိုတာကတော့ len ဆိုတဲ့ function ဟာ item ရဲ့ array size ကို ဖော်ပြတာပါ။ item ထဲမှာ စုစုပေါင်း array အခန်း ဘယ် ၂ ခုကို ရှိတယ်ဆိုတာကို သိနိုင်ပါတယ်။

အခု ကျွန်တော်တို့ stack ကို စမ်းကြည့်ရအောင်။

```
#stack.py

class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
```

```
            return self.items.pop()
        def peek(self):
            return self.items[len(self.items) - 1]
        def size(self):
            return len(self.items)

s = Stack()

print(s.is_empty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print(s.size())
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

```
s = Stack()
```

s ကို Stack class အနေနဲ့ ကြေငြာထားပါတယ်။

```
print(s.is_empty())
```

ပထမဆုံး stack က empty ဖြစ်နေလားဆိုတာကို ထုတ်ပြထားတယ်။ ဒါကြောင့် True ဆိုပြီး ပြပါလိမ့်မယ်။

```
s.push(4)
s.push('dog')
```

နောက်ပြီးတော့ ကျွန်တော်တို့တွေ 4 , dog စသည်ဖြင့် ထည့်လိုက်ပါတယ်။

```
print(s.peek())
```

peek ကတော့ နောက်ဆုံး ထည့်ထားတဲ့ အခန်းကို ဖော်ပြတာပါ။

```
print(s.size())
```

s.size() ဆိုတာကတော့ ကျွန်တို့တွေ size ဘယ်လောက်ရှိပြီလဲ ဆိုတာကို ထုတ်ကြည့်တာပါ။

```
print(s.pop())
```

pop ဆိုတာကတော့ နောက်ဆုံး ထည့်ထားတဲ့ data တွေကို ထုတ်လိုက်တာပါ။

Simple Balanced Parentheses

ကျွန်တော် တို့ အပေါင်း အနှုတ် အမြောက် အစားတွေ လုပ်တဲ့ အခါမှာ ကွင်းစ ကွင်းပတ်တွေကို အသုံးပြုပါတယ်။

ဥပမာ။။

```
(5+6)∗(7+8)/(4+3)
```

ကျွန်တော်တို့ဟာ user က ထည့်လိုက်တဲ့ ကွင်းစ ကွင်းပိတ် တွေ မှန် မမှန် စစ်ဖို့ အတွက် stack ကို အသုံးပြုပြီး ရေးပါမယ်။

မှန်ကန်တဲ့ ကွင်းစကွင်းပိတ်တွေဟာ

```
(()()()())
```

```
(((())))
```

```
(()((())()))
```

လိုမျိုး ဖြစ်ပါလိမ့်မယ်။

```
((((((())
```

```
()))
```

```
(()()(()
```

လိုမျိုးတွေကတော့ မှားယွင်းသည်လို့ သတ်မှတ်ရပါမယ်။

```
# parChecker.py

from stack import Stack

def parChecker(symbolString):
    s = Stack()
    balanced = True
    index = 0
    while index < len(symbolString) and balanced:
        symbol = symbolString[index]
        if symbol == "(":
            s.push(symbol)
```

```
        else:
            if s.is_empty():
                balanced = False
            else:
                s.pop()

        index = index + 1

    if balanced and s.is_empty():
        return True
    else:
        return False

print(parChecker('((()))'))
```

ဒီ code ကို ကျွန်တော်တို့ လေ့လာကြည့်ရအောင်။

ကျွန်တော်တို့ `stack.py` ကို `parChecker.py` နဲ့ file နေရာ အတူတူ ထားဖို့ လိုပါတယ်။

ဥပမာ။။ `parChecker.py` ဟာ `/Users/python/` မှာ ရှိတယ် ဆိုရင် `stack.py` ဟာလည်း `/Users/python/` နေရာမှာ ရှိဖို့ လိုပါတယ်။

ဒါမှသာ

```
from stack import Stack
```

က အလုပ်လုပ်ပါမယ်။

Python မှာ external file import format က

```
from filename import ClassName
```

ဒါကြောင့် class name က Stack ဖြစ်ပြီး file name က `stack.py` ဖြစ်သည့် အတွက်ကြောင့်

```
from stack import Stack
```

ဆိုပြီး ရေးထားတာပါ။

parChecker function ထဲမှာ စာလုံးရေ ရှိသလောက်ကို ကျွန်တော်တို့တွေ while loop နဲ့ ပတ်လိုက်တယ်။ နောက်ပြီးတော့ balanced က True ဖြစ်နေ သ၍ loop ပတ်ပါတယ်။ balanced က stack empty ဖြစ် မဖြစ် စစ် ဖို့ အတွက်ပါ။ ပြီးတော့ စာလုံးတွေကို တစ်လုံးခြင်း စီ ယူပါတယ်။ ကျွန်တော်တို့ trace လိုက်ကြည့်ရအောင်။

ကျွန်တော်တို့ ထည့်လိုက်တဲ့ String က `(()` ။

ပထမဆုံး စာလုံးက `(`

```
    symbol = symbolString[index]
    if symbol == "(":
        s.push(symbol)
    else:
        if s.is_empty():
            balanced = False
        else:
            s.pop()

    index = index + 1
```

ဒါကြောင့် `s.push("(")` ဝင်သွားပါတယ်။ တကယ်လို့သာ ပထမဆုံး စာလုံးက (မဟုတ်ခဲ့ရင် balanced က False ဖြစ်ပြီးတော့ loop ထဲကနေ ထွက်သွားမှာပါ။

ဒုတိယ စာလုံး `(` ကို ထပ်ယူတယ်။ push ထပ်လုပ်တယ်။ Array ထဲမှာ အခန်း ၂ ခန်း ဖြစ်သွားပြီ။

တတိယ စာလုံး `)` ကို ယူတယ်။ pop လုပ်တယ်။ Array ထဲမှာ အခန်း ၁ ခန်း ကျန်သေးတယ်။

စာလုံးရေ ကုန်သွားသည့် အတွက် loop ထွက်သွားတယ်။

```
    if balanced and s.is_empty():
        return True
    else:
        return False
```

`if balanced and s.is_empty():` မှာ ရေးထားသည့် condition ကြောင့် `return False` ပြန်ပါတယ်။

ကျွန်တော်တို့တွေဟာ `()` ကို စစ်သည့် အခါမှာ အဖွင့် နှင့် အပိတ် အကြိမ် အရေ အတွက် တူဖို့ လိုတယ်။ ဒါကြောင့် `(` ကို stack ထဲမှာ push လုပ်ပြီး `)` ကို stack ထဲကနေ ပြန်ထုတ်ပါတယ်။ `()` အရေအတွက် ညီရင် နောက်ဆုံး stack က empty ဖြစ်သွားပါမယ်။ နောက်ပြီးတော့ balanced ကလည်း True ဖြစ်ပြီးတော့ array ထဲကနေ ထွက်လာပါလိမ့်မယ်။

Balanced Symbol

ကျွန်တော်တို့တွေ (နှင့်) အတွက်ကိုတော့ ရေးပြီးပါပြီ။ နောက်တဆင့် အနေနဲ့ {[နှင့်]} ကို ထည့်ပြီး ရေးဖို့ လိုပါတယ်။ သင်္ချာ ကွင်းတွေက

```
    {[()]}
```

ဆိုပြီး ရှိပါတယ်။ ကျွန်တော်တို့ လက်သည်းကွင်း အတွက် ရေးပြီးပါပြီ။ အခြား ကွင်းတွေပါ ပါလာရင် စစ်ဖို့ အတွက် ထပ်ပြီး ပြင်ရပါမယ်။

အဲဒီအတွက် ပြီးခဲ့တဲ့ code အတိုင်း ရေးလို့ မရတော့ပါဘူး။

```
from stack import Stack

def parChecker(symbolString):
    s = Stack()
    balanced = True
    index = 0
    while index < len(symbolString) and balanced:
        symbol = symbolString[index]
        if symbol in "([{":
            s.push(symbol)
        else:
            if s.is_empty():
                balanced = False
            else:
                top = s.pop()
                if not matches(top,symbol):
                       balanced = False
        index = index + 1
    if balanced and s.is_empty():
        return True
    else:
        return False

def matches(open,close):
    opens = "([{"
    closers = ")]}"
    return opens.index(open) == closers.index(close)


print(parChecker('{{([][])}()}'))
print(parChecker('[{()]'))
```

ဒီ code လေးကို တချက်ကြည့်ရအောင်။

ပြီးခဲ့သည့်တုန်းကလိုပဲ ကျွန်တော်တို့ ရေးထားတာပါ။ သို့ပေမယ့် အခု အခါမှာတော့ `(` တစ်ခု တည်းက မဟုတ်တော့ပါဘူး။ `{[]}` ပါ ထပ်ပါလာပါပြီ။ ဒါကြောင့် ကွင်းစတွေဖြစ်သည့် `([{` တွေ သာ ဖြစ်ခဲ့ရင် stack ထဲ ထည့်ပါတယ်။ မဟုတ်ခဲ့ဘူးဆိုရင် stack ထဲကနေ pop လုပ်တာ တစ်ခုတည်း မရတော့ပါဘူး။ နောက်ဆုံးဖြည့်ထားတာက `[` ဖြစ်ရင် ကွင်းပြန်ပိတ်တာက `]` ဖြစ်ကို ဖြစ်ရပါမယ်။ ဒါကြောင့် အဖွင့် ကွင်း မဟုတ်ခဲ့လို့ အပိတ်ကွင်း သာ ဖြစ်ခဲ့ရင် ရှေ့ဘက်က ဖွင့် ထားတဲ့ ကွင်း နဲ့ တူ မလား စစ်ဖို့ လိုအပ်ပါတယ်။

```
top = s.pop()
if not matches(top,symbol):
    balanced = False
```

အဖွင့် ဖြစ်ခဲ့ရင် အပိတ်ဖြစ်သလား စစ်ဖို့ matches ဆိုတဲ့ function ကို ရေးထားပါတယ်။

```
def matches(open,close):
    opens = "([{"
```

```
    closers = ")]}"
    return opens.index(open) == closers.index(close)
```

အဲဒီမှာ index ဆိုတာ ပါလာပါပြီ။ index ဆိုတာ ထည့်လိုက်တဲ့ စာလုံးက ဘယ်နေရာမှာ ရှာတွေ့တယ်ဆိုတဲ့ နံပတ် ပြန်လာတာပါ။

```
opens = "([{"
closers = ")]}"

print(opens.index("("))
print(opens.index("["))
print(closers.index("]"))
print(closers.index("}"))
```

ဒါကြောင့် အဖွင့် နှင့် အပိတ် index တူသလား နဲ့ လွယ်လင့် တကူ စစ်ထားတာပါ။ သို့မဟုတ်ရင် `if condition` နှင့် open က `[` ဖြစ်ခဲ့ရင် close က `]` ဖြစ်ရမယ် ဆိုပြီး စစ်နေဖို့ လိုပါတယ်။ တစ်ခါတစ်လေ programming ရေးသားရာမှာ ဖြတ်လမ်းလေးတွေ သုံးပြီး အများကြီး ရေးမယ့် အစား အခုလို အတိုလေး ရေးလို့ ရနိုင်သည့် နည်းလမ်းလေးတွေ ရှိပါတယ်။

အဖွင့် နဲ့ အပိတ် က သာ မတူခဲ့ရင်တော့ loop က ထွက်ပြီး `False` ပြန်ပေးလိုက်ရုံပါပဲ။

Decimal To Binary

အခု ကျွန်တော်တို့တွေ Decimal (Base 10) ကနေ Binary (Base 2) value ပြောင်းတာလေး ရေးကြည့်ရအောင်။ Computer သမား တစ်ယောက် ဖြစ်ရင်တော့ binary ဆိုတာ ဘာလဲ သိဖို့ လိုအပ်ပါတယ်။ Computer ဟာ binary value နဲ့ ပဲ အလုပ်လုပ်ပါတယ်။ value က 0 နှင့် 1 ပဲ ရှိပါတယ်။ Decimal value ကတော့ 0-9 ဂဏန်းတွေရှိပါတယ်။ ဥပမာ။ 668 ဆိုတဲ့ decimal value ကို binary ပြောင်းရင် 1010011100 ဆိုတဲ့ binary value ရပါတယ်။ ဘယ်လို ပြောင်းရ သလဲဆိုရင်တော့

668 ကို 2 နဲ့ စားသွားပါတယ်။ နောက်ဆုံး သုည ရသည့် အထိပါ။

```
668/2 = 334 , Mod : 0
334/2 = 167,  Mod: 0
167/2 = 83,   Mod: 1
83/2 = 41,    Mod: 1
41/2 = 20,    Mod: 1
20/2 = 10,    Mod: 0
10/2 = 5,     Mod: 0
5/2 =  2,     Mod: 1
2/2 = 1,      Mod: 0
1/2 = 0,      Mod: 1
```

ရလာတဲ့ mod တွေကို ပြောင်းပြန် ယူလိုက်တော့ 1010011100 ထွက်ပါတယ်။ code ဘယ်လို ရေးလို့ ရမလဲ ဆိုတာကို သဘောပေါက်လောက်ပြီ ထင်ပါတယ်။

လွှယ်ပါတယ်။ ၂ နဲ့ စားသွားမယ်။ အကြွင်းတွေကို stack ထဲ ထည့်သွားမယ်။ ပြီးရင် pop ပြန်လုပ်လိုက်မယ်။ Stack က first in last out ဖြစ်သည့် အတွက်ကြောင့် ပထမဆုံး အကြွင်း ရလာဒ်က နောက်ဆုံးမှ ထွက်ပါမယ်။ Decimal to Binary ပြောင်းဖို့ အတွက် stack ကို အသုံးပြုနိုင်ပါတယ်။

```
from stack import Stack

def divideBy2(decNumber):
    remstack = Stack()

    while decNumber > 0:
        rem = decNumber % 2
        remstack.push(rem)
        decNumber = decNumber // 2

    binString = ""
    while not remstack.is_empty():
        binString = binString + str(remstack.pop())

    return binString

print(divideBy2(668))
```

code လေးကတော့ ရှင်းပါတယ်။ စားလဒ်က သုည မဖြစ်မခြင်း loop ပတ်မယ်။ နောက်ပြီးတော့ အကြွင်းတွေကို stack ထဲမှာ ထည့်ထားမယ်။

ပြီးသွားတဲ့ အခါ stack မကုန် မခြင်း loop ပတ်ပြီးတော့ pop လုပ်ပေးလိုက်ပါတယ်။

အခုလောက်ဆိုရင်တော့ Stack ကို သုံးတတ်လောက်ပြီ ထင်ပါတယ်။

လေ့ကျင့်ခန်း ၄

မေးခွန်း ၁။

Base 8 ကို ပြောင်းသည့် divdeBy8 function ကို ရေးပါ။ 668 (base 10) ၏ base 8 တန်ဖိုးသည် တန်ဖိုးသည် 1234 ဖြစ်သည်။ divideBy8(668) ၏ အဖြေသည် 1234 ထွက်လာရမည်။

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)
```

မေးခွန်း ၂။

Base 16 ကို ပြောင်းသည့် divdeBy16 function ကို ရေးပါ။ 668 (base 10) ၏ base 16 တန်ဖိုးသည် တန်ဖိုးသည် 29C ဖြစ်သည်။ divdeBy16(668) ၏ အဖြေသည် 29C ထွက်လာရမည်။ base 16 သည်

Base 16	Base 10
A	10
B	11
C	12
D	13
E	14
F	15

ဖြစ်သည်။ ထို့ကြောင့် အကြွင်း ကို if condition သုံးကာ 10 မှ 15 value များကို ပြောင်းပေးရန် လိုသည်။

အခန်း ၅ ။ Queue

ဒီအခန်းမှာတော့ ကျွန်တော် Queue အကြောင်းကို အဓိက ရှင်းပြသွားပါမယ်။ Queue တစ်ခု ကို ဘယ်လို ရေးရသလဲ နောက်ပြီးတော့ ဘာတွေ ပါဝင်တယ်ဆိုတာတွေကို လေ့လာနိုင်မှာပါ။

Queue

ကျွန်တော်တို့ Stack နဲ့ ပတ်သက်ပြီးတော့ လေ့လာပြီးပါပြီ။ အခု Queue အကြောင်းလေ့လာ ရအောင်။ Stack ဟာ First In Last Out ဖြစ်ပြီးတော့ Queue ဟာ First In First Out ဖြစ်ပါတယ်။ Queue ဟာ ဘာနဲ့ တူသလဲ ဆိုတော့ ဆိုင်တွေမှာ တန်းစီပြီး မုန့်ဝယ် သလို အရင် တန်းစီတဲ့လူ အ ရင် ရပါတယ်။ Stack ကတော့ တစ်ကား တစ်ပေါက်ထဲရှိတဲ့ bus ကား လိုမျိုး နောက်ဆုံးဝင်သည့် သူက အရင် ထွက် လို့ ရပါတယ်။

Queue Abstract Data Type

Queue တစ်ခု ဖန်တီးဖို့ ကျွန်တော်တို့ ဘာတွေ လိုမလဲ အရင် စဉ်းစားကြည့်ရအောင်။

Queue() Queue class တစ်ခု ဖန်တီးဖို့ လိုအပ်ပါတယ်။ ပထမဆုံး class ကို create လုပ်သည့်အခါ မှာတော့ empty ဖြစ်နေဖို့ လိုပါတယ်။
enqueue(item) item အသစ်ထည့်လိုက်ရင် နောက်ဆုံးမှာ data တွေသွားပြီး append လုပ်ပါမယ်။
dequeue() ဖျက်ပြီဆိုရင် ရှေ့ဆုံးက data ကို ထုတ်ဖို့ လိုအပ်ပါတယ်။
isEmpty() queue တစ်ခု ဟာ empty ဖြစ်မဖြစ် စစ်ဖို့လိုပါတယ်။ Boolean value return ပြန်လာပါ မယ်။
size() queue ထဲမှာ item ဘယ်နှစ်ခု ရှိလဲဆိုတာကို သိဖို့ အတွက်ပါ။

ကျွန်တော်တို့ စပြီးတော့ implement လုပ်ကြည့်ရအောင်။

Implementing A Queue

ကျွန်တော်တို့ Stack တစ်ခု ကို ဖန်တီးထားဖူးသည့် အတွက်ကြောင့် Queue တစ်ခု ဖန်တီးဖို့ မ ခက်ခဲလှပါဘူး။

`pyqueue.py` တစ်ခု ကို ဖန်တီးပြီး အောက်ပါ code ထည့်လိုက်ပါမယ်။

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
```

```
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)

q=Queue()
q.enqueue(6)
q.enqueue('cat')
q.enqueue(True)
print(q.size())
print(q.dequeue())
print(q.dequeue())
print(q.size())
```

```
3
6
cat
1
```

ဆိုပြီး ထွက်လာပါမယ်။ စုစုပေါင်း ၃ ခု ရှိပါတယ်။ ပထမဆုံးထည့်လိုက်သည့် value က 6 ဖြစ်သည့်အတွက် 6 ထွက်လာပါမယ်။ ဒုတိယ အကြိမ် က cat ဆိုပြီး ထွက်လာပါမယ်။ အခု အချိန်မှာတော့ queue တစ်ခု ပဲ ရှိပါတော့တယ်။

Priority Queue

Queue မှာ နောက်ထပ် တစ်မျိုး ရှိတာကတော့ Priority Queue ပါ။ အရင်လာ အရင်ထွက် ဆိုပေမယ့် အရေးကြီးတာကို ဦးစားပေးပြီး နောက်မှလာပေမယ့် အရင် ထွက်ဖို့ လိုပါတယ်။ ထည့်လိုက်သည့် data ရဲ့ Priority ပေါ်မှာ မူတည်ပြီးတော့ ပြန်ထုတ်ဖို့ လိုပါတယ်။

`pyqueue.py` နဲ့ အတူတူ `pqueue.py` ကို ဖန်တီးပါတယ်။

```
from pyqueue import Queue

class PQueue:
    def __init__(self):
        self.items = {}

    def isEmpty(self):
        return len(self.items) == 0

    def enqueue(self, item,priority=0):
        if priority not in self.items:
            self.items[priority] = Queue()

        queue = self.items[priority]
        queue.enqueue(item)
```

```
    def dequeue(self):
        keys = list(self.items.keys())

        if len(keys) > 0 :
            cursor = keys[-1]

            myqueue = self.items[cursor]
            val = myqueue.dequeue()

            if myqueue.size() == 0 :
                del self.items[cursor]
            return val
        return ""

    def size(self):
        size = 0
        for key in self.items.keys():
            size = size + self.items[key].size()
        return size
```

Priority Queue ဟာ queue တွေကို Priority အလိုက် စီပြီးတော့ ပြန် ထုတ်ပေးတာပါ။ Code ကို မရှင်းပြခင်မှာ အရင်ဆုံး စမ်းကြည့် ရအောင်။

```
from pqueue import PQueue

p = PQueue()

p.enqueue("HELLO",1)
p.enqueue("H",2)
p.enqueue("E",5)
p.enqueue("L",3)
p.enqueue("O",3)
p.enqueue("Sample",9)

# 7
print(p.size())

#Sample
print(p.dequeue())

#E
print(p.dequeue())

#4
print(p.size())

#L
print(p.dequeue())

#O
print(p.dequeue())

#H
print(p.dequeue())


#1
print(p.size())
```

enqueue လုပ်တဲ့ အခါမှာ value နဲ့ priority ပါပါတယ်။ သို့ပေမယ့် ပြန်ထုတ်သည့် အခါမှာ Priority အမြင့်ဆုံး က အရင် ထွက်လာပါလိမ့်မယ်။

```
p = PQueue()

p.enqueue("HELLO",1)
p.enqueue("H",2)
p.enqueue("E",5)
p.enqueue("L",3)
p.enqueue("O",3)
p.enqueue("Sample",9)
```

အဲဒီမှာ priority အမြင့်ဆုံးက `"Sample"` ပါ။ ဒါကြောင့် နောက်ဆုံးမှ ထည့်ပေမယ့် ပထမဆုံး ထွက်လာတာကို တွေ့ပါလိမ့်မယ်။

အခု ကျွန်တော်တို့ PQueue() ကို လေ့လာကြည့်ရအောင်။

ကျွန်တော်တို့ဟာ Queue တွေကို priority အလိုက် စီရမှာ ဖြစ်တဲ့ အတွက် array အစား dictionary ကို အသုံးပြုထားပါတယ်။ နောက် အခန်းမှာ Dictionary အကြောင်းကို ရှင်းပြပေးပါမယ်။ enqueue လုပ်တဲ့ အခါမှာ item dictionary ရဲ့ Priority အခန်း မှာ Queue Object ရှိမရှိ စစ်ပါတယ်။

```
if priority not in self.items:
    self.items[priority] = Queue()
```

ဆိုတာက self.items ထဲမှာ key name က priority ရှိ မရှိကြည့်တာပါ။ မရှိဘူးဆိုရင် Queue Object အသစ် တစ်ခု ဆောက်ပြီး ထည့်လိုက်ပါတယ်။

```
queue = self.items[priority]
queue.enqueue(item)
```

item ထဲကနေ ပြီးတော့ priority key ပေါ်မှာ မူတည်ပြီးတော့ Queue object ကို ထုတ်လိုက်ပါတယ်။ ပြီးတော့ Queue ထဲမှာ enqueue လုပ်ပေးပါတယ်။

```
p.enqueue("HELLO",1)
p.enqueue("H",2)
p.enqueue("E",5)
p.enqueue("L",3)
p.enqueue("O",3)
p.enqueue("Sample",9)
```

အဲဒီလို ဆိုရင် data တွေ အနေနဲ့

```
{
    1 : ["HELLO"],
```

```
        2 : ["H"],
        3 : ["L","O"],
        5 : ["E"],
        9 : ["Sample"]
    }
```

ဆိုပြီး ရှိနေပါမယ်။

dequeue အပိုင်းကတော့ နည်းနည်း ရှုပ်ထွေးပါတယ်။

```
keys = list(self.items.keys())
```

keys ထဲမှာ [1,2,3,5,9] ဆိုပြီး ရောက်လာပါမယ်။

```
if len(keys) > 0 :
```

keys မှာ အခန်း ရှိမရှိ စစ်ပါတယ်။

```
cursor = keys[-1]
```

`-1` ဆိုတာကတော့ နောက်ဆုံး အခန်းကို ဆွဲထုတ်လိုက်ပါတယ်။

```
myqueue = self.items[cursor]
val = myqueue.dequeue()
```

ပြီးတော့ queue object ကို ဆွဲထုတ်ပါတယ်။ ပြီးတော့ `dequeue()` လုပ်ပြီးတော့ data ကို ဆွဲထုတ်လိုက်ပါတယ်။

```
if myqueue.size() == 0 :
    del self.items[cursor]
```

queue ထဲမှာ data မရှိတော့ရင် အခန်းကို ဖျက်ချလိုက်ပါတယ်။

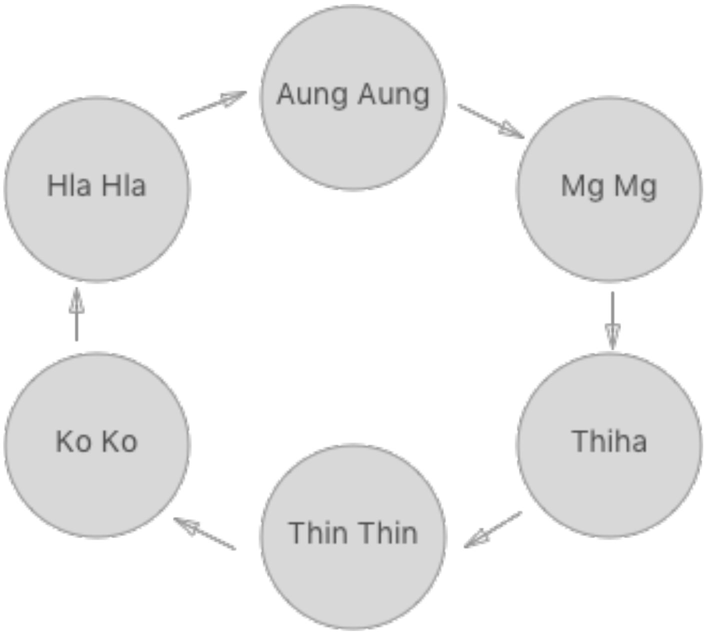
```
size = 0
for key in self.items.keys():
    size = size + self.items[key].size()
```

size တွက် ကတော့ ရိုးရိုးလေးပါပဲ။ item ထဲမှာ ရှိတဲ့ queue တွေ အားလုံးရဲ့ size ကို ပေါင်းပြီးတော့ return ပြန်ပေးလိုက် ရုံပါပဲ။

Hot Potato

ကျွန်တော်တို့အခု Hot Potato game လေး တစ်ခု ဖန်တီးကြည့်ရအောင်။ Hot Potato game ဆိုတာ က အားလူး ကို အကြိမ် အရေအတွက် တစ်ခု အထိ သွားပြီး လက်ထဲမှာ အားလူး ရှိတဲ့ သူက ထွက်ရပါတယ်။ နောက်ဆုံး တစ်ယောက်တည်း ကျန်တဲ့ အထိပါ။

ကျွန်တော်တို့ အခု အဲဒီ အတွက် program လေးကို Queue ကို အသုံးပြုပြီး ရေးပါမယ်။ မရေးခင် စဉ်းစားကြည့်ရအောင်။



Aung Aung, Mg Mg , Thiha , Thin Thin , Ko Ko , Hla Hla ဆိုပြီး ရှိတယ်။ အာလူး ၈ ကြိမ် မြောက် လူက ထွက် ရမယ် ဆိုရင် Thiha က အရင် ထွက်ရပါမယ်။ နောက်တစ်ကြိမ်ဆိုရင် Aung Aung ထွက်ရမယ်။ ပြီးရင် Mg Mg။ ပြီးတော့ Hla Hla။ ပြီးတော့ Thin Thin။ နောက်ဆုံး Ko Ko သာကျန်ခဲ့ပါမယ်။

အဲဒီ Program လေးကို စဉ်းစားရင် အာလူးဟာ တစ်ယောက်ပြီး တစ်ယောက် ပြောင်းသွားတယ်။ နောက်ဆုံး Hla Hla ဆီ ရောက်သွားရင် Aung Aung ကနေ ပြန်စတယ်။

တနည်းပြောရင် ပထမဆုံး က လူကို နောက်ကနေ ပြန်ပြီး တန်းစီ ခိုင်းတာနဲ့ အတူတူပါပဲ။ အာလူးကို နောက်တစ်ယောက်ကို ပေးပြီးပြီ ဆိုရင် နောက်ကို ပြန်သွားပြီး တန်းစီ ပုံစံပါပဲ။

Aung Aung, Mg Mg , Thiha , Thin Thin , Ko Ko , Hla Hla ဆိုပါဆို့။ Aung Ang ကနေ Mg Mg ကို အာလူး ပေးပြီးရင် Queue က Mg Mg , Thiha , Thin Thin , Ko Ko , Hla Hla, Aung Aung ဖြစ်သွားမယ်။ ပြီးရင် Thiha , Thin Thin , Ko Ko , Hla Hla, Aung Aung, Mg Mg ဖြစ်မယ်။

ဒါကို သဘောပေါက်ပြီဆိုရင် ကျွန်တော်တို့ code ရေးလို့ရပါပြီ။

```
from pyqueue import Queue

def hotPotato(namelist, num):
    simqueue = Queue()
```

```
    for name in namelist:
        simqueue.enqueue(name)

    while simqueue.size() > 1:
        for i in range(num):
            simqueue.enqueue(simqueue.dequeue())
        simqueue.dequeue()
    return simqueue.dequeue()


print(hotPotato(["Aung Aung", "Mg Mg" , "Thiha" , "Thin Thin" , "Ko Ko" , "Hla Hla"],8))
```

Code လေးက ရှင်းပါတယ်။ ထွေထွေ ထူးထူး မရှင်းပြတော့ပါဘူး။

```
simqueue.enqueue(simqueue.dequeue())
```

အဲဒီ code လေးက dequeue လုပ်ပြီးတာနဲ့ ချက်ခြင်း enqueue ပြန်လုပ်ဆိုတဲ့ သဘောပါ။ အခုဆို ရင်တော့ Queue အကြောင်း အနည်းငယ်သဘောပေါက်လောက်ပြီ ထင်ပါတယ်။

လေ့ကျင့်ခန်း ၅။

မေးခွန်း ၁။

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)

q=Queue()
q.enqueue(12)
q.enqueue('dog')
q.enqueue(True)

print(q.dequeue())
```

အဖြေသည်

A . 12

B. dog

C. True

အခန်း ၆ ။ List နှင့် Dictionary

ဒီအခန်းမှာ အဓိကအားဖြင့် List နှင့် Dictionary ကို ဖော်ပြပေးသွားပါမယ်။ List နဲ့ Dictionary ဟာ တကယ့် program တွေ ရေးသည့် အခါမှာ မရှိမဖြစ် သိဖို့ လိုအပ်ပါတယ်။ List ကို Array လို့ လည်း ဆိုနိုင်ပါတယ်။ နောက်ပိုင်း sorting တွေ အပိုင်းမှာ Array ရဲ့ အရေးပါပုံတွေ ကို တွေ့ရပါ လိမ့်မယ်။ ပြီးခဲ့တဲ့ အခန်းတွေကလည်း List ကို အသုံးပြုပြီးတော့ Stack နဲ့ Queue ကို ဖန်တီးခဲ့ တာကို မှတ်မိမယ်လို့ ထင်ပါတယ်။ သို့ပေမယ့် List အကြောင်းကို သေချာ မရှင်းပြခဲ့ပါဘူး။ အခု အခန်းမှာတော့ List နဲ့ Dictionary အကြောင်းကို သေချာစွာ ရှင်းပြပေးပါမယ်။

Lists

List ဆိုတာကတော့ အခန်းကြီးထဲမှာ အခန်း အသေးလေးတွေ ရှိသည့် သဘောပါ။ List ကို programming langauge တွေမှာတော့ Array လို့လည်း ခေါ်ကြပါတယ်။ List ကိုတော့ ကျွန်တော် တို့တွေ Stack အပိုင်းတွေမှာ သုံးခဲ့ဖူးပါတယ်။ တကယ့်လို့ Array/List စတာတွေက programming language မှာ မရှိဘူး ဒါမှမဟုတ် ကိုယ်ပိုင် programming language ကို ဖန်တီးသည့်အခါမှာ Array ဘယ်လို အလုပ်လုပ်လဲ သိအောင် ကိုယ်ပိုင် ဖန်တီးပြီးတော့ ရေးတတ်ဖို့ လိုပါတယ်။

Unordered List Abstract Data Type

Unorder List ဆိုတာကတော့ list ထဲမှာ အစီအစဉ် တကျမဟုတ်ပဲ ဒီ အတိုင်းထည့်ထားတာပါ။ ဥပမာ ။ `[4,3,6,1,90,404]` စသည့် ဖြင့်ပေါ့။

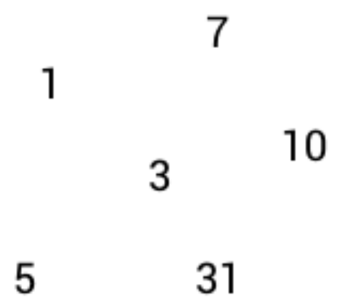
ကျွန်တော်တို့ ကိုယ်ပိုင် List တစ်ခု ဖန်တီးဖို့ အတွက်

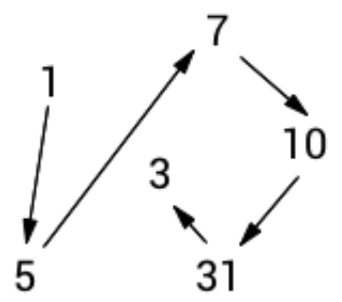
- `List()` ဆိုတဲ့ class ဖန်တီးဖို့လိုမယ်။ init လုပ်သည့်အခါမှာ empty data ဖြစ်နေရမယ်။
- `add(item)` ကတော့ item ကို list ထဲမှာ ထည့်မယ်။ ရှေ့ဆုံးနေရာမှာ ထည့်မှာပါ။
- `remove(item)` ကတော့ item ကို list ထဲကနေ ထုတ်ဖို့ အတွက်ပါ။
- `search(item)` ကတော့ item ဟာ list ထဲမှာ ရှိပြီးပြီလား မရှိသေးဘူးလား စစ်ဖို့ပါ။
- `is_empty()` ကတော့ List ထဲမှာ item တွေ မရှိတော့ဘူးလား ဆိုပြီး စစ်ဖို့အတွက်ပါ။ `true/false` boolean value ကို return ပြန်ပါမယ်။
- `size()` ကတော့ item အရေအတွက်ကို return ပြန်ပေးပါမယ်။ Integer value ကို return ပြန် ပေးပါမယ်။
- `append(item)` ကတော့ နောက်ဆုံး အခန်းမှာ ထည့်ဖို့ပါ။
- `index(item)` ကတော့ item ရဲ့ position ကို ရှာပြီးတော့ return ပြန်မှာပါ။

- `insert(pos,item)` ကတော့ item ကို ကိုယ်ထည့်ချင်သည့် နေရာမှာ ထည့်ဖို့ အတွက်ပါ။
- `pop()` ကတော့ နောက်ဆုံး အခန်းထဲကနေ ဆွဲထုတ်ဖို့ အတွက်ပါ။ pop အတွက်က ဘာမှ return ပြန် ဖို့ မလိုပါဘူး။
- `pop(pos)` ကတော့ အခန်း နံပတ်က ဟာကို ဖျက်မယ်။ ပြီးရင် အဲဒီက data ကို return ပြန်ပေးမယ်။

Implementing an Unordered List: Linked Lists

Unordered List ကို ပုံမှန်အားဖြင့် linked list လို့ ခေါ်ကြပါတယ်။ Value တွေ ဟာ နေရာ အတည် အကျမဟုတ်ပဲ နေရာစုံတွေမှာ ရှိနေပါတယ်။ item တစ်ခုက နောက်ထပ် iteam တစ်ခုကို ထပ်ပြီး ညွှန်းထားပါတယ်။ ဥပမာ အောက်က ပုံကို ကြည့်လိုက်ပါ။





နံပတ်တွေဟာ အစီအစဉ်အတိုင်း မဟုတ်ပဲ ချိတ်ဆက်ထားတာကို တွေ့နိုင်တယ်။ ဒါဆိုရင် class တစ်ခုမှာ လက်ရှိ value နဲ့ နောက်ထပ် value တစ်ခု ကို တွဲပြီး သိမ်းဖို့ လိုတယ်။ နောက်ထပ် value ကလည်း value နဲ့ next ကို သိမ်းဖို့ လိုတယ်။ တနည်းပြောရင် node လေးတွေ ဆက်ထားတာပဲ။

အဲဒီ အတွက် ကျွန်တော်တို့တွေ Node class တစ်ခု တည်ဆောက်ဖို့ လိုလာပြီ။

```
class Node:
    def __init__(self,init_data) :
        self.data = init_data
        self.next = None

    def get_data(self):
        return self.data
```

```
    def get_next(self):
        return self.next

    def set_data(self,new_data) :
        self.data = new_data

    def set_next(self, new_next) :
        self.next = new_next
```

Class လေးကတော့ ရှင်းရှင်းလေးပါပဲ။ လက်ရှိ ရှိနေသည့် data ကို store လုပ်ထားမယ်။ next data ကို မှတ်ထားမယ်။

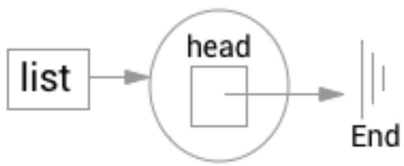
code လေးကို အရင်ဆုံး အလုပ်လုပ်လား စမ်းကြည့်ရအောင်။

```
from node import Node

temp = Node(93)
print(temp.get_data())
```

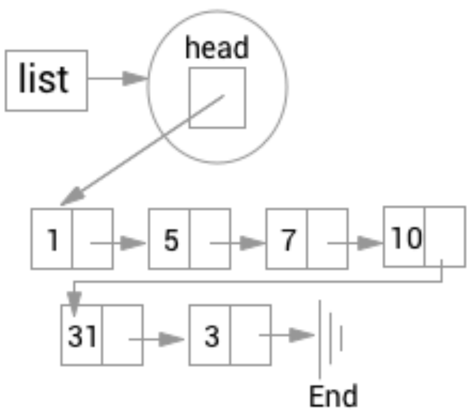
အခု Node တစ်ခု ရပြီ။ အဲဒီတော့ Unordered List Class ကို ဆောက်ကြမယ်။ Node ထဲမှာ value သိမ်းမယ်။ Node ရဲ့ next value က နောက်ထပ် node တစ်ခုကို ချိတ်ထားမယ်။ ဒါဆိုရင် ကျွန်တော်တို့တွေ Linked List တစ်ခု ဖန်တီးနိုင်ပြီ။

Unordered List Class

Unordered List ကို Node နဲ့ ဖန်တီးပြီးတော့ တစ်ခုခြင်းစီကို ချိတ်ဆက်သွားရုံပဲ။ ပုံလေးနဲ့ စဉ်းစားကြည့်ရင် အောက်ကလို ပုံလေးပဲ။



List ကသာ empty ဖြစ်နေရင် head က end နှင့် ချိတ်ထားပါလိမ့်မယ်။



မဟုတ်ဘူးဆိုရင်တော့ head က လက်ရှိ ရှေ့ဆုံး node ကို ညွှန်ထားမယ်။ ရှေ့ node ရဲ့ value က 1 ဖြစ်ပြီးတော့ next ကိုတော့ နောက် ထပ် node တစ်ခု နဲ့ ထပ်ပြီးတော့ ညွှန်ထားတယ်။ ဒီပုံကို မြင်တော့ ကျွန်တော်တို့တွေ ဘာဖြစ်လို့ node class ကို ဆောက်ခဲ့သလည်းဆိုတာကို သဘောပေါက်လောက်ပါပြီ။ အခု unorder list ဖန်တီးကြည့်ရအောင်။

```
from node import Node

class UnorderedList:
    def __init__(self):
        self.head = None

mylist = UnorderedList()
```

ဒါကတော့ အရိုးအရှင်းဆုံး ဦးစွာ class တစ်ခု ဖန်တီးလိုက်တာပေါ့။ head ထဲမှာ None ကိုထည့်ထားတယ်။ ဘာဖြစ်လို့လည်းဆိုတော့ object ကို ဆောက်လိုက်တာနဲ့ empty list တစ်ခုကို ဖန်တီးခြင်လို့ပါ။

Empty

ကျွန်တော်တို့တွေ List ကို empty ဖြစ်မဖြစ် စစ်ဖို့ အတွက် function တစ်ခု ဖန်တီး ရအောင်။ function ကလည်း လွယ်ပါတယ်။ head ကသာ None ဖြစ်နေရင် List က empty ဖြစ်နေတယ်ဆိုတဲ့ အဓိပ္ပာယ်ပါပဲ။

```
def is_empty(self):
    return self.head == None
```

code ကတော့

```
from node import Node

class UnorderedList:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

mylist = UnorderedList()
print(mylist.is_empty())
```

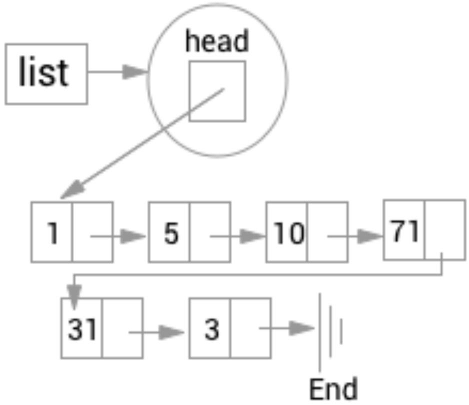
ရိုးရိုးလေးပါပဲ။ အခု နောက်တစ်ဆင့် သွားရအောင်။

Add

အခု အဆင့်မှာတော့ add function ကို ဖန်တီးကြမယ်။

```
mylist. = UnorderedList()
mylist.add(3)
mylist.add(31)
mylist.add(71)
mylist.add(10)
mylist.add(5)
mylist.add(1)
```

ဆိုရင် list က အောက်က ပုံလို ပေါ်ဖို့လိုပါတယ်။



အသစ်ထပ်ဖြည့်လိုက်တိုင်း အနောက်ကို ရောက်ရောက်သွားမယ်။

ပထမဆုံး အကြိမ်မှာ 3 ပဲ ရှိတယ်။ 31 ထပ်ဖြည့်တော့ 31,3 ဖြစ်သွားတယ်။ 71 ထပ်ဖြည့်တော့ 71,3,1,3 ဖြစ်သွားတယ်။ အဲဒီ အတွက် ကျွန်တော်တို့တွေ function တစ်ခု ရေးဖို့ စဉ်းစား ရအောင်။

ဘယ်လို ရေးရင် ရမလဲ။ မရေးခင် အရင် ဆုံး စဉ်းစားကြည့်ဖို့ လိုပါတယ်။

variable တစ်ခု ထည့်လိုက်မယ်။

ကျွန်တော်တို့ Node object တစ်ခု ဆောက်ရမယ်။ အဲဒီ ထဲကို ပေးလိုက်သည့် variable ထည့် မယ်။

လက်ရှိ ရှိနေသည့် head ကို ထည့်မယ် Node ရဲ့ next ထဲမှာ ထည့်လိုက်မယ်။

list ရဲ့ head ကို temp မှာထည့်မယ်။ အဲဒါဆိုရင် ရပြီ။

code မရေးခင်မှာ တစ်ဆင့်ခြင်းဆီ စဉ်းစားပြီး ရေးသည့် အခါမှာ အမှားပြန်ပြင်ရတာ ပိုပြီး လွယ်သလို အမှားလည်း နည်းလာနိုင်သည့် အတွက် programming စလေ့လာကာစ လူတွေ အနေနဲ့ အဆင့်တိုင်း စဉ်းစားသွားဖို့ လိုပါတယ်။

ကဲ အခု ကျွန်တော်တို့တွေ add function ရေးကြည့်ရအောင်။

```
def add(self,item):
    temp = Node(item)
    temp.set_next(self.head)
```

```
        self.head = temp
```

code အပြည့်အစုံက

```
from node import Node

class UnorderedList:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def add(self,item):
        temp = Node(item)
        temp.set_next(self.head)
        self.head = temp

mylist = UnorderedList()
mylist.add(3)
mylist.add(31)
mylist.add(71)
mylist.add(10)
mylist.add(5)
mylist.add(1)
```

code ကို မရှင်းဖူး ဆိုရင် အောက်က အဆင့်လေးတွေကို ကြည့်ကြည့်ပါ။

၁။ အရင်ဆုံး list head မှာ None ရှိတယ်။

၂။ 3 ကို ထည့်လိုက်တော့ , `temp = Node(3)` ဆိုပြီး temp object ကို ဆောက်လိုက်တယ်။ အဲဒီ အချိန်မှာ temp ရဲ့ data က 3 ဖြစ်ပြီးတော့ next ကတော့ None ဖြစ်နေမယ်။

၃။ `temp.set_next(self.head)` လို့ ဆိုသည့်အတွက် temp ရဲ့ next ထဲမှာ လက်ရှိ head ကို ဝင် သွားပြီ။ head က None ဖြစ်သည့်အတွက် next ကလည်း None ဖြစ်နေမှာပဲ။

၄။ `self.head` ကို temp ထည့်လိုက်သည့်အတွက်ကြောင့် `self.head` က Node(3) ဖြစ်သွားပြီ။

၅။ 31 ကို ထပ်ဖြည့်တော့လည်း ဒီအတိုင်းပဲ။ သို့ပေမယ့် `temp.set_next(self.head)` ကြောင့် `Node(31)` ရဲ့ next က ပြီးခဲ့ `Node(3)` ဖြစ်သွားတယ်။

၆။ `self.head` က `Node(31)` ဖြစ်သွားတာကြောင့် `self.head` ထဲမှာ `Node(31)->Node(3)` ဆိုပြီး ဖြစ်သွားပါပြီ။

ဒါဆိုရင်တော့ Add လုပ်သည့် ကိစ္စကို နားလည်လောက်ပြီ။ အခု size (အရေအတွက်) ဘယ်လောက်ရှိလဲ ဆိုတာကို သိရအောင် function ရေးကြည့်ရအောင်။

Size

အခု self.head ထဲမှာ Node တွေ အများကြီးရှိနေပြီ။ Size ကို သိဖို့ကတော့ Node အရေအတွက် ဘယ်လောက် ရှိလဲ ဆိုတာ သိဖို့လိုတယ်။ Node တွေက တစ်ခုနဲ့ တစ်ခုချိတ်ထားပြီးတော့ နောက်ဆုံး next က None ဖြစ်သွားသည့် အထိပဲ။

Pseudo code လေးနဲ့ စဉ်းစားကြည့်ရအောင်။

```
Set current is head
Set count is zero
Loop Until current is None
    Increase count
    current = current.get_next()
Return count
```

Pseudo code အရ ဆိုရင် current ထဲမှာ head ကို ထည့်မယ်။ ပြီးရင် count ကို သုညကနေ စမှတ်မယ်။ current ကို None မဖြစ်မခြင်း loop ပတ်မယ်။ loop ထဲရောက်တိုင်း count ကို ၁ တိုးသွားမယ်။ ပြီးရင် current ကို လက်ရှိ current ရဲ့ next ကို ထည့်မယ်။ loop က ထွက်သွားရင် count ကို return ပြန်ပေးမယ်။ code လေးက ရှင်းရှင်းလေးပါ။ အဲဒါကို python နဲ့ ပြောင်းရေးကြည့်ရအောင်။

```
def size(self):
    current = self.head
    count = 0
    while current != None:
        count = count + 1
        current = current.get_next()
    return count
```

code အပြည့်အစုံက

```
from node import Node

class UnorderedList:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def add(self,item):
        temp = Node(item)
        temp.set_next(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.get_next()
        return count
```

```
mylist = UnorderedList()

print(mylist.size())

mylist.add(3)
mylist.add(31)
mylist.add(71)
mylist.add(10)
mylist.add(5)
mylist.add(1)

print(mylist.size())
```

တကယ့်ကို လွယ်လွယ်လေးပါ။ အခု ကျွန်တော်တို့တွေ ရှိသမျှ node တွေ ကုန်အောင် loop ပတ်လို့ ရသွားပြီ။ ဒါဆိုရင် search လုပ်လို့ ရပြီပေါ့။

Search

Search လုပ်မယ်ဆိုရင် ပြီးခဲ့တဲ့ size အတိုင်း loop ပတ်ဖို့ လိုတယ်။ တွေ့ခဲ့ရင် loop ထဲက ထွက်မယ်။ ဒါပဲ ကွာလိမ့်မယ်။

```
def search(self,item):
    current = self.head
    found = False
    while current != None and not found:
        if current.get_data() == item:
            found = True
        else:
            current = current.get_next()
    return found
```

လက်ရှိ ရှိသည့် code မှာ ပြောင်းလိုက်ရင်

```
from node import Node

class UnorderedList:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def add(self,item):
        temp = Node(item)
        temp.set_next(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.get_next()
```

```
        return count

    def search(self,item):
        current = self.head
        found = False
        while current != None and not found:
            if current.get_data() == item:
                found = True
            else:
                current = current.get_next()
        return found

mylist = UnorderedList()

mylist.add(3)
mylist.add(31)
mylist.add(71)
mylist.add(10)
mylist.add(5)
mylist.add(1)

print(mylist.search(10))
print(mylist.search(12))
```

Size ကို နားလည်တယ်ဆိုရင် search code ကလည်း ရိုးရှင်းပါတယ်။ current က None ရောက်သည့်အထိ loop ပတ်တယ်။ loop ထဲက ဘယ်အချိန်ထွက်မလဲဆိုတော့ current က None ဖြစ်သွားချိန် ဒါမှမဟုတ် found က true ဖြစ်သွားချိန်ပေါ့။

```
while current != None and not found:
```

ဒီ code မှာ `and` ကို အသုံးပြုထားတာ တွေ့နိုင်ပါတယ်။ `and` ရဲ့ သဘောက တစ်ခု `False` ဖြစ်ရင် အကုန် false ပဲ။ ၂ ခု လုံး true ဖြစ်မှ true ဖြစ်သည့် သဘောကို သိကြပါလိမ့်မယ်။ code အရ `current != None` ကလည်း `True` ဖြစ်ရမယ်။ not found ဆိုသည့် အတွက် found ကလည်း false ဖြစ်ရမယ်။ found က false ဖြစ်မှသာ not false ဆိုပြီး true ကို ရမှာပါ။ ၂ ခုလုံး true ဖြစ်နေသ၍ looping က အလုပ်လုပ်နေပါမယ်။

ကိုယ်ရှာနေသည့် item ကိုသာ တွေ့ရင် found က true ဖြစ်သွားပြီးတော့ looping ထဲကနေ ထွက်သွားပါလိမ့်မယ်။

Remove

အခု remove အပိုင်းကို စဉ်းစားကြရအောင်။ item တစ်ခုပေးလိုက်မယ်။ အဲဒီ item ကို ဖျက်ဖို့ လိုတယ်။ သူ့ရဲ့ အရှေ့က သူ့ကို ချိတ်ထားသည့် Node နဲ့ သူ့ရဲ့ အနောက်က သူ့ကို ချိတ်ထားသည့် Node ၂ ခုကို ချိတ်ပေးဖို့လိုတယ်။ သူကတော့ နည်းနည်း ရှုပ်သွားပြီ။

ကျွန်တော်တို့မှာ

```
head -> 31 -> 10 -> 8 -> 4 -> 3 -> None
```

ဆိုပြီးရှိရင် 8 ကို remove လုပ်လိုက်ရင် အောက်ကလို ဖြစ်သွားမယ်။

```
head -> 31 -> 10 -> 4 -> 3 -> None
```

အဲဒီတော့ ကျွန်တော်တို့ 8 ကို ဖျက်ဖို့ အတွက် 8 ရှေ့ ရဲ့က Node ကို မှတ်ထားမယ်။ အပေါ်က ပုံ အတိုင်း ဆိုရင်တော့ 10 ပေါ့။ Node(10) ရဲ့ next ကို Node(8) အစား Node(4) ချိတ်ပေးလိုက်ရုံပဲ။ Node(4) ဆိုတာကတော့ Node(8) ရဲ့ next ပါ။

ဒီတော့ Pseudo code လေး ရေးကြည့်ရအောင်။

```
SET current = head
SET previous = None
SET found = false
Loop Until found OR current is None
    IF current.data == item THEN
        found = true
    ELSE
        previous = current
        current = current.next
IF found == true THEN
    IF previous == None THEN
        head = current.next
    ELSE
        previous.next = current.next
```

ကျွန်တော်တို့တွေဟာ အရင် အတိုင်း loop ကို current ဟာ None ဖြစ်နေသည့် အချိန် သို့မဟုတ် not found ဖြစ်သွားသည့်အချိန် ထိ loop ပတ်ဖို့ လိုပါတယ်။ အကယ်၍ ရှာတွေ့ခဲ့ရင် ဖြစ်နိုင်ခြေ ၂ ခု ရှိတယ်။ ပထမ အခန်းဖြစ်နိုင်တာ ရယ် သို့မဟုတ် ပထမ အခန်း မဟုတ်တာရင်။ ပထမ အခန်းဆိုရင်တော့ previous က None ဖြစ်နေမှာပါ။ အဲဒီ အခါမှာ head ကို next နဲ့ ချိတ်ပေးလိုက်ရုံပဲ မဟုတ်ရင်တော့ previous ရဲ့ next ကို current ရဲ့ next နဲ့ ချိတ်ပေးဖို့ လိုပါတယ်။

python နဲ့ ရေးကြည့်ရအောင်။

```
def remove(self,item) :
    current = self.head
    previous = None
    found = False
    while current != None and not found:
        if current.get_data() == item:
            found = True``````````````````
        else:
            previous = current
            current = current.get_next()
    if found :
        if previous == None:
            self.head = current.get_next()
        else:
            previous.set_next(current.get_next())
```

code တွေအကုန်ပြန် စုလိုက်ရင်

```
from node import Node

class UnorderedList:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def add(self,item):
        temp = Node(item)
        temp.set_next(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.get_next()
        return count

    def search(self,item):
        current = self.head
        found = False
        while current != None and not found:
            if current.get_data() == item:
                found = True
            else:
                current = current.get_next()
        return found

    def remove(self,item) :
        current = self.head
        previous = None
        found = False
        while current != None and not found:
            if current.get_data() == item:
                found = True
            else:
                previous = current
                current = current.get_next()
        if found :
            if previous == None:
                self.head = current.get_next()
            else:
                previous.set_next(current.get_next())

mylist = UnorderedList()

mylist.add(3)
mylist.add(31)
mylist.add(71)
mylist.add(10)
mylist.add(5)
mylist.add(1)

print(mylist.size())
mylist.remove(5)
print(mylist.size())
```

```
mylist.remove(100)
print(mylist.size())
```

python code လေးကလည်း ရိုးရှင်းပါတယ်။ အခုဆိုရင်တော့ ကျွန်တော်တို့တွေ remove အပိုင်း ပြီးသွားပါပြီ။

ကျန်တဲ့ append,insert,index,pop စတာတွေကိုတော့ လေ့ကျင့်ခန်း အနေနဲ့ ကိုယ်တိုင် ရေးဖို့ လိုအပ်ပါတယ်။

The Ordered List Abstract Data Type

အခု ကျွန်တော်တို့ ထပ်ပြီးတော့ ordered list ကို ဖန်တီးကြည့်ရအောင်။ Ordered List ဆိုတာ ကတော့ unordered list လို မဟုတ်ပဲ နံပတ်စဉ် လိုက်တိုင်း စီထားသည့် list ပေါ့။ Order List မှာ ဘာတွေပါမလဲဆိုတော့

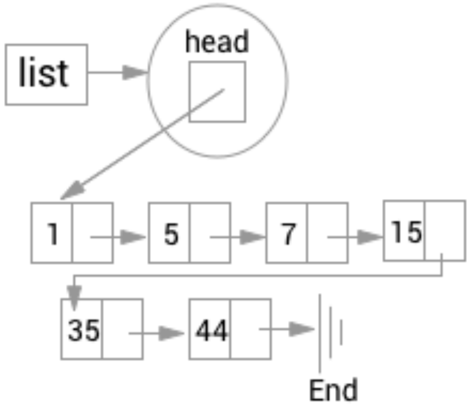
- `OrderList()` ဆိုတဲ့ class တစ်ခု ဖန်တီးမယ်။ return ကတော့ empty list ပြန်မယ်။
- `emove(item)` ကတော့ list ထဲမှာ ရှိသည့် item ကို ဖျက်မှာပါ။
- `search(item)` ကတော့ list ထဲမှာ item ပါမပါ ရှာပါလိမ့်မယ်။
- `is_empty()` ကတော့ list က empty ဟုတ်မဟုတ် အတွက်ပါ။
- `size()` ကတော့ list ထဲမှာ item ဘယ်လောက် ရှိသလဲဆိုတာကို သိဖို့ အတွက်ပါ။
- `index(item)` ကတော့ list ထဲမှာ item က ဘယ် position , ဘယ် အခန်း မှာ ရှိလဲ ဆိုတာကို return ပြန်ပေးမှာပါ။
- `pop()` ကတော့ နောက်ဆုံး အခန်းကို ထုတ်မယ်။ အဲဒီ value ကို return ပြန်ပေးမယ်။
- `pop(pos)` ကတော့ နောက်ဆုံး အခန်းမဟုတ်ပဲ ပေးလိုက်သည့် အခန်း နံပတ်ကို ဖျက်မယ်။ ပြီးရင် အဲဒီ အခန်းက value ကို return ပြန်ပေးပါမယ်။

Implementing an Ordered List: Linked Lists

Unordered List ကို ဖန်တီးထားပြီးပြီ ဆိုတော့ ကျွန်တော်တို့တွေ အတွက် Ordered List ဖန်တီးဖို့ မခက်ခဲတော့ပါဘူး။ Ordered List ကတော့ နံပတ်တွေကို အစီအစဉ်လိုက် စီထားသည့် list ပါ။ Unordered List မှာကတော့ နံပတ်စဉ်တွေ အတိုင်း list ထဲမှာ ရှိနေတာ မဟုတ်ပါဘူး။ ဒါကြောင့် မတူညီတာကတော့ list ထဲကို item ထည့်တော့မယ်ဆိုရင် ထည့်မယ့် value ထက် ကြီးတာကို သွားရှာရမယ်။ သူ့ရဲ့ အရှေ့မှာ သွားထည့်ရမယ်။ Unordered List လိုမျိုး ထည့်ချင်သလို ထည့် လို့ရတာ မဟုတ်ပါဘူး။

Ordered List ပုံစံကို ကြည့်ရအောင်

Unordered List နဲ့ ဆင်သယောက်ပါပဲ။ ကွာတာကတော့ သူက ကြီးစဉ်ငယ်လိုက် အစီအစဉ် လိုက် စီထားတာပါ။



အခု Class တစ်ခု ကို ဖန်တီးကြည့်ရအောင်

```
class OrderedList:
   def __init__(self):
      self.head = None
```

ဒါကတော့ အရင်အတိုင်းပဲ။ ပုံမှန် class တစ်ခု ဖန်တီးထားတာပါ။

Unordered List အကြောင်းသိပြီးပြီ ဖြစ်သည့် အတွက် ကျွန်တော်တို့တွေ add ကို နောက်မှ ရေး မယ်။ အခု search လေးက စလိုက်ရအောင်။

```
def search(self, item):
    current = self.head
    found = False
    stop = False
    while current != None and not found and not stop:
    if current.get_data() == item:
        found = True
    else:
        if current.get_data() > item:
        stop = True
        else:
        current = current.get_next()
   return found
```

code ဖတ်လိုက်တာနဲ့ အခုဆို နားလည်လောက်ပြီလို့ ထင်ပါတယ်။ Search လုပ်တယ် ၊ ရှာတယ် ဆိုသည့် သဘောကတော့ ရှင်းရှင်းလေးပါ။ တစ်ခန်းခြင်းစီမှာ ဒီ value ဟုတ်လား ၊ မဟုတ်ခဲ့ရင် value က အခု လက်ရှိ အခန်းထက် ကြီးနေလားဆိုပြီး စစ်ပါတယ်။ ဘာလို့ စစ်ရလဲ ဆိုတော့ ဂဏန်းတွေက ကြီးစဉ်ငယ်လိုက် ရှိနေတော့ ကြီးသွားရင်တော့ သေချာပြီ နောက်ဘက်အခန်း တွေမှာ လည်း မရှိတော့ဘူး။ အကယ်၍ မရှိခဲ့ဘူး ဆိုရင်တော့ နောက်အခန်းကို ဆက်သွားပြီး ရှာ ဖို့ လိုပါလိမ့်မယ်။

ဘာကြောင့် Search ကို အဓိက ထားပြီး အရင်ပြောရသလဲ ဆိုတော့ Search ပိုင်းနားလည် သဘောပေါက်မှ Ordered List မှာ Add အပိုင်း ထည့်လို့ ရပါလိမ့်မယ်။ Ordered List က ထည် မယ်ဆိုရင် ထည့်မယ် value ထက် ကြီးထက်တာကို ရှာရမယ်။ ပြီးရင် အဲဒီ အရှေ့မှာ ထည့်ဖို့ လို ပါတယ်။

```
def add(self, item):
   current = self.head
   previous = None
   stop = False
   while current != None and not stop:
      if current.get_data() > item:
         stop = True
      else:
         previous = current
         current = current.get_next()
   temp = Node(item)
   if previous == None:
      temp.set_next(self.head)
      self.head = temp
   else:
      temp.set_next(current)
      previous.set_next(temp)
```

အခု add function ကို ရေးပြီးပါပြီ။ code လေး တချက်လောက် ကြည့်ရအောင်။ Search မှာ ကတော့ found ဆိုပြီး အသုံးပြုထားပြီးတော့ add မှာကတော့ previous ကို အသုံးပြုထားပါ တယ်။ အခု လက်ရှိ အခန်းမတိုင်ခင်က အခန်းပေါ့။ ဒါမှသာ ကျွန်တော်တို့တွေဟာ လက်ရှိ အခန်းနဲ့ သူ့ရဲ့ ပြီးခဲ့တဲ့ အခန်းကြားမှာ value ကို ထည့်လိုက်ရင် ရပါပြီ။

Search အတိုင်းပါပဲ။ ကျွန်တော်တို့တွေဟာ Loop ပတ်ပြီးတော့ ထည့်မယ် item ထက်ကြီးတာကို ရှာတယ်။ နောက်ဆုံး အခန်း မရောက်မခြင်း ရှာပါတယ်။ ဒါမှမဟုတ် current item က လက်ရှိ item ထက်ကြီးသွားမလားဆိုပြီးတော့လည်း ရှာပါတယ်။ မကြီးဘူးဆိုရင်တော့ previous ထဲမှာ current ကို ထည့်တယ်။ current ကိုတော့ current ရဲ့ next ကို ထည့်ပါတယ်။

ပြီးသွားပြီဆိုရင် နောက်ဆုံး အခန်းရောက်သွားလား သိရင်အောင်

```
if previous == None:
```

ဆိုပြီးရှာပါတယ်။ နောက်ဆုံး အခန်းဆိုရင်တော့ နောက်ဆုံး အခန်းမှာ ထည့်လိုက်ရုံပဲပေါ့။

မဟုတ်ခဲ့ဘူးဆိုရင်တော့ item ရဲ့ next ကို current ထည့်မယ်။ previous ရဲ့ next ကိုတော့ item ရဲ့ Node လေး ချိတ်ပေးလိုက်ရုံပါပဲ။ Code က လွယ်လွယ် နဲ့ ရိုးရိုး ရှင်းရှင်းပါပဲ။ ကျွန်တော် အဓိက Search ရဲ့ Add ပဲ ပြောသွားတယ်။ ကျန်တာတွေကို Unordered List နဲ့ ပေါင်းလို့ ရတယ်။

Unordered List မှာ exercise လုပ်ဖြစ်သည့်သူတွေ အနေနဲ့ pop အပိုင်းကို စဉ်းစားဖူးပါလိမ့်မယ်။

pop ကို ရေးသားဖို့အတွက် စဉ်းစားကြည့်ရအောင်။

ပထမဆုံး စဉ်းစားရမှာက နောက်ဆုံး အခန်းကို ဘယ်လိုသွားမလဲ ?
နောက်ဆုံး အခန်းကို ဘယ်လို ဖျက်မလဲ
ကျွန်တော်တို့ အရင်က ရေးထားသလိုပါပဲ။ နောက်ဆုံး အခန်းက None ဖြစ်တယ်။ ဒါဆိုရင်တော့ လက်ရှိ Node ရဲ့ next value ကသာ None ဖြစ်သွားခဲ့ရင် အဲဒါက နောက်ဆုံး အခန်းပဲ။ ဒီတော့ current ရဲ့ next က None မဖြစ်မခြင်း Loop ပတ်ဖို့ လိုတယ်။

```
while current.get_next() != None :
```

နောက်တစ်ခုက နောက်ဆုံးခန်း ဘယ်လိုဖျက်မလဲ ဆိုတာက လွယ်သွားပြီ။ လက်ရှိ current ရဲ့ ရှေ့ အခန်းမှာ next ကို None ပေးလိုက်ရုံပါပဲ။

```
def pop(self) :
        current = self.head
        previous = None
        while current.get_next() != None :
            previous = current
            current = current.get_next()

        previous.set_next(None)
        return current.get_data()
```

ကဲ အခု ကျွန်တော်တို့ Ordered List class တစ်ခု လုံး စမ်းကြည့်ရအောင်။

```
from node import Node

class OrderedList:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def add(self, item):
        current = self.head
        previous = None
        stop = False
        while current != None and not stop:
            if current.get_data() > item:
                stop = True
            else:
                previous = current
                current = current.get_next()
        temp = Node(item)
        if previous == None:
            temp.set_next(self.head)
            self.head = temp
        else:
            temp.set_next(current)
            previous.set_next(temp)

    def size(self):
        current = self.head
```

```
        count = 0
        while current != None:
            count = count + 1
            current = current.get_next()
        return count

    def search(self, item):
        current = self.head
        found = False
        stop = False
        while current != None and not found and not stop:
            if current.get_data() == item:
                found = True
            else:
                if current.get_data() > item:
                    stop = True
                else:
                    current = current.get_next()
        return found

    def remove(self,item) :
        current = self.head
        previous = None
        found = False
        while current != None and not found:
            if current.get_data() == item:
                found = True
            else:
                previous = current
                current = current.get_next()
        if found :
            if previous == None:
                self.head = current.get_next()
            else:
                previous.set_next(current.get_next())

    def pop(self) :
        current = self.head
        previous = None
        while current.get_next() != None :
            previous = current
            current = current.get_next()

        previous.set_next(None)
        return current.get_data()

mylist = OrderedList()

mylist.add(3)
mylist.add(31)
mylist.add(71)
mylist.add(10)
mylist.add(5)
mylist.add(1)

print(mylist.size())
mylist.remove(5)
print(mylist.size())
print(mylist.pop())
print(mylist.size())
```

Dictionary

အခု List ပိုင်းကို ကျွန်တော်တို့တွေ နားလည် သဘောပေါက်လောက် ရှိပါပြီ။ Programming မှာ Array , List အပြင် လူသုံးများသည့် နောက်ထပ် data type တစ်ခုကတော့ Dictionary ပါ။ Java မှာတော့ HashMap အနေနဲ့ သုံးတယ်။ PHP မှာတော့ associative array လို့ ခေါ်တယ်။ Dictionary ရဲ့ အဓိက ပိုင်းကတော့ Key Value ပါ။ value တွေကို Key နဲ့ သိမ်းပြီးတော့ ပြန်ထုတ်မယ်ဆိုရင် အခန်းနံပတ်နဲ့ မဟုတ်ပဲ Key နဲ့ ပြန်ထုတ်မှ ရပါမယ်။

```
dict = {'Name': 'Aung Ko', 'Age': 7}

print("NAME: " + dict["Name"])
print("Age: " + str(dict["Age"]))
```

အဲဒီ code မှာ ကြည့်လိုက်ရင် Name,Age စတာတွေက Key ပါ။ Key ကို ထည့်လိုက်သည့် အခါ မှာ Value ရလာတာကို တွေ့ရမှာပါ။ `dict["Name"]` အစား `dict["Aung Ko"]` ကို ခေါ်ရင် Name ဆိုပြီး ထွက်လာမှာ မဟုတ်ပါဘူး။ Value ကနေ Key ကို ပြန်ခေါ်လို့ မရပါဘူး။

နောက်ထပ် ဥပမာ ကြည့်ရအောင် ဗျာ။

```
person1 = {'Name': 'Aung Ko', 'Age': 7}
person2 = {'Name': 'Ko Ko', 'Age': 8}

room = [person1,person2]

for person in room:
    print("NAME: " + person["Name"])
    print("Age: " + str(person["Age"]))
    print("======")
```

ကျွန်တော်တို့ Dictionary ကို array ထဲမှာ ထည့်လိုက်တယ်။ တနည်းပြောရင် အခန်းထဲမှာ ရှိသည့် လူတွေ အကုန် array ထဲမှာ ထည့်ပြီးတော့ ပြန်ထုတ်ထားသည့် သဘောပေါ့။

```
for person in room:
```

ဆိုတာကတော့ room array ကို loop ပတ်မယ်။ အထဲမှာ ရှိသည့် data ကို person ထဲမှာ ထည့် မယ်။ ဒါဆိုရင် person က dictionary ဖြစ်သွားပါပြီ။ အဲဒီ ထဲကနေ key နဲ့ ပြန်ပြီး ဆွဲထုတ်ထား တာပါ။ ပြီးခဲ့တဲ့ code နဲ့ သဘောတရား အတူတူပါပဲ။

Updating

Dictionary မှာ Value ကို အမြဲပြန်ပြင်ပြီး Update လုပ်လို့ ရပါတယ်။

```
dict = {'Name': 'Aung Ko', 'Age': 7}
```

```
dict["Age"] = 9

print("NAME: " + dict["Name"])
print("Age: " + str(dict["Age"]))
```

Update လုပ်သည့်အခါမှာလည်း Key နဲ့ တိုက်ရိုက် update လုပ်နိုင်ပါတယ်။

Delete Dictionary

Key ကို ပြန်ပြီး ဖျက်ချင်ရင်

```
dict = {'Name': 'Aung Ko', 'Age': 7}

dict["Age"] = 9

del dict['Name']

print("NAME: " + dict["Name"])
print("Age: " + str(dict["Age"]))
```

ဒီ code မှာ ဆိုရင် Name ကို ဖျက်လိုက်သည့်အတွက် ပြန်ထုတ်သည့် အခါမှာ Error ဖြစ်ပါလိမ့်မယ်။

ဒီလောက်ဆိုရင်တော့ Dictionary ကို အနည်းငယ် သဘောပေါက်လောက်ပါပြီ။ Dictionary ဟာ နောက်ပိုင်း programming တွေ ရေးသည့် အခါမှာ မဖြစ်မနေ အသုံးဝင်လာပါလိမ့်မယ်။ Web Development ပိုင်းတွေ သွားသည့်အခါမှာ API နဲ့ ချိတ်ဆက်ပြီး ရေးသည့် အပိုင်းမှာ Dictionary ရဲ့ အရေးပါပုံတွေကို တွေ့လာရလိမ့်မယ်။ အခုစာအုပ်မှာတော့ အခြေခံ သဘောတရားလေးကိုသာ ကျွန်တော် ဖော်ပြထားပါတယ်။

JSON

Dictionary ကို နားလည်ပြီဆိုရင်တော့ JSON အကြောင်းကို နားလည်လွယ်ပါပြီ။ JSON ဟာ web application အတွက် API တွေရေးရာမှာ အသုံးများသည့် format တစ်ခုပါ။ API ဆိုတာကတော့ Apllication Program Interface ဖြစ်ပါတယ်။ ဥပမာ။။ user information ပေးပါ ဆိုပြီး client က request ပြုလုပ်လျှင် ဒါတွေကတော့ user information ပါ ဆိုပြီး return ပြန်ပေးပါလိမ့်မယ်။

ဥပမာ။

```
{
        "id" : 1,
        "name" : "Kaung Kaung",
        "age" : 8
}
```

ဒါကို ကြည့်လိုက်ရင် Dictionary နှင့် တူပါသလား ဆိုပြီး သဘောပေါက်နိုင်ပါတယ်။ Dictionary နဲ့ JSON ကွာခြားချက်ကတော့ Dictionary ဟာ python မှာ ရှိသည့် data type တစ်ခု ဖြစ်ပါသည်။ JSON ကတော့ format ရှိသည့် string သို့မဟုတ် text ပါ။ Dictionary ၏ data type မှာ python ရဲ့ instant object ကို ထည့်သွင်းနိုင်သည်။ JSON မှာကတော့ Number, String , Boolean, JSON Array နှင့် JSON Object ကို သာထည့်သွင်းနိုင်ပါသည်။

ဥပမာ။

```
{
        "id" : 1,
        "name" : "Kaung Kaung",
        "age" : 22
}
```

အထက်ပါ ပုံစံသည် JSON object ပါ။

```
[
  {
    "id" : 1,
    "name" : "Kaung Kaung",
    "age" : 8
  },
  {
    "id" : 2,
    "name" : "Yan Kyi Naung",
    "age" : 9
  },
  {
    "id" : 3,
    "name" : "Zaw Htet",
    "age" : 10
}
]
```

အထက်ပါ ပုံစံသည် JSON Array ဖြစ်ပါသည်။ JSON Array ကို `[` နှင့် စပြီး `]` နှင့် ဆုံးပါသည်။

```
{
  "id" : 1,
  "name" : "Kaung Kaung",
  "age" : 8,
  "address" : {
    "no" : 129,
    "street" : "31 St",
    "city" : "Yangon",
    "country" : "Myanmar"
  }
}
```

JSON သည် အထက်တွင် ဖော်ပြထားသည့် ပုံစံမျိုး JSON Object ထဲတွင် နောက်ထပ် JSON Object ရှိနိုင်သည်။

```
{
        "id": 1,
        "name": "Kaung Kaung",
        "age": 8,
        "addresses": [{
                        "no": 129,
                        "street": "31 St",
                        "city": "Yangon",
                        "country": "Myanmar"
                },
                {
                        "no": 89,
                        "street": "23 St",
                        "city": "Yangon",
                        "country": "Myanmar"
                }
        ]
}
```

JSON သည် အထက်တွင် ဖော်ပြထားသည့် ပုံစံမျိုး JSON Object ထဲတွင် နောက်ထပ် JSON Array ရှိနိုင်သည်။

JSON ကို အဓိကအားဖြင့် data parsing အဖြင့် အသုံးများပါသည်။ နောက်ပိုင်း web development ပြုလုပ်သည့် အခါ ပိုမို နားလည်လာပါမည်။.

အခု JSON string ကို Dictionary ပြောင်းလဲပါမည်။ json အတွက် အောက်ပါ code ကို import လုပ်ရန် လိုအပ်ပါသည်။

```
import json
```

JSON string တစ်ခု တည်ဆောက်ပါမည်။

```
json_data = '{"a": 1, "b": 2, "c": 3, "d": 4, "e": 5}'
```

ထို JSON string အား Dictionary ပြောင်းလဲပါမည်။

```
sample_dict = json.loads(json_data)
```

အခုဆိုလျှင် json string သည် Dictionary ဖြင့်သွားပါမည်။

```
print(sample_dict['a'])
```

အခုဆိုလျှင် `1` ကို ဖော်ပြပေးမည် ကို တွေ့ရပါမည်။

အခု dictionary object ကို JSON ပြောင်းကြည့်ပါမည်။

```
print(json.dumps(sample_dict))
```

အထက်ပါ code တွင် `json.dumps` သည် dictionary object အား json string ပြောင်းပေးခြင်း ဖြစ်သည်။ သတိထားရမှာသည် dictionary ထဲတွင် string,number နှင့် boolean value သာ ဖြစ်ရမည် ဖြစ်သည်။

JSON ကို config file , API စသည်တို့တွင် အသုံးများသည့် အတွက် အခြေခံ သဘောတရားနားလည်အောက် ဖော်ပြထားခြင်းဖြစ်ပါသည်။

လေ့ကျင့်ခန်း ၆။

၁။ အောက်ပါ code တွင် gender သည် male ဖြစ်သော သူများကို ဖော်ပြပါ။

```
room = [{'Name': 'Aung Ko', 'Age': 7, 'Gender' : 'male'}, {'Name': 'Ko Ko', 'Age': 8, 
'Gender' : 'male'},{'Name': 'Aye Aye', 'Age': 7, 'Gender' : 'female'},{'Name': 'Htet 
Htet', 'Age': 8, 'Gender' : 'female'},{'Name': 'Win Aung', 'Age': 7, 'Gender' : 'male'}]
```

၂။ အထက်ပါ code တွင် room ထဲတွင် ရှိသော လူများ၏ စုစုပေါင်း အသက်ကို ဖော်ပြသော code ရေးပြပါ။

အခန်း ၇ ။ Recursion

အခန်း ၇ ရောက်လာပြီဆိုတော့ programming နဲ့ ပတ်သက်ပြီးတော့ နားလည်လောက်ပါပြီ။ အခု chapter ကနေ စပြီးတော့ သေသေချာချာ လိုက်စဉ်းစားဖို့ လိုလာပြီ။ အခု chapter မှာတော့ recursion အကြောင်းကို ရှင်းပြသွားပါမယ်။

Recursion ဆိုတာလဲ

Recursion ဆိုတာကတော့ programming မှာ တူညီသည့် ပြဿနာကို တူညီသည့် ဖြေရှင်းမှု နဲ့ ထပ်ခါ ထပ်ခါ ရှင်းသည့် ပုံစံပါ။ တနည်းအားဖြင့် looping နဲ့ ဆင်သလိုပါပဲ။ looping ကတော့ စမှတ်ရှိပြီး ဆုံးမှတ်ရှိပါတယ်။ Recursion မှာလည်း ဘယ်အချိန်မှာ ဒီ function ထဲကို ပြန်ထွက်မယ်ဆိုပြီး ရှိပါတယ်။ recurrsion ကတော့ ဒီ function ကိုပဲ ထပ်ခါ ထပ်ခါ ခေါ်နေပြီးတော့ နောက်ဆုံး function ကနေ ပြန်ထွက်သွားမယ့် အထိပါ။

Recursion ကို ဘယ်လို သုံးလဲ

ကျွန်တော်တို့ အောက်က ဥပမာလေးကို ကြည့်ရအောင်။

```
for x in range(0, 5):
    print ("Hello World",x)
```

code လေးကတော့ ရှင်းပါတယ်။ looping ပတ်ပြီးတော့ Hello World ကို နံပတ်လေးနဲ့ ထုတ်ထားတာပါ။ ကျွန်တော်တို့ ပုံစံ ပြောင်းရေးကြည့်ရအောင်

```
def recursive(string, num):
    if num == 5:
        return
    print (string,num)
    recursive(string,num+1)

recursive("Hello World",0)
```

ဒါကတော့ recursive နဲ့ ပြန်ပြင်ရေးထားတာပါ။ code လေးကို သဘောပေါက်အောင် ကြည့်ကြည့်ပါ။

```
recursive("Hello World",0)
```

ဆိုပြီး function ကို လှမ်းခေါ်လိုက်တယ်။

```
if num == 5:
    return
```

num က 5 မဟုတ်သည့် အတွက်ကြောင့် return မဖြစ်ဘူး။ num က သာ 5 ဖြစ်ခဲ့မယ်ဆိုရင် function က ဆက်ပြီးတော့ အလုပ်လုပ်မှာ မဟုတ်ဘူး။

```
print (string,num)
recursive(string,num+1)
```

ပြီးတော့ print ထုတ်လိုက်တယ်။ ထပ်ပြီးတော့ ဒီ function ပဲ ထပ်ခေါ်တယ်။ num လေးကို 1 ပေါင်းပြီးတော့ ထပ်ခေါ်လိုက်ပါတယ်။

ဒီ code ၂ခု မှာတော့ ရလဒ်ကတော့ အတူတူပါပဲ။ ကွာခြားပုံကတော့ looping နဲ့ ရေးတာနဲ့ recursion ပုံစံ ရေးတာပဲ ကွာခြားသွားပါတယ်။

Calculating the Sum of a List of Numbers

အခု ထပ်ပြီးတော့ လေ့လာကြည့်ရအောင်။ ကျွန်တော်တို့မှာ function တစ်ခုရှိမယ်။ [1,2,5,9,7] ဆိုသည့် array ထဲက number တွေ အကုန် ပေါင်းသည့် function လေးပါ။

code လေးကို ကြည့်ရအောင်

```
def listsum(numList):
    sum = 0
    for i in numList:
        sum = sum + i
    return sum

print(listsum([1,2,5,9,7]))
```

```
sum = 0
for i in numList:
    sum = sum + i
```

loop ကတော့ numList ထဲမှာ ပါသည့် number တွေ အကုန် loop ပတ်မယ်။

အဆင့်တွေ အနေနဲ့

```
sum = 0 + 1
sum = 1 + 2
sum = 3 + 5
sum = 8 + 9
```

```
sum = 17 + 7
```

ရှင်းရှင်းလေးပါ။ နောက်ဆုံး `sum = 24` ထွက်လာပါတယ်။

ဒီ code လေးကို ထပ်ပြီးတော့ နားလည် အောင် ကြည့်ရအောင်။

ပထမဆုံး နံပတ် နဲ့ နောက်က နံပတ် စဉ်တွေ ကို ပေါင်းသွားတာနဲ့ မတူဘူးလား။ အဲဒီ sum ကို ကျွန်တော်တို့တွေ ဖြန့်ပြီး ရေးကြည့်ရင်

```
sum = (1 + (2 + (5 + (9 + 7))))
```

အဲလို အတန်းကြီး ရပါတယ်။ အဲဒီ သဘောတရားဟာ အောက်ကလို ပုံစံ နဲ့ တူတယ်

```
listsum(numList) = first(numList) + listsum(rest(numList))
```

ရှေ့ဆုံး အခန်းကို ယူတယ်။ ပြီးရင် ကျန်တဲ့ အခန်းနဲ့ ပေါင်းတယ်။ နောက်တစ်ခေါက် ရှေ့ဆုံး အခန်းကို ယူတယ်။ ကျန်တဲ့ အခန်းနဲ့ ပေါင်းတယ်။ ဘယ်အချိန်ထိလဲ ဆိုတော့ list ထဲမှာ ၁ ခုပဲ ကျန်သည့် အထိပဲ။ တစ်ခု ပဲ ကျန်ရင်တော့ ထပ်ပြီး မပေါင်းတော့ဘူး။ ဒီ အတိုင်းပဲ ရှိနေတယ်။

ဒါလေးကို သဘောပေါက်ရင် ကျွန်တော်တို့တွေ ရေးလို့ ရပါပြီ။ မရေးခင် မှာ ကျွန်တော်တို့ သိထားသင့်တာက python မှာ ရှေ့အခန်းမပါပဲ ကျန်တဲ့ အခန်းတွေ အကုန် list ထဲကနေ ဘယ်လို ယူရမလဲ ဆိုတာပါ။

```
k = [1,2,5,9,7]

print(k[1:])
print(k[2:4])
print(k[:3])
```

```
k[1:]
```

list ထဲမှာ ရှေ့ဆုံး တစ်ခန်းက လွဲပြီး ကျန်တဲ့ အခန်း အကုန်ပါ ဆိုသည့် အဓိပ္ပာယ်ပါ။

```
k[2:]
```

ဆိုရင်တော့ ရှေ့ဆုံး ၂ ခန်း ပေါ့။

```
k[2:4]
```

ဒီ code ရဲ့ အဓိပ္ပာယ်ကတော့ ရှေ့ ၂ ခန်း မယူဘူး။ နောက်ပြီး ၄ လုံး မြောက် အထိပဲ ယူမယ်။ ဒါကြောင့် [5, 9] ဆိုပြီး ထွက်လာပါတယ်။ ရှေ့ဆုံး ၂ ခန်း မပါဘူးဆိုတော့ တတိယ အခန်း 5 က နေ စတယ်။ ၄ ခု မြောက်ထိပဲ ဆိုတော့ 9 မှာ ဆုံးတယ်။ ဒါကြောင့် [5,9] ဖြစ်သွားတာပါ။

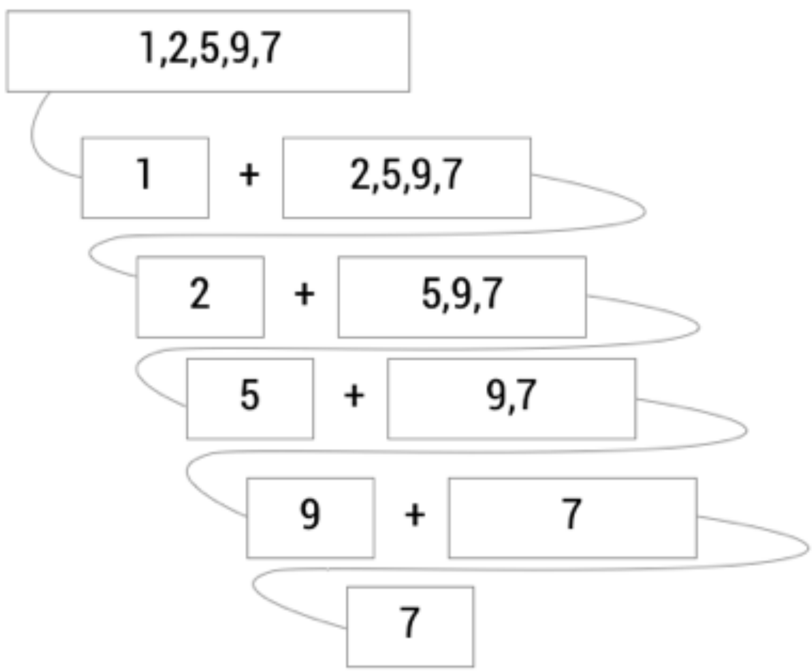
```
k[:3]
```

ကတော့ ရှေ့က အကုန်ယူမယ်။ ဒါပေမယ့် ၃ လုံး မြောက်နေရာ အထိပဲ ယူမယ်။ဒါကြောင့် `[1, 2, 5]` ဆိုပြီးတော့ ထွက်လာပါတယ်။

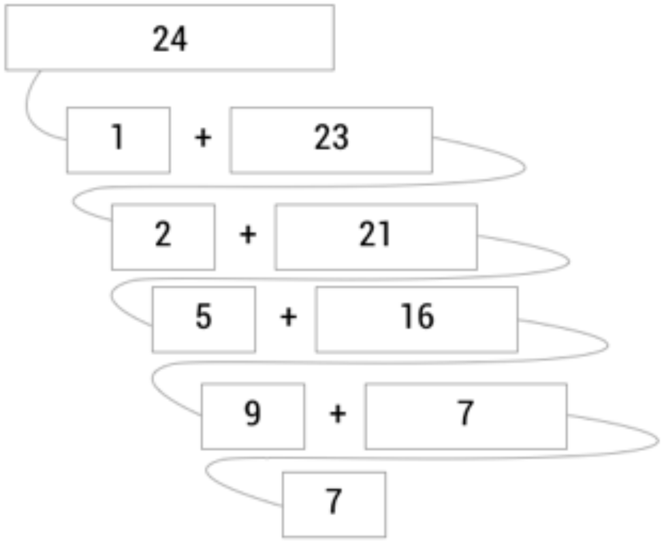
ဒီသဘောတရားကို သိပြီဆိုရင် ကျွန်တော်တို့ recursive ရေးထားသည့် function ကို ကြည့် ရအောင်။

```
def listsum(numList):
   if len(numList) == 1:
        return numList[0]
   else:
        return numList[0] + listsum(numList[1:])

print(listsum([1,2,5,9,7]))
```

အဲဒီ code လေးကို ရှင်းဖို့ အောက်ကပုံ ကိုကြည့်ကြည့်ပါ။



ကျွန်တော်တို့ list လေး ပေးလိုက်တယ်။ ဒီ function ကို ပဲ အဆင့်ဆင့်လေး ခေါ်သွားလိုက်တော့ ပုံထဲက အတိုင်းလေး ဖြစ်သွားတယ်။ နောက်တော့ return ပြန်လာတယ်။



ဒီပုံကို အောက်ကနေ အပေါ် return ပြန်လာသည့် ပုံပါ။ အောက်ဆုံးကနေ အပေါ်ဆုံးကို တဆင့်ခြင်းဆီ အဆင့်ဆင့် return ပြန်ရင်းနဲ့ အဖြေထွက်လာတာကို တွေ့နိုင်ပါတယ်။

ကျွန်တော်တို့တွေ ဆက်ပြီးတော့ အခြား ဥပမာ ကို လေ့လာကြည့်ရအောင်။

Fibonacci sequence

သင်္ချာမှာ fibonacci sequence ကို မှတ်မိသေးလား မသိဘူး။ fibonacci sequence ဆိုတာကတော့ `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 134` စသည်ဖြင့်ပေါ့။ 0 ကနေ စတယ်။ ပြီးရင် 1 လာတယ်။ 0 နှင့် 1 ပေါင်းတော့ 1 ရတယ်။ ဒါကြောင့် တတိယက 1 ဖြစ်တယ်။ နောက်ထပ်တစ်ခု အတွက် 1+1 ပေါင်းတယ်။ 2 ရတယ်။ ပြီးတော့ 1+2 ပေါင်းတယ်။ 3 ရတယ်။ သဘောကတော့ ရှေ့ ၂ လုံး ပေါင်းပြီးတော့ နောက်ထပ် တစ်လုံးကို ဖော်ပြပေးတာပဲ။

fibonacci sequence အလုံး ၁၀ မြောက်က value ကို လိုချင်တယ် လို့ ပြောလာရင် ကျွန်တော်တို့ program က ပြန်ထုတ်ပေးနိုင်အောင် ရေးဖို့ လိုပါတယ်။ Looping နဲ့ ရေးတာကိုတော့ လေ့ကျင့်ခန်း အနေနဲ့ ကိုယ်တိုင် ရေးကြည့်ပါ။

```
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
```

ဒါဆိုရင် ကျွန်တော်တို့ အနေနဲ့ အောက်က ပုံစံ အတိုင်း ရေးလို့ ရတယ်။

```
fib(6) = fib(5) + fib(4)
```

တနည်းအားဖြင့်

```
fib(n) = fib(n-1) + fib(n-2)
```

လို့ တောင် ဆိုနိုင်တယ်။ သို့ပေမယ့် အမြဲ မမှန်ဘူး။ ဘယ်အချိန်မှ အဲဒါကို သုံးလို့ ရလည်း ဆို တော့ fib(2) ကမှ စပြီး အသုံးပြုနိုင်မယ်။ တနည်းအားဖြင့်

```
if n < 2
    return n
```

ဆိုပြီး ပြောလို့ ရတယ်။

အခုဆိုရင် ကျွန်တော်တို့ program မရေးခင်မှာ ဘယ်လို ရေးရမလဲ ဆိုတာကို သဘောပေါက်ပြီ။ စဉ်းစားလို့ ရပြီ။ code ချရေးကြည့်ဖို့ပဲ လိုတော့တယ်။

```
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)

print(fib(10))
```

Recursive သဘောတရားနှင့် စဉ်းစားပုံက looping နဲ့ ကွာတယ်ဆိုတာကို fibonacci sequence ကို looping နဲ့ လေ့ကျင့်ခန်း လုပ်ကြည့်ရင် သိနိုင်ပါတယ်။

The Three Laws of Recursion

Recursion ကို အသုံးပြုမယ် ဆိုရင် လိုက်နာရမည့် နည်း ဥပဒေသ ၃ ခု ရှိပါတယ်။

၁။ recursive function ထဲကနေ ပြန်လည်ထွက်ခွာ ရမည့် အခြေအနေ ရှိရမည်။ ၂။ recursive algorithm ဟာ အခု အခြေအနေကနေ နောက်ထပ် အခြေအနေ တစ်ခုကို ပြောင်းလဲ သွားပြီး တော့ recursive function ကနေ ထွက်မယ့် အခြေအနေထိ ရောက်ဖို့လိုတယ်။ ၃။ recursive function ဟာ တူညီသည့် function ကို ပဲ ထပ်ခါထပ်ခါ ခေါ်နေဖို့လိုတယ်။

ဒီအတိုင်း ရေးပြရင်တော့ သဘောပေါက်မှာမဟုတ်ဘူး။ Fibonacci sequence က code လေးနဲ့ ကြည့်ရအောင်။

```
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

```
print(fib(10))
```

ပထမဆုံး ဥပဒေသ ၁ အရ function ထဲကနေ ထွက်ခွာဖို့ အခြေအနေ ရှိဖို့ လိုပါတယ်။

```
if n < 2:
    return n
```

ဒါကြောင့် အထက်ပါ code လေးက ဒီ function ကို ထပ်မခေါ်တော့ပဲ recursive ကို အဆုံး သတ်ခဲ့ပါတယ်။

ဒုတိယ ဥပဒေသ အရ recursive function ကနေ ထွက်ခွာသွားရမယ့် အခြေအနေကို ကူးပေါင်းရမယ်။

```
return fib(n-1) + fib(n-2)
```

ဒီမှာ ဆိုရင် `fib(n)` ကနေ ထပ်ပြီး `n-1` နှင့် `n-2` ကို ရေးထားတာတွေ့နိုင်ပါတယ်။ `n < 2` အထိ ရောက်သွားနိုင်သည့် အခြေ အနေပါ။

အထက်ပါ code အတိုင်း ဥပဒေသ ၃ အရ ဒီ function ကိုပဲ ထပ်ခါ ထပ်ခါ ခေါ်ထားတာကို ရေးထားတာတွေ့နိုင်ပါတယ်။

Recursive ဟာ စလေ့လာကစလူတွေ အနေနဲ့ အနည်းငယ် ရှုပ်ထွေးစေနိုင်တယ်။ သို့ပေမယ့် တကယ်လက်တွေ့ လုပ်ငန်းခွင်မှာ အသုံးပြုသည့် အခါမှာ code တွေကို နည်းသွားပြီး ကျစ်လစ် ရှင်းလင်းအောင် အထောက်အကူပြုပေးတယ်။ သို့ပေမယ့် အခြားတစ်ယောက် အနေနဲ့ အချိန် အနည်းငယ်ပေးပြီး နားလည် သဘောပေါက်အောင်တော့ code တွေ ပြန်ဖတ်ဖို့ လိုပါလိမ့်မယ်။

အခန်း ၈ ॥ Data Structure

အခုဆိုရင် ကျွန်တော်တို့ Programming ကို တီးမိခေါက်မိ ဖြစ်အောင် ရေးတတ်နေပါပြီ။ Programming ရေးရာမှာ ပုံမှန် ရေးတတ်ရုံနဲ့ အလုပ်မဖြစ်ပါဘူး။ ကိုယ်ပိုင် စဉ်းစားပြီး ပြဿနာကို အဖြေရှာနိုင်ဖို့ လိုပါတယ်။ ဒီအခန်းမှာ ကျွန်တော်တို့တွေ အနေနဲ့ စဉ်းစားပြီးတော့ အဖြေရှာနိုင်အောင် Searching အမျိုးမျိုး, Sorting အမျိုးမျိုး တို့ကို လေ့လာရမှာပါ။

Searching

Search လုပ်တယ်ဆိုတာကတော့ ကိုယ်ရှာချင်သည့် value ရှိလား မရှိဘူးလား ဆိုတာကို သိဖို့ အတွက်ပါ။

```
print(15 in [3,5,2,4,1])
```

အဲဒီ code လေးဆိုရင်တော့ `False` ဆိုပြီး return ပြန်လာပါလိမ့်မယ်။ ဘာကြောင့်လဲဆိုတော့ 15 ဟာ array အခန်းထဲမှာ မရှိလို့ပါ။

```
print(4 in [3,5,2,4,1])
```

ဆိုရင်တော့ `True` ဆိုပြီး return ပြန်လိမ့်မယ်။ 4 က array အခန်းထဲမှာ ရှိလို့ပါ။

အခု ကျွန်တော်တို့ သုံးထားသည့် ပုံစံကတော့ python မှာပါသည့် array အခန်းထဲမှာ ရှိလား မရှိဘူးလားဆိုပြီး ရှာသည့် ပုံစံပါ။ အဲဒီအစား ကိုယ်ပိုင် search ကို program ရေးကြည့်ရအောင်။

Search အပိုင်းကို ဒီစာအုပ်မှာ ၂ မျိုး ဖော်ပြပေးပါမယ်။ Sequential Searching နဲ့ Binary Search ပါ။

Sequential Searching

Sequential Searching ဆိုတာကတော့ array အခန်းထဲမှာ ကိုယ်ရှာလိုသည့် value ရှိလား ဆိုပြီး တစ်ခန်းခြင်းဆီ စစ်ဆေးသည့် အပိုင်းပါ။ ပထမဆုံး အခန်းကနေပြီးတော့ နောက်ဆုံး အခန်းထိ စစ်သည့် သဘောပါ။ အကယ်၍ နောက်ဆုံး အခန်းမတိုင်ခင်မှာ တွေ့ခဲ့ရင် search လုပ်နေသည့် loop ထဲက ထွက်လိုက်ရုံပါပဲ။

```
k = [1,2,51,34,37,78]
s = 37
```

```
for i in k:
    if i == s :
        print("found")
        break
```

code လေးက အရမ်းကို ရှင်းပါတယ်။ array ကို loop ပတ်တယ်။ တွေ့ခဲ့ရင် တွေ့တယ်လို့ ပြောတယ်။

သို့ပေမယ့် ကျွန်တော်တို့တွေ ဘယ်အခန်း နံပတ်မှာ ရှာတွေ့သလဲ ဆိုတာကို သိဖို့ အတွက် code ကို အောက်ကလို ပြင်ရေးပါမယ်။

```
k = [1,2,51,34,37,78]
s = 37

for (index,value) in enumerate(k):
    if value == s :
        print("found at ",index)
        break
```

အဲဒီမှာ `enumerate(k)` ဆိုတာ ပါလာပါတယ်။ enumerate ဆိုတာကတော့ python ရဲ့ built in function တစ်ခုပါ။ သူက index နဲ့ value ကို return ပြန်လာပေးပါတယ်။

အခုဆိုရင်တော့ ကျွန်တော်တို့ array ထဲမှာ search အတွက် ရေးတတ်ပြီ။ သို့ပေမယ့် ပိုပြီး သပ်ရပ်သွားအောင် function ခွဲရေးပါမယ်။

```
def search(array,search_value):
    for (index,value) in enumerate(array):
        if value == search_value :
            return index
    return -1

res = search([1,2,51,34,37,78],371)
if res != -1 :
    print("Found at ",res)
else:
    print("Not found")
```

ရှာတွေ့ခဲ့ရင် အခန်း နံပတ် ပြန်လာပြီးတော့ ရှာမတွေ့ရင် -1 ပြန်လာတာကို တွေ့ပါလိမ့်မယ်။ function ဖြစ်သည့် အတွက်ကြောင့် return ပြန်လိုက်တာနဲ့ looping ထဲက ထွက်သွားတာကြောင့် break လုပ်ဖို့ မလိုအပ်ပါဘူး။

ဒီလောက်ဆိုရင်တော့ Sequential Searching ကို သဘောပေါက်လောက်ပြီ ထင်ပါတယ်။

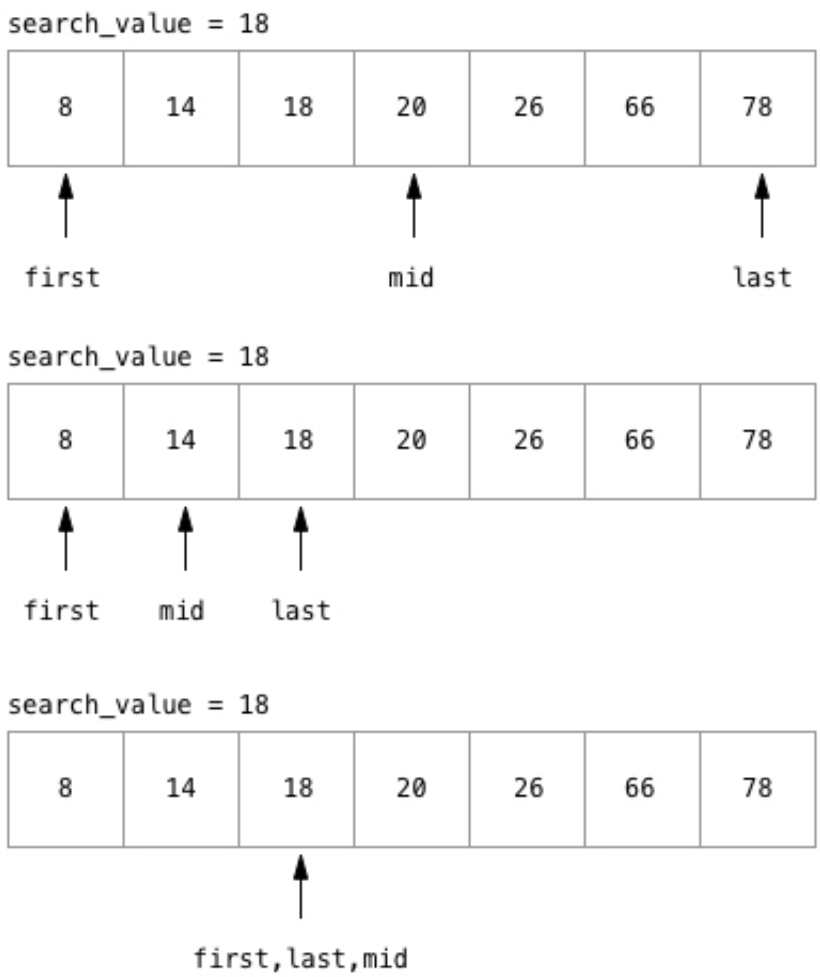
Binary Search

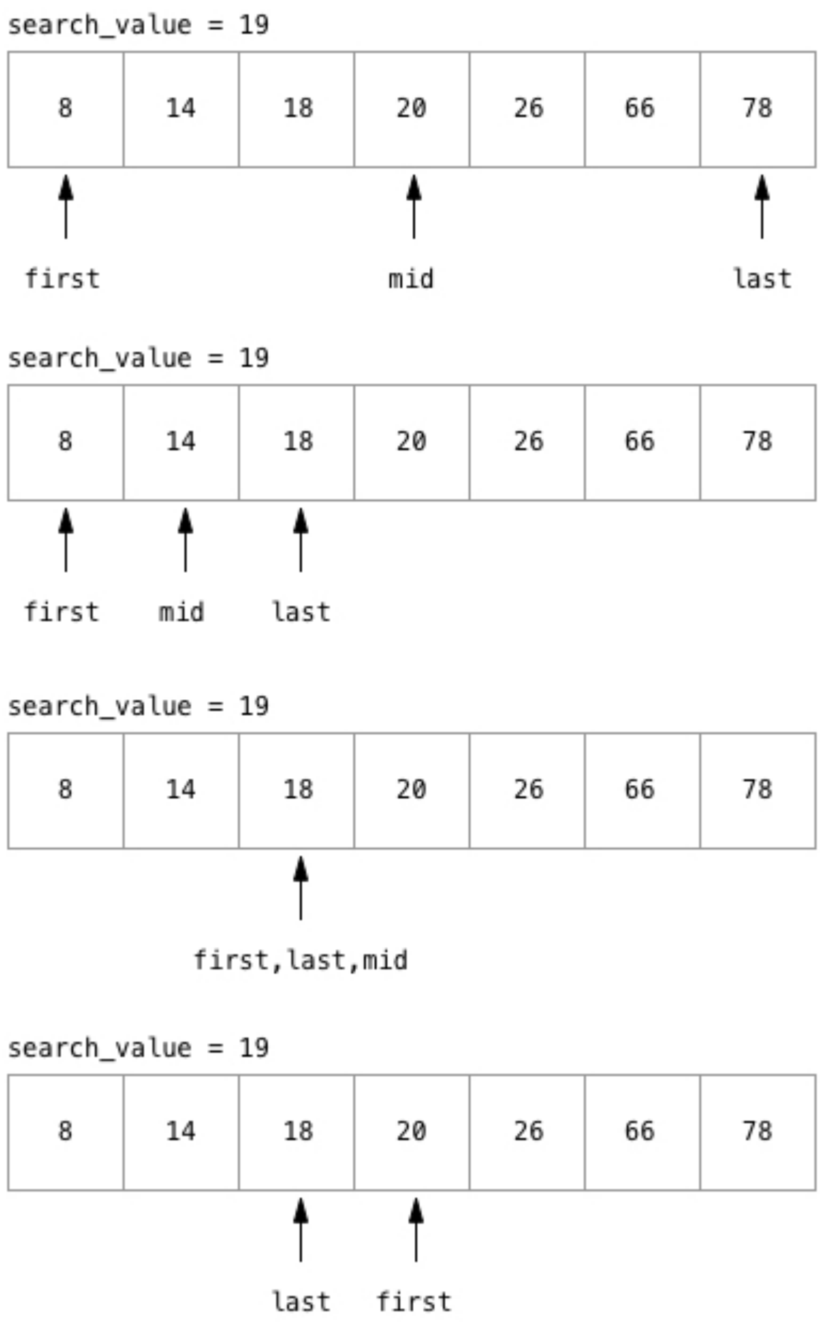
ကျွန်တော်တို့ Sequential Searching မှာတုန်းက array ဟာ sorting စီထားခြင်း မရှိပါဘူး။ သို့ပေမယ့် Binary Search ကို အသုံးပြုတော့မယ်ဆိုရင် array ဟာ မဖြစ်မနေ sorting စီထားမှသာ ဖြစ်ပါမယ်။

Binary Search ဟာ sorting စီထားသည့် array ကို အလယ်ကို ထောက်လိုက်ပါတယ်။ mid value က ရှာမယ့် value နဲ့ တူခဲ့ရင် ရှာတွေ့ပြီပေါ့။ အကယ်၍ မတွေ့ခဲ့ရင် mid value က ရှာမယ့် value နဲ့ ငယ်လား ကြီးလာစစ်တယ်။ အကယ်၍ ငယ်ခဲ့ရင် mid value ရှေ့က အခန်းက နောက်ဆုံး အခန်း ဖြစ်ပြီး ရှေ့ဆုံးက အခန်းက ပထမ အခန်းဖြစ်တယ်။ ပြီးရင် အလယ်ကို ပြန်ထောက်ပြီး ရှာပါတယ်။ အကယ်၍ ကြီးခဲ့ရင် ပထမဆုံး အခန်းက mid value ရဲ့ နောက်က အခန်းဖြစ်ပြီးတော့ နောက်ဆုံး အခန်း နဲ့ ပထမဆုံး အခန်းရဲ့ အလယ်ကို ပြန်ထောက်ပြီး ရှာပါတယ်။

သဘောတရားလေးက ရှင်းပါတယ်။ ဒါကြောင့် သူက sorting စီထားဖို့ လိုပါတယ်။ အခန်းတွေ အကုန်မသွားတော့ပဲ အလယ်အခန်းက ရှာဖွေ သွားတာပါ။

သေချာနားလည် သွားအောင် အောက်က ပုံလေးကို ကြည့်ကြည့်ပါ။





ပထမပုံကတော့ search လုပ်သည့် အခါမှာ ရှာတွေ့သည့် flow လေးပါ။

ဒုတိယပုံကတော့ ရှာမတွေ့သည့် flow ပါ။ အကယ်၍ first ကသာ last ထက် position ကြီးသွားမယ်ဆိုရင်တော့ ဆက်ပြီးတော့ ရှာတော့မှာ မဟုတ်ပါဘူး။

ပုံလေး ၂ ပုံကို နားလည်တယ်ဆိုရင်တော့ ကျွန်တော်တို့တွေ code ကို စပြီး ရေးနိုင်ပါပြီ။

```
k = [8,14,18,20,26,66,78]
s = 18
found = False
first = 0
last = len(k) - 1
while first <= last and not found:
    mid = (first + last)//2
    mid_value = k[mid]
```

```
        if mid_value == s:
            found = True
        else:
            if s < mid_value :
                last = mid - 1
            else:
                first = mid + 1

    print(found)
```

first က last ထက် ကြီးသည့် အခါ အလုပ်လုပ်မယ် ၊ ဒါမှမဟုတ် ရှာတွေ့ခဲ့ရင် loop ထဲက ထွက်မယ် ဆိုတာကြောင့်

```
    while first <= last and not found:
```

ဆိုပြီး ရေးထားတာပါ။ looping က firs က last ထက် ငယ်နေရင် ဒါမှမဟုတ် တူနေရင် အလုပ်လုပ်မယ်။ နောက်ပြီးတော့ found ကလည်း False ဖြစ်နေရမယ်။ not found ဆိုသည့် အခါမှာ ကတော့ not False ဆိုရင် True ဖြစ်သွားသည့် အတွက်ကြောင့် condition မှာ ၂ ခု လုံး True ဖြစ်နေသ၍ အလုပ်လုပ်မယ့် သဘောပါ။

```
    mid = (first + last)//2
```

အဲဒီ code မှာ // ဆိုတာက ဒသမ ကိန်း မယူဘူးလို့ ဆိုတာပါ။ 3/2 ဆိုရင် 1.5 ရပါတယ်။ 3//2 ဆိုရင်တော့ 1 ပဲ ရပါတယ်။

ကျွန်တော်တို့ code ကို ရှင်းသွားအောင် function ခွဲရေးပါမယ်။

```
    def binary_search(array,item):
        first = 0
        last = len(array) - 1
        while first <= last:
            mid = (first + last)//2
            mid_value = array[mid]
            if mid_value == item:
                return True
            else:
                if item < mid_value :
                    last = mid - 1
                else:
                    first = mid + 1
        return False

    print(binary_search([8,14,18,20,26,66,78],18))
    print(binary_search([8,14,18,20,26,66,78],19))
```

ဒီလိုဆိုရင် code လေးက အတော်လေးကို ရှင်းသွားပါပြီ။ ဘယ် အခန်းမှာ ရှာတွေ့လဲဆိုသည့် code အတွက်ကတော့ လေ့ကျင့်ခန်း အနေနဲ့ ကိုယ်တိုင် ရေးကြည့်ပါ။

ကျွန်တော် ထပ်ပြီးတော့ recursive နဲ့ ရေးကြည့်ရအောင်။

```
def binary_search(array,item):

    if len(array) == 0:
        return False

    mid = len(array)//2
    mid_value = array[mid]
    if mid_value == item:
        return True
    else:
        if item < mid_value :
            return binary_search(array[:mid],item)
        else:
            return binary_search(array[mid+1:],item)


print(binary_search([8,14,18,20,26,66,78],18))
print(binary_search([8,14,18,20,26,66,78],19))
```

first နှင့် last အစား array ရဲ့ size ကို တဝက်ဝက်ပြီးတော့ mid ကို ယူထားတာကို တွေ့နိုင်ပါတယ်။ array[:mid] နှင့် array[mid+1:] အကြောင်းကို တော့ ကျွန်တော် recursive အခန်းမှာ ရေးထားပြီးသား ဖြစ်သည့် အတွက် နားလည်မယ်လို့ ထင်ပါတယ်။ array ကို ထပ်ခါ ထပ်ခါ ပိုင်းပြီး ရှာနေပြီး နောက်ဆုံး array ကုန်သွားရင် သို့မဟုတ် ရှာတွေ့ခဲ့မှသာ recursive function ထဲက နေ ထွက်မှာကို တွေ့နိုင်ပါတယ်။

Sorting

ကျွန်တော်တို့ program တွေ ရေးသည့်အခါမှာ sorting ဟာလည်း အရေးပါပါတယ်။ Array ထဲမှာ ရှိသည့် တန်ဖိုးတွေကို ကိုယ်တိုင် sorting ပြန်ရေးနိုင်ဖို့ရှိပါတယ်။ python မှာ sorting အတွက် built-in function တွေ ပါထားသော်လည်း ကျွန်တော်တို့ အနေနဲ့ programming ကို လေ့လာနေသည့် အဆင့်ဖြစ်သည့် အတွက်ကြောင့် sorting နည်း အမျိုးမျိုးကို သုံးပြီးတော့ array ကို sorting လုပ်သွားပါမယ်။

Bubble Sort

Sorting ထဲမှာ အရိုးအရှင်းဆုံး နည်းလမ်းပါပဲ။ ကျွန်တောတို့မှာ

```
[9,1,8,6,7,4]
```

ဆိုတဲ့ array ရှိတယ်။ အဲဒါကို sorting စီ လိုက်ရင်

```
[1,4,6,7,8,9]
```

ဆိုပြီး ရမယ်။ ပုံမှန် ကျွန်တော်တို့ sorting ဘယ်လို စီသွားသလဲ ဆိုတာကို ကြည့်ရအောင်။

အရင်ဆုံး ပထမ ဆုံး အခန်းကို ကြည့်တယ်။ ဒုတိယ အခန်း နဲ့ စစ်မယ်။ 9 နှင့် 1 နှစ်ခု ကို ယှဉ်မယ်။ 9 က 1 ထက် ကြီးမလား စစ်မယ်။ မကြီးဘူးဆိုရင် ဒီ အတိုင်းထားမယ်။ ကြီးရင် နေရာလဲမယ်။ အဲလိုနဲ့ ရှေ့အခန်း နောက်အခန်း ကြီးသလား ဆိုပြီး အခန်း ကုန်သွားသည့် အထိ စစ်သွားမယ်။

ဒါဆိုရင် အောက်ကလို မျိုး တဆင့် ခြင်းစီ ဖြစ်သွားပါမယ်။

```
[9,1,8,6,7,4]
[1,9,8,6,7,4]
[1,8,9,6,7,4]
[1,8,6,9,7,4]
[1,8,6,7,9,4]
[1,8,6,7,4,9]
```

အခု နောက်ဆုံး အခန်းက အကြီးဆုံး ကို ရပါပြီ။ နောက် တဆင့် အစကနေ ပြန်စမယ်။ ဒါပေမယ့် နောက်ဆုံး အခန်းထိ စစ်ဖို့ မလိုတော့ဘူး။ ဘာဖြစ်လို့လည်း ဆိုတော့ နောက်ဆုံး အခန်း က အကြီး ဆုံး value ဖြစ်နေတာ သေချာသွားပါပြီ။

```
[1,8,6,7,4,9]
[1,8,6,7,4,9]
[1,6,8,7,4,9]
[1,6,7,8,4,9]
[1,6,7,4,8,9]
```

အခု ဆိုရင် နောက်ဆုံး အခန်း ၂ ခန်းက sorting ဖြစ်နေပြီ။ ဒီလို ပုံစံ နဲ့ တောက်လျှောက် စစ်ပြီးတော့ နေရာ ရွှေ့ သွားမယ်။ နောက်ဆုံး ၂ ခန်း မပါပဲ ထပ်ပြီး ရွှေ့ ရင် အောက်ကလို ရလာလိမ့်မယ်။

```
[1,6,7,4,8,9]
[1,6,7,4,8,9]
[1,6,7,4,8,9]
[1,6,4,7,8,9]
```

ကျွန်တော် နောက် တဆင့် ထပ်ရွှေ့မယ်။ ဘယ်အချိန်ထိ ကျွန်တော်တို့ တွေ ဒီလို ရွှေ့မှာလဲ ဆိုရင်တော့ နောက်ကနေ စပြီး ရှေ့ဆုံး အခန်း ရောက်အောင် ထိပဲ။

```
[1,6,4,7,8,9]
[1,6,4,7,8,9]
[1,4,6,7,8,9]
[1,4,6,7,8,9]
[1,4,6,7,8,9]
```

ကဲ အခု ဆိုရင်တော့ ကျွန်တော်တို့တွေ sorting စီပုံကို သိထားပြီ။ အဲဒီ အတွက် code ရေးဖို့ လုပ်ရအောင်။

```
def bubble_sort(array):
    for num in range(len(array) - 1,0,-1):
        for i in range(num):
            if array[i] > array[i+1]:
                temp = array[i]
                array[i] = array[i+1]
                array[i+1] = temp
    return array

alist = [1,6,4,7,8,9]
print(bubble_sort(alist))
```

code လေးကတော့ ရိုးရှင်းပါတယ်။ ထူးခြားတာကတော့ range(len(array) - 1,0,-1) ပါ။ အဲဒီ အဓိပ္ပာယ်ကတော့ len(array) - 1 ကနေ 1 အထိ loop ပတ်မယ်လို့ ဆိုလိုတာပါ။

အခြား language တွေမှာတော့

```
for (i=len(array) - 1, i > 0 ; i--) {

}
```

ဆိုပြီး ရေးကြပါတယ်။

python ဖြစ်သည့် အတွက်ကြောင့်

```
for num in range(len(array) - 1,0,-1):
```

ဆိုပြီး ရေးသားထားတာပါ။

Array ကို nested loop ပတ်ထားတာကို တွေ့နိုင်ပါလိမ့်မယ်။ ပထမ loop ကတော့ နောက်ဆုံး အခန်းကနေ စတယ် ပထမဆုံး အခန်းထိ။ ဒုတိယ loop ကတော့ ပထမ အခန်းကနေ စတယ်။ `range(num)` ထိ loop ပတ်ပါတယ်။ ဘာကြောင့် အဲလို loop ပတ်သလဲ ဆိုတာကိုတော့ ကျွန်တော်တို့ အထက်မှာ ရှင်းပြပြီးပါပြီ။ ပထမဆုံး အကြိမ်မှာ နောက်ဆုံး အခန်းက အကြီးဆုံး value ဝင်တယ်။ နောက်တစ်ကြိမ်ဆိုရင် နောက်ဆုံး အခန်း ထည့်တွက် ဖို့ မလိုတော့ဘူး။ ဒါကြောင့်ပါ။

ဒီမှာ ဘာပြဿနာရှိလဲ ဆိုတော့ sorting စီထားပြီးသားဆိုရင်ပေမယ့် ထပ်ပြီး အကုန်လုံး loop ပတ်နေတာကို တွေ့နိုင်ပါတယ်။ sortng စီထားပြီးမှန်း ဘယ်လို သိသလဲဆိုတော့ ဒုတိယ loop ထဲမှာ တစ်ခါမှ data အပြောင်းအလဲ မလုပ်ရရင်တော့ sorting စီထားပြီးမှန်း သိနိုင်ပါတယ်။

ဒါကြောင့် ကျွန်တော်တို့ အခုလို ပြင်ရေးပါမယ်။

```
def bubble_sort(array):
    exchange = True
    num = len(array) - 1
    while num > 0 and exchange:
        exchange = False
        for i in range(num):
            if array[i] > array[i+1]:
                exchange = True
                temp = array[i]
                array[i] = array[i+1]
                array[i+1] = temp
        num = num - 1
    return array

alist = [1,2,3,4,8,9]
print(bubble_sort(alist))
```

ကျွန်တော်တို့ exchange ဆိုသည့် variable လေးထည့်ပြီးတော့ loop ထပ်လုပ်မလုပ်ဆိုတာကို ထိန်းထားတာကို တွေ့နိုင်ပါတယ်။ တစ်ကြောင်းခြင်းစီကိုတော့ ကိုယ်တိုင် လိုက်ကြည့်ပြီးတော့ နားလည် နိုင်မယ်လို့ ထင်ပါတယ်။

Selection Sort

Selection Sort ကတော့ bubble sort နဲ့ ဆင်ပါတယ်။ တူကတော့ အပိုင်း ၂ ပိုင်းခွဲလိုက်တယ်။ sort လုပ်ပြီးသား အပိုင်းနဲ့ sort မလုပ်ရသေးသည့် အပိုင်း။ sort လုပ်ပြီးသား အပိုင်းကို ထပ်ပြီးတော့ sort မလုပ်တော့ဘူး။ ပထမဆုံး အနေနဲ့ အငယ် ဆုံး တန်ဖိုးကို ရှာပြီးတော့ ပထမဆုံး အခန်းထဲမှာ ထည့်လိုက်တယ်။ နောက်တစ်ကြိမ် ဒုတိယ အခန်းကနေ စပြီးရှာမယ်။ အငယ်ဆုံး တန်ဖိုး ကို ဒုတိယ အခန်းထဲမှာ ထည့်မယ်။

```
def selection_sort(array):
    for num in range(len(array)):
        pos_of_min = num
        for loc in range(num,len(array)) :

            if array[loc] < array[pos_of_min]:
                pos_of_min = loc

        temp = array[num]
        array[num] = array[pos_of_min]
        array[pos_of_min] = temp

    return array

alist = [1,20,31,444,8,9]
print(selection_sort(alist))
```

code လေးကို အရမ်းကို ရှင်းပါတယ်။ ပထမဆုံး အခန်းကို အငယ်ဆုံး တန်ဖိုးလို့ သဘောထား တယ်။ ပြီးတော့ သူ့ထက်ငယ်သည့် array ရှိလားဆိုပြီး ရှာတယ်။ တွေ့ခဲ့ရင် အငယ်ဆုံး တန်ဖိုးရဲ့ location ကို လဲလိုက်တယ်။ အကုန်လုံးပြီးသွားခဲ့ရင် array အခန်းကို နေရာလဲလိုက်ပါတယ်။

Array ပုံစံက အောက်ကလို ရွှေ့သွားပါမယ်။

```
[1, 20, 31, 444, 8, 9]
0: [1, 20, 31, 444, 8, 9]
1: [1, 8, 31, 444, 20, 9]
2: [1, 8, 9, 444, 20, 31]
3: [1, 8, 9, 20, 444, 31]
4: [1, 8, 9, 20, 31, 444]
5: [1, 8, 9, 20, 31, 444]
```

Insertion Sort

နောက်ထပ် sort တနည်းကတော့ Insertion sort လို့ ခေါ်ပါတယ်။ သူကတော့ selection sort နဲ့ ဆင်တယ်။ sorted အပိုင်း နဲ့ unsorted အပိုင်း ၂ ပိုင်း ခွဲပြီးတော့ sort လုပ်ပါတယ်။

```
[5,6,4,7,12,9]
```

ဆိုရင် ပထမဆုံး အခန်းကို sorted လုပ်ထားတယ်ဆိုပြီး ယူလိုက်တယ်။ ဒါကြောင့် စပြီဆိုရင် 5 နှင့် 6 နဲ့ကို ယှဉ်တယ်။ ငယ်သလား။ မငယ်ဘူး။ ဒါဆိုရင် sorted ပိုင်းက 5,6 ဖြစ်သွားပြီ။

```
[5,6,|4,7,12,9]
```

အခု နောက်တစ်ကြိမ်မှာဆိုရင်တော့ ဒုတိယ အခန်းက စမယ်။ 6 နဲ့ 4 ယှဉ်တယ်။ 4 က 6 ထက် ငယ်တယ်။ ဒီတော့ 6 နှင့် 4 နေရာလဲတယ်။

ပြီးရင် 5 နဲ့ 4 ထပ်ယှဉ်တယ်။ 4 က 5 ထပ်ငယ်တယ်။ ဒီတော့ ထပ်နေရာလဲတယ်။

```
[4,5,6,|7,12,9]
```

sorted ပိုင်းက 4,5,6 ဖြစ်သွားပြီးတော့ unsorted ကတော့ 7,12,9 ပေါ့။

6 နှင့် 7 ယှဉ်တယ်။ နောက်က အခန်းက မကြီးဘူး။ ဒီတော့ ဒီအတိုင်းထားတယ်။

```
[4,5,6,7,|12,9]
```

sorted က 4,5,6,7 နှင့် unsorted ကတော့ 12,9 ပါ။

7 နှင့် 12 ယှဉ်တယ်။ 7 က ငယ်နေတယ်။ ဒီတော့ နေရာ မရွှေ့ဘူး။

```
[4,5,6,7,12,|9]
```

sorted က 4,5,6,7,12 နှင့် unsorted ကတော့ 9 ပါ။

12 နှင့် 9 ယှဉ်တယ်။ 9 က ငယ်နေတယ်။ ဒီတော့ ရွှေ့တယ်။ 7 နှင့် 9 ယှဉ်တယ်။ 7 က ငယ်တော့ ထပ်မရွှေ့တော့ဘူး။ ဒီတော့ အကုန်လုံး ပြီးသွားပြီးတော့ result က အောက်ကလို ထွက်လာပါပြီ။

```
[4,5,6,7,9,12]
```

ဒီ Insertion sort မှာ အဓိက ကတော့ ရှေ့အလုံး နဲ့ နောက် အလုံးကို ယှဉ်တယ်။ ငယ်ရင် အဲဒီ အလုံးကို ကိုင်ထားပြီးတော့ ရှေ့က အလုံး နဲ့ ငယ်လား စစ်နေတာပါပဲ။

အခုဆိုရင်တော့ Insertion Sort အကြောင်းကို သဘောပေါက်လောက်ပါပြီ။ ဒီတော့ code ရေး ကြည့်ရအောင်။

ဒီ code က နည်းနည်း ရှုပ်သည့် အတွက် ကျွန်တော်တို့ Pseudo code လေး ရေးကြည့်ရအောင်။

```
for i = 1 to n-1
element = array[i]
j = i
while (j > 0 and array[j-1] > element) {
    array[j] = array[j-1]
    j = j -1
}
array[j] = element
```

ဒီ code မှာ

```
for i = 1 to n-1
```

ပထမဆုံး အခန်းက sorted လုပ်ထားပြီးလို့ သဘောထားသည့် အတွက် ကျွန်တော်တို့တွေ အခန်း ကို 1 ကနေ စပြီး loop ပတ်ထားပါတယ်။

```
element = array[i]
j = i
while (j > 0 and array[j-1] > element) {
```

အခု loop ထဲမှာ ရောက်နေသည့် position ပေါ်မှာ မူတည်ပြီးတော့ value ယူလိုက်တယ်။ နောက်ထပ် loop တစ်ခု ကို i အစား ကျွန်တော်တို့တွေ variable j ထဲမှာ ထည့်ထားလိုက်တယ်။ ဘာလို့လည်းဆိုတော့ j က နောက်က အခန်းနဲ့ စစ်ပြီးတော့ လျှော့လျှော့သွားဖို့လိုတယ်။ ဘယ် အထိ လျှော့ရမှာလဲ ဆိုတော့ နောက်က အခန်းက အခု အခန်းထက် ကြီးနေသ၍။ နောက်ပြီး တော့ j က 0 ထက်ကြီးနေသ၍ပေါ့။ 0 ဖြစ်သွားရင်တော့ ထပ်လျှော့လို့ မရတော့ဘူး။ ရှေ့ဆုံး အခန်း ရောက်နေပြီ။

```
array[j] = array[j-1]
j = j - 1
```

ဒါကတော့ အခန်းရွှေ့သည့် သဘောပါ။ နောက် အခန်းက ကြီးနေရင် နောက်အခန်းကို အခု အခန်းထဲ ထည့်။ ပြီးရင် j ကို တခန်း ထပ်လျော့။ ပြီးရင် သူ့ရဲ့ နောက်က value နဲ့ လက်ရှိ element နဲ့ ကြီးလား ထပ်စစ်တယ်။ ကြီးတယ်ဆိုရင် ထပ်ရွှေ့။

```
array[j] = element
```

loop ပြီးသွားရင်တော့ element ကို နောက်ဆုံး j ရှိသည့် အခန်းမှာ ထည့်လိုက်ရုံပါပဲ။

code ကို နားလည်ပြီဆိုရင် python နဲ့ ရေးကြည့်ရအောင်။

```
def insertionsort(array):
    for i in range(1,len(array)):
        element = array[i]
        j = i
        while j > 0 and array[j-1] > element :
            array[j] = array[j-1]
            j = j - 1

        array[j] = element
    return array

print(insertionsort([5,6,4,7,12,9]))
```

အခုဆိုရင်တော့ ကျွန်တော်တို့ insertion sort အကြောင်းကို သိလောက်ပါပြီ။

Shell Sort

ကျွန်တော်တို့ insertion sort မှာ ဂဏန်းတွေကို အစ ကနေ စတယ်။ ပြီးတော့ နောက်တစ်ခု နဲ့ တိုက်သွားတယ်။ အကယ်၍ အငယ်ဆုံးက နောက်ဆုံးမှာ ရှိတယ် ဆိုရင် နောက်ဆုံး ထိ ရောက်ပြီးမှ နောက်ဆုံးမှာ ရှိသည့် အငယ်ဆုံး ဂဏန်းကလည်း ရှေ့ ဂဏန်းထက် ငယ်ထား ဆိုပြီး ရှေ့ဆုံးထိ ရောက်အောင် ပြန်သွားရတယ်။

အခု shell sort ကတော့ insertion sort နဲ့ တူပါတယ်။ သို့ပေမယ့် gap လေးတွေ ထည့်ထားပါတယ်။ gap ပေါ်မှာ မူတည်ပြီးတော့ အခန်းကို ယှဉ်ပြီး ရွှေ့ပါတယ်။

code အပိုင်းကို မသွားခင်မှာ အရင်ဆုံး shell sort ဆိုတာ ဘာလဲ ကြည့်ရအောင်။

ကျွန်တော်တို့ မှာ အောက်ဖော်ပြပါ အတိုင်း array အခန်း ရှိပါတယ်။

```
[5,6,4,7,12,9,1,8,32,49]
```

စုစု ပေါင်း အခန်း ၁၀ ခု ပေါ့။

gap ကို အရင် ရှာဖို့ လိုတယ်။

```
gap = len(array)//2
```

// ဖြစ်သည့် အတွက်ကြောင့် 3.5 ဖြစ်ခဲ့ရင် 3 ကို ပဲ ယူမယ်လို့ ဆိုထား ပါတယ်။

အခု က အခန်း ၁၀ ခန်း ရှိသည့် အတွက်ကြောင့်

```
gap = 5
```

shell sort မှာ gap ကို gap နဲ့ တဝက်ပိုင်းပြီးတော့ နောက်ဆုံး 1 ထိ ရောက်အောင် သွားပါတယ်။

ဒီတော့

```
gap = 5
gap = 5//2 = 2
gap = 2//2 = 1
```

အခု ပထမဆုံး gap = 5 ကနေ စရအောင်။ အခန်း 1 နဲ့ gap ကို စပြီး ယှဉ်ပါတယ်။

```
gap = 5
[5<-,6,4,7,12,9<-,1,8,32,49]
```

5 နှင့် 9 ကို ယှဉ်တယ်။ 9 က မငယ်သည့် အတွက် ဒီ အတိုင်းထားမယ်။ နောက် တခန်းကို ထပ် ရွှေ့မယ်။

```
[5,6<-,4,7,12,9,1<-,8,32,49]
```

6 နှင့် 1 ကို ယှဉ်တယ်။ 1 က ငယ်သည့် အတွက် လဲ မယ်။ insertion sort လိုပဲ ငယ်တာကို ထပ်ပြီး တော့ နောက်မှာ သူ့ထပ် ငယ်တာ ရှိလား ဆိုပြီး စစ်တယ်။ insertion sort နဲ့ ကွာတာက နောက်က တစ်ခန်းကို မဟုတ်ပဲ gap value အကွာနဲ့ စစ်တယ်။ အခု လက်ရှိ အခန်းက 1 - gap ဆိုတော့ 1-5 = -4 ပေါ့။ 0 ထက် ငယ်နေသည့် အတွက် ထပ်ပြီး မစစ်တော့ဘူး။ အကယ်၍ 0 သို့မဟုတ် 0 ထက် ကြီးနေရင် အဲဒီ အခန်းနဲ့ 1 ကို ထပ်ပြီး စစ်ဖို့ လိုပါတယ်။ အဲဒီတော့

```
[5,1,4,7,12,9,6,8,32,49]
```

အခု နောက်ထပ် တခန်းကို ထပ်တိုးမယ်။

```
[5,1,4<-,7,12,9,6,8<-,32,49]
```

4 နှင့် 8 ကို ယှဉ်တယ်။ 8 က ကြီးနေတော့ ဒီ အတိုင်းထား။ နောက် တစ်ခန်းကို ထပ်တိုး။

```
[5,1,4,7<-,12,9,6,8,32<-,49]
```

7 နဲ့ 32 လည်း လဲ ဖို့ မလိုဘူး။

```
[5,1,4,7,12<-,9,6,8,32,49<-]
```

12 နဲ့ 49 လည်း လဲ ဖို့ မလိုဘူး။ အခု ရှေ့ဆုံး အခန်းကနေ ရွှေ့တာ gap နေရာ ရောက်ပါပြီ။ ဒါ ကြောင့် ထပ် မရွှေ့တော့ပါဘူး။ အခု gap တန်ဖိုး ကို ပြန်ပြောင်းပါမယ်။

```
gap = gap//2 == 5//2 = 2
```

အခု gap တန်ဖိုးက 2 ဖြစ်သည့် အတွက် အခန်း 0 နဲ့ အခန်း 2 ကို ယှဉ်တာကနေ စမယ်။

```
[5<-,1,4<-,7,12,9,6,8,32,49]
```

5 နှင့် 4 ကို ယှဉ်တယ်။ ငယ်သည့် အတွက်နေရာလဲတယ်။ 0 - 2 က 0 ထက် ငယ်သွားသည့် အတွက် ဆက်ပြီး မယှဉ်တော့ဘူး။ နောက် အခန်း ထပ်သွားတယ်။

```
[4,1<-,5,7<-,12,9,6,8,32,49]
```

1 နှင့် 7 က လဲ ဖို့ မလိုဘူး။ နောက် အခန်း ထပ်သွားတယ်။

```
[4,1,5<-,7,12<-,9,6,8,32,49]
```

5 နဲ့ 12 လဲဖို့ မလိုဘူး။

```
[4,1,5,7<-,12,9<-,6,8,32,49]
```

7 နဲ့ 9 လဲ ဖို့ မလိုဘူး။

```
[4,1,5,7,12<-,9,6<-,8,32,49]
```

12 နဲ့ 6 မှာ 12 က ကြီးတော့ လဲမယ်။ ပြီးတော့ လက်ရှိ ရောက်နေသည့် အခန်းနံပတ် 4 - 2 (gap) = 2 ဆိုတော့ အခန်း 2 က value နဲ့ ထပ်ယှဉ်တယ်။ 4 နဲ့ 6 ဆိုတော့ လဲ စရာမလိုဘူး။

```
[4,1,5,7,6,9<-,12,8<-,32,49]
```

9 နှင့် 8 ကို ယှဉ်တယ်။ အခန်းလဲ တယ်။ ပြီးတော့ 7 နဲ့ 8 ကို ထပ်ယှဉ်တယ်။ 7 က ငယ်နေတော့ မလဲဘူး။

```
[4,1,5,7,6,8,12<-,9,32<-,49]
```

12 နှင့် 32 လဲ ဖို့ မလိုဘူး။

```
[4,1,5,7,6,8,12,9<-,32,49<-]
```

9 နှင့် 49 လဲဖို့ မလိုဘူး။

အခု gap က ရွှေ့ဖို့ အခန်း မရှိသည့် အတွက်ကြောင့် gap ကို ထပ်ပြီးတော့ 2 နဲ့ စားမယ်။

```
gap = gap//2 = 2//2 = 1
```

အခု gap က တစ်ခုပဲ ဖြစ်သွားသည့် အတွက် ကြောင့် ၁ ခန်း စီပဲ လဲ တော့မယ်။

```
[4<-,1<-,5,7,6,8,12,9,32,49]
```

4 နဲ့ 1 လဲမယ်။

```
[1,4<-,5<-,7,6,8,12,9,32,49]
```

4 နှင့် 7 ကို မလဲဘူး။

```
[1,4,5<-,7<-,6,8,12,9,32,49]
```

5 နှင့် 7 ကို မလဲဘူး။

```
[1,4,5,7<-,6<-,8,12,9,32,49]
```

7 နဲ့ 6 လဲတယ်။ ပြီးတော့ 5 နဲ့ 6 ထပ် ယှဉ်တယ်။ မငယ်တော့ မလဲတော့ဘူး။

```
[1,4,5,6,7<-,8<-,12,9,32,49]
```

7 နှင့် 8 ယှဉ်တယ်။ 7 က ငယ်တော့ မလဲဘူး။

```
[1,4,5,6,7,8<-,12<-,9,32,49]
```

8 နဲ့ 12 မလဲဘူး။

```
[1,4,5,6,7,8,12<-,9<-,32,49]
```

12 နဲ့ 9 လဲမယ်။ ပြီးတော့ 8 နဲ့ 9 ယှဉ်တယ်။ 8 က ငယ်တော့ မလဲဘူး။

```
[1,4,5,6,7,8,9,12<-,32<-,49]
```

12 နဲ့ 32 ယှဉ်တယ်။ မလဲဘူး။

```
[1,4,5,6,7,8,9,12,32<-,49<-]
```

32 နှင့် 49 ယှဉ်တယ်။ မလဲဘူး။

gap က ဆက်ပြီးသွားလို့ မရတော့ဘူး။ 2 နဲ့ ပြန်စားတော့ 0 ရတော့ loop မလုပ်တော့ပဲ လက်ရှိ ရှိပြီးသား array ကို sorting စီပြီးသားလို့ သတ်မှတ်လိုက်ပါပြီ။

အခု

```
[1,4,5,6,7,8,9,12,32,49]
```

ဆိုပြီး sorted array ထွက်လာပါပြီ။

အခု ကျွန်တော်တို့ python code လေးကို ကြည့်ရအောင်။

```
def shellSort(array):

    gap = len(array) // 2
    # loop over the gaps
    while gap > 0:
        # insertion sort
        for i in range(gap, len(array)):
```

```
            val = array[i]
            j = i
            while j >= gap and array[j - gap] > val:
                array[j] = array[j - gap]
                j -= gap
            array[j] = val
        gap //= 2
    return array

print(shellSort([5,6,4,7,12,9,1,8,32,49]))
```

```
code ကတော့ insertion sort နဲ့ တူတူပါပဲ။ ကွာခြားချက်ကတော့
```

```
while gap > 0:
```

ဆိုပြီး gap ပေါ်မှာ မူတည်ပြီး loop ပတ်ထားပါတယ်။ insertion sort တုန်းကတော့ - 1 ကို သုံးထားပြီးတော့ အခု shell sort မှာတော့ - gap ကို အသုံးပြုထားပါတယ်။

code တစ်ဆင့်ခြင်း ကိုတော့ ကိုယ်တိုင် ပြန်ပြီးတော့ trace လိုက်ကြည့်ပါ။

Merge Sort

အရင်ဆုံး merge sort အကြောင်းကို မပြောခင်မှာ sorted array ၂ ခုကို merge နဲ့ sort တစ်ခါတည်း လုပ်သည့် အကြောင်း ရှင်းပြပါမယ်။

```
sorted_array1 = [4,5,9,18]
sorted_array2 = [8,10,11,15]
```

ဒီလို sorted array ၂ ခု ရှိပါတယ်။ merge လုပ်ဖို့ အတွက် array ၂ ခု ရဲ့ ရှေ့ဆုံး ၂ ခန်းကို ယှဉ်မယ်။ ငယ်သည့် အခန်းကို ယူပြီး sorted ထဲမှာ ထည့်မယ်။

အခု ရှေ့ဆုံး အခန်း 4 နဲ့ 8 မှာ ဆိုရင် 4 ကို ယူမယ်။ ပြီးရင် 5 နဲ့ 8 မှာ ဆိုရင် 5 ကို ယူမယ်။ array အခန်း ၂ ခု လုံးကို ကုန်သွားအောင် ထိ စီသွားပါမယ်။

```
sorted_array1 = [5,9,18]
sorted_array2 = [8,10,11,15]
sorted = [4]
sorted_array1 = [9,18]
sorted_array2 = [8,10,11,15]
sorted = [4,5]
sorted_array1 = [9,18]
sorted_array2 = [10,11,15]
sorted = [4,5,8]
sorted_array1 = [18]
sorted_array2 = [10,11,15]
sorted = [4,5,8,9]
sorted_array1 = [18]
sorted_array2 = [11,15]
sorted = [4,5,8,9,10]
sorted_array1 = [18]
sorted_array2 = [15]
sorted = [4,5,8,9,10,11]
```

```
sorted_array1 = [18]
sorted_array2 = []
sorted = [4,5,8,9,10,11,15]
sorted_array1 = []
sorted_array2 = []
sorted = [4,5,8,9,10,11,15,188]
```

အခု ဆိုရင် merge လုပ်ပြီး sort လုပ်တာကို နားလည် လောက်ပါပြီ။ အခု merge sort အကြောင်း ထပ်ရှင်းပါမယ်။

```
array = [5,4,18,9,8,10,15,11]
```

ဒီလို array ကို sort လုပ်ဖို့အတွက် ထက်ဝက် ပိုင်းသွားပါမယ်။

```
[5,4,18,9]
[8,10,15,11]
[5,4]
[18,9]

[8,10]
[15,11]
[5]
[4]

[18]
[9]

[8]
[10]

[15]
[11]
```

အခု လို အပိုင်းလေးတွေ ပိုင်းပြီးတော့ ၁ ခန်းထဲ ကျန်သည့် အထိ ပိုင်းသွားပါတယ်။ ပြီးတော့ merge လုပ်သည့် အခါမှာ တစ်ခါတည်း sort လုပ်ပါမယ်။

```
[4,5]

[9,18]

[8,10]

[11,15]
[4,5,9,18]

[8,10,11,15]
[4,5,8,9,10,11,15,18]
```

ဒါဆိုရင်တော့ merge sort အကြောင်း နားလည်သွားပြီ။

ကျွန်တော်တို့ ဒီ code လေးကို recursive သုံးပြီး ရေးသားပါမယ်။ အဓိကတော့ array ၂ ခုကို ပြန်ပြီး merge ရသည့် အပိုင်းပါ။

```
def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0
    while left_idx < len(left) and right_idx < len(right):

        if left[left_idx] <= right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    if left_idx < len(left):
        result.extend(left[left_idx:])
    if right_idx < len(right):
        result.extend(right[right_idx:])
    return result

def mergesort(w):

    if len(w)<2:
        return w
    else:
        mid=len(w)//2
        return merge(mergesort(w[:mid]), mergesort(w[mid:]))

print(mergesort([4,5,1,3,9,7,2,10]))
```

အခြား code တွေကိုတော့ နားလည် ပါလိမ့်မယ်။ ထူးခြားတာကတော့ အောက်က code လေးပါ။

```
result.extend(left[left_idx:])
```

extend ကတော့ လက်ရှိ array ထဲကို နောက်ထပ် array က လာပြီးတော့ ထည့်သည့် သဘောပါ။

```
array_1 = [12,34,44]
array_2 = [6,8]
array_1.extend(array_2)
print(array_1)
```

အောက်က code လေးကို အနည်းငယ် ထပ်ရှင်းပြပါအုံးမယ်။

```
if left_idx < len(left):
    result.extend(left[left_idx:])
if right_idx < len(right):
    result.extend(right[right_idx:])
```

ကျွန်တော်တို့ array ကို merge လုပ်သည့် အခါမှာ အငယ်ဆုံးတွေကို အရင် merge လုပ်သွားသည့် အတွက် အချို့ အခန်းတွေ အကုန် မပြီးသွားဘူးပေါ့။

ဥပမာ

```
left = [2,4,5,8] #left_idx =  4
right = [6,9,10] #right_idx = 1
result = [2,4,5,6,8] #9,10 not merge yet
```

အခု code မှာ ဆိုရင် left က ကုန်သွားပြီ။ right က [9,10] ကျန်သေးတယ်။ အဲဒီတော့ extend ကို အသုံးပြုပြီး result ထဲကို merge လုပ်လိုက်တာပါ။

အခု ဆိုရင်တော့ ကျွန်တော်တို့ merge sort ကို ရေးလို့ ရသွားပါပြီ။ code ကို နားလည်အောင် ကိုယ်တိုင် တဆင့်ခြင်းစီ trace လိုက်ဖို့ လိုပါတယ်။

Quick Sort

Quick sort ဆိုတာကတော့ pivot နဲ့ ယှဉ်ပြီးတော့ အပိုင်း ၂ ပိုင်း ခွဲပါတယ်။ pivot ကတော့ ပုံမှန် အားဖြင့် array အခန်းထဲက အခန်းတစ်ခုကို random ချပြီးတော့ အဲဒီ အခန်းထဲက value ကို ယူပါတယ်။ အခု ကျွန်တော်တို့ random မချပဲနဲ့ ပထမဆုံး အခန်းက value ကို ယူပါမယ်။

```
array = [4,7,8,1,6,3,2]
```

ဆိုပြီး array ရှိပါတယ်။

ကျွန်တော်တို့ pivot အနေနဲ့ ရှေ့ဆုံး အခန်းကိုပဲ ယူမယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = []
more = []
pivotList = []
```

less ဆိုတာကတော့ pivot ထက် ငယ်သည့် value တွေ အတွက်ပါ။ more ကတော့ pivot ထက် ကြီးသည့် value အတွက်ပါ။ pivotList ကတော့ pivot နဲ့ တူသည့် value တွေ အတွက်ပါ။

ပထမဆုံး 4 ကို array ထဲမှာ ကြည့်တယ်။ တူနေရင် pivotList ထဲကို ထည့်မယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = []
more = []
pivotList = [4]
```

ပြီးရင် နောက် တစ်ခန်းသွားမယ်။ 7 ကို pivot နဲ့ စစ်တယ်။ ကြီးနေသည့်အတွက်ကြောင့် more ထဲကို ထည့်မယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = []
more = [7]
pivotList = [4]
```

နောက်တခန်း 8 ကလည်း ကြီးနေသည့် အတွက်ကြောင့် more ထဲကို ထည့်မယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = []
more = [7,8]
pivotList = [4]
```

နောက်တစ်ခန်း ဆက်ပြီးသွားတော့ 1 ထဲမှာဖြစ်တယ်။ 4 နဲ့ ယှဉ်တော့ ငယ်တယ်။ ဒါကြောင့် less ထဲမှာ ထည့်တယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = [1]
more = [7,8]
pivotList = [4]
```

နောက်တစ်ခန်းမှာတော့ 6 ဖြစ်တယ်။ 4 နဲ့ ယှဉ်တယ်။ ကြီးသည့် အတွက် more ထဲမှာ ထည့်တယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = [1]
more = [7,8,6]
pivotList = [4]
```

နောက်တစ်ခန်းတိုးပြီးတော့ စစ်တော့ 3 က pivot ထက် ငယ်တော့ less ထဲမှာ ထည့်တယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = [1,3]
more = [7,8,6]
pivotList = [4]
```

နောက်တစ်ခန်း 2 ကလည်း pivot ထက်ငယ်တော့ less မှာ သိမ်းတယ်။

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = [1,3,2]
more = [7,8,6]
pivotList = [4]
```

အခု ဆိုရင် ကျွန်တော်တို့ array ၃ ခု ရပြီ။ pivot နဲ့ တူတာ။ ငယ်တာ။ ကြီးတာ ဆိုပြီးတော့ ရပါတယ်။ ငယ်သည့် array ကိုလည်း recursive လုပ်ပြီးတော့ ထပ်စီ ဖို့လိုတယ်။ pivot value 1 ကို ထားပြီးတော့ [1,3,2] ကို ခွဲပြီး စီဖို့လိုတယ်။ နောက်ဆုံး array အခန်း တစ်ခုပဲ ကျန်သည့် အထိ recusrive စစ်ဖို့ လိုပါတယ်။

အဲလို စစ်လိုက်ရင် နောက်ဆုံး

```
array = [4,7,8,1,6,3,2]
pivot = 4
less = [1,2,3]
more = [6,7,8]
pivotList = [4]
```

ဆိုပြီး ဖြစ်သွားမယ်။ အဲဒီအခါ less+pivotList+more ဆိုတဲ့ array ၃ ခု ပေါင်းပြီး တော့ sorting ရလဒ်ကို ထုတ်နိုင်ပါပြီ။

```
def quickSort(arr):

    less = []
    pivotList = []
    more = []
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        for i in arr:
            if i < pivot:
                less.append(i)
            elif i > pivot:
                more.append(i)
            else:
                pivotList.append(i)
        less = quickSort(less)
        more = quickSort(more)

        return less + pivotList + more

qs = [4,7,8,1,6,3,2]
print(quickSort(qs))
```

ဒါကတော့ ကျွန်တော် အလွယ်သဘော ရှင်းလိုက်တာပါ။ ဒီ ပုံစံ အတိုင်း ဆိုရင်တော့ memory နေရာ အများကြီး ယူပါတယ်။

သမာရိုးက quick sort ပုံစံကတော့

```
array = [4,7,8,1,6,3,2]
```

ဆိုပြီး ရှိမယ်။ ကျွန်တော်တို့ pivot ယူရပါမယ်။ ပြီးတော့ left , right ဆိုပြီး index ထောက်ဖို့ ရှိပါမယ်။ left ကတော့ အစကနေ စပြီးတော့ right ကတော့ နောက်ဆုံး အခန်းကတော့ စပါမယ်။

```
pivot = 6
left = 0
right = 6
```

ပြီးရင် array[left] က pivot ထက် ငယ်သလား စစ်မယ်။ ငယ်ရင် နောက် တစ်ခန်းတိုးပေါ့။ ကြီးတာကို မတွေ့မခြင်း ရှေ့တိုးသွားပါမယ်။ right ကတော့ pivot ထက်ကြီးသလား ဆိုပြီး စစ်ပါမယ်။ ငယ်တာ မတွေ့မခြင်း ရှေ့ကို တိုးလာပါမယ်။ quick sort ရဲ့ အဓိက အပိုင်းကတော့ ဘယ်ဘက်မှာ pivot ထက် ငယ်တာ ဖြစ်ပြီးတော့ ညာဘက်မှာတော့ pivot ထက်ကြီးသည့် value တွေ ဖြစ်ရပါမယ်။

```
pivot = 6
left = 1 # 7 > 6
right = 6
```

အခု left အခန်းနဲ့ right အခန်းကို လဲပါမယ်။ ပြီးရင် left ကို တစ်ခန်းတိုးပြီးတော့ right ကိုတော့ တစ်ခန်း ထပ်လျော့ပါမယ်။

```
array = [4,2,8,1,6,3,7]
pivot = 6
left = 2
right = 5
```

ပြီးရင် အစကနေ ပြန်စစ်မယ်။ ဘယ်အချိန်ထိလဲ ဆိုတော့ left က right ကို ကျော်သွားသည့် အထိပါ။

```
array = [4,2,8,1,6,3,7]
pivot = 6
left = 2 # 8 > 6
right = 5 # 3 < 6
```

အခု ရလာသည့် value ကို ထပ်စစ်တယ်။ ပြီးတော့ လဲတယ်။ left ကို တစ်ခန်းတိုး။ right ကို တစ်ခန်းလျော့။

```
array = [4,2,3,1,6,8,7]
pivot = 6
left = 3
right = 4
```

အစကနေ ပြန်စစ်မယ်။

```
array = [4,2,3,1,6,8,7]
pivot = 6
left = 4
right = 4
```

အခု left နဲ့ right တူသွားပြီ။ နေရာလဲပေမယ့် အခုနေရာက နေရာပဲ။ left ကို တစ်ခန်းတိုး။ right ကို တစ်ခန်းထပ်လျော့။

```
array = [4,2,3,1,6,8,7]
pivot = 6
left = 5
right = 3
```

အခု left က right ထက်ကြီးသွားပြီ။ ဒါဆိုရင် pivot ရဲ့ ဘယ်ဘက်က pivot value ထက် ငယ်တာတွေ ဖြစ်ပြီးတော့ ညာဘက်ကတော့ သူ့ထက်ကြီးတာတွေပါ။

တစ်နည်းပြောရင် partion ၂ ခု ဖြစ်သွားပြီ။ pivot ရဲ့ left , right ပေါ်မှာ မူတည်ပြီး ၂ ခု ခွဲလိုက်ပြီးတော့ ထပ်စီရပါမယ်။ အခန်း 0 ကိုနေ right ထိက တစ်ခု။ left ကနေ ပြီးတော့ array အခန်း ကုန်ထိ က တစ်ခုပါ။

```
[4,2,3,1]
[8,7]
```

ကို ကျွန်တော်တို့ အစက စီသလို ပြန်ပြီး စီဖို့ လိုပါတယ်။

ကျွန်တော်တို့ pseudo code လေး ကို ကြည့်ရအောင်။

```
function quicksort(array)
    if length(array) > 1
        pivot := select any element of array
        left := first index of array
        right := last index of array
        while left <= right
            while array[left] < pivot && left < lenght of array
                left := left + 1
            while array[right] > pivot && right >= 0
                right := right - 1
            if left < right
                swap array[left] with array[right]
```

```
            left = left + 1
            right = right - 1
        else if left == right
            left = left + 1
    quicksort(array from first index to right)
    quicksort(array from left to last index)
```

ဒီ pseudo လေးကို အခြေခံပြီးတော့ python code ပြန်ရေးကြည့်ရအောင်။

```
from random import randint

def quickSort(array):
    if len(array) <= 1:
        return array

    room = randint(0,len(array) - 1)

    pivot = array[room]

    left = 0
    right = len(array) - 1

    while left <= right:

        while left < len(array) and array[left] < pivot:
            left += 1

        while right >= 0 and array[right] > pivot:
            right -= 1

        if left < right:
            temp = array[left]
            array[left] = array[right]
            array[right] = temp
            left += 1
            right -= 1
        elif left == right:
            left += 1

    leftSort = quickSort(array[0:right+1])
    rightSort = quickSort(array[left:len(array)])
    return leftSort+rightSort

array = [4,2,3,1,6,8,7]
print(quickSort(array))
```

```
from random import randint
```

ဆိုတာကတော့ random number ကို generate လုပ်ဖို့အတွက်ပါ။

```
room = randint(0,len(array) - 1)
```

ဆိုတာကတော့ အခန်း နံပတ် 0 ကနေ စပြီးတော့ array အခန်း နောက်ဆုံးထိ random number ကို generate လုပ်မယ်လို့ ဆိုတာပါ။

အခု ဆိုရင်တော့ quick sort သဘောတရားကို နားလည်လောက်ပါပြီ။ အခု chapter မှာ array ထဲမှာ value ရှာခြင်း နှင့် array sorting ပိုင်းကို ရှင်းပြသွားပါတယ်။ အခု အခန်းမှာ ပါသည့် အကြောင်း အရာ တစ်ခု ခြင်း စီကို သေချာ လိုက်ဖတ်ပြီးတော့ ကိုယ်တိုင် ရေးနိုင်မယ် တွေးနိုင်မယ်ဆိုရင်တော့ programming ရဲ့ ပြဿနာတွေကို စဉ်းစားတတ်လာပါလိမ့်မယ်။

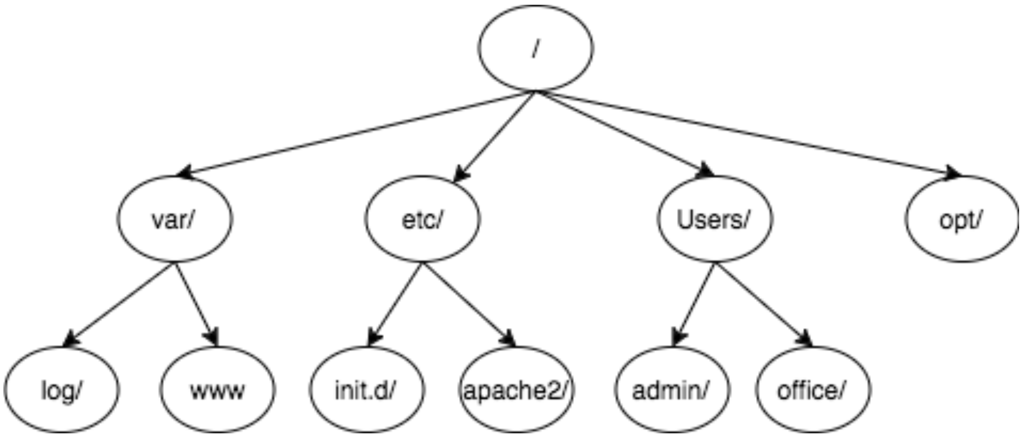
အခန်း ၉ ။ Tree

ကျွန်တော်တို့တွေဟာ ပြီးခဲ့သည့် အခန်းတွေမှာ data structure ပိုင်းတွေ ဖြစ်သည့် stack , queue , search , sort စသည့် အပိုင်းတွေကို သိပြီးသွားပါပြီ။ အခု အခါမှာတော့ data stucture ပိုင်းမှာ မဖြစ်မနေ သိသင့်သည့် tree အကြောင်းကို ဖော်ပြပါမယ်။

Tree ကို computer science ပိုင်းတွေ နေရာ တော်တော်များများ မှာ အသုံးပြုကြပါတယ်။ Operating Systems, graphic, database system နှင့် အခြား computer networking စသည့် နေရာ အတော်များများမှာ Tree data structure က မပါမဖြစ်ပါ။ ဒါကြောင့် Programming ကို လေ့လာမည့် သူများ အနေနှင့် Tree အကြောင်းကို မဖြစ်မနေ သိထားဖို့ လိုအပ်ပါတယ်။

သစ်ပင် တစ်ခုမှာ အောက်ခြေမှာ root (အမြစ်) ရှိပြီးတော့ အထက်ပိုင်းမှာ branches(ကိုင်းများ) ခွဲထွက်ပါတယ်။ ကျွန်တော်တို့ အခု tree မှာတော့ အထက်ပိုင်းက root ဖြစ်ပြီးတော့ အောက်ဘက်မှာ branches တွေ ခွဲ ပါတယ်။

ဥပမာ Linux က file system တစ်ခု ရဲ့ ပုံစံ အကြမ်းသဘောတရားလေးကို ကြည့်ရအောင်။



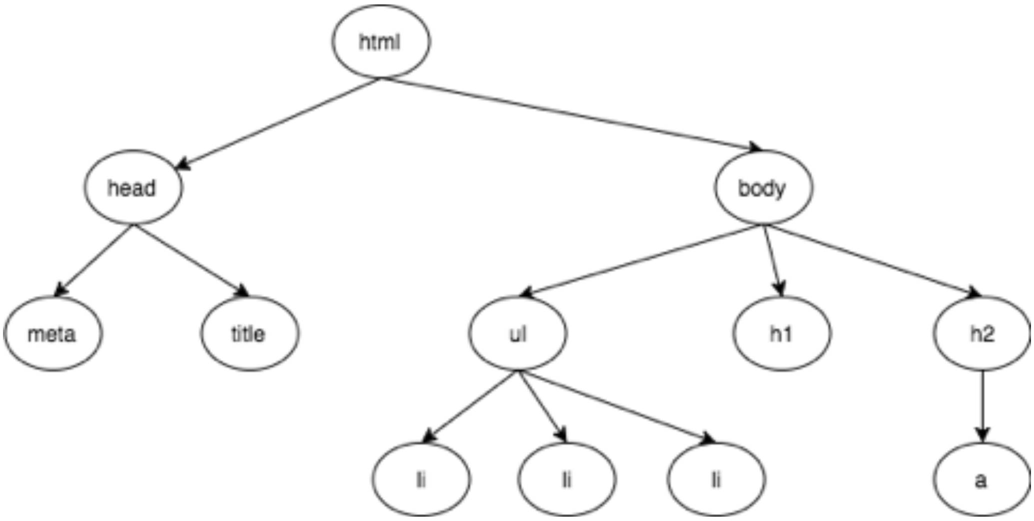
ထိပ်ဆုံးမှာ root (/) ရှိပါတယ်။ သူ့အောက်မှာ အခြား folder တွေ ဖြစ်သည့် var,etc,Users,opt စသည့် folder တွေ ပါဝင်ပါတယ်။ အဲဒီ folder တွေ အောက်မှာ အခြား folder တွေ ထပ်ပြီးတော့ ရှိသေးတယ်။ အဲဒါက tree system တစ်ခုပါပဲ။

နောက်ပြီးတော့ ကျွန်တော်တို့ နေ့စဉ် တွေ့မြင်နေကျ ဖြစ်သည့် website တွေကို HTML ဖြင့် တည်ဆောက်ထားပါတယ်။ HTML code example လေးကို အောက်မှာ ဖော်ပြထားပါတယ်။

```
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Simple</title>
    </head>
    <body>
```

```
            <h1>Simple Website</h1>
            <ul>
                <li>List item one</li>
                <li>List item two</li>
            </ul>
            <h2><a href="https://www.comquas.com">COMQUAS</a></h2>
        </body>
    </html>
```

HTML ဟာလည်း tree structure ပါပဲ။ HTML ကို tree structure နဲ့ ဆွဲကြည့်ရင် အောက်က ပုံလို မြင်ရပါမယ်။



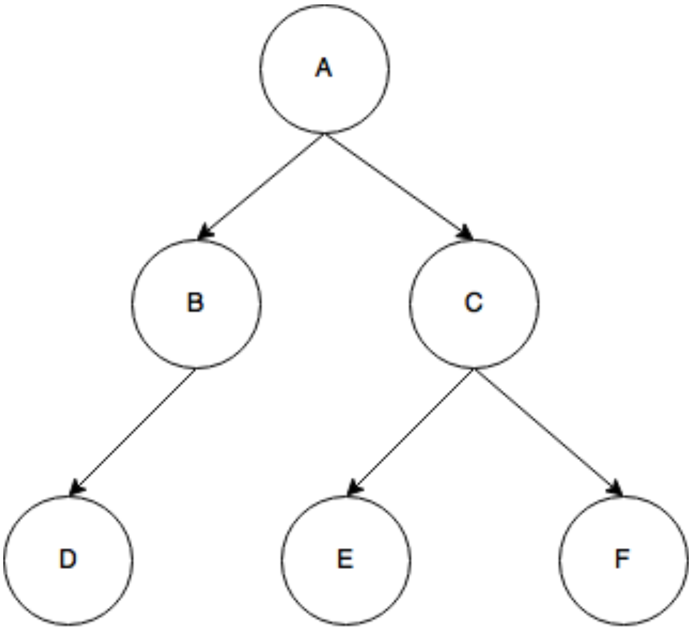
Tree structure ဟာ နေရာမျိုးစုံမှာ လိုသလို အသုံးပြုနေရပါတယ်။ ဒီ အခန်းမှာတော့ binary tree ကို အဓိကထားပြီးတော့ ဖော်ပြပေးသွားမှာပါ။

Binary Tree

Tree အကြောင်းကို လေ့လာတော့မယ်ဆိုရင် ဦးစွာ Binary Tree အကြောင်းကို နားလည် ဖို့ လိုပါတယ်။

Binary Tree ဆိုတာကတော့ Node တစ်ခု အောက်မှာ branch ၂ ခု ပဲ ရှိရမယ်။ အနည်းဆုံး branch 0 ကနေ 2 အထိပဲ ရှိသည့် tree system တစ်ခုပါ။

အခု Binary Tree ဥပမာ လေး ကြည့်ရအောင်။



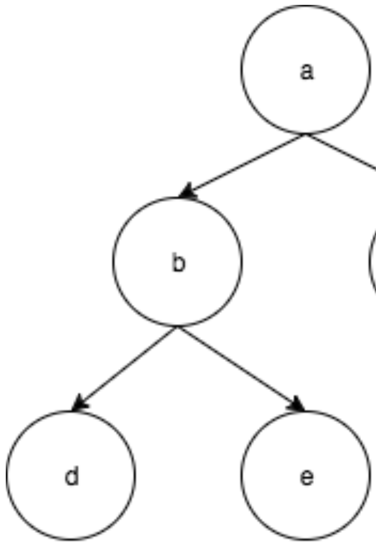
A ရဲ့ အောက်မှာ B နှင့် C ရှိတယ်။ B အောက်မှာ D ရှိတယ်။ C အောက်မှာတော့ E နဲ့ F ရှိပါတယ်။ ဒီ ပုံ တစ်ခုလုံးကိုတော့ tree လို့ ဆိုနိုင်တယ်။ A,B,C,D,E,F တွေကတော့ Node တွေပါ။ Binary tree ရဲ့ Node မှာ left နဲ့ right ရှိပါတယ်။ left ဘက်က child နဲ့ right ဘက်က child ပေါ့။ Node A ရဲ့ left ကတော့ B Node ဖြစ်ပြီးတော့ right ကတော့ C Node ပေါ့။ B Node ရဲ့ left ကတော့ Node D ဖြစ်ပြီး right ကတော့ empty ဖြစ်နေပါတယ်။ C ရဲ့ left ကတော့ E Node ဖြစ်ပြီးတော့ right node ကတော့ F ဖြစ်နေပါတယ်။

ဒါဆိုရင် ကျွန်တော်တို့ binary tree တစ်ခု ကို တည်ဆောက်ကြည့်ရအောင်။ ကျွန်တော်တို့ဆီမှာ BinaryTree class တစ်ခုရှိမယ်။ left_child နဲ့ right_child ရှိမယ်။ အဲဒီ ၂ ခု လုံးက လည်း BinaryTree class တွေ ဖြစ်ရမယ်။ နောက် တစ်ခုက လက်ရှိ root key ရှိရမယ်။

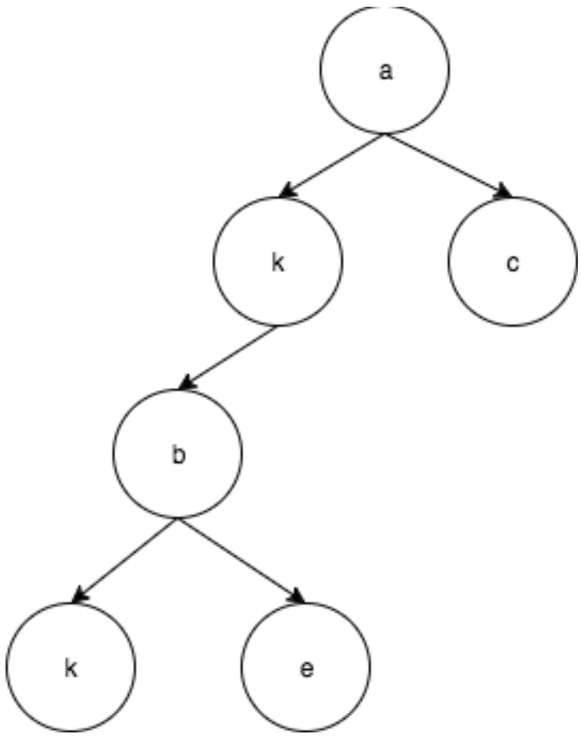
```
class BinaryTree:
    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None
```

နောက်တဆင့် အနေနဲ့ စဉ်းစားရင် left နှင့် right ထည့်ဖို့ လိုမယ်။ အဲဒီမှာ ဘာကို ထပ်ပြီး စဉ်းစားဖို့ လိုလဲဆိုတော့ left ထဲမှာ data ရှိနေရင် အသစ်ထည့်လိုက်သည့် tree ထဲမှာ append သွားလုပ်ဖို့ လိုတယ်။ right ထဲမှာ data ရှိနေရင်လည်း အသစ်ထည့်မယ့် tree ထဲမှာ append လုပ်ဖို့ လိုပါတယ်။

ဥပမာ။



အထက်က ပုံလေးဆိုရင် Node အသစ် မထည့်ရသေးဘူး။ အဲဒီ အထဲမှာ ကျွန်တော် k Node လေး ကို left ဘက်မှာ ထည့်လိုက်ရင် အောက်ကလို ဖြစ်သွားပါမယ်။



ဒါကြောင့် insert မှာ ကျွန်တော်တို့ အနေနဲ့ data ရှိပြီးသားလား မရှိရသေးဘူးလား စစ်ဖို့ လိုမယ်။ မရှိသေးဘူးဆိုရင် တစ်ခါတည်း ထည့်မယ်။ ရှိမယ် ဆိုရင်တော့ လက်ရှိ ရှိနေသည့် node ကို node အသစ်မှာ လာထည့်ဖို့လိုပါလိမ့်မယ်။

```python
class BinaryTree:
    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
```

```
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t
```

အခု ထပ်ပြီးတော့ ရှိသည့် left, right ကို ဆွဲထုတ်ဖို့ ရေးရအောင်။

```
class BinaryTree:
    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t

    def get_right_child(self):
        return self.right_child

    def get_left_child(self):
        return self.left_child

    def set_root_val(self,obj):
        self.key = obj

    def get_root_val(self):
        return self.key
```

အခု အချိန်မှာတော့ code လေးတွေက ခက်ခက်ခဲခဲမဟုတ်ပဲ နဲ့နားလည် နိုင်ပါတယ်။

ကျွန်တော်တို့ရဲ့ Binary Tree ကို စမ်းကြည့်ရအောင်။

```
class BinaryTree:

    def __repr__(self):
        return "Binary Tree, Key is " + self.key
```

```
    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t

    def get_right_child(self):
        return self.right_child

    def get_left_child(self):
        return self.left_child

    def set_root_val(self,obj):
        self.key = obj

    def get_root_val(self):
        return self.key



    def postorder(self):
        if self != None:
            if self.get_left_child() != None:
                self.get_left_child().postorder()
            if self.get_right_child() != None:
                self.get_right_child().postorder()
            print(self.get_root_val())


from binarytree import BinaryTree

root = BinaryTree('a')

print(root)

print(root.get_root_val())
print(root.get_left_child())

root.insert_left('b')
print(root.get_left_child().get_root_val())

root.insert_right('c')
print(root.get_right_child().get_root_val())

root.get_right_child().set_root_val('hello')
print(root.get_right_child().get_root_val())

root.insert_left('d')
```

```
print(root.get_left_child())
print(root.get_left_child().get_left_child().get_root_val())
```

အဲဒီမှာ

```
def __repr__(self):
        return "Binary Tree, Key is " + self.key
```

ဆိုတာလေးကို တွေ့ရလိမ့်မယ်။ အဲဒါကတော့ ကျွန်တော်တို့ object ကို print နဲ့ ထုတ်သည့် အခါ မှာ `<__main__.BinaryTree object at 0x10293b5f8>` ဆိုပြီး မပေါ်ချင်ပဲ key ကို ထုတ်ပြချင်သည့် အတွက်ကြောင့် `__repr__` ဆိုသည့် function မှာ ဝင်ရေးထားတာပါ။

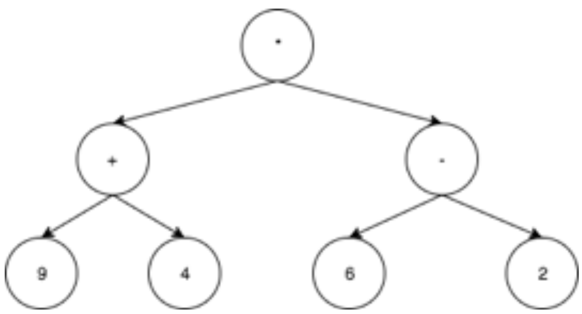
root.insert_left('d')

print(root.get_left_child())

print(root.get_left_child().get_left_child().get_root_val())
ဒီ code ဆိုရင်လည်း a ရဲ့ left မှာ b ရှိတယ်။ ထပ်ပြီးတော့ d ကိုထည့်လိုက်တယ်။ ပြီးမှ a ရဲ့ left child ရဲ့ left child ကို ပြန်ပြီးတော့ print ထုတ်ထားတာပါ။ a ရဲ့ left child က d ဖြစ်သွားပြီးတော့ d ရဲ့ left child ကတော့ b ဖြစ်သွားတာကို တွေ့ရပါလိမ့်မယ်။

Parse Tree

အခု Tree ကို အသုံးပြုပြီးတော့ parse tree တစ်ခု ကို ဖန်တီးရအောင်။ နားလည် အောင် သဘော ပြောရရင်တော့ `((9 + 4) * (6 - 2))` ကို parse tree ထုတ်လိုက်ရင် အောက်ပါ ပုံအတိုင်း ထွက်လာပါလိမ့်မယ်။



အထက်ပါ diagram ကို ကြည့်လိုက်ရင် ရိုးရှင်းပါတယ်။ ၉ နှင့် ၄ ကို အရင် ပေါင်းမယ်။ ပြီးရင် ၆ ထဲ က ၂ ကို နှုတ်မယ်။ ပြီးရင် ပြန်မြှောက်မယ်။

အခု ကျွန်တော်တို့တွေအနေနဲ့ User က ပေးလိုက်သည့် parser ပေါ်မှာ မူတည်ပြီးတော့ parse tree တစ်ခု တည်ဆောက်ပါမယ်။

ဦးစွာ ကျွန်တော်တို့တွေ တည်ဆောက်ဖို့အတွက် rule လေးတွေ ကြည့်ပါမယ်။

၁။ `(` ပါ လာခဲ့ရင် node အသစ်တစ်ခုကို လက်ရှိ node ရဲ့ ဘယ်ဘက် child မှာ တည်ဆောက်ဖို့ လိုပါတယ်။

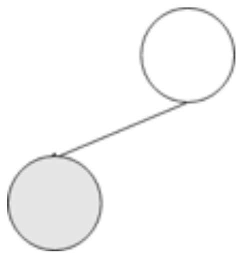
၂။ `+ , - , * , /` တစ်ခု ဖြစ်ရင်တော့ လက်ရှိ node ရဲ့ value ကို ရလာသည့် သင်္ကေတ ကို ထည့်ပါမယ်။ ညာဘက်မှာ node အသစ်တစ်ခု တည်ဆောက်ပါမယ်။

၃။ value ကတော့ နံပတ် ဖြစ်နေရင် လက်ရှိ node ရဲ့ value ကို အဲဒီ နံပတ် ထည့်မယ် ပြီးရင် သူ့ရဲ့ parent node ကို ပြန်သွားပါမယ်။ ဒါကြောင့် current node က သူ့ရဲ့ parent node ဖြစ်သွားပါလိမ့် မယ်။

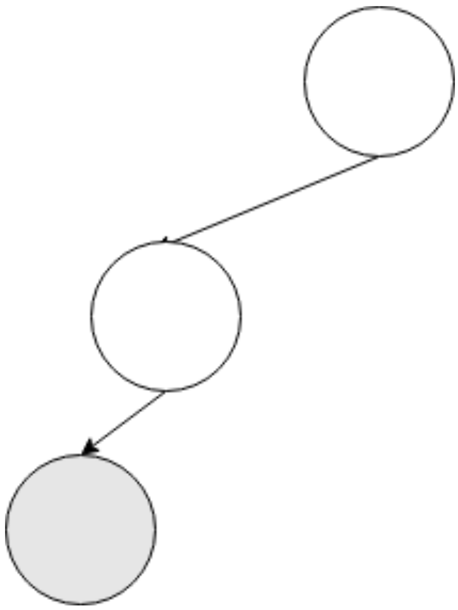
၄။ `)` တွေ့ခဲ့ရင်လည်း parent node ဆီ ပြန်သွားဖို့ပါပဲ။

အခု ကျွန်တော်တို့တွေ `((9 + 4) * (6 - 2))` ကို စမ်းကြည့်ရအောင်။

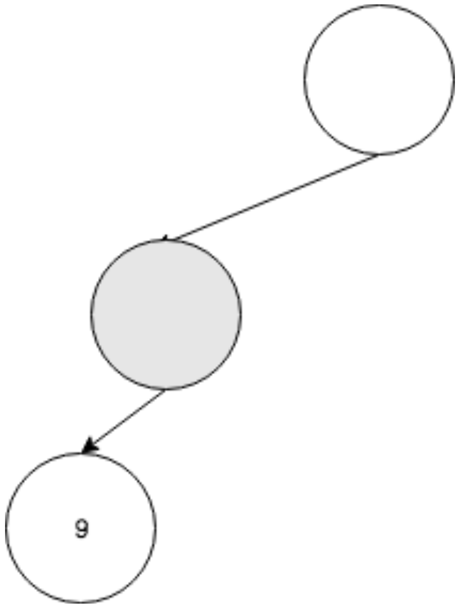
`(` တွေ့သည့် အတွက် left child တစ်ခု တည်ဆောက်ပါမယ်။



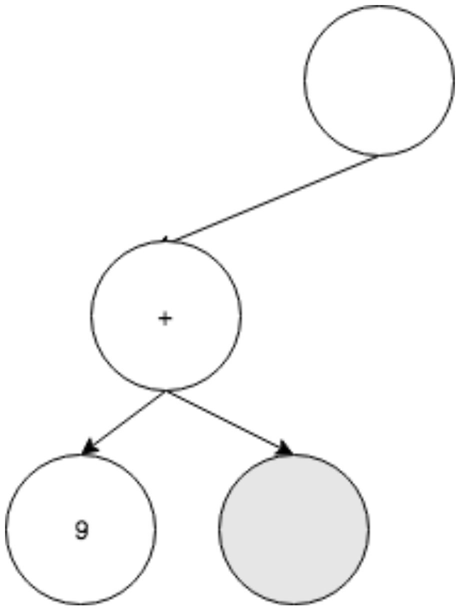
`(` ထပ်တွေ့သည့် အတွက် ထပ်ပြီးတော့ left child တစ်ခု ထပ်ဆောက်ပါမယ်။



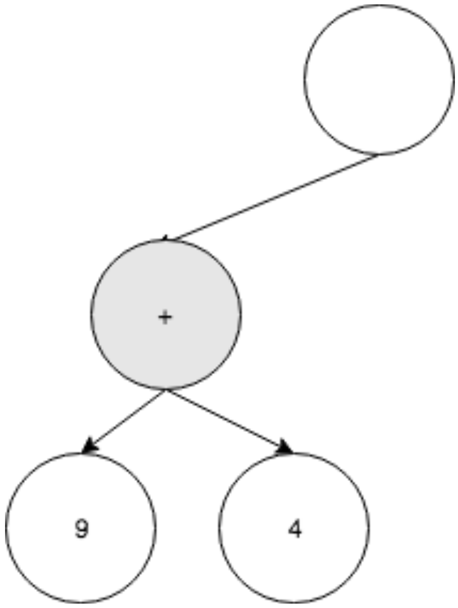
`9` ကို တွေ့သည့်အတွက် value ကို ထည့်ပြီးတော့ parent node ကို ပြန်သွားပါမယ်။



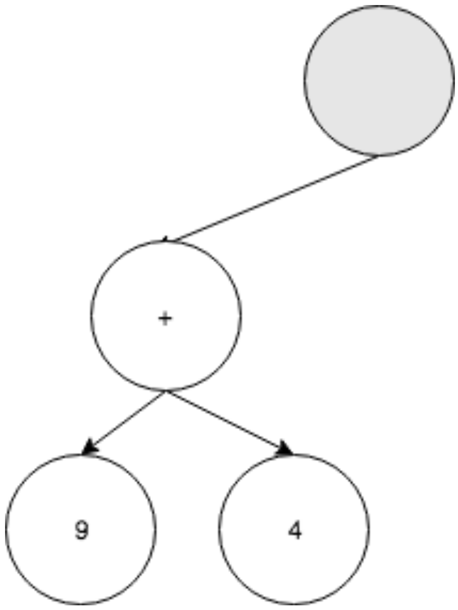
+ ကို တွေ့သည့် အတွက်ကြောင့် value ထည့်ပြီးတော့ right child တစ်ခု ဆောက်ပါမယ်။



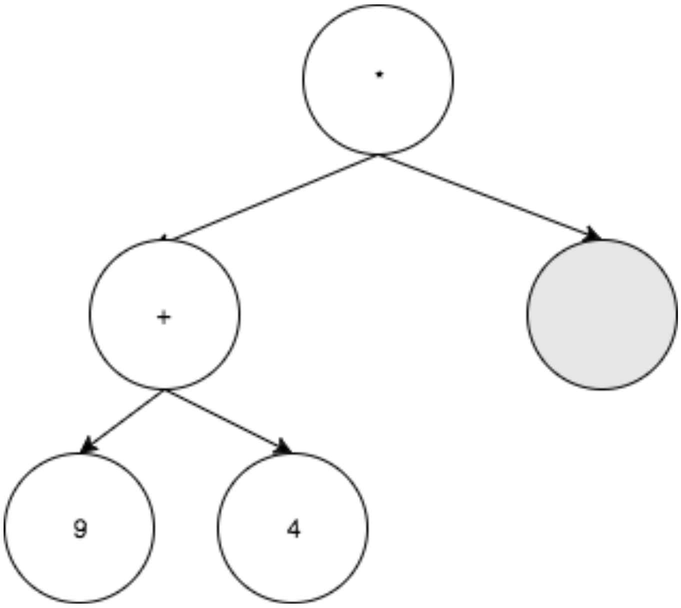
4 ကို တွေ့သည့် အတွက်ကြောင့် value ထည့်ပြီးတော့ parent ကို ပြန်သွားပါမယ်။



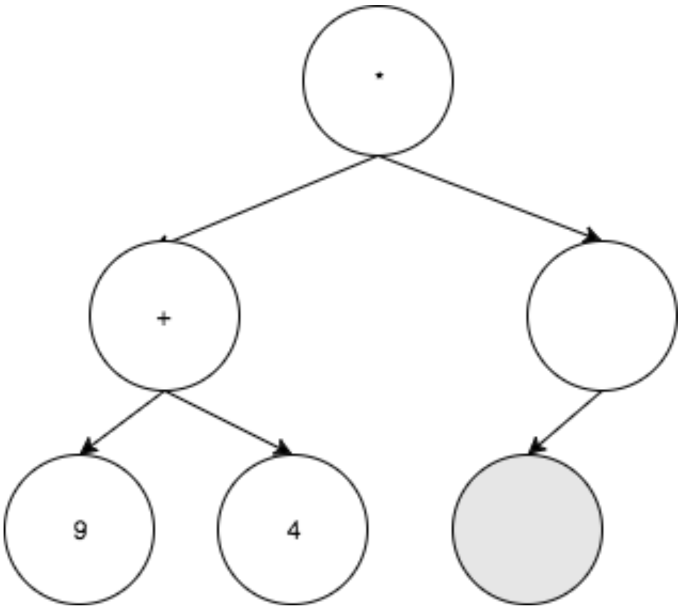
) ကို တွေ့သည့် အတွက်ကြောင့် parent node ကို ပြန်သွားပါမယ်။



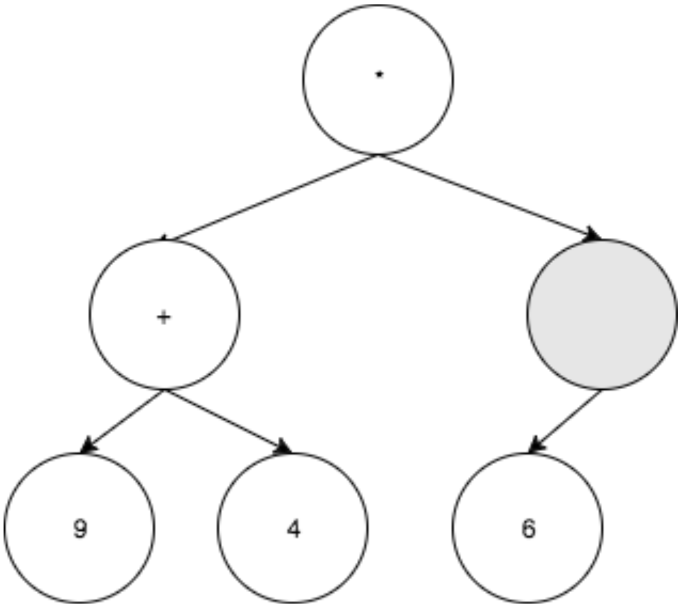
* ကို တွေ့သည့် အတွက်ကြောင့် value ထည့်ပြီးတော့ right child ကို ဆောက်ပါမယ်။



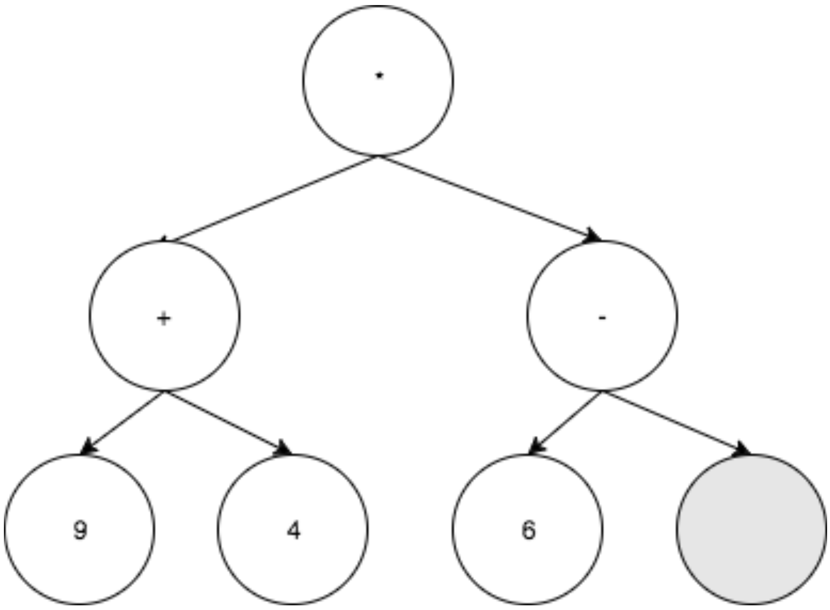
(ကို တွေ့သည့် အတွက် ကြောင့် left child ဆောက်ပါမယ်။



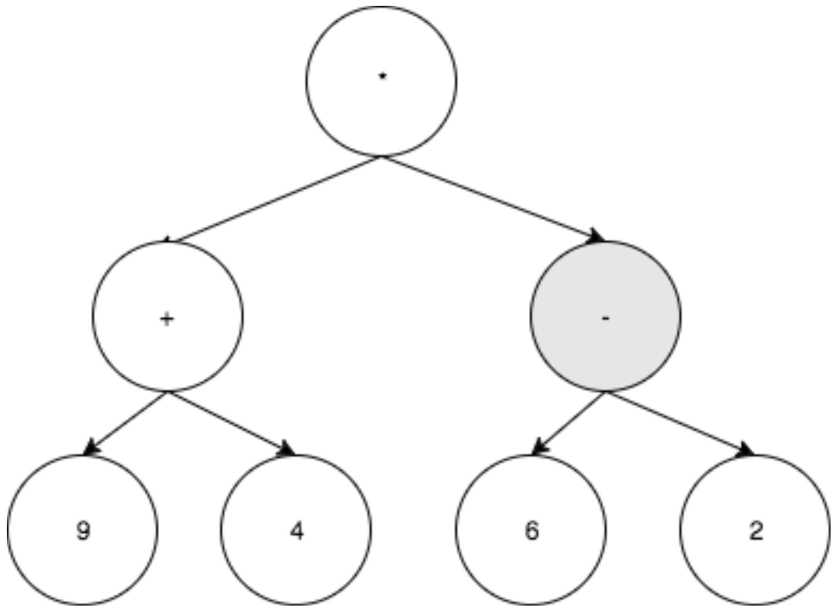
6 ကို တွေ့သည့် အတွက်ကြောင့် value ထည့်မယ်။ parent ကို ပြန်သွားပါမယ်။



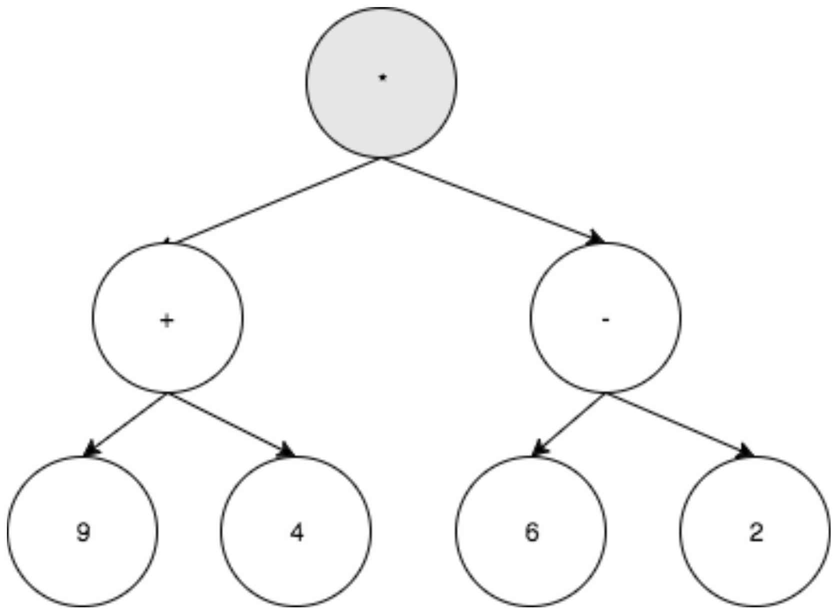
- ကို တွေ့သည့် အတွက်ကြောင့် value ထည့်မယ်။ right child ကို ဆောက်ပါမယ်။



2 ကို တွေ့သည့် အတွက်ကြောင့် value ထည့်မယ်။ parent node ကို ပြန်သွားမယ်။



`)` ကို တွေ့သည့် အတွက်ကြောင့် parent node ကို ထပ်သွားပါမယ်။



`)` ကို တွေ့တယ်။ parent ကို ထပ်သွားတယ်။ ဒါပေမယ့် parent မရှိတော့သည့် အတွက် current node မှာ ဘာမှ မရှိတော့ပါဘူး။ ကျွန်တော်တို့လည်း parse tree ဆွဲလို့ပြီးပါပြီ။

အခု ကျွန်တော်တို့ parse tree ကို code အနေနဲ့ ရေးကြည့်ရအောင်။

```
from binarytree import BinaryTree
from stack import Stack

def build_parse_tree(fp_exp):
    fp_list = fp_exp.split()
    p_stack = Stack()
    e_tree = BinaryTree('')
    p_stack.push(e_tree)
    current_tree = e_tree
    for i in fp_list:

        if i == '(':
            current_tree.insert_left('')
```

```
            p_stack.push(current_tree)
            current_tree = current_tree.get_left_child()
        elif i not in ['+', '-', '*', '/', ')']:
                current_tree.set_root_val(int(i))
                parent = p_stack.pop()
                current_tree = parent
        elif i in ['+', '-', '*', '/']:
            current_tree.set_root_val(i)
            current_tree.insert_right('')
            p_stack.push(current_tree)
            current_tree = current_tree.get_right_child()
        elif i == ')':
            current_tree = p_stack.pop()
        else:
            raise ValueError
    return e_tree

pt = build_parse_tree("( ( 9 + 4 ) * ( 6 - 2 ) )")
pt.postorder()
```

အခု code မှာ ကျွန်တော်တို့တွေ binary tree နဲ့ stack ကို သုံးထားတာကို တွေ့နိုင်ပါတယ်။ အဲဒီမှာ ထူးထူးခြားခြား postorder ဆိုတာ ပါလာပါတယ်။ ဒီအကြောင်းကို နောက် အခန်းမှာ ဆက်ရှင်းပြပါမယ်။

Tree Traversals

Tree traversals ဆိုတာကတော့ tree တစ်ခုမှာ node တစ်ခု ခြင်းဆီ ကို သွားသည့် process လို့ ဆိုရပါမယ်။ Tree Traversals ၂ မျိုး ရှိပါတယ်။

Depth-first search

Breadth-first search

ဆိုပြီး ရှိပါတယ်။

Depth-first search (DFS)

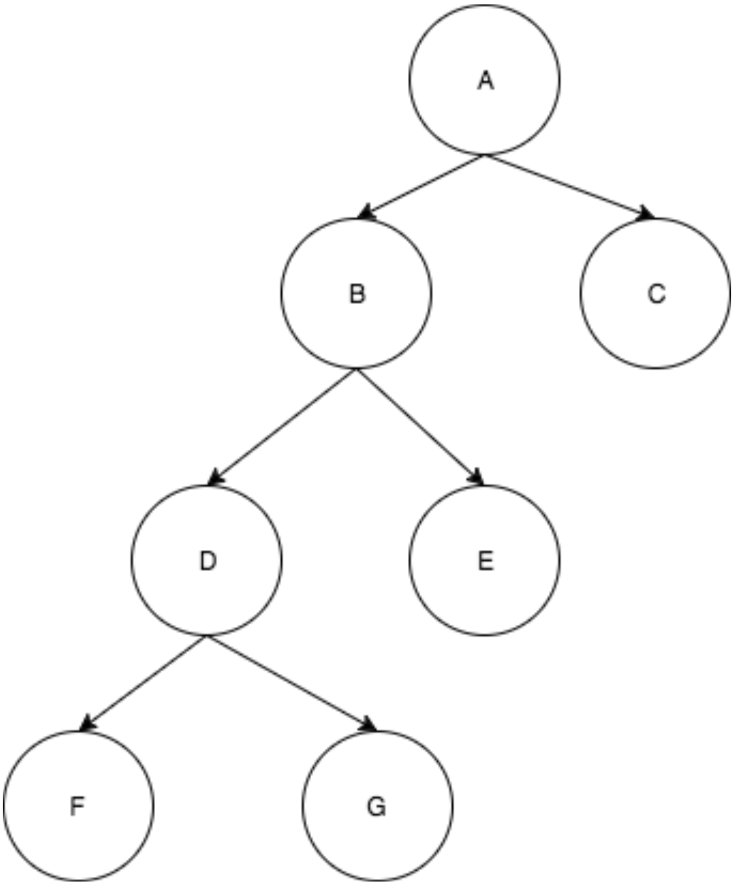
DFS ကို အသုံးပြုပြီး data တွေကို ပြန်ပြီး ထုတ်ပြဖို့ အတွက် ထုတ်ပြနည်း ၃ ခု ရှိပါတယ်။

- in order
- pre order
- post order

ဆိုပြီး ရှိပါတယ်။

In Order

In order ဆိုတာကတော့ ဘယ်ဘက် က data ကို ပြမယ်။ ပြီးမှ center ကို ပြမယ်။ ပြီးရင် ညာဘက် ကို ပြမယ်။



In order အစီ ထုတ်မယ် ဆိုရင်

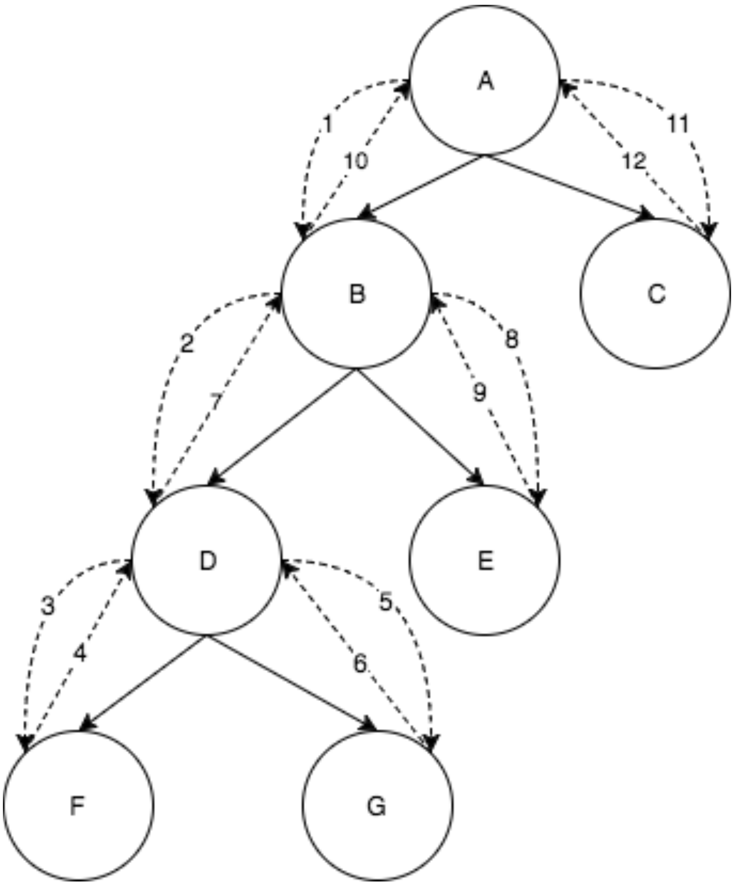
- F
- D
- G
- B
- E
- A
- C

ဆိုပြီး ထုတ်ပါမယ်။ ဘယ်ဘက် က အောက်ဆုံးကို အရင် ထုတ်တယ်။ ပြီးတော့ သူ့ရဲ့ parent ကို ထုတ်ပြတယ်။ ပြီးတော့ ညာဘက် က node ကို ထုတ်တယ်။

ဘယ်ဘက် အောက်ဆုံးက F ဖြစ်ပြီးတော့ F ရဲ့ parent က D ပါ။ ပြီးတော့ ညာဘက် G ကို ထုတ်မယ်။ ညာဘက် node က ဆက်မရှိတော့သည့် အတွက် parent ကို ပြန်သွားမယ်။ parent က လည်း ကိစ္စ ပြီးပြီ ဖြစ်သည့် အတွက် သူ့ရဲ့ parent ကို ပြန်သွားမယ်။ အဲဒီ parent က node ကို B ကို ထုတ်ပြပါတယ်။ ပြီးတော့ ညာဘက် က E ကို ထုတ်ပြတယ်။ parent ကို ပြန်သွားတယ်။ A ကို ထုတ်ပြတယ်။ ပြီးတော့ ညာဘက်က C ကို ထုတ်ပြတယ်။

In order ကတော့ အောက်ဆုံးမှာ ရှိသည့် ဘယ်ဘက် က node ကို အရင်ပြမယ်။ ပြီးရင် parent ကို ပြမယ်။ ပြီးရင် ညာဘက် အောက်ဆုံး ထိ ဆင်းပြီးမှ ပြမယ်။

သွားသည့် flow လေးကို ကြည့်ရအောင်။



အဲဒီ flow လေးကို ကြည့်လိုက်ရင် in order ကို ကောင်းမွန်စွာ နားလည်သွားပါလိမ့်မယ်။

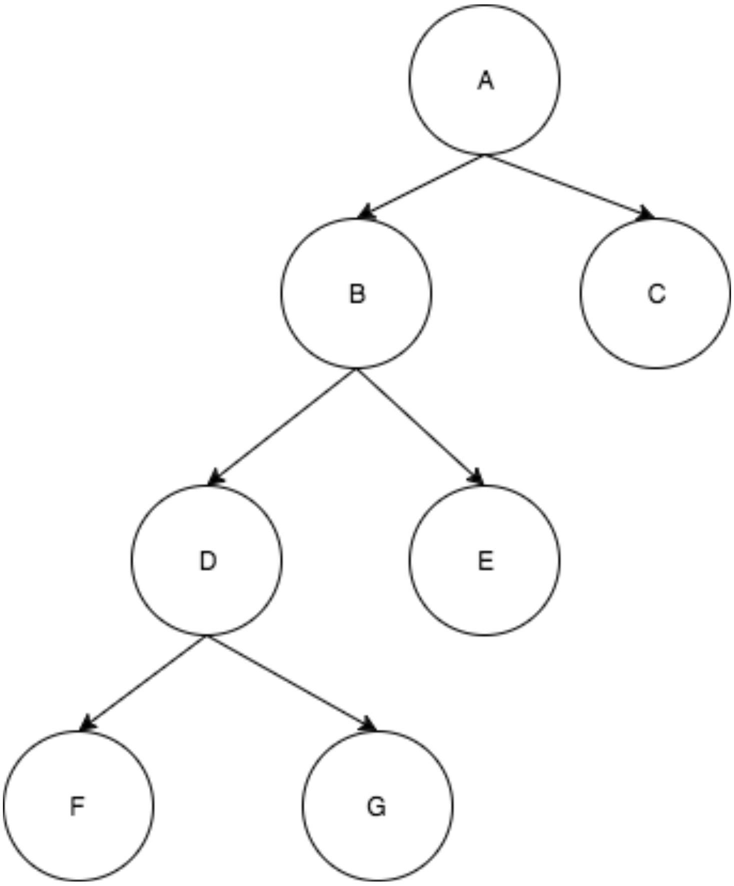
အခု code လေး ကို ကြည့် ရအောင်။

```
def inorder(self):
    if self != None:
        if self.get_left_child() != None:
            self.get_left_child().inorder()

        print(self.get_root_val())

        if self.get_right_child() != None:
            self.get_right_child().inorder
```

ကျွန်တော်တို့ recursive ကို သုံးပြီးတော့ left child တွေ အကုန် သွားပါတယ်။ နောက်ဆုံး အဆင့်မှာ root value ကို print ထုတ်ထားတယ်။ ပြီးသွားမှာ parent ရဲ့ value ကို ထုတ်ထားတယ်။ ပြီးရင် right တွေ အကုန်ပြန် ဆင်းချထားတာကို တွေ့နိုင်ပါတယ်။

Pre Order

In Order ကို နားလည်သွားရင်တော့ pre order က node value ကို အရင်ထုတ်ပြီးမှ left ကို သွားတာပါ။ အနည်းငယ် ကွာခြားသွားတယ်။



အဲဒီ ပုံလေးကို pre order အရ ထုတ်မယ်ဆိုရင်တော့

- A
- B
- D
- F
- G
- E
- C

ဆိုပြီး ထွက်လာပါလိမ့်မယ်။

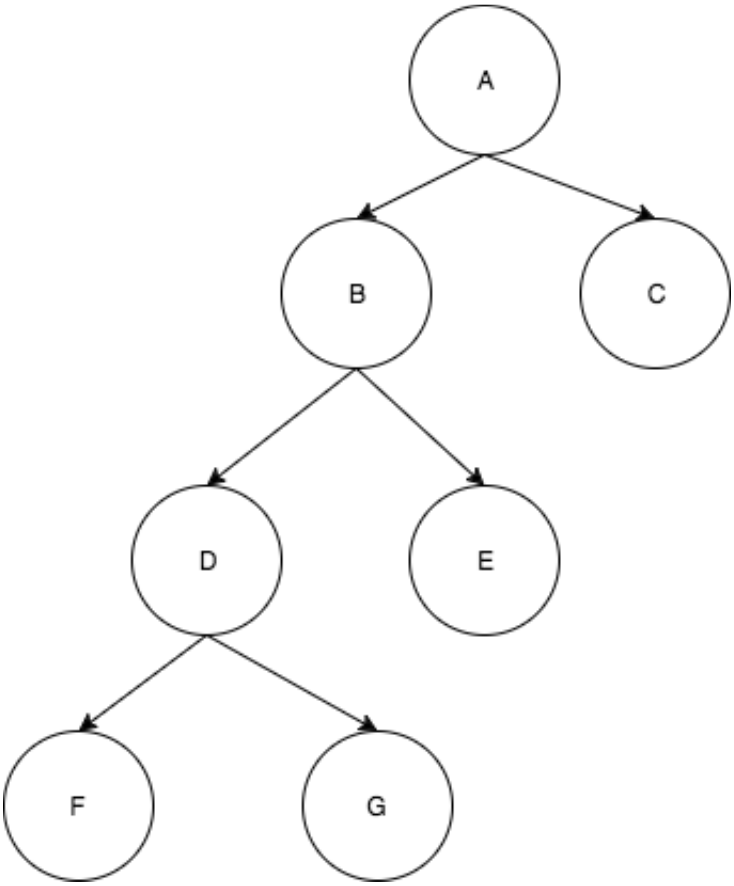
In order ကို နားလည်ထားပြီးပြီတော့ အသေးစိတ် မရှင်းတော့ပါဘူး။ code လေးကို ကြည့် ရအောင်။

```
def preorder(self):
    if self != None:
        print(self.get_root_val())

        if self.get_left_child() != None:
            self.get_left_child().preorder()

        if self.get_right_child() != None:
            self.get_right_child().preorder()
```

အရင်ဆုံး node ရဲ့ value ကို ထုတ်လိုက်ပါတယ်။ ပြီးမှ left ကို သွားပါတယ်။ left အကုန်ပြီးမှ right ကို သွားတာကို တွေ့ပါလိမ့်မယ်။

Post Order

Pre order နဲ့ အနည်းငယ်သာ ကွဲပြားပါတယ်။ အရင်ဆုံး left ကို ထုတ်တယ်။ ပြီးမှ right ကို ထုတ်တယ်။ ပြီးမှ node ရဲ့ value ကို ထုတ်မယ်။



အထက်ပါ binary tree ကို ထုတ်မယ်ဆိုရင်

- F
- G
- D
- E
- B
- C
- A

ဆိုပြီး ထွက်လာပါမယ်။

အောက်ဆုံး F က အရင် လာမယ်။ ပြီးရင် ညာဘက်က G လာမယ်။ ပြီးမှ သူ့ရဲ့ parent D လာပါမယ်။ ပြီးရင် ညာဘက် က E လာမယ်။ ပြီးမှ parent B လာပါမယ်။ B ရဲ့ ညာဘက်က C လာမယ်။ ပြီးမှ parent A လာပါမယ်။

code ကတော့ ဆင်တူပါပဲ။

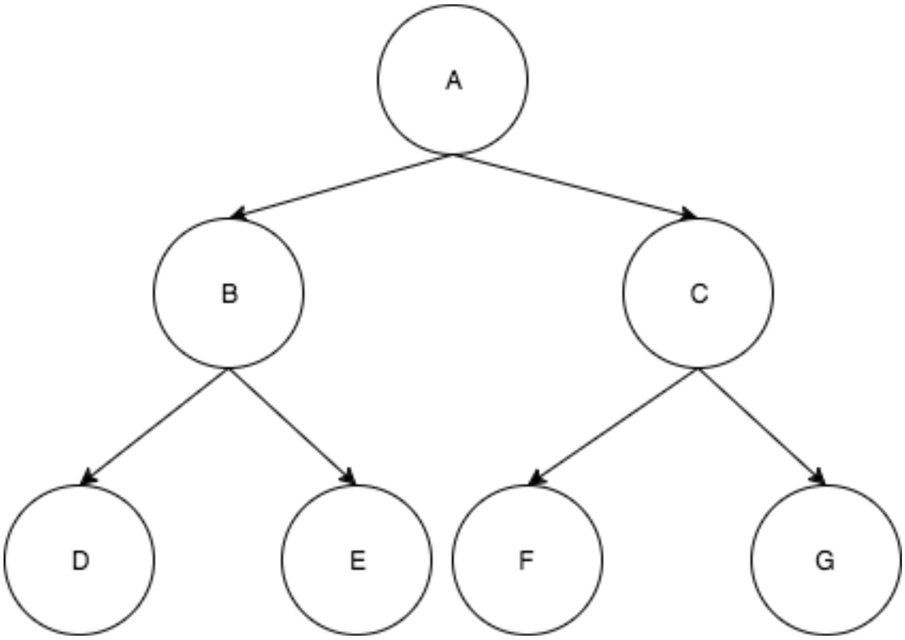
```
def postorder(self):
    if self != None:
        if self.get_left_child() != None:
            self.get_left_child().postorder()

        if self.get_right_child() != None:
            self.get_right_child().postorder()

        print(self.get_root_val())
```

အခု ဆိုရင်တော့ ကျွန်တော်တို့တွေ binary Tree တစ်ခု လုံး ကို သွားတတ်နေပါပြီ။ code လေးကို ကြည့်ရအောင်။

```
class BinaryTree:

    def __repr__(self):
        return "Binary Tree, Key is " + self.key

    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t

    def get_right_child(self):
        return self.right_child

    def get_left_child(self):
        return self.left_child

    def set_root_val(self,obj):
        self.key = obj

    def get_root_val(self):
        return self.key
```

```
    def inorder(self):
        if self != None:
            if self.get_left_child() != None:
                self.get_left_child().inorder()

            print(self.get_root_val())

            if self.get_right_child() != None:
                self.get_right_child().inorder()


    def postorder(self):
        if self != None:
            if self.get_left_child() != None:
                self.get_left_child().postorder()

            if self.get_right_child() != None:
                self.get_right_child().postorder()

            print(self.get_root_val())

    def preorder(self):
        if self != None:
            print(self.get_root_val())

            if self.get_left_child() != None:
                self.get_left_child().preorder()

            if self.get_right_child() != None:
                self.get_right_child().preorder()

root = BinaryTree("A")

root.insert_left("B")
root.insert_right("C")

b = root.get_left_child()

b.insert_left("D")
b.insert_right("E")

d = b.get_left_child()

d.insert_left("F")
d.insert_right("G")

print("---- In Order ----")
root.inorder()
print("---- Pre Order ----")
root.preorder()
print("---- Post Order ----")
root.postorder()
```

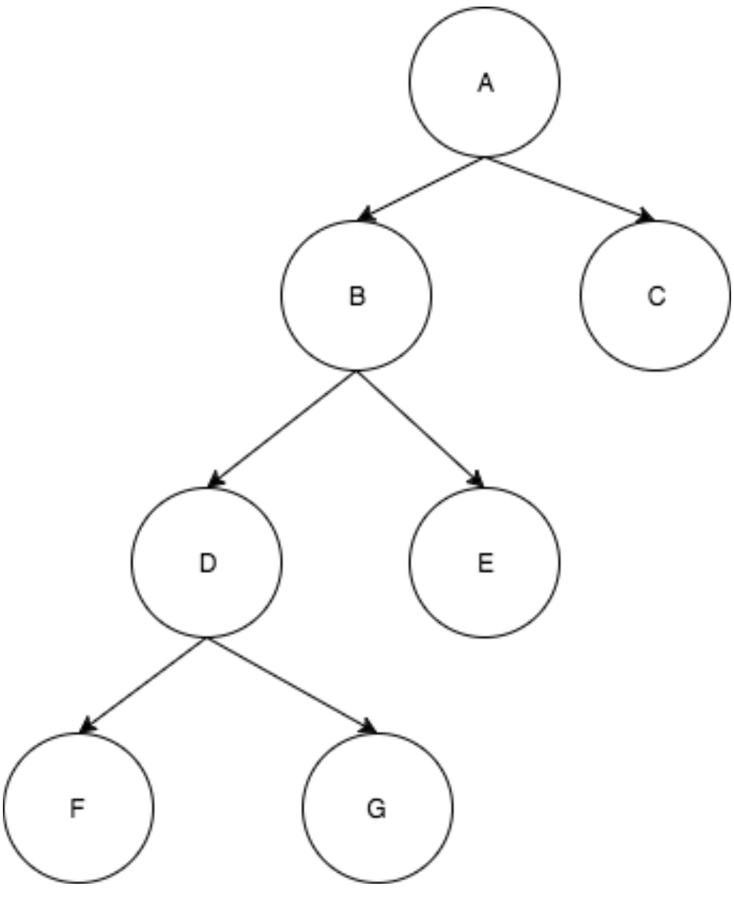
လေ့ကျင့်ခန်း ၉-၁



၁။ အထက်ပါ binary tree တစ်ခု တည်ဆောက်ပါ။ ထို binary tree အတွက် dfs ကို သုံးပြီး search function ကို ရေးပါ။ ဥပမာ ။ F လို့ ထည့်လိုက်လျှင် binary tree တွင် ပါဝင်သောကြောင့် true ဟု return ပြန်ပါမည်။ H ဟု ထည့်လိုက်လျှင် ရှာ မတွေ့သောကြောင့် false ဟု return ပြန်ရမည်။

Breadth-first search (BFS)

အခု ကျွန်တော်တို့ နောက်တနည်းဖြစ်သည့် BFS ကို အသုံးပြုပြီးတော့ Binary Tree က node တွေကို အဆင့်လိုက်သွားပါမယ်။



ပုံမှာ BFS အရ ဆိုရင်တော့

- A
- B , C
- D , E
- F , G
- ဆိုပြီးတော့ level အဆင့်လိုက် ထုတ်ပြပါလိမ့်မယ်။

ကျွန်တော်တို့တွေ အနေနဲ့ ပထမဆုံး root ကနေ စပါမယ်။ root က A ပါ။ A ရဲ့ left နှင့် right ကို array ထဲမှာ မှတ်ထားတယ်။ B,C ပေါ့။ ပြီးရင် B ရဲ့ left နှင့် right ကို array ထဲမှာ မှတ်ထားမယ်။ D,E ပါ။ C ရဲ့ left နှင့် right ကို ထပ်ပြီးတော့ မှတ်ထားမယ်။ သို့ပေမယ့် C မှာ child မရှိသည့် အတွက်ကြောင့် D,E ပဲ ရှိပါတော့မယ်။ အကယ်၍ C မှာ left နှုင့် right မှာ child ရှိခဲ့လျှင် မှတ်ထားသည့် value က D,E,left Of C, right of C ဖြစ်သွားပါမယ်။ အခုတော့ D,E မှာ D ရဲ့ left နဲ့ right F,G ကို မှတ်ထားပါတယ်။ E မှာ child မရှိသည့် အတွက်ကြောင့် မှတ်ထားသည့် array ထဲမှာ F,G ပဲ ရှိမယ်။ F မှာ child မရှိတော့ဘူး။ G မှာလည်း child မရှိတော့ပါဘူး။ array အခန်းထဲမှာ မရှိတော့သည့် အတွက်ကြောင့် loop ကနေ ထွက်သွားပါမယ်။ node တွေ အားလုံးကိုလည်း ရောက်ခဲ့ပြီးပါပြီ။

Pseudo code နဲ့ စဉ်းစားကြည့်ရအောင်ဗျာ။

```
current_level = [Root_A]
Loop Until current_level is not empty
    next_level = [] //create empty array for saving
    level_data = [] //to store current level value

    For node in current_level
        level_data.append(node.value)

        if node.left_child is not empty
            next_level.append(node.left_child)

        if node.right_child is not empty
            next_level.append(node.right_child)
    End For Loop

    Print level_data

    current_level = next_level // start again for child data

End Loop
```

အဲဒါလေးတွေကတော့ စဉ်းစားပြီး ရေးချထားသည့် pseudo code တွေပါ။ ပထမဆုံး ထိပ်ဆုံး root ကနေ စတယ်။ ပြီးရင် သူ့အောက် level က child ကို array ထဲမှာ ထည့်တယ်။ အစကနေ loop ပြန်ပတ်တယ်။ child ထဲမှာ ရှိသည့် left,right ကို array ထဲကို ထည့်တယ်။ အကုန်ပြီးသွားရင် အစကနေ loop ပြန်ပတ်ထားတာကို တွေ့နိုင်ပါတယ်။ ဘယ် အချိန် loop ပတ်နေလဲ ဆိုတော့ child တွေ တစ်ခုမှ မရှိတော့သည့် အခိ loop ပတ်နေပါတယ်။

python code ပြောင်းရေးကြည့်ရအောင်။

```
class BinaryTree:

    def __repr__(self):
        return "Binary Tree, Key is " + self.key

    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t

    def get_right_child(self):
        return self.right_child

    def get_left_child(self):
        return self.left_child

    def set_root_val(self,obj):
        self.key = obj

    def get_root_val(self):
        return self.key

    def bfs(self):
        thislevel = [self]
        while thislevel:
            nextlevel = []
            level = []
            for n in thislevel:
                level.append(n.get_root_val())
                if n.get_left_child() != None:
                    nextlevel.append(n.get_left_child())
                if n.get_right_child() != None:
                    nextlevel.append(n.get_right_child())
            print(",".join(level))
```

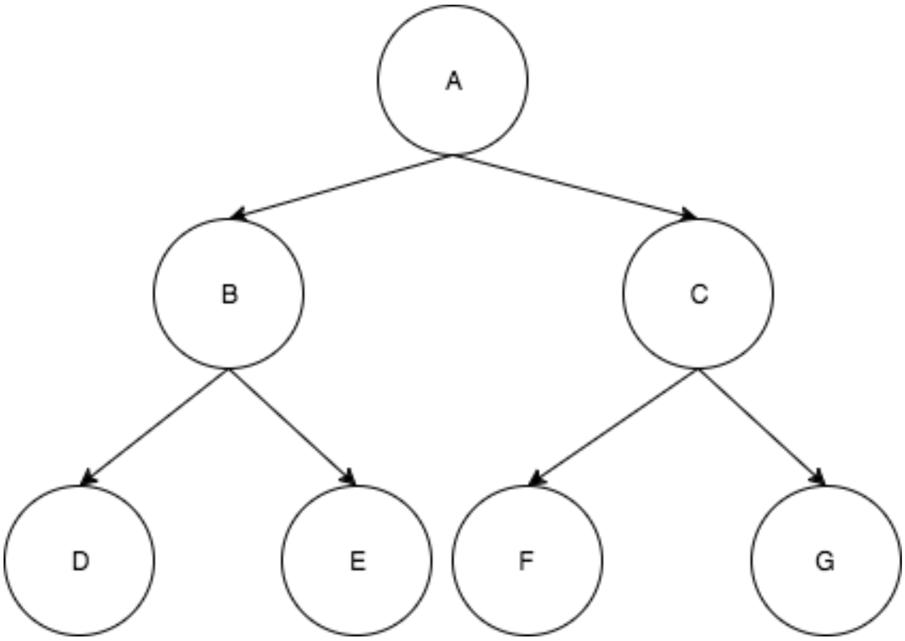
```
            thislevel = nextlevel


root = BinaryTree("A")

root.insert_left("B")
root.insert_right("C")

b = root.get_left_child()

b.insert_left("D")
b.insert_right("E")

d = b.get_left_child()

d.insert_left("F")
d.insert_right("G")

root.bfs()
```

အထက်ပါ code မှာ `print(",".join(level))` ဆိုသည်မှာ array အား string အနေဖြင့် ပြရန် အတွက် ဖြစ်သည်။ array မှ data များအား comma(,) ဖြင့် ဖော်ပြရန် အတွက် `",".join` အား အသုံးပြုထားခြင်း ဖြစ်သည်။

အခု ဆိုရင်တော့ BFS ကို အသုံးပြုပြီးတော့ binary tree ရဲ့ node တွေ အားလုံးကို သွားတတ်ပြီလို့ ထင်ပါတယ်။

Binary tree ဟာ left နှင့် right ၂ ခု ပဲရှိပါတယ်။ ကိုယ်တိုင် binary tree မဟုတ်ပဲ တစ်ခုထက် မက node တွေကို child အဖြစ်ထည့်သွင်းသည့် class လည်း အခု အချိန်မှာ လွယ်လင့် တကူ ဖန်တီး နိုင်ပါပြီ။ ထို့ အတူ DFS , BFS ကို ထို Tree structure မှာ ပြန်လည် အသုံးချနိုင်ပါလိမ့်မယ်။

လေ့ကျင့်ခန်း ၉-၂



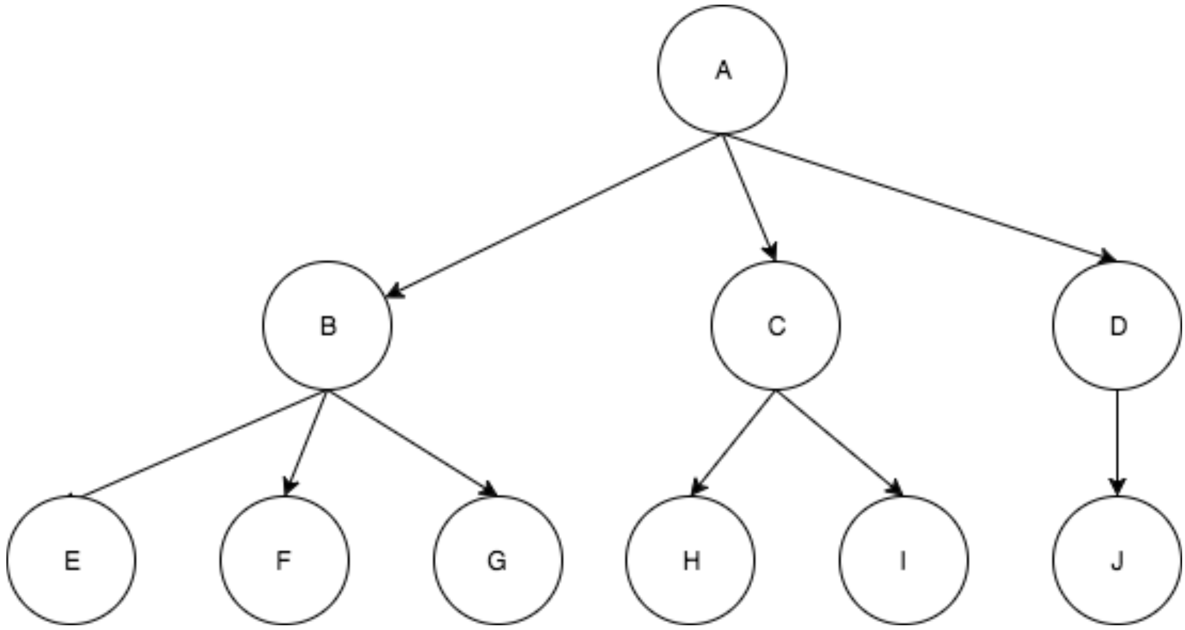
၁။ အထက်ပါ binary tree တစ်ခု တည်ဆောက်ပါ။ ထို binary tree အတွက် bfs ကို သုံးပြီး search function ကို ရေးပါ။ ဥပမာ ။ F လို့ ထည့်လိုက်လျှင် binary tree တွင် ပါဝင်သောကြောင့် true ဟု return ပြန်ပါမည်။ H ဟု ထည့်လိုက်လျှင် ရှာ မတွေ့သောကြောင့် false ဟု return ပြန်ရမည်။

Tree

ကျွန်တော်တို့ binary tree ကို ပြန်လည်ပြုပြင်ပြီးတော့ Tree တစ်ခု ဖန်တီးပါမယ်။ Binary Tree နဲ့ ကွာခြားတာကတော့ Tree မှာ children တွေက တစ်ခု ထက်မက ပါဝင်ပါတယ်။ Binary Tree မှာ left,right အစား child ပဲ ရှိပါတော့မယ်။

ကျွန်တော်တို့ အနေနဲ့ Binary Tree Class အစား Node class ကို ဖန်တီးပါမယ်။

Node class ထဲမှာတော့ child တွေကို ထည့်ဖို့ list ပါ ပါမယ်။ Node class ထဲမှာတော့ အရင်ကလို value ကို တိုက်ရိုက် မထည့်တော့ပဲ Node class ကိုသာ ထည့်သွင်းပါမယ်။



အထက်ပါ ပုံအတိုင်း code လေး ရေးကြည့်ပါမယ်။

```
class Node:
    def __init__(self,value):
        self.value = value
        self.child = []

    def __repr__(self):
        return "Value is " + self.value

    def insert_child(self,node):
        self.child.append(node)

    def get_child(self):
        return self.child


root = Node("A")
```

```
b = Node("B")
c = Node("C")
d = Node("D")

root.insert_child(b)
root.insert_child(c)
root.insert_child(d)


e = Node("E")
f = Node("F")
g = Node("G")


b.insert_child(e)
b.insert_child(f)
b.insert_child(g)

h = Node("H")
i = Node("I")

c.insert_child(h)
c.insert_child(i)

j = Node("J")

d.insert_child(j)

print(root.value)
print(root.child)
print(root.child[0].value)
print(root.child[0].child[0].value)
```

child တွေ အကုန်လုံးက list ဖြစ်သည့် အတွက် root.child[0].value ဆိုပြီး ခေါ်နိုင်ပါတယ်။ list ဖြစ်သည့်အတွက် child အရေအတွက် သိချင်ရင်တော့ len(root.child) ဖြင့် အသုံးပြုနိုင်ပါတယ်။

အခု BFS ကို အသုံးပြုကြည့်ရအောင်။

```
class Node:
    def __init__(self,value):
        self.value = value
        self.child = []

    def __repr__(self):
        return "Value is " + self.value

    def insert_child(self,node):
        self.child.append(node)

    def get_child(self):
        return self.child

    def bfs(self):
        thislevel = [self]
        while thislevel:
            nextlevel = []
            level = []
            for n in thislevel:
```

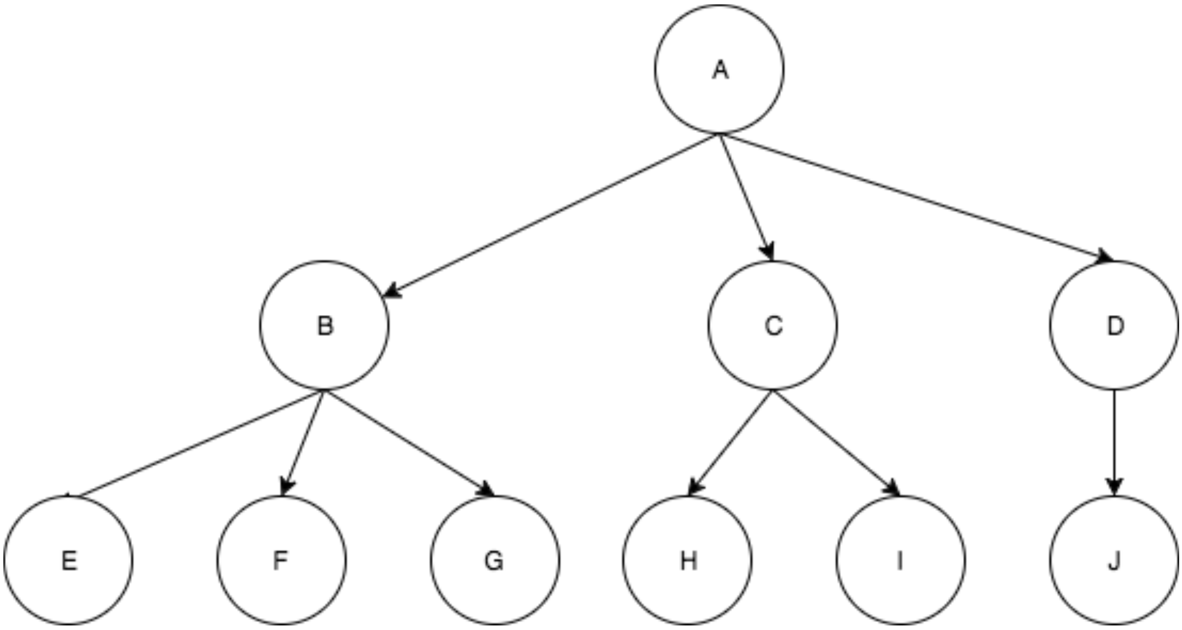
```
                level.append(n.value)
                if len(n.child) > 0:
                    nextlevel = nextlevel + n.child


            if len(level) > 0 :
                print(",".join(level))

            thislevel = nextlevel


root = Node("A")

b = Node("B")
c = Node("C")
d = Node("D")

root.insert_child(b)
root.insert_child(c)
root.insert_child(d)


e = Node("E")
f = Node("F")
g = Node("G")


b.insert_child(e)
b.insert_child(f)
b.insert_child(g)

h = Node("H")
i = Node("I")

c.insert_child(h)
c.insert_child(i)

j = Node("J")

d.insert_child(j)

root.bfs()
```

Node အောက်မှာ child တွေက array ဖြစ်သည့် အတွက် nextlevel array ကို child နှင့် merge လုပ်ဖို့ လိုပါတယ်။

```
nextlevel = nextlevel + n.child
```

python မှာ array ၂ ခုကို merge လုပ်ချင်ရင်တော့ `merge = array + array` ဆိုပြီး အသုံးပြု နိုင်ပါတယ်။ အထက်ပါ BFS ကို တဆင့်ဆီ ကိုယ်တိုင် trace လိုက်ကြည့်ဖို့ လိုပါတယ်။

လေ့ကျင့်ခန်း ၉-၃



၁။ အထက်ပါ Tree ကို DFS ဖြင့် ရေးသားပါ။ တစ်ခုထက်မက child တွေ ဖြစ်နိုင်သည့် အတွက် preoder နှင့် postorder သာ အသုံးပြုနိုင်ပါသည်။ ထို့ကြောင့် preoder နှင့် postorder function ရေးသားပါ။

အခန်း ၁၀ ။ Algorithm Analysis

အခုဆိုရင်တော့ စာအုပ်၏ နောက်ဆုံး အခန်းကို ရောက်လာပါပြီ။ အခု အခန်းထိ ရောက်လာသည့် အချိန်မှာတော့ program တစ်ခုကို ဘယ်လို ရေးရမလဲဆိုတာကိုတော့ စဉ်းစားတတ်နေပါပြီ။ အခု အခန်းမှာတော့ program တစ်ခု ရေးဖို့ထက် program တစ်ခု ကို analyst လုပ်ဖို့ အဓိက ပါဝင်ပါမယ်။ ကျွန်တော်တို့ တွေဟာ program တစ်ခုကို ရေးတတ်ရုံသာမကပဲ ဒီ program တစ်ခု ဟာ ဘယ်လောက်ကြာနိုင်မလဲ ဆိုတာကို သိဖို့ လိုပါတယ်။ ဥပမာ Array ဟာ အခန်း တွေများလာလေလေ array loop ပတ်တာ ကြာလေလေ ဖြစ်ပါလိမ့်မယ်။

What Is Algorithm Analysis?

ကျွန်တော်တို့တွေ အနေနဲ့ တစ်ယောက် နှင့် တစ်ယောက် program တွေကို နှိုင်းယှဉ်ကြပါတယ်။ ဘယ် program က ပိုကောင်းတယ် ပိုမြန်တယ်လို့ ယှဉ်တတ်ပါတယ်။ နှိုင်းယှဉ်သည့် အခါမှာ တော့ ကျွန်တော်တို့တွေဟာ n integer များ၏ ပေါင်းပြီး ရသည့် ရလဒ်ပေါ်မှာ မူတည်ပြီး ဆုံးဖြတ်ပါတယ်။

```
def sum_of_n(n):
    the_sum = 0
    for i in range(1,n+1):
        the_sum = the_sum + i
    return the_sum

print(sum_of_n(10))
```

အခု code လေးကို ကြည့်ရအောင်။ ပုံမှန် အားဖြင့် 1 ကနေ စပြီးတော့ n အရေအတွက် ထိ ပေါင်းတာပါ။ ဥပမာ 5 ဆိုရင် 1+2+3+4+5 ပေါ့။

ဒီ code ဘယ်လောက်ကြာလဲ သိရအောင် ကျွန်တော်တို့တွေ အနေနဲ့ function ပြီးသည့် အချိန်ထဲ က program စသည့် အချိန်ကို နှုတ်ကြည့်မှ သာ သိနိုင်ပါလိမ့်မယ်။

```
import time

def sum_of_n(n):
    start = time.time()
    the_sum = 0
    for i in range(1,n+1):
        the_sum = the_sum + i
    end = time.time()
    return the_sum,end-start

for i in range(5):
```

```
    print("Sum is %d , %.7f seconds" % sum_of_n(100000))
```

အထက်ပါ code မှာ အချိန်ကို တွက်ဖို့ အတွက်

```
import time
```

ဆိုပြီး ထည့်ထားပါတယ်။

```
return the_sum,end-start
```

ပြီးတော့ sum ရလဒ် နှင့် ကြာချိန် ကို return ပြန်ထားပေးပါတယ်။

```
for i in range(5):
    print("Sum is %d , %.7f seconds" % sum_of_n(100000))
```

မှာတော့ parameter ၂ ခု return ပြန်ပေးသည့် value ကို တစ်ခါတည်း ထုတ်ပြထားတာပါ။ %d ကတော့ integer value ဖြစ်ပြီးတော့ %.7f ကတော့ ဒသမ ကို ၇ နေရာထိ ယူမယ်လို့ ဆိုတာပါ။ % ကတော့ ပြန်လာသည့် tuple ကို print ထဲမှာ အစား ထိုးဖို့ပါ။ %d, %.7f ဖြစ်သည့် အတွက် tuple ထဲမှာ ရှိသည့် (sum,time) ၂ ခုမှာ sum က ပထမ ဖြစ်လို့ %d ထဲ ရောက်သွားပြီးတော့ time က ဒုတိယဖြစ်လို့ %.7f နေရာမှာ ဖော်ပြမှာပါ။

အခု အဲဒီ program နဲ့ နောက်ထပ် program တစ်ခု နှိုင်းယှဉ်ကြည့်ရအောင်။

```
1 + 2 + 3 + 4 +.....+ n
```

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

သည် `n * (n+1) / 2` နှင့် တူပါတယ်။

ဒါကြောင့် ကျွန်တော်တို့ ဟာ program loop မပတ်တော့ပဲ အောက်ကလို ပြင်ရေးပါမယ်။

```
import time

def sum_of_n(n):
    start = time.time()
    the_sum = n * (n+1) / 2
    end = time.time()
    return the_sum,end-start

for i in range(5):
    print("Sum is %d , %.7f seconds" % sum_of_n(100000))
```

အဖြေက အတူတူပါပဲ။ သို့ပေမယ့် processing time က သိသိသာသာ ကွာသွားတာကို တွေ့နိုင်ပါတယ်။ ဒီ ကြာချိန် ကွာခြား ချက်က ဘာကို ပြောပြ နေသလဲ ဆိုတော့ looping ဟာ များလာလေလေ အလုပ်လုပ်ရသည့် အချိန် ပိုကြာလေလေ ဖြစ်တာကို တွေ့နိုင်ပါတယ်။ ဒုတိယ program ဟာ looping မသုံးပဲနဲ့ သင်္ချာ equation ကို အသုံးပြုထားသည့် အတွက် အများကြီး ပိုမို မြန်ဆန်တာကို တွေ့နိုင်ပါတယ်။ Program တစ်ခုဟာ data များလာလေလေ နှေးလာလေလေ ဖြစ်နိုင်သလား ဆိုတာကို သိနိုင်ဖို့ အတွက် ကျွန်တော်တို့တွေဟာ Big O Notation ကို အသုံးပြုကြပါတယ်။

Big-O Notation

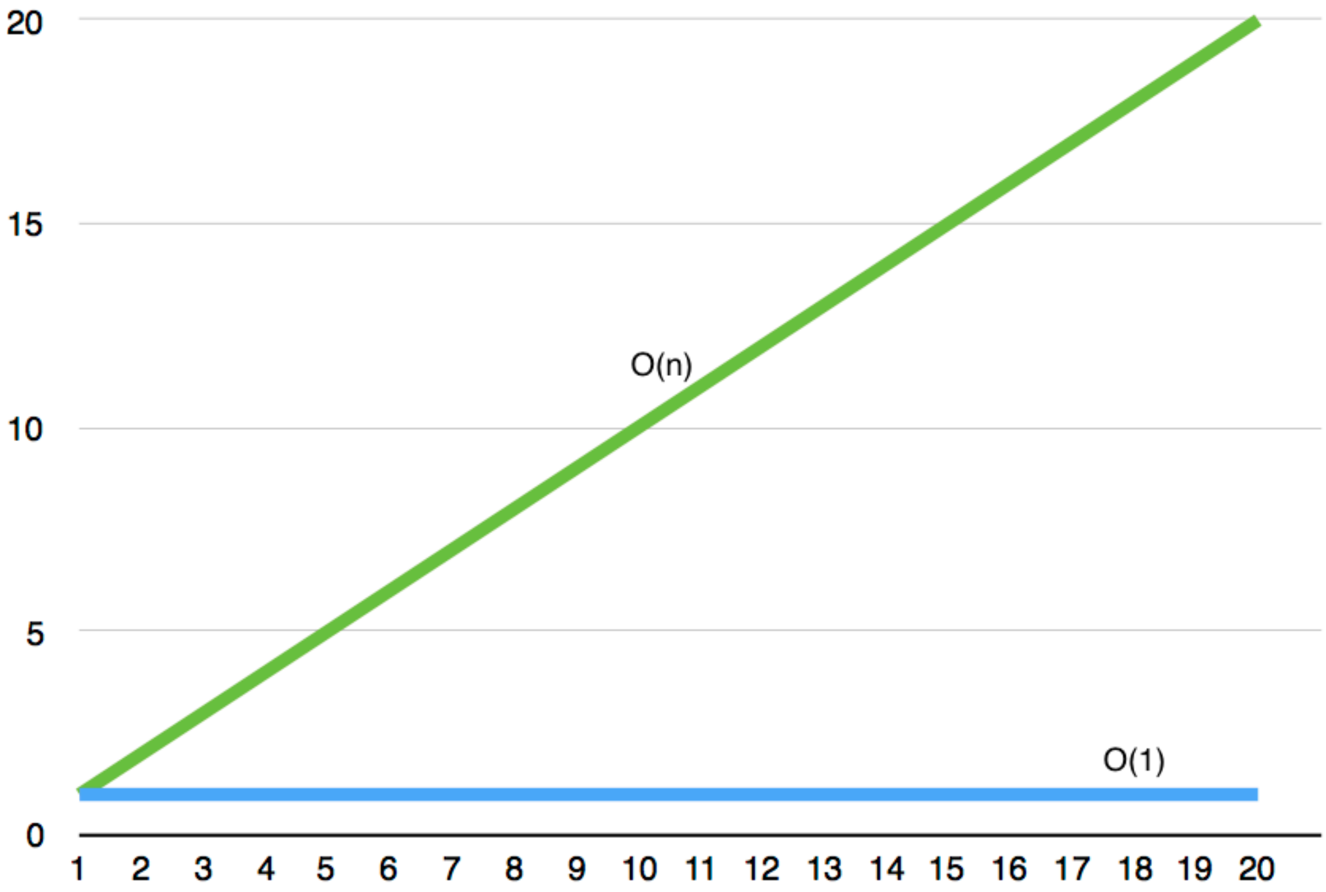
Big-O Notation ဟာ program တစ်ခုဟာ ဘယ်လောက် ထိ ထိရောက်မှုရှိလဲ။ Data များလာလေလေ ဘယ်လောက် ထိ အဆင်ပြေနိုင်မလဲ ဆိုတာကို ဖော်ပြပေးဖို့ အတွက် အသုံးပြုကြပါတယ်။ ပြီးခဲ့တဲ့ program ၂ ခုမှာ ဆိုရင်တော့ ပထမ program ဟာ n ရဲ့ size ပေါ်မှာ မူတည်ပြီးတော့ ကြာမြင့်ပါတယ်။

```
import time

def sum_of_n_loop(n):
    the_sum = 0
    for i in range(1,n+1):
        the_sum = the_sum + i

    return the_sum

def sum_of_n_eq(n):
    return n * (n+1) / 2

start = time.time()
for i in range(100000,100100):
    sum_of_n_loop(i)
end = time.time()

print("Time is %.7f second" % (end-start))

start = time.time()
for i in range(100000,100500):
    sum_of_n_eq(i)
end = time.time()

print("Time is %.7f second" % (end-start))
```

function ၂ ခု ကို ယှဉ်ကြည့်ရင် သိသိသာသာ ကွာခြားတာကို တွေ့နိုင်ပါတယ်။

ပထမ function ဟာ 1 ကနေ ပြီးတော့ n ထိ သွားပါတယ်။ တနည်းပြောရင် အကြိမ်အရေအတွက် n ထိ အလုပ်လုပ်ရတယ်။ array ၁၀၀ ရှိရင် အကြိမ် ၁၀၀ အလုပ်လုပ်ရတယ်။ ဒီတော့ ကျွန်တော်တို့O(n) လို့ သတ်မှတ်ပါမယ်။

ဒုတိယ function ကတော့ ၁ ကြိမ်သာ အလုပ်လုပ်တယ်။ n က 1000 ဖြစ်နေလည်း ၁ ကြိမ်သာ အလုပ်လုပ်တယ်။ အမြဲတန်း constant ပဲ။ n ရဲ့ တန်ဖိုးပေါ်လိုက်ပြီး ပေါင်းလဲမှုမရှိဘူး။ ဒါကြောင့် O(1) လို့ ဆိုပါတယ်။

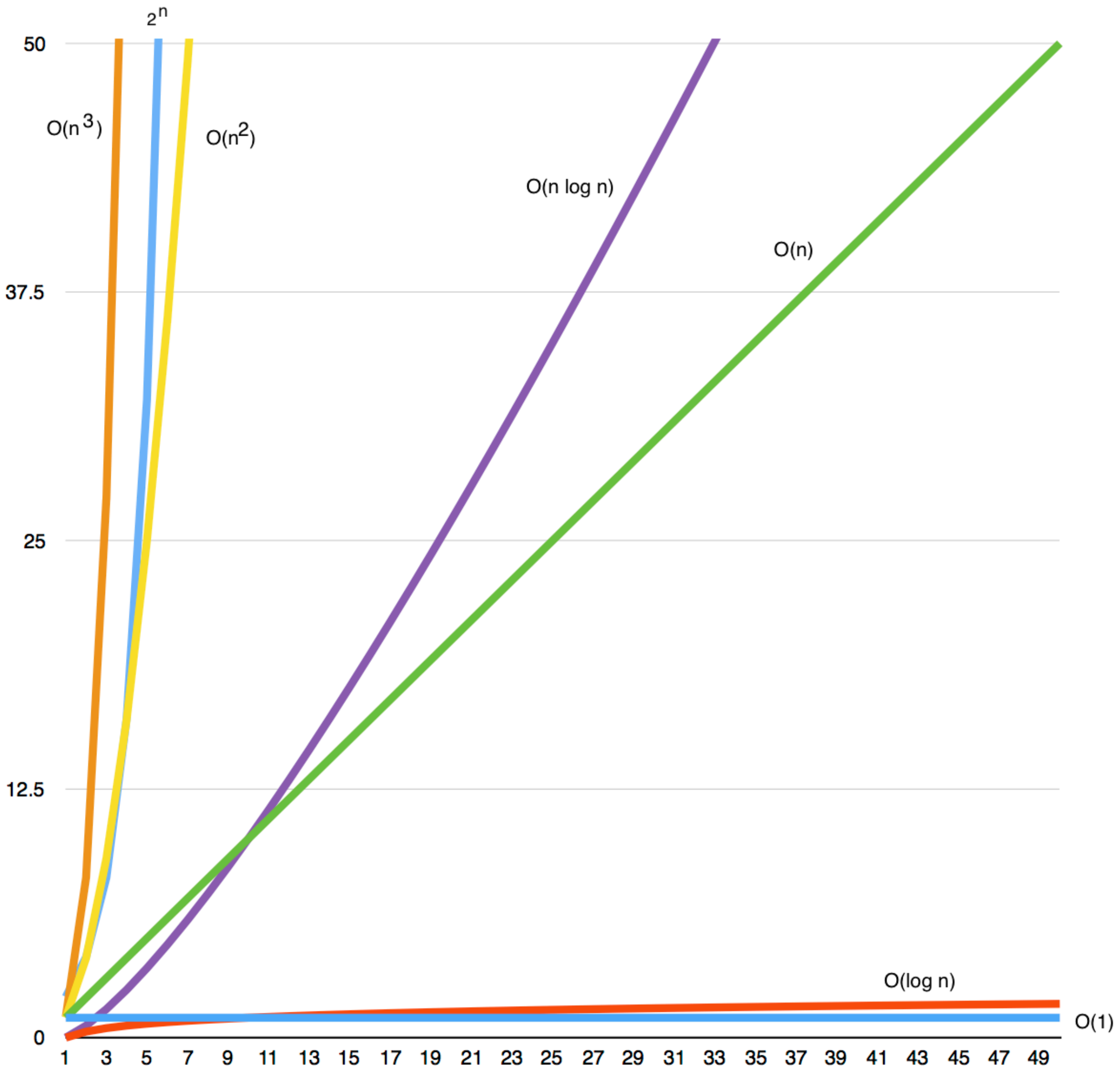


ကျွန်တော်တို့ အနေနဲ့ graph ဆွဲကြည့်လိုက်ရင် အထက်ပါ ပုံ အတိုင်း မြင်ရပါလိမ့်မယ်။

Big-O Notation မှာ အောက်ပါ function တွေ ရှိပါတယ်။

f(n)	**Name**
1	Constant
log(n)	Logarithmic
n	Linear
n log(n)	Log Linear
n ^ 2	Quadratic
n ^ 3	Cubic
2 ^ n	Exponential

graph အနေနဲ့ ကြည့်မယ် ဆိုရင် အောက်ပါ ပုံအတိုင်း တွေ့နိုင်ပါတယ်။



1 နှင့် log(n) က အကောင်းဆုံး algorithm တွေပါ။ n ကတော့ ပုံမှန် ပေါ့။ n log(n) ဟာ အစ ပိုင်း မှာ n ထက် မြန်နိုင်ပေမယ့် data များလာရင် နှေးလာပါလိမ့်မယ်။ n ^ 2 နှင့် 2 ^ n ဟာ အစပိုင်းမှာ မကွာပေမယ့် နောက်ပိုင်း data များလာလေလေ ကွာလာလေလေ ကို တွေ့နိုင်ပါတယ်။ n ^ 3 ကတော့ အနှေးဆုံးလို့ ဆိုရပါမယ်။

Function

Big O မှာ ဘယ် function တွေ ဟာ ဘာအတွက်လဲဆိုတာကို လေ့လာရအောင်။

Constant

Constant ကတော့ ရှင်းပါတယ်။

```
a = b + 1
```

looping တွေ ပါဝင်မနေပါဘူး။ processing ကို တစ်ကြောင်းတည်းနှင့် အလုပ်လုပ်ပါတယ်။

Logarithmic

array ကို တဝက်ပိုင်းပြီး loop ပတ်သည့် algorithm တွေကို log(n) နှင့် သတ်မှတ်ပါတယ်။ ဥပမာ binary search ပါ။

```
while (n > 1):
    n = n // 2
```

Linear

Looping တစ်ခုတည်းပါရင်တော့ linear ပါ။

```
for i in range(len(array)):
    print(i)
```

ဒါမျိုးဟာ `O(n)` ဖြစ်ပြီးတော့ linear ဖြစ်ပါတယ်။

Log Linear

log linear ဟာ merge sort, quick sort လိုမျိုး sorting တွေမှာ တွေ့ရပါမယ်။ Array အခန်းအား တစ်ဝက်ပိုင်းသည့် အခါတွင် Log Liner ပါ။

```
x = n
while ( x > 0 ) {
    y = n
    while ( y > 0 ) {
        y = y / 2
    }
    x = x - 1
}
```

```
x = n
while ( x > 0 ) {
    y = n
    while ( y > 0 ) {
        y = y - 1
    }
    x = x / 2
}
```

Quadratic

ဒါကတော့ looping ၂ ထပ် အတွက်ပါ။

```
for i in range(len(array)):
    for k in range(len(array)):
        print(k)
```

looping ၂ ထပ် ကိစ္စတွေဟာ `O(n^2)` နှင့် တူညီပါတယ်။

Cubic

ဒါကတော့ looping ၃ ထပ် ကိစ္စတွေပေါ့။

```
for i in range(len(array)):
    for k in range(len(array)):
        for w in range(len(array)):
            print(i+k+w)
```

Exponential

ဒါကတော့ တွေ့ရတာ ရှားပါတယ်။ password တွေကို ဖြစ်နိုင်သည့် combinations တွေ ပေါင်းပြီး generate လုပ်သည့် algorithm တွေမှာ တွေ့ရတတ်ပါတယ်။

Big-O Notiation ကို ဘယ်လို တွက်မလဲ

အခု ကျွန်တော်တို့ Big-O Notiation အကြောင်း အနည်းငယ် သိပါပြီ။ ကျွန်တော်တို့ အနေနဲ့ ဘယ်လို တွက်ရမလဲ ဆိုတာကို သိဖို့လိုပါတယ်။

1. Different steps get added

အကယ်၍ algorithm မှာ မတူညီသည့် အဆင့်တွေ ပါလာခဲ့ရင် Big O ကို ပေါင်းပေးရပါတယ်။

```
doStep1() #O(a)
doStep2() #O(b)
```

အဲဒါဆိုရင် `O(a+b)` ဖြစ်ပါတယ်။

2. Drop constant

Big O မှာ constant တန်ဖိုးတွေပါဝင်ခဲ့ရင် ဖြုတ်လိုက်ဖို့ လိုက်ပါတယ်။

```
def minmax1(array):
    min = 0
    max = 0
    for k in array:
        min = MIN(k,min)
    for k in array:
        max = MAX(k,max)
def minmax2(array):
    min = 0
    max = 0
    for k in array:
        min = MIN(k,min)
        max = MAX(k,max)
```

ဒီ function ၂ ခုကို ယှဉ်လိုက်ရင် ပထမ function ဟာ O(n+n) နှင့် ဒုတိယကတော့ O(n) လို့ ဆိုနိုင်ပါတယ်။ O(n+n) = O(2n) ဖြစ်ပါတယ်။ သို့ပေမယ့် Big O Nototation တွက်သည့် အခါမှာ constant တန်ဖိုးတွေကို ဖြုတ်ချခဲ့ရပါတယ်။ ဒါကြောင့် program ၂ ခုလုံးဟာ O(n) လို့ပဲ သတ်မှတ်ပါတယ်။

3. Different Input, different variable

```
for c in array1:
    for h in array2:
        x = x + 1
```

ဒီ code လေးကို ကြည့်လိုက်ရင် array ရှိသလောက်သွားတယ်။ looping ၂ ခု ဆိုတော့ n ^ 2 ဖြစ်မယ်လို့ထင်စရာ ဖြစ်ပါတယ်။ တကယ်တန်းတော့ O(a*b) ပါ။ a ကတော့ array1 ရဲ့ size ဖြစ်ပြီး b ကတော့ array2 ရဲ့ size ပါ။ အကယ်၍ variable တူခဲ့ရင်တော့ n ^ 2 ဖြစ်ပါမယ်။

```
for c in array1:
    for h in array1:
        x = x + 1
```

ဒီ code ဆိုရင် looping ရဲ့ variable တူပါတယ်။ ဒီ array size ကိုပဲ ၂ ထပ် ပတ်ရတာကို တွေ့နိုင်ပါတယ်။ ဒါကြောင့် (n*n) ဖြစ်သည့်အတွက်ကြောင့် O(n^2) ဖြစ်ပါတယ်။

4. Drop non-dominate terms

အကယ်၍ n တွေဟာ တစ်ခု ထက်မက ပါခဲ့ရင် တန်ဖိုး တစ်ခုကိုပဲ ယူပါတယ်။ ဥပမာ။

```
min = 0
for c in array1:
    min = MIN(c,min)

for c in array1:
    for h in array1:
```

```
        print(c,h)
```

ဒီ code မှာ ပထမ loop က O(n) ဖြစ်ပါတယ်။ ဒုတိယ loop ကတော့ O(n^2) ဖြစ်ပါတယ်။ ဒီတော့ ၂ ခုပေါင်းတော့ O(n+n^2) ရပါတယ်။

Big O notation ဟာ upper bound ဖြစ်သည့် အတွက် n နှင့် n^2 မှာ တန်ဖိုး ပိုကြီးသည့် n^2 ကို သာယူပါတယ်။ ဒါကြောင့် program ရဲ့ Big O Notation ဟာ O(n^2) ဖြစ်ပါတယ်။

Array Sorting Algorithm

ကျွန်တော်တို့ ပြီးခဲ့တဲ့ အခန်းတွေမှာ array ကို sorting လုပ်ခဲ့ပါတယ်။ Array sorting Big O Notation ကို အောက်ပါ ဇယားမှာ တွေ့နိုင်ပါတယ်။

Name	Big O Notation
Bubble Sort	O(n^2)
Selection Sort	O(n^2)
Insertion Sort	O(n^2)
Shell Sort	`O(n*log(n))` သည် အကောင်းဆုံး ရနိုင်ခြေဖြစ်ပြီး `O(n ^ 1.25)` သည် ဖြစ်နိုင်ခြေရှိသည့် ပျမ်းမျှ တန်ဖိုးဖြစ်သည်
Merge Sort	O(n log(n))
Quick Sort	O(n log(n))

အခု sorting algorithm အချို့ကို Big O နဲ့ ထုတ်ကြည့်ရအောင်။

Bubble Sort

Bubble sort algorithm ကို ပြန်ကြည့်ရအောင်။

```
def bubble_sort(array):
    for num in range(len(array) - 1,0,-1):
        for i in range(num):
            if array[i] > array[i+1]:
                temp = array[i]
                array[i] = array[i+1]
                array[i+1] = temp
    return array
```

Bubble sort ဟာ array ကို ပထမ အကြိမ်မှာ array အခန်း တစ်ခု လျှော့ပတ်တယ်။ ဒုတိယ အကြိမ် ၂ ခုလျှော့ပတ်တယ်။ နောက်ဆုံး 0 ရောက်သည့် အထိ loop ပတ်တယ်။ တနည်းပြောရ ရင်

```
(n-1) + (n-2) + ... + 1 + 0
```

လို့ဆိုနိုင်ပါတယ်။ အဲဒါဟာ n(n-1)/2 နှင့် တူပါတယ်။ တနည်းဆိုရင် 1/2(n^2-n) နှင့် တူတယ်လို့ ဆိုနိုင်ပါတယ်။ ဒါကြောင့် O(1/2(n^2-n)) ဖြစ်ပါတယ်။ constant ဖြုတ်ချ ဖို့လိုသည့် အတွက် O(n^2-n) ဖြစ်ပါတယ်။ n^2 က n ထက် ပိုကြီးသည့်အတွက် bubble sort ဟာ O(n^2) ဖြစ်ပါ တယ်။

Merge Sort

Merge sort ဟာ အခြား sorting algorithm တွေထက် ပိုမြန်ပါတယ်။ သူက O(nlogn) ဖြစ်သည့် အတွက်ကြောင့်ပါ။ Merge sort algorithm ကို ပြန်ကြည့် ရအောင်။

```
def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0
    while left_idx < len(left) and right_idx < len(right):

        if left[left_idx] <= right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    if left_idx < len(left):
        result.extend(left[left_idx:])
    if right_idx < len(right):
        result.extend(right[right_idx:])
    return result

def mergesort(w):

    if len(w)<2:
        return w
    else:
        mid=len(w)//2
        return merge(mergesort(w[:mid]), mergesort(w[mid:]))
```

recursive လုပ်ထားတယ်။ တနည်းအားဖြင့် array size အတိုင်း ပထမ အဆင့် loop ပတ်နေတာကို တွေ့နိုင်ပါတယ်။ သို့ပေမယ့် ဒုတိယ loop မှာ array size ကို တဝက်ချိုးလိုက်တာကို တွေ့ရပါလိမ့် မယ်။ array size တစ်ဝက် ချိုးလိုက်သည်များကို `log(n)` ဟု ဆိုခဲ့ပါတယ်။ `log(n)` တွေဟာ array size အကြိမ် အရေ အတွက် အလုပ်လုပ်ရပါတယ်။ ဒါကြောင့် `n log(n)` ဖြစ်ပါတယ်။ Big O အရ ဆိုရင် `O(n log(n))` ဖြစ်ပါတယ်။

အခုဆိုရင်တော့ Big O notation အကြောင်း အနည်းငယ် တီးမိ ခေါက်မိပါပြီ။ ကျွန်တော် အခု ဖော်ပြထားသည်မှာ Big O notation ၏ အကြောင်းအရာ အနည်းငယ်မျှ သာ ဖြစ်ပါတယ်။

အခန်း ၁၁ ။ Regular Expression

Programming မှာ အခြေခံအား ဖြင့် regular expression ကို မဖြစ်မနေ သိထားသင့်ပါတယ်။ Regular Expression ကို Regex လို့ အတိုခေါက် လည်း ခေါ်ကြပါတယ်။ Regular expression ဟာ validation စစ်သည့် နေရာတွေမှာ အသုံးပြုနိုင်သလို search , replace နေရာတွေမှာလည်း အသုံးပြုနိုင်ပါတယ်။ Regular expression ဟာ search pattern ကို အသုံးပြုပြီး စစ်ဆေးခြင်း နှင့် ရှာဖွေ ခြင်း တို့ကို လုပ်ဆောင်နိုင်ပါတယ်။

Regular Expression အယူအဆကို ၁၉၅၀ ဝန်းကျင်မှာ အမေရိကန် သင်္ချာပညာရှင် Stephen Cole Kleene က regular language ဆိုပြီး စတင်ခဲ့ပါတယ်။ unix ရဲ့ text-processing utilities မှာ အများအားဖြင့် အသုံးပြုခဲ့ကြပါတယ်။ 1980 ဝန်းကျင်မှာတော့ မတူညီသည့် syntax တွေ နဲ့ regular expressions ရှိခဲ့ပြီး POSIX standard နှင့် လူသုံးများသည့် Perl syntax တွေ ရှိလာခဲ့ပါတယ်။ 1997 မှာတော့ Philip Hazel က PCRE (Perl Compatible Regular Expressions) ကို developd လုပ်ခဲ့ပါတယ်။ PCRE ကို အခုခေတ် နေရာ အတော်များများ မှာ အသုံးပြုကြပါတယ်။

Patterns

Regular Expression ကို ရေးသားဖို့ အတွက် အရင်ဆုံး Patterns သဘောတရားကို သိဖို့ လိုပါတယ်။ Pattern ဆိုဟာ အထပ်ထပ်အခါခါ ဖြစ်ပေါ်နေသည့် အကြောင်းအရာ တွေကို အကျဉ်းချုပ် ပုံစံ ဖန်တီးတယ်လို့ ဆိုရပါမယ်။

ဥပမာ။ အင်္ဂလိပ်စာမှာ နာမည်ဆိုရင် A ကနေ Z စာလုံးကြီးနဲ့ စတယ်။ ကြားမှာကတော့ a ကနေ z အထိ စာလုံးများစွာ ဖြစ်နိုင်တယ်။ space သို့မဟုတ် . နဲ့ ဆုံးတယ်။ အဲလို အတွက် pattern ကို `[A-Z][a-z]*(\s|\.)` ဆိုပြီး ရေးနိုင်ပါတယ်။

အကြိမ်အရေအတွက် zero or more ဖြစ်သည့် အတွက် ကြောင့် * ကို သုံးပါတယ်။ space သို့မဟုတ် `.` ဆိုသည့် အတွက်ကြောင့် `(\s|\.)` ကို အသုံးပြုထားပါတယ်။

Boolean or

apple သို့မဟုတ် orange ဆိုသည့် ကိစ္စများအတွက် vertical bar ကို အသုံးပြုပါတယ်။ ဥပမာ `apple|orange` ဟု ရေးပါသည်။

Grouping

Group ဖွဲ့ချင်ရင် လက်သည်းကွင်း နှင့် အသုံးပြုပါတယ်။ `gray | grey` ဟု ရေးမည့် အစား `gr(a|e)y` ဟု group ဖွဲ့ပြီး ရေးသားနိုင်ပါသည်။

A ကနေ Z အထိ ကဲ့သို့ range လိုမျိုးကို [A-Z] ဟုရေးနိုင်တယ်။ A ကနေ Z , a ကနေ z , 0 ကနေ 9 ကဲ့သို့ range မျိုးအတွက် အကုန်ပေါင်းရေးနိုင်သည်။ [A-Za-z0-9] ။ စာလုံးသည် [A-Za-z0-9] ထဲတွင် ရှိသည် ဟု ဆိုလိုခြင်း ဖြစ်သည်။ ထို့အပြင် အခြား စာလုံးများလည်း ပေါင်းရေး နိုင်သည်။ [A-Za-z0-9#%] ဟု ဆိုလျှင် A-Z , a-z , 0-9 အပြင် #% စာလုံးများလည်း ဖြစ် နိုင်သည် ဟု ဆိုလိုပါသည်။

Quantification

Quantifier တွေ ကို အသုံးပြုရင် တစ်ခုသည်း အသုံးပြုလို့ မရပါဘူး။ character , group စသည့် တစ်ခုခု နဲ့ တွဲပြီး အသုံးပြုရပါတယ်။ အသုံးပြုလို့ ရသည့် quantifier တွေကတော့ အောက်ပါ အတိုင်းဖြစ်ပါသည်။

Expression	Description
?	question mark ကတော့ zero or one အခြေအနေ ကို ရေးသားခြင်းဖြစ်ပါသည်။
*	zero or more အခြေအနေ ကို ရေးသားရာတွင် အသုံးပြုသည်။
+	one or more အခြေအနေကို ရေးသားရာတွင် အသုံးပြုသည်။
^ **`	စာကြောင်း၏ အစ စာလုံး ကို သတ်မှတ် ရာတွင် အသုံးပြုသည်။
$	စာကြောင်း၏ အဆုံး စာလုံးကို သတ်မှတ်ရာတွင် အသုံးပြုသည်။
[^]	range ထဲက စာလုံးများ မဟုတ်သည့် စာလုံးများ အတွက် အသုံးပြုသည်။
{n}	စာလုံး ပါဝင်မည့် အရေအတွက် ကို စစ်ဆေးရန် အတွက် အသုံးပြုသည်။
{min,}	အနည်းဆုံး ပါဝင်ရမည့် စာလုံး အရေအတွက် အသုံးပြုသည်။
{min,max}	အနည်းဆုံး နှင့် အများဆုံး ပါဝင်ရမည့် စာလုံး အရေအတွက် စစ်ရာတွင် အသုံးပြုသည်။

Wildcard

Expression	Description
.	မည်သည့် character အတွက် မဆို အသုံးပြုလိုလျှင် dot (.) လေးကို အသုံးပြုနိုင်သည်။
\d	ကိန်းဂဏန်း အားလုံးအတွက် \d ကိုအသုံးပြုနိုင်သည်။ [0-9] နှင့် အတူတူ ဖြစ်ပါသည်။
\D	ဂဏန်း မဟုတ်သော် စာလုံးများကို စစ်ဆေးရာတွင် အသုံးပြုသည်။ [^0-9] နှင့် အတူတူဖြစ်သည်။
\w	word ထဲတွင် ပါသော character များအတွက်အသုံးပြုသည်။ [A-Za-z0-9_] နှင့် တူညီသည်။
\W	word ထဲတွင် မပါသော character များ အတွက် ဖြစ်သည်။ [^A-Za-z0-9_] နှင့် တူညီသည်။
\s	white space character နှင့် တူညီ မှု အတွက် အသုံးပြုသည်။ [\r\n\t\f\v] နှင့် တူညီ သည်။
\S	white space character မဟုတ်ခြင်း အတွက် အသုံးပြုသည်။ [^\r\n\t\f\v] နှင့် တူညီသည်။

အသုံးပြုပုံ

ကျွန်တော်တို့ ဖုန်းနံပတ်စစ်သည့် regular expression လေးရေးကြည့်ရအောင်။ Telenor ဖုန်းဟုတ် မဟုတ် စစ်ဖို့အတွက် rule က +959 သို့မဟုတ် 959 သို့မဟုတ် 09 သို့မဟုတ် 9 နဲ့ စမယ်။ နောက်မှာ 79 သို့မဟုတ် 78 သို့မဟုတ် 77 ဖြစ်ပါမယ်။ အနောက်မှာ 0 ကနေ 9 ထိ ဂဏန်း ၇ လုံး ပါမယ်။ အဲဒီ အတွက် ကျွန်တော်တို့တွေ အောက်မှာ ဖော်ပြထားသည့် အတိုင်း ရေးပါမယ်။

```
(\+?95|0?9)7(9|8|7)\d{7}$
```

က one ore more အခြေအနေ ကို စစ်ခြင်း နှင့် ရောနေသည့် အတွက်ကြောင့် `backslash (\)` ကို အသုံးပြုထားပါတယ်။ `\+` ဆိုသည်မှာ `+` သည် စာလုံးဖြစ်ခြင်း ဟု ဖော်ပြထားခြင်း ဖြစ်သည်။ `+` နှင့် 0 သည် zero or one ဖြစ်နိုင်သည့် အခြေအနေ အတွက် `?` နှင့် သုံးထားသည် ကို တွေ့နိုင်ပါသည်။ `$` က တော့ စာလုံး အဆုံးကို ဆိုလိုပါတယ်။ ဂဏန်း နဲ့ မဆုံး ပဲ အခြား character နှင့် ဆုံးခဲ့ရင် မဟုတ်တော့ပါဘူး။

- `\d` သည် digit ကို ဆိုလိုခြင်း ဖြစ်သည် ။ digit သည် 0 မှ 9 အထိ စာလုံးများ ပါဝင်သည်။
- `\d{7}` သည် digital ၇ လုံး ဖြစ်ရမည် ဟု ဆိုလိုခြင်း ဖြစ်သည်။
- `\d{1,7}` ဟု ရေးလျှင် အနည်းဆုံး digit ၁ လုံးမှ ၇ လုံး ထိ ပါရမည် ဆိုလိုခြင်းဖြစ်သည်။
- `\d{1,}` ဟု ရေးလျှင် အနည်းဆုံး digit ၁ လုံးပါရမည် ဟု ဆိုလိုသည်။

Match

အထက်ပါ regular expression ကို python နှင့် တွဲပြီး အသုံးပြုကြည့်ပါမယ်။

```python
import re

telenor = re.compile("(\+?95|0?9)7(9|8|7)\d{7}$")
if telenor.match("09791234567") == None :
    print("Not telenor")
else:
    print("telenor")

if telenor.match("09971234567") == None :
    print("Not telenor")
else:
    print("telenor")
```

regular expression အသုံးပြုမယ် ဆိုရင် `import re` ကို ထည့်ဖို့ လိုပါတယ်။ `re.compile` မှာ တော့ ကျွန်တော်တို့ pattern ကို ထည့်ဖို့ လိုပါတယ်။ match လုပ်သည့်အခါမှာတော့ pattern နှင့် မ ကိုက်လျှင် None return ပြန်ပြီး pattern နှင့် ကိုက်ညီလျှင် match object return ပြန်ပါတယ်။ match object ကတော့ စာကြောင်းရဲ့ ဘယ်နေရာမှာ match ဖြစ်နေလဲ ဆိုတာကို ပြောပြပေးပါတယ်။

အခု နောက်ထပ် တစ်ခု စမ်းကြည့်ရအောင်။ ကျွန်တော်တို့ ထည့်ပေးထားသည့် စာလုံးထဲမှာ HTML စာလုံးဖြစ်သည့် bold ပါမပါ စစ်ကြည့်ပါမယ်။

```
import re

boldregex = re.compile(".*<b>(.*)</b>.*")
text = "Hello <b>World</b>"
match = boldregex.match(text)
print(match)
```

pattern ပုံစံကို လေ့လာကြည့်ရအောင်။

```
.*<b>(.*)</b>.*
```

လို့ရေးသားထားပါတယ်။ စာသားအစက ကြိုက်တာ ဖြစ်နိုင်ပြီးတော့ အဆုံးကလည်း ကြိုက်တာ ဖြစ်နိုင်ပါတယ်။ `<b>` နှင့် `</b>` ကြားမှာတော့ စာလုံးကို group ဖွဲ့ထားပါတယ်။ match ကို print ထုတ်ကြည့်သည့် အခါမှာတော့ match object ကို ထုတ်ပြတာ တွေ့နိုင်ပါတယ်။

```
<_sre.SRE_Match object; span=(0, 18), match='Hello <b>World</b>'>
```

ကျွန်တော်တို့ အခု match object ကနေ ပြီး group ဖွဲ့ထားသည့် စာလုံးကို ဆွဲထုတ်ပါမယ်။ Group ဖွဲ့ထားသည့် ပုံအရ ဆိုရင်တော့ World ဆိုသည့် စာလုံး ဖော်ပြပေးဖို့ လိုပါတယ်။ ပထမဆုံး match ဖြစ်သည့် group ကို ထုတ်ကြည့်ရအောင်။

```
print(match.group(0))
print(match.group(1))
```

```
Hello <b>World</b>
World
```

ဆိုပြီး ထွက်လာပါမယ်။

findall

match က ကျွန်တော်တို့တွေ အပြည့်အစုံ အတိအကျ မှန်မှသာ အသုံးပြုပါတယ်။ စာလုံးကို ရှာချင်ရင်တော့ ကျွန်တော်တို့တွေ findall ကို အသုံးပြုနိုင်ပါတယ်။

```
import re

boldregex = re.compile("<b>(.*)</b>")
```

```
text = "Hello <b>World</b>"
match = boldregex.findall(text)
print(match)
```

အထက်ပါ code မှာ ဆိုရင် result က World ဆိုပြီး array return ပြန်ပါမယ်။ ကျွန်တော်တို့ text ကို နည်းနည်း ထပ်ပြင်ပြီး စမ်းကြည့်ပါမယ်။

```
import re

boldregex = re.compile("<b>(.*)</b>")
text = "Hello <b>World</b>! This is <b>bold</b>"
match = boldregex.findall(text)
print(match)
```

ဆိုရင်တော့ result က

```
['World</b>! This is <b>bold']
```

ဆိုပြီး ဖြစ်နေတယ်။ `<b>` နဲ့ စပြီး `</b>` ဆိုရင် ဖြစ်သည့် အတွက်ကြောင့် အခုလို ပြန်နေပါတယ်။ regular expression pattern ကို ပြန်ပြင်ပါမယ်။

```
import re

boldregex = re.compile("<b>(.*?)</b>")
text = "Hello <b>World</b>! This is <b>bold</b>"
match = boldregex.findall(text)
print(match)
```

ဆိုရင် result က

```
['World', 'bold']
```

ကျွန်တော်တို့ လိုချင်သည့် အဖြေရပါပြီ။ code မှာ `.*` အစား `.*?` လို့ ပြင်လိုက်ပါတယ်။ `*?` က အကုန်လုံးကို match လုပ်မယ်။ ဘယ်အထိလည်း ဆိုတော့ အနောက်က `< /b>` မပါလာသေးသည့် အထိ လို့ ဆိုလိုပါတယ်။ `.*` ဆိုရင် နောက်ဆုံး `</b>` ဖြစ်သည့် အထိပါ။ ဒါကြောင့် ကြားမှာ `</b>` ပါဝင် နေပါတယ်။

အခုဆိုရင် match နှင့် findall ကို နားလည်မယ် ထင်ပါတယ်။ match ကတော့ အတိကျမှန်သည့် စာကြောင်း အတွက် အသုံးပြုပါတယ်။ findall ကတော့ ပေးလိုက်သည့် pattern နှင့် တူညီသည့် စာများကို ရှာပေးပါတယ်။

အခု search နှင့် replace ကို လေ့လာပါမယ်။ ကျွန်တော်တို့တွေ Markdown ကနေ HTML ပြောင်းသည့် code လေးရေးပါမယ်။ အရင်ဆုံး Markdown code ကို အနည်းငယ် ရှင်းပြပါမယ်။

Markdown Parser

Markdown ဆိုတာကတော့ developer တွေ HTML အစား အသုံးများသည့် format တစ်ခုပါ။ Github မှာ README.md ဆိုပြီး README file ကို markdown နှင့် ရေးသားပါတယ်။ HTML ထက် ရှင်းလင်းပြီး ရေးသားရာမှာ မြန်ဆန် ပါတယ်။ HTML ကို လွယ်လင့် တကူ ပြောင်းပေးနိုင်ပါတယ်။

Header အတွက် ဆိုရင် # နဲ့ စပါတယ်။

```
# H1
## H2
### H3
#### H4
```

အထက်ပါ စာ တွေက အောက်ပါ HTML နှင့် ညီပါတယ်။

```
<h1>H1</h1>
<h2>H2</h2>
<h3>H3</h3>
<h4>H4</h4>
```

bold အတွက်ဆိုရင်တော့

```
**bold**
```

က

```
<b>bold</b>
```

နှင့် ညီပါတယ်။ Italic အတွက်

```
_italic_
```

က

```
<i>italic</i>
```

နှင့် ညီပါတယ်။

code အတွက်ဆိုရင်တော့

\ \ \`js

var k = 10;

\ \ \`

က

```
<pre>
<code class="js">
var k = 10;
</code>
</pre>
```

အခု code တွေက Markdown မှာ အသုံးပြုသည့် code အချို့ပါ။ ကျွန်တော်တို့တွေ ဒီစာအုပ်မှာ တော့ ဥပမာ အနေနဲ့ အကုန်လုံးကို မရေးပြပဲ အထက်ပါ markdown code အချို့ကို html ပြောင်း သည့် parser လေး ရေးပါမယ်။

ပထမဆုံး Header ကို ရေးကြည့်ရအောင်။

`#` H1 ဖြစ်သည့် အတွက်ကြောင့် `#` နှင့် စမယ် space လာမယ် ပြီးရင် ကြိုက်သည့်စာဖြစ်နိုင် တယ်။ ဒါဆိုရင် <h1>H1</h1> အနေဖြင့် ပြောင်းပေးဖို့ လိုပါတယ်။ သတိထားသည့်တာကတော့ စာကြောင်း အစ က # ဖြစ်ဖို့ လိုပါတယ်။ ဒါကြောင့် ^ ကို သုံးဖို့ လိုအပ်ပါတယ်။ အဲဒီအတွက် pattern က

```
^#\s(.*)
```

ပြောင်းမယ့် ပုံစံကတော့

```
<h1>\1<\h1>
```

`\1` ဆိုတာကတော့ group 1 ကို ရည်ညွှန်းပါတယ်။ group တွေကတော့ လက်သည်းကွင်း နဲ့ သတ်မှတ်ပါတယ်။

Python code နဲ့ ပြောင်းရေးကြည့်ပါမယ်။

```
import re

parser = re.compile("^#\s(.*)")
text = "# Header 1"
replacement = parser.sub("<h1>\\1</h1>",text)
print(replacement)
```

python မှာ `\` ကို ဖော်ပြဖို့ အတွက် `\\` ဆိုပြီး အသုံးပြုဖို့ လိုပါတယ်။ ဒီထက် ပိုပြီး ရိုးရှင်းသည့် ပုံစံ ပြောင်းရေးပြပါမယ်။

```
import re

text = "# Header 1"
replacement = re.sub("^#\s(.*)","<h1>\\1</h1>",text)
print(replacement)
```

`re.sub` ဖြင့် တိုက်ရိုက်လည်း အသုံးပြုနိုင်ပါတယ်။ ဒါဆိုရင်တော့ h2 ကနေ h4 ထိ ရေးနိုင်မယ်လို့ ယုံကြည်ပါတယ်။

အခု bold နှင့် italic ကို ရေးကြည့်ရအောင်။ bold နှင့် italic က ပုံစံ အတူတူပါပဲ။ ကွာခြားချက်ကတော့ **နှင့် _ ပဲ ကွာပါတယ်။ pattern က လည်း ဆင်ပါတယ်။** လာမယ်။ ပြီးရင် ကြိုက်သည့် string ဖြစ်နိုင်တယ်။ ပြီးလျှင် **နှင့် ပြီးမယ်။ သတိထားသင့်တာကတော့** လာသည် နှင့် ပြီးဖို့ လိုပါတယ်။

Bold အတွက်

```
\*\*(.*?)\*\*
```

Italic အတွက်

```
_(.*?)_
```

Bold အတွက် code က

```
import re

text = "This is **bold**. This is another **bold again**."
replacement = re.sub("\*\*(.*?)\*\*","<b>\\1</b>",text)
print(replacement)

Italic အတွက် code က

import re

text = "This is _italic_. This is another _italic again_."
replacement = re.sub("_(.*?)_","<i>\\1</i>",text)
print(replacement)
```

အခု အခါမှာတော့ regex က နားလည် သည့် အခါမှာ လွယ်လင့် တကူ ရေးနိုင်တယ် ဆိုတာကို သတိပြုမိပါလိမ့်မယ်။

အခုနောက်ဆုံး code အတွက် ရေးကြည့်ရအောင်။ code က \ \ \ နှင့်စတယ်။ ပြီးလျှင် language တစ်ခု လိုက်တယ်။ တစ်ကြောင်းဆင်းတယ်။ ပြီးလျှင် code တွေ လာတယ်။ ပြီးလျှင် တစ်ကြောင်းဆင်းတယ်။ နောက်ပြီး \ \ \ လာတယ်။ အဲဒီ အတွက် pattern လေးရေးကြည့်ရအောင်။

`\ \ \ (.*)(\n)+(.*)(\n)+\ \ \`

Replace အတွက် pattern ကတော့

```
<pre><code class="\1">\2\3\4</code></pre>
```

အခု python code နဲ့ တွဲပြီး စမ်းရေးကြည့်ရအောင်။

```
import re
text = """```js
var k = 10;
```"""
replacement = re.sub("```(.*)(\n)+(.*)(\n)+```","<pre><code
class=\"\\1\">\\2\\3\\4</code></pre>",text)
print(replacement)
```

code လေးက ရိုးရိုးရှင်းရှင်းပါပဲ။ ကျွန်တော်တို့ အပေါ်မှာ ပြောထားသည့် အတိုင်း regular expresison နှင့် search and repalce လုပ်သွားတာပါ။

အခုဆိုရင်တော့ regular expression ကို အနည်းငယ် သိပြီလို့ ထင်ပါတယ်။ reguler expression ဟာ email တွေစစ်ဆေးခြင်း အခြား number validation , string validation စတာတွေ အတွက် လည်း အသုံးဝင်ပါတယ်။

အခု လေ့ကျင့်ခန်း အချို့ကို ရေးသားကြည့်ရအောင်။

## လေ့ကျင့်ခန်း ၁၁-၁

၁။ ကားလိုင်စင် စစ်ဆေးသည့် regular expression တစ်ခု ရေးသားပါ။ A-Z နှင့် စမယ်။ / လာမယ်။ ဂဏန်း ၅ လုံးပါမယ်။ `/` ပါမယ်။ ဂဏန်း ၂ လုံးပါမယ်။ ဥပမာ `B/11111/16` ပုံစံပါ။

၂။ mention ခေါ်သည့် ပုံစံကို regular expression မှာ ရေးကြည့်ရအောင်။ @mgmg ဆိုပြီး ရေးထားသည့် စာတွေကို html အနေနဲ့ ပြန်ပြောင်းပါမယ်။ `<a href="/username/mgmg">@mgmg</a>` ဆိုပြီး ဖော်ပြပေးပါမည်။

၃။ markdown က code တစ်ကြောင်းအတွက် \ `code here\` ဆိုရင် `<code>code here</code>` ဆိုပြီးပြောင်းပါတယ်။
```

နိဂုံး

အခုဆိုရင်တော့ ကျွန်တော်တို့တွေ programming အခြေခံ သဘောတရားကို နားလည်လောက်ပါပြီ။ အခု စာအုပ်မှာ programming ကို စလေ့လာဖို့ အတွက် အခြေခံ သဘောတရား အဆင့်သာ ရှိပါသေးတယ်။ Programmer တစ်ယောက် ဖြစ်ချင်ရင်တော့ နောက်ထပ် ထပ်ပြီးတော့ လေ့လာစရာတွေ အများကြီး ကျန်ပါသေးတယ်။ မဖြစ်မနေ လေ့လာသင့်တာတွေကတော့ HTML , CSS , Javascript နှင့် Database , UML စတာတွေကို ထပ်မံပြီးတော့ လေ့လာဖို့ လိုအပ်သေးပါတယ်။ အခု စာအုပ်ဟာ တစ်ခါမှ programming မလေ့လာဖူးသူတွေကို အခြေခံ သဘောတရား မိတ်ဆက်အဆင့်သာ ဖြစ်ပေမယ့် နောက်ထပ် အဆင့်တွေကို နားလည်လွယ်ကူစွာ လေ့လာနိုင်မယ်လို့ မျှော်လင့် ပါတယ်။ နောက်အဆင့်တွေ အနေနဲ့ မိမိ နှစ်သက်ရာ programming language ကို လေ့လာကြည့်ပါ။ လူသုံးများသည့် langauge တွေကတော့ php , java, c# , javascript (node js) စသည်တို့ ရှိပါသည်။ language တစ်ခု နှင့် တစ်ခု ဟာ syntax တွေ တူညီ မှု မရှိပါဘူး။ အခြေခံအဆင့်ကို နားလည်ထားရင် မည်သည့် language မဆို အလွယ်တကူသင်ယူနိုင်ပါလိမ့်မယ်။

ကျေးဇူးတင်ပါသည်။

လေ့ကျင့်ခန်း အမေးအဖြေ များ

လေ့ကျင့်ခန်း ၁

မေးခွန်း ၁။

အောက်ပါ program မှာ k ရဲ့ value က ဘာဖြစ်ပါသလဲ။

```
i = 7 + 3
k = i + 2
```

အဖြေက

A. 7

B. 10

C. 12

—

C.12

မေးခွန်း ၂။

လူတစ်ယောက်၏ မွေးဖွားသည့် နှစ်ကို လက်ခံပါ။ ထို့နောက် ၂၀၁၈ တွင် ရောက်ရှိနေသည့် အသက်ကို ဖော်ပြပါ။

```
birthday = int(input("Please enter your birth year: "))
print("Your age is ", 2018 - birthday)
```

လေ့ကျင့်ခန်း ၂

လေ့ကျင့်ခန်း ၂ - ၁

```
x = input("Enter first value : ")
y = input("Enter second value : ")
op = input("Operator (+ - * /) : ")
try :
    x = int(x)
```

```
        y = int(y)

        output = True
        if op == "+" :
            result = x+y
        elif op == "-" :
            result = x-y
        elif op == "*" :
            result = x*y
        elif op == "/" :
            result = x/y
        else :
            output = False
            print("Wrong Operator")

        if output :
            print("Result is ",result)


    except ValueError:
        print("Please enter number only")
        print(ValueError);
```

အထက်တွင် ဖော်ပြထားသော Calculator code ကို flow chart ဆွဲပါ။

လေ့ကျင့်ခန်း ၂ - ၂

မေးခွန်း ၁။

```
total = 0
for x in range(10):
    total = total + x
print(total)
```

program ကို run လိုက်ရင် Total က

A. 0

B. 10

C. 45

C. 45

မေးခွန်း ၂။

```
total = 0;
for x in range(10):
    total = total + 1
print(total)
```

program ကို run လိုက်ရင် Total က

A. 0
B. 10
C. 45

—

B. 10

မေးခွန်း ၃။

range(5) ဆိုရင်

A. 0 ကနေ 5 ထိ
B. 0 ကနေ 4 ထိ
C. 1 ကနေ 5 ထိ
D. 1 ကနေ 4 ထိ

—

B. 0 ကနေ 4 ထိ

မေးခွန်း ၄။

While loop က အနည်းဆုံး တစ်ကြိမ် အလုပ်လုပ်တယ်။

A. မှန်
B. မှား

—

A.

မေးခွန်း ၅။

While loop အသုံးပြုဖို့ အခေါက် အရေအတွက် အတိအကျ ရှိရမယ်။

A. မှန်
B. မှား

—

B.

မေးခွန်း ၆။

Fibonacci program လို့ ခေါ်ရအောင်။ ကျွန်တော်တို့တွေ user ဆီက နံပတ် လက်ခံမယ်။ နံပတ် က 5 ဖြစ်ရင်
Fibonacci sequence အရ နံပတ် ၅ ခု ထုတ်ပြမယ်။

```
1 1 2 3 5
```

7 ဖြစ်ခဲ့ရင်တော့

```
1 1 2 3 5 8 13
```

လို့ ထုတ်ပြမယ်။

Fibonacci sequence ဆိုတာကတော့ ရှေ့က နံပတ် ၂ ခု ကို ပေါင်းပြီးတော့ နောက်ထပ် ဂဏန်း တစ်ခု ရပါတယ်။

```
n = input("Enter number: ")
n = int(n)
f = 0
s = 1
for i in range(n):
    t = f
    f = s
    s = t + s
    print(f,end=" ")
```

မေးခွန်း ၇။

Even/odd စစ်ထုတ်တဲ့ program ပါ။ user ဆီကနေ ဂဏန်း လက်ခံမယ်။ ပြီးရင် 1 ကနေ စပြီး တော့ even ဖြစ်လား odd ဖြစ်လား ဆိုပြီး ထုတ်ပြရမယ်။

ဥပမာ user က 3 လို့ ရိုက်လိုက်ရင်

```
1 is Odd
2 is Even
3 is Odd
```

ဆိုပြီး ထုတ်ပြမယ်။

တကယ်လို့ 5 လို့ ရိုက်လိုက်ရင်

```
1 is Odd
2 is Even
3 is Odd
4 is Even
5 is Odd
```

ဆိုပြီး ထုတ်ပြရပါမယ်။

```
n = input("Enter number: ")
n = int(n)
f = 0
s = 1
for i in range(1,n+1):
    if i % 2 == 0 :
        print(i,"is Even")
    else:
        print(i,"is Odd")
```

လေ့ကျင့်ခန်း ၂-၃

မေးခွန်း ၁။

Max Number လိုမျိုး အငယ်ဆုံး ဂဏန်းကို ရှာတဲ့ code ရေးကြည့်ပါ။

```
list = [1255,2125,1050,2506,1236,1048,2010,1055]

minnumber = list[0]

for x in list:
    if minnumber > x :
        minnumber = x

print("MAX number in array is",minnumber)
```

မေးခွန်း ၂။

```
list = [1,5,2,7,8,9,200,155]
print(len(list))
```

အဖြေသည်

A. ၉

B. ၈

C. ၇

B

မေးခွန်း ၃။

`Array [3,4,1,2,9,7]` ဆိုပြီး ရှိပါသည်။ user ဆီက နံပတ်ကို လက်ခံပြီး array အခန်းထဲတွေ တွေ့မတွေ့ user ကို print ထုတ်ပြပါမည်။ တွေ့ခဲ့ပါက အခန်း ဘယ်လောက်မှာ တွေ့ခဲ့သည်ကို print ထုတ်ပြပါမည်။

```
n = input("Enter number: ")
n = int(n)
arr = [3,4,1,2,9,7]
```

```
room = 0
while room < len(arr):
    if arr[room] == n:
        print ("Found at ", room)
        exit()
    room = room + 1
print("Not found")
```

မေးခွန်း ၄။

Max number ကို ရှာပါ။ ဘယ်နံပတ်က အကြီးဆုံးလဲ။ ဘယ်အခန်းမှာ ရှိတာလဲ ဆိုတာကို print ရိုက် ပြပါ။

```
list = [1255,2125,1050,2506,1236,1048,2010,1055]

maxnumber = list[0]
foundAt = 0
room = 0
while room < len(list):
    x = list[room]
    if maxnumber < x :
        maxnumber = x
        foundAt = room
    room = room + 1

print("MAX number in array is",maxnumber," Room is",foundAt)
```

လေ့ကျင့်ခန်း ၂-၄

မေးခွန်း ၁။

minus ဆိုသည့် function ရေးပြပါ။ ဂဏန်း ၂ လုံး ပို့လိုက်ပြီး ရလာဒ်ကို return ပြန်ပေးရမည်။

```
def minus(val1,val2):
    return val1-val2
print("5 - 2 =",minus(5,2))
```

မေးခွန်း ၂။

triangle_star ဆိုတဲ့ function ကို ရေးပါ။ user ဆီက နံပတ်တောင်းပါ။ 3 လို့ရိုက်ရင် triangle_star(3) ဆိုပြီး ပို့ပေးပါ။ triangle_star မှ အောက်ပါ အတိုင်း ရိုက်ထုတ်ပြပါ။

```
*
**
***
```

အကယ်၍ 5 လို့ ရိုက်ထည့်လျှင် ၅ လိုင်း ထုတ်ပြပါမည်။

```
def triangle_star(stars):
  for start in range(stars):
    for star in range(start+1):
      print("*", end="")
    print("")

triangle_star(5)
```

လေ့ကျင့်ခန်း ၃

မေးခွန်း ၁။

ပေါင်းနှုတ်မြှောက်စား ပါဝင်သည့် MyMath class တစ်ခုကို တည်ဆောက်ပါ။

```
class MyMath:
    def sum(self,x,y):
        return x+y
    def minus(self,x,y):
        return x-y
    def multiply(self,x,y):
        return x*y
    def division(self,x,y):
        return x/y

math = MyMath()
print("4+3",math.sum(4,3))
print("4-3",math.minus(4,3))
print("4*3",math.multiply(4,3))
print("4/3",math.division(4,3))
```

မေးခွန်း ၂။

MyMath class ကို base ယူပြီး MyMathExt ဆိုသည့် class ကို ဆောက်ပါ။ ၂ ထပ် ၊ ၃ ထပ် တွက်ရန် ထည့်သွင်းပါ။

```
class MyMath:
    def sum(self,x,y):
        return x+y
    def minus(self,x,y):
        return x-y
    def multiply(self,x,y):
        return x*y
    def division(self,x,y):
        return x/y

class MyMathExt(MyMath):
    def sqaure(self,x):
        return x**2
    def three(self,x):
        return x**3

math = MyMathExt()
print("4+3",math.sum(4,3))
print("4-3",math.minus(4,3))
```

```
print("4*3",math.multiply(4,3))
print("4/3",math.division(4,3))
print("4 square",math.sqaure(4))
print("4 power 3 is",math.three(4))
```

လေ့ကျင့်ခန်း ၄

မေးခွန်း ၁။

Base 8 ကို ပြောင်းသည့် divdeBy8 function ကို ရေးပါ။ 668 (base 10) ၏ base 8 တန်ဖိုးသည် တန်ဖိုးသည် 1234 ဖြစ်သည်။ divideBy8(668) ၏ အဖြေသည် 1234 ထွက်လာရမည်။

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)

def divideBy8(number):
    remstack = Stack()

    base = 8
    while number > 0:
        rem = number % base
        remstack.push(rem)
        number = number // base

    binString = ""
    while not remstack.is_empty():
        binString = binString + str(remstack.pop())

    return binString
print(divideBy8(668))
```

မေးခွန်း ၂။

Base 16 ကို ပြောင်းသည့် divdeBy16 function ကို ရေးပါ။ 668 (base 10) ၏ base 16 တန်ဖိုးသည် တန်ဖိုးသည် 29C ဖြစ်သည်။ divdeBy16(668) ၏ အဖြေသည် 29C ထွက်လာရမည်။ base 16 သည် အောက် ဖော်ပြပါ အတိုင်း ပြောင်းလဲ ပေးရန် လိုသည်။

ထို့ကြောင့် အကြွင်း ကို if condition သုံးကာ 10 မှ 15 value များကို ပြောင်းပေးရန် လိုသည်။

```
class Stack:
    def __init__(self):
```

```
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)

def divideBy16(number):
  remstack = Stack()

  base = 16
  while number > 0:
        rem = number % base
        if rem == 10:
            rem = "A"
        elif rem == 11:
            rem = "B"
        elif rem == 12:
            rem = "C"
        elif rem == 13:
            rem = "D"
        elif rem == 14:
            rem = "E"
        elif rem == 15:
            rem = "F"
        remstack.push(rem)
        number = number // base

  binString = ""
  while not remstack.is_empty():
      binString = binString + str(remstack.pop())

  return binString
print(divideBy16(987))
```

မေးခွန်း ၁။

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)

q=Queue()
q.enqueue(12)
q.enqueue('dog')
```

```
q.enqueue(True)

print(q.dequeue())
```

အဖြေသည်

12

လေ့ကျင့်ခန်း ၆

မေးခွန်း ၁

အောက်ပါ code တွင် gender သည် male ဖြစ်သော သူများကို ဖော်ပြပါ။

```
room = [{'Name': 'Aung Ko', 'Age': 7, 'Gender' : 'male'}, {'Name': 'Ko Ko', 'Age': 8,
'Gender' : 'male'},{'Name': 'Aye Aye', 'Age': 7, 'Gender' : 'female'},{'Name': 'Htet
Htet', 'Age': 8, 'Gender' : 'female'},{'Name': 'Win Aung', 'Age': 7, 'Gender' : 'male'}]
```

Answer:

```
room = [{'Name': 'Aung Ko', 'Age': 7, 'Gender' : 'male'}, {'Name': 'Ko Ko', 'Age': 8,
'Gender' : 'male'},{'Name': 'Aye Aye', 'Age': 7, 'Gender' : 'female'},{'Name': 'Htet
Htet', 'Age': 8, 'Gender' : 'female'},{'Name': 'Win Aung', 'Age': 7, 'Gender' : 'male'}]

for obj in room:
    if obj["Gender"] == "male":
        print(obj["Name"])
```

မေးခွန်း ၂

အထက်ပါ code တွင် room ထဲတွင် ရှိသော လူများ၏ စုစုပေါင်း အသက်ကို ဖော်ပြသော code ရေးပြပါ။

Answer:

```
room = [{'Name': 'Aung Ko', 'Age': 7, 'Gender' : 'male'}, {'Name': 'Ko Ko', 'Age': 8,
'Gender' : 'male'},{'Name': 'Aye Aye', 'Age': 7, 'Gender' : 'female'},{'Name': 'Htet
Htet', 'Age': 8, 'Gender' : 'female'},{'Name': 'Win Aung', 'Age': 7, 'Gender' : 'male'}]

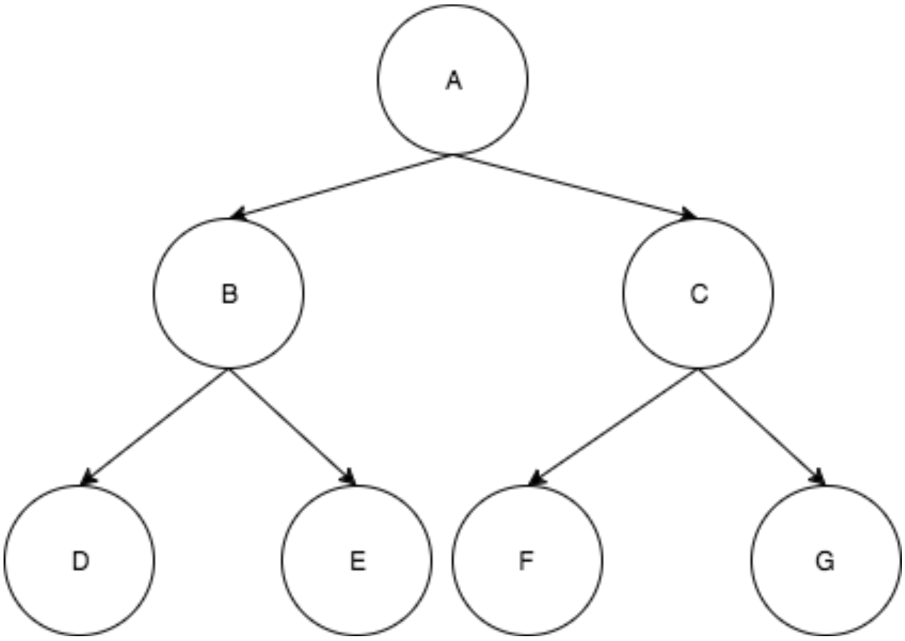
total_age = 0

for obj in room:
    total_age = total_age + obj["Age"]

print(total_age)
```

လေ့ကျင့်ခန်း ၉

လေ့ကျင့်ခန်း ၉-၁



မေးခွန်း ၁

အထက်ပါ binary tree တစ်ခု တည်ဆောက်ပါ။ ထို binary tree အတွက် dfs ကို သုံးပြီး search function ကို ရေးပါ။ ဥပမာ ။ F လို့ ထည့်လိုက်လျှင် binary tree တွင် ပါဝင်သောကြောင့် true ဟု return ပြန်ပါမည်။ H ဟု ထည့်လိုက်လျှင် ရှာ မတွေ့သောကြောင့် false ဟု return ပြန်ရမည်။

```
class BinaryTree:

    def __repr__(self):
        return "Binary Tree, Key is " + self.key

    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t

    def get_right_child(self):
```

```
            return self.right_child

        def get_left_child(self):
            return self.left_child

        def set_root_val(self,obj):
            self.key = obj

        def get_root_val(self):
            return self.key

        def search(self,value):
            if self != None:
                if self.get_left_child() != None:
                    if self.get_left_child().search(value):
                        return True

                if(self.get_root_val() == value):
                    return True

                if self.get_right_child() != None:
                    if self.get_right_child().search(value):
                        return True
                return False


    root = BinaryTree("A")

    root.insert_left("B")
    root.insert_right("C")

    b = root.get_left_child()

    b.insert_left("D")
    b.insert_right("E")

    d = b.get_left_child()

    d.insert_left("F")
    d.insert_right("G")

    found = root.search("F")
    if found :
        print("FOUND")
    else:
        print("NOT FOUND")
```

လေ့ကျင့်ခန်း ၉-၂

မေးခွန်း ၁

အထက်ပါ binary tree တစ်ခု တည်ဆောက်ပါ။ ထို binary tree အတွက် bfs ကို သုံးပြီး search function ကို ရေးပါ။ ဥပမာ ။ F လို့ ထည့်လိုက်လျှင် binary tree တွင် ပါဝင်သောကြောင့် true ဟု return ပြန်ပါမည်။ H ဟု ထည့်လိုက်လျှင် ရှာ မတွေ့သောကြောင့် false ဟု return ပြန်ရမည်။

```
    class BinaryTree:

        def __repr__(self):
```

```
        return "Binary Tree, Key is " + self.key

    def __init__(self,root):
        self.key = root
        self.left_child = None
        self.right_child = None

    def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.left_child = self.left_child
            self.left_child = t

    def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            t = BinaryTree(new_node)
            t.right_child = self.right_child
            self.right_child = t

    def get_right_child(self):
        return self.right_child

    def get_left_child(self):
        return self.left_child

    def set_root_val(self,obj):
        self.key = obj

    def get_root_val(self):
        return self.key

    def search(self,value):
        thislevel = [self]
        while thislevel:
            nextlevel = []
            level = []
            for n in thislevel:
                if (n.get_root_val() == value):
                    return True
                level.append(n.get_root_val())
                if n.get_left_child() != None:
                    nextlevel.append(n.get_left_child())
                if n.get_right_child() != None:
                    nextlevel.append(n.get_right_child())
            thislevel = nextlevel
        return False


root = BinaryTree("A")

root.insert_left("B")
root.insert_right("C")

b = root.get_left_child()

b.insert_left("D")
b.insert_right("E")

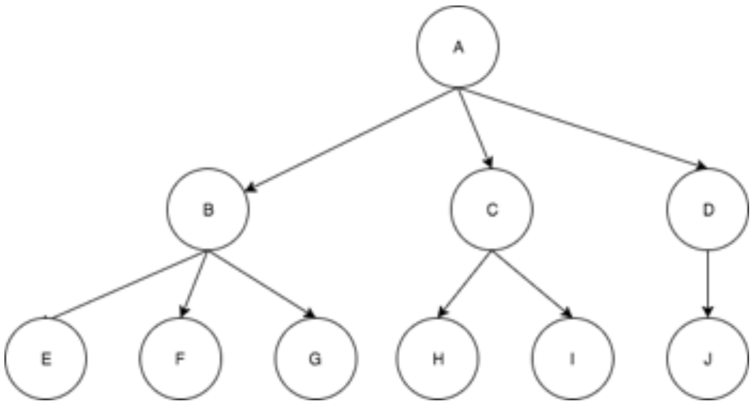
d = b.get_left_child()
```

```
d.insert_left("F")
d.insert_right("G")

found = root.search("H")
if found :
    print("FOUND")
else:
    print("NOT FOUND")
```

လေ့ကျင့်ခန်း ၉-၃

မေးခွန်း ၁



၁။ အထက်ပါ Tree ကို DFS ဖြင့် ရေးသားပါ။ တစ်ခုထက်မက child တွေ ဖြစ်နိုင်သည့် အတွက် preoder နှင့် postorder သာ အသုံးပြုနိုင်ပါသည်။ ထို့ကြောင့် preoder နှင့် postorder function ရေးသားပါ။

```
class Node:
    def __init__(self,value):
        self.value = value
        self.child = []

    def __repr__(self):
        return "Value is " + self.value

    def insert_child(self,node):
        self.child.append(node)

    def get_child(self):
        return self.child

class Node:
    def __init__(self,value):
        self.value = value
        self.child = []

    def __repr__(self):
        return "Value is " + self.value

    def insert_child(self,node):
        self.child.append(node)
```

```
    def get_child(self):
        return self.child

    def postorder(self):
        if self != None:
            if len(self.get_child()) > 0:
                for child in reversed(self.get_child()):
                    child.postorder()
            print(self.value)

    def preorder(self):
        if self != None:
            print(self.value)
            if len(self.get_child()) > 0:
                for child in self.get_child():
                    child.preorder()

root = Node("A")

b = Node("B")
c = Node("C")
d = Node("D")

root.insert_child(b)
root.insert_child(c)
root.insert_child(d)


e = Node("E")
f = Node("F")
g = Node("G")


b.insert_child(e)
b.insert_child(f)
b.insert_child(g)

h = Node("H")
i = Node("I")

c.insert_child(h)
c.insert_child(i)

j = Node("J")

d.insert_child(j)

print("---- PRE OREDER ----")
root.preorder()
print("---- POST OREDER ----")
root.postorder()
```

လေ့ကျင့်ခန်း ၁၁

လေ့ကျင့်ခန်း ၁၁-၁

မေးခွန်း ၁

ကားလိုင်စင် စစ်ဆေးသည့် regular expression တစ်ခု ရေးသားပါ။ `A-Z` နှင့် စမယ်။ `/` လာမယ်။ ဂဏန်း ၅ လုံးပါမယ်။ `/` ပါမယ်။ ဂဏန်း ၂ လုံးပါမယ်။ ဥပမာ `B/11111/16` ပုံစံပါ။

```
import re

text = "K/12344/32"

if re.match("[A-Z]\/[0-9]{5}\/[0-9]{2}",text) == None:
    print("NOT MATCH")
else:
    print("MATCH")
```

မေးခွန်း ၂

mention ခေါ်သည့် ပုံစံကို regular expression မှာ ရေးကြည့်ရအောင်။ @mgmg ဆိုပြီး ရေးထားသည့် စာတွေကို html အနေနဲ့ ပြန်ပြောင်းပါမယ်။ `<a href="/username/mgmg">@mgmg</a>` ဆိုပြီး ဖော်ပြပေးပါမည်။

```
import re

text = "@mgmg"
replacement = re.sub("@(.*)\s*","<a href=\"/username/\\1\">@\\1</a>",text)

print(replacement)
```

မေးခွန်း ၃

markdown က code တစ်ကြောင်းအတွက် \ `code here\` ဆိုရင် `<code>code here</code>` ဆိုပြီး ပြောင်းပါတယ်။

```
import re

text = "hello `code here` world"
replacement = re.sub("`(.*)`","<code>\\1</code>",text)

print(replacement)
```

Reference

- https://docs.python.org/3/
- http://en.wikipedia.org/wiki/Programming_language_generations
- http://www.rff.com/flowchart_shapes.htm
- http://study.cs50.net
- Problem Solving with Algorithms and Data Structures By Brad Miller and David Ranum
- https://en.wikibooks.org
- https://rosettacode.org
- https://math.stackexchange.com/questions/2260/proof-for-formula-for-sum-of-sequence-123-ldotsn
- http://bigocheatsheet.com/
- https://www.youtube.com/watch?v=v4cd1O4zkGw
- https://en.wikipedia.org/wiki/Sorting_algorithm
- https://www.khanacademy.org/computing/computer-science/algorithms
- http://www.cs.wcupa.edu/rkline/ds/shell-comparison.html

ကျေးဇူးတင်လွှာ

ကျွန်တော့်အား programming ကို ပထမဆုံး သင်ကြားပေးခဲ့သော ဗိုလ်တစ်ထောင် ကောလိပ် လူ့စွမ်းအား အရင်း အမြစ် မှ သင်ဆရာ မြင်ဆရာ ကြားဆရာများ ကို ကျေးဇူးတင်ရှိပါတယ်။ ယခု စာအုပ်မှာ ကျောင်းမှ သင်ကြားပေးခဲ့သည် ဆရာ ဆရာမ များ၏ သင်ကြားမှု ပုံစံ ၊ သူတို့၏ ရှင်းလင်းထားမှုတွေကို အခြေခံထားတာတွေ ပါဝင်ပါတယ်။

ကျွန်တော့်အား စာအုပ် ထုတ်ဖို့ အမြဲပြောကြားသည့် Myanmar Link မှ ကိုစိုးသီဟနောင် ၊ စာအုပ်ထုတ်ဖို့ လိုအပ်သည်များကို ကူညီပေးသည့် ကျွန်တော့် ဇနီး ယဉ်ယဉ်အေး နှင့် ကျွန်တော့် ပညာရေး အတွက် ပြုစု ပျိုးထောင် ပေးခဲ့သော အဖေ နှင့် အမေ တို့ကို ဤစာအုပ် ဖြစ်မြောက် လာသည့် အတွက် ကျေးဇူးတင်ရှိပါတယ်။

Saturngod

စာရေးသူ၏ ကိုယ်ရေး အကျဉ်း

စာရေးသူ၏ အမည်ရင်းမှာ ထိန်လင်းရွှေ ဖြစ်ပြီး ရန်ကုန် မြို့တွင် မွေးဖွားပါသည်။ ၂၀၀၅ ခုနှစ်တွင် အ.ထ.က (၁) ဒဂုံ မှ တက္ကသိုလ်ဝင် စာမေးပွဲ အောင်မြင်ခဲ့ပြီး University Of Computer Studies (UCSY) , 15 Batch, Class of 2006 တွင် ဆက်လက်တက်ရောက်ခဲ့ပါသည်။ ၄ နှစ်တာ တက်ရောက်ပြီး B.C.Sc (Hons.) ဘွဲ့ရရှိခဲ့ပါသည်။ ထို့နောက် စင်္ကာပူ နိုင်ငံတွင် ၇ နှစ်ကျော် အလုပ်လုပ်ကိုင်ခဲ့ပါသည်။ အဓိကအပိုင်းအားဖြင့် Web Application Development , iOS App Development ပိုင်းကို လုပ်ကိုင်ခဲ့ပါသည်။

၂၀၁၇ တွင် မြန်မာနိုင်ငံ ပြန်လည်ရောက်ရှိခဲ့ပြီး COMQUAS CO., Ltd ကို တည်ထောင်ကာ Mobile Apps များရေးစွဲခြင်း ၊ နည်းပညာဝန်ဆောင်မှုများ ပေးခြင်း စသည်တို့ကို လုပ်ကိုင်လျက်ရှိပါသည်။ ဤစာအုပ် ရေးသားချိန်တွင် COMQUAS Co., Ltd တွင် CEO ရာထူး ထမ်းဆောင်လျက်ရှိပါသည်။

၂၀၀၆ မှ စတင်ပြီး Saturngod အမည်ဖြင့် blog ရေးသားခဲ့ပါသည်။ https://blog.saturngod.net တွင် စာရေးသူ၏ ပုံမှန် ရေးသားသည့် စာများကို ဖတ်ရှုနိုင်ပါသည်။ စာရေးသူအား တိုက်ရိုက်ဆက်သွယ်လိုပါက saturngod@gmail.com သို့ email ပေးပို့နိုင်ပါသည်။