

Begin Modern Programming

with

C
O
D
I
N
G
A

Pyi Soe

မာတိကာ

၁	ခက်ရပ်ကားရဲလှုပြင့် ပရိုဂရမ်းမင်းမိတ်ဆက်	၁
၁.၁	ခက်ရပ် ကားရဲလှု	၁
၁.၂	Meet Karel ပရိုဂရမ်	၃
၁.၃	ကားရဲလှု ပရိုဂရမ် run ခြင်း	၈
၁.၄	Python အင်စတောလ်လုပ်တဲ့အခါ ဘာတွေပါလဲ	၉
၁.၅	Move Beeper to Other Side	၁၀
၂	ကားရဲလှုနှင့် ကွန်ထရီးလ် စတိတ်မန်များ	၁၃
၂.၁	for loop	၁၃
၂.၂	အင်ဒန်ထလုပ်ခြင်းနှင့် ကွန်စထရက်ချာ	၁၆
၂.၃	while loop	၁၆
၂.၄	if စတိတ်မန်	၁၉
၂.၅	if...else စတိတ်မန်	၂၀
၂.၆	Nested ကွန်ထရီးလ် စတိတ်မန်များ	၂၂
၃	ဖန်ရှင်များ (Functions)	၂၄
၃.၁	ဖန်ရှင် သတ်မှတ်ခြင်း	၂၄
၃.၂	ပရိုကွန်ဒီရှင်နှင့် ပို့စ်ကွန်ဒီရှင်	၂၂
၃.၃	ဖန်ရှင်များဖြင့် abstraction များ တည်ဆောက်ခြင်း	၂၅
၃.၄	Bottom-up Programming	၂၅
၃.၅	Top-down Programming	၂၃
၄	ကားရဲလှုနှင့် ရိုကားဆိုင် ဖန်ရှင်များ	၂၁
၄.၁	ရိုကားဆိုင် ဖန်ရှင် ဘယ်လို အလုပ်လုပ်လဲ	၂၀
၄.၂	ရိုကားဆိုင်စွဲ့ဖြင့် ပုံစံဖြန့်ရှင်းခြင်း	၂၃
၅	အေတာများနှင့် ဖန်ရှင်များ	၂၅
၅.၁	ကိန်းဂဏ်းများ	၂၅

၉.၂ ဖေရီရေတဲ့လုံး	၇၃
၉.၃ ဓမ္မားများ	၇၅
၉.၄ အီပိုစ်ပရက်ရှင်များ	၇၇
၉.၅ ဘူလီယန် ထိုက်နှင့် ဘူလီယန် အီပိုစ်ပရက်ရှင်	၇၈
၉ အော်ဂျက်များ	၈၁
၉.၁ date, time and datetime	၈၂
၉.၂ list	၉၀
၁၀ ကွန်ထရီးလ် စတိတ်မန္ဒားများ	၉၃
၁၀.၁ if စတိတ်မန္ဒား	၉၃
၁၀.၂ for Loop	၉၅
၁၀.၃ Python Arcade Library	၁၀၅
၁၀.၄ while loop	၁၀၀
၁၀.၅ လောက့်ခန်း ဥပမာများ	၁၀၄
၁၁ ဖန်ရှင်များ	၁၂၀
၁၁.၁ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်များ	၁၂၀
၁၁.၂ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်များ	၁၂၅
၁၁.၃ Exceptions	၁၂၇
၁၁.၄ ဖန်ရှင်နှင့် ပရိုဂရမ်ဒီဇိုင်း ဥပမာ (၁) “Insurance Premium”	၁၃၀
၁၁.၅ ဖန်ရှင်နှင့် ပရိုဂရမ်ဒီဇိုင်း ဥပမာ (၂) “Colorful House”	၁၃၃
၁၁.၆ ပါရာမီတာ ခွေးချယ်သတ်မှတ်ခြင်း	၁၄၂
၁၁.၇ တန်ဖိုးပြန်ပေးခြင်း၊ မပေးခြင်း	၁၄၅
၁၁.၈ return	၁၄၆
၁၂ From Classes to Objects (ကလပ်စ်များမှ အော်ဂျက်များသိ)	၁၄၉
၁၂.၁ Account Class (အကောင့် ကလပ်စ်)	၁၅၉
၁၂.၂ Access Control	၁၆၄
၁၂.၃ Composition or Has-a Relationship	၁၆၆
၁၂.၄ Properties	၁၇၀
၁၂.၅ Class Attributes and Methods	၁၇၃
၁၂.၆ Static Methods	၁၇၆
၁၂.၇ Object-Oriented Programming	၁၈၀

၁၀	Inheritance and Polymorphism	၁၉၅
၁၀.၀	Inheritance ဥပမာ ‘Account Class Hierarchy ^၅	၁၉၅
၁၀.၂	Overriding.	၁၉၉
၁၀.၃	Multilevel Inheritance	၂၀၀
၁၀.၄	Class Hierarchy and UML	၂၀၀
၁၀.၅	Is-A Relationship and Inheritance	၂၀၀
၁၀.၆	အသုံးချ ဥပမာ (၁) Breakout Game.	၂၀၀
၁၀.၇	Breakout တည်ဆောက်ခြင်း	၂၀၂
က	လိုအပ်သည့် အသေးစိတ်များ ထည့်သွင်းခြင်း	၂၂၅
ခ	ကားရဲလ်ပရိုဂရမ် ဖီချာများ	၂၃၁

အခန်း ၁

စက်ရုပ်ကားရဲလိုဖြင့် ပရီဂရမ်းမင်းမိတ်ဆက်

ကွန်ပျိုတာတွေဟာ သက်မဲ့ စက်ပစ္စည်းတွေပါပဲ။ ကားတို့ လေယာဉ်တို့နဲ့ မတူတာက ကွန်ပျိုတာတွေဟာ စက်ချဉ်းသက်သက် ဘာအစွမ်းမှ မယ်မယ်ရရ မရှိဘူး။ ဒါပေမဲ့ ဆောင်ရွက်လိုတဲ့ ကိစ္စအဝေဝအတွက် ပရီဂရမ်းမျိုးမျိုး ထည့်ပေးလိုက်တဲ့ အခါမာ သူ့အစွမ်းက အတိုင်းအဆမဲ့ပဲ။ နေရာမျိုးစံ၊ နယ်ပါယ်မျိုးစံ မှာ အကူးအညီပေးနိုင်တဲ့ စွယ်စုံသုံး ပစ္စည်းတစ်ခုဖြစ်သွားတယ်။ ဂိတ်သံစဉ်တွေကို ဖွင့်ပေးနိုင်သလို အသံလည်းသွင်းပေးနိုင်တယ်။ ရုပ်ရှင်တည်းဖြတ် လုပ်ချင်တာလား။ ပြဿနာမရှိဘူး၊ ကူညီပေးနိုင်တယ်။ နျှော လီးယား ပါတ်ပေါင်းဖို့တွေကို ဖို့မိန့်သလို မောင်းသူမဲ့ အုံပျံတွေကိုလည်း ပဲထိန်းပေးနိုင်တယ်။

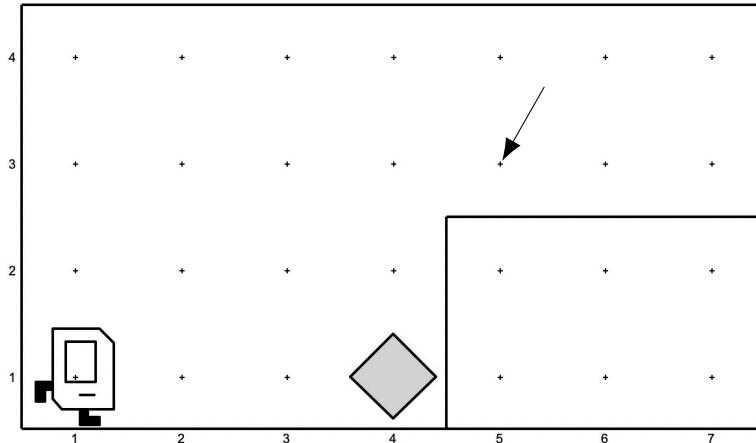
ကျွန်တော်တို့တွေ နိစ္စဓာဝ အသုံးပြုနေကြတဲ့ ကား၊ စမတ်နှင့်း၊ လက်ပါတ်နာရီ၊ မိုက်ခရှိဝေဖွံ့ဖို့ အဝတ်လျှော့စက် စတဲ့ စက်ပစ္စည်း အမျိုးမျိုးဟာလည်း ကွန်ပျိုတာတွေနဲ့ မကင်းပြန်ပါဘူး။ “ကွန်ပျိုတာနည်းပညာ အကူးအညီမပါတဲ့ အတိုင်းဆက်သံစိတ်ထွင်မှုဆိုတာ မရှိဘူး” လို့ ဆိုနိုင်ပါတယ်။

တစ်ချက်တစ်ချက် ရိုက်ခတ်လိုက်တဲ့ ကွန်ပျိုတာနည်းပညာ လိုင်းလုံးကြီးတွေဟာ ကမ္မာတစ်စုံမှုးလုံး ပုံစံပြောင်းသွားလောက်အောင် အဟုန်ပြင်းထန်လာတယ်။ ဘီလီယံချွဲ့ခြုံတဲ့ လူတွေ ဆီရှယ်မီဒီယာတွေပေါ်က နေ ရုပ်သံတွေနဲ့ ချိတ်ဆက်ပြောင်းလုံး ရစေတာဟာလည်း ကွန်ပျိုတာစနစ်တွေပါပဲ။ Artificial Intelligence (AI) နည်းပညာကြောင့် သက်ရှိတွေမှာပဲတွေ့ရတဲ့ ညာက်ရည်မျိုးကို ကွန်ပျိုတာတွေမှာလည်း တွေ့လာရပါပြီ။ သံချုပ်စွာတွေ ဖြေရှင်းခြင်း၊ စစ်တုရင်ထိုးခြင်း စတဲ့ ကိစ္စမျိုးတွေအပြင် ပန်းချီဆွဲခြင်း၊ ကဗျာရေးစပ်ခြင်း၊ သီချင်းရေးဖွံ့ခြင်း ကဲ့သို့ အနုပညာဖန်တီးမှုတွေကိုပါ AI က လုပ်ဆောင်ပေးနိုင်ပါတယ်။ နှစ်ဆယ်တစ်ရာစုံ၊ အထူးမြှားဆုံး AI နည်းပညာလိုင်းဟာ အရှိန်အဟုန်ပြင်းပြင်း ရိုတ်ခတ်ဖို့ အားယူစ ပြုနေပါပြီ။

‘ကွန်ပျိုတာ’ လိုပြောတဲ့ အခါ စက်ပစ္စည်းသက်သက် မဟုတ်ဘဲ ကွန်ပျိုတာမှုတ်ညာက်ထဲက ပရီဂရမ်တွေလည်း ပါဝင်တယ်ဆိုတာ သတိချုပ်ပါမယ်။ ကွန်ပျိုတာတွေ တစ်စုံတစ်ရာ စွမ်းဆောင်နိုင်စေတဲ့ ပရီဂရမ်တွေ ရေးတဲ့ အလုပ်ကို ပရီဂရမ်းမင်း (Programming) လို့ခေါ်တယ်။

၁.၁ စက်ရုပ် ကားရဲလို

ပရီဂရမ်းမင်းမင်းမှုတာ ဘယ်လိုမျိုးလဲ သဘောပေါက်အောင် စာတွေတစ်သီးပြီးရေး ရှင်းပြတာထက် ပရီဂရမ်လေးတွေ လက်တွေ့ ရေးကြည်လိုက်တာ ပိုပြီးထိရောက်ပါတယ်။ ဒါကြောင့် စက်ရုပ်ကားရဲလိုကို ပရီဂရမ်လေးတွေရေးပြီး အလုပ်တွေခုံးကြည့်ကြမယ်။ ပုံ (၁.၁) မှာ တွေ့ရတာက ကားရဲလို ရောက်ရှိနေတဲ့ နဗုံနာကမ္မာတစ်ခုပါ။ မီးခါးရောင် မှုန်ကူကုံးပုံလေးကို ဘိပါ (beeper) လို့ ခေါ်တယ်။ အဲဒီဘိပါကို မြားပြထားတဲ့ နေရာကို ရွှေခိုင်းချင်တယ်။ မျှေားမည်းအထူးတွေက နံရံတွေပါ။ ကားရဲလိုကို ကိစ္စတစ်ခု ဆောင်ရွက်စေ



ပုံ ၁.၁ ခက်ရှုပ်လေး ကားရဲ့

ချင်တဲ့အခါ အခြေခံ ကားရဲ့လိုက် တွေ့ကို အသုံးပြုရပါတယ်။ ကွန်မန်းတွေ့ကို နှုတ်နှုံးပြောပြီး ခိုင်းရတာ မဟုတ်ဘဲ ပရိုကရမ်းပြီး ခိုင်းရတာပါ။ ကားရဲ့နားလည်တဲ့ ကွန်မန်းတွေ့ကို ကြည့်ကြရအောင်။

ကားရဲ့လိုက်မန်းများ

မဖြစ်မနေ သိထားရမဲ့ အခြေခံ ကားရဲ့လိုက်မန်း လေးခုပဲ ရှိတယ်။ move, turn_left, put_beeper နဲ့ pick_beeper တို့ဖြစ်တယ်။ အခြား ကားရဲ့လိုက်မန်း တွေ့လည်း ရှိပါသေးတယ်။ ဒါပေမဲ့ ကားရဲ့လိုက်များမင်း စလေ့လာဖို့ ဒီလေးခုနဲ့ပဲ လုံလောက်ပါပြီ။

move ကွန်မန်းက ကားရဲ့လိုက် ရှေ့တွေ့ကွန်နာကို ရွှေ့ခိုင်းတာ။ ကားရဲ့လိုက်မှာ တစ်ခုနဲ့တစ်ခု အကွာအဝေးတူ ခြားထားတဲ့ အတန်းလိုက် အတန်းလိုက် အစက်ကလေးတွေ့ဟာ ကွန်နာ (corner) တွေ ဖြစ်တယ်။ ကဲ့များကို မျှေားမည်းအထူ နံရုံတွေ့နဲ့ ထောင့်မှုန်းစတုဂံပဲ ပါတ်လည် ဘောင်ခတ်ထားတယ်။ ကွန်နာတွေ့ကြားမှာလည်း နံရုံတွေ့နိုင်တယ်။ နူးနာကဲ့မှာ ဘေးတိုက် နံပါတ်စဉ် ၄ နဲ့ ၅ ကြား ထောင်လိုက် နံရုံတွေ့ခဲ့ အထက်အောက် နံပါတ်စဉ် ၂ နဲ့ ၃ ကြား အလျေားလိုက် နံရုံတွေ့ခဲ့ကို တွေ့ရှုပါမယ်။ ကွန်နာရှေ့မှာ နံရုံကေနေရင် ကားရဲ့လိုက် ပေါ်ခဲ့လို့မရပါဘူး။

put_beeper က ကားရဲ့လိုက် လက်ရှိ ရှိနေတဲ့ ကွန်နာမှာ ‘ဘိပါတစ်ခုချု’ ထားခိုင်းတာ၊ pick_beeper က ပုံပေါ်နေတဲ့ ကွန်နာမှာ ‘ဘိပါတစ်ခုကောက်’ ခိုင်းတာပါ။ ကွန်နာမှာ ဘိပါရှိနေမှ ကောက်ခိုင်းလို့ရမှုပါ။ မရှုရင် ကောက်ခိုင်းလို့ မရှုဘူး။ ဘိပါချိခိုင်းရင်လည်း ကားရဲ့လိုက် ဘိပါရှိမှ ချိခိုင်းလို့ရတယ်။ ကားရဲ့လိုက် ဘိပါတွေ့ လို့သလောက် ဖြည့်ပေးထားတယ်လို့ ယူဆပါ။ turn_left က ‘ဘယ်လှည့်’ ခိုင်းတာ။

ဘိပါကို ဘယ်လိုပြောခိုင်းမလဲ

ပုံ (၁.၁) အနေအထားကနေ ရေ့ကို သုံးနေရာရွှေ့ ဘိပါကောက်၊ ဘယ်ဘက်လှည့်၊ အပေါ် နှစ်နေရာရွှေ့ ညာဘက်လှည့်၊ ရေ့တစ်နေရာထပ်ရွှေ့ပြီး ဘိပါချုထားခိုင်းလိုက်ရင် အလုပ်ပြီးသွားပါပြီ။

ကားရဲ့လိုက် ညာဘက်လှည့်ခိုင်းဖို့ turn_right ကွန်မန်း မရှိဘူး။ ဒါပေမဲ့ ဘယ်သုံးခါလှည့်တာဟာ ညာဘက်လှည့်တာနဲ့ တူတူပါပဲ။ ဒါကြောင့် ညာဘက်ချင်တဲ့အခါ ဘယ်သုံးခါလှည့်ခိုင်းလို့ရတယ်။

၁.၂ Meet Karel ပရိုဂရမ်

ပရိုဂရမ် ရေးတယ်ဆိုတာ ကွန်ပူးတာကို ကိစ္စတစ်ခုခဲ့ အောက်ရှုက်ပေးဖို့ ခိုင်းစေတဲ့ ညွှန်ကြားချက်တွေ ရေးတာပါပဲ။ ဒီလို ညွှန်ကြားချက်တွေကို ပရိုဂရမ်ကုဒ် (program code) လို ခေါ်တယ်။ ပရိုဂရမ်ကုဒ် တွေကို ကွန်ပူးတာနားလည်တဲ့ programming language တစ်ခုခဲ့ ရေးရတယ်။ ဒီစာအပ်မှာ အသုံးပြုမဲ့ programming language ကတော့ Python ပါ။ Programming language တစ်မျိုးပဲ ရှိတာ မဟုတ်ပါဘူး။ ရာနဲ့ချိပြီး ရှိတာပါ။ လူဘာသာစကားတွေ အမျိုးမျိုးရှိသလိုပဲပေါ့။ Python ဟာ ဒီလို ရာနဲ့ချိတဲ့ထဲက လက်ရှိအသုံးအများဆုံး ထိပ်ဆုံးဆယ်ခု ထဲမှာ ပါဝင်တယ်။ Python နဲ့ ဘိပါရွှေခြင်းတဲ့ ပရိုဂရမ်ကို လေ့လာကြည့်ရအောင်။ ကားရဲလ်နဲ့ ပထမဆုံး မိတ်ဆက်ပေးတဲ့ ပရိုဂရမ်မို့လို ဒီပရိုဂရမ် နံမည်ကို 'Meet Karel' လို ခေါ်ပါမယ်။

```
# File: meet_karel.py
# About: This is
from stanfordkarel import *

def main():
    """Karel code goes here!"""
    move()
    move()
    move()
    pick_beeper()
    turn_left()
    move()
    move()

    turn_left()
    turn_left()
    turn_left()

    move()
    put_beeper()
# End of main

if __name__ == "__main__":
    run_karel_program("meet_karel")
```

ဒါဟာ 'Meet Karel' ပရိုဂရမ်အတွက် Python နဲ့ရေးထားတဲ့ ပရိုဂရမ်ကုဒ် တွေဖြစ်ပါတယ်။ 'Python ကုဒ်' လို အတိုချိုးပဲ ပြောတာများတယ်။ Python 'စာ/စကား' မတတ်ရင် ဒီ 'Python ကုဒ်' တွေကိုလည်း နားလည်မှာ မဟုတ်ဘူး။ ဒီတော့ Python 'စာ/စကား' အခြေခံက စပြီး လေ့လစိုးလိုပါမယ်။

ကွန်းမာန် (Comment)

ပထမဆုံး # သက်တနဲ့ စတဲ့ စာကြောင်းတွေက ကွန်းမန်တွေပါ။ ကွန်းမန်တွေက ကွန်ပူးတာ အောင်ရှုကပေးရမဲ့ ညွှန်ကြားချက်တွေ မဟုတ်ဘူး။ ပရိုဂရမ်ကုဒ်နဲ့ ပါတ်သက်ပြီး ကုဒ် ဖတ်ရှုသူ အတွက် မှတ်ချက်

ရေးတာ သို့မဟုတ် ရှင်းပြထားတာပါ။ တနည်းအားဖြင့် ဖတ်ရှုသူ (လူ) ပရိုကရမ်မာအတွက် ရည်ရွယ်တာ။ ကွန်ပျူးတာ (စက်) အတွက် ရည်ရွယ်တာ မဟုတ်ဘူး။ ကွန်ပျူးတာက ကုဒ်ထဲက ကွန်းမန်တွေ အားလုံးကို လစ်လျှော့ရှုမှုပါ။ ဒါပေမဲ့ ပရိုကရမ်ကုဒ်ကို ဖတ်တဲ့လူ နားလည်ဖို့ အထောက်အကြုံ ဖြစ်စေတဲ့အတွက် ကွန်းမန်ရေးတာကို ပေါ့ပေါ့တန်တန် အရေးမပါသလို သဘောထားလို့ မရပါဘူး။ မိမိရေးတဲ့ ကုဒ်ကို ရှင်းပြဖို့ လိုအပ်ရင် ကွန်းမန်ရေးရပါမယ်။ ရေးသင့်တဲ့ နေရာတွေကိုလည်း မကြာခင်တွေရမှုပါ။

import စတိတ်မန်

```
from stanfordkarel import *
```

ကတော့ အငို့စတိတ်မန် ဖြစ်ပါတယ်။ “stanfordkarel လိုက်ဘရီမှ အာလုံးကို ထည့်သွင်းပေးပါ” လို တောင်းဆိုတဲ့ အဓိပ္ပာယ်။ * သက်တကို ‘အားလုံး’ လို ယူဆပါ။ stanfordkarel လိုက်ဘရီမှာ ကား ရဲလ်ပရိုကရမ်အတွက် လိုအပ်တာအားလုံး ပါဝင်တယ်။ ဒီလိုက်ဘရီကို အင်ပိုလုပ်ထားမှ ကားရဲလ်ကွန်းမန်း တွေ သုံးလို့ရမှုပါ။ သီးခြား ကားရဲလ်ပရိုကရမ် တစ်ခုစီတိုင်းအတွက် အင်ပိုလုပ်ရမှာ ဖြစ်တယ်။

လိုက်ဘရီ

လိုက်ဘရီ (library) ဆိုတာ ပညာရပ်နယ်ပယ် တစ်ခုအတွက် ရည်ရွယ်ရေးထားတဲ့ ပရိုကရမ်ကုဒ်တွေ ပါပဲ။ သချိအတွက်အချက် လိုက်ဘရီ ဂိမ်းရေးဖို့ လိုက်ဘရီ 2D/3D ဂရပ်ဖစ်ဆွဲဖို့ လိုက်ဘရီ၊ အေ အိုင်အတွက် လိုက်ဘရီ စသည်ဖြင့် နယ်ပယ်အသီးသီး၊ ကိစ္စရပ်အဖို့ဖို့အတွက် သက်ဆိုင်ရာ ကျမ်းကျင် ပညာရှင်တွေ ထုတ်လုပ်ဖြန့်ချုပ်ပေးထားတဲ့ လိုက်ဘရီတွေရှိတယ်။ Matrix အော်ပရေးရှင်းတွေအတွက် numpy၊ ဂရပ်ဖွဲ့မယ်ဆိုရင် matplotlib စတဲ့ လိုက်ဘရီတွေကို အင်ပိုလုပ် အသုံးပြနိုင်ပါတယ်။ မေ့ထရ် A ကို B နဲ့ မြောက်ရင် ဒီလိုပါ

```
from numpy import *

A = array([[1, 1, 2, 2],
           [2, 2, 1, 1],
           [2, 2, 1, 1]])
B = array([[2, 2],
           [2, 2],
           [1, 1],
           [2, 2]])

result = matmul(A, B)

print(result)
```

ရလဒ် အခုလိုထွက်ပါမယ်။

```
[[10 10]
 [11 11]
 [11 11]]
```

ဒါကတော့ ဘားချုပ် အတွက် numpy နဲ့ matplotlib လိုက်ဘရီ သုံးထားတာပါ။

```

from matplotlib.pyplot import *
from numpy import *

# make data:
x = 0.5 + arange(8)
y = [4.8, 5.5, 3.5, 4.6, 6.5, 6.6, 2.6, 3.0]

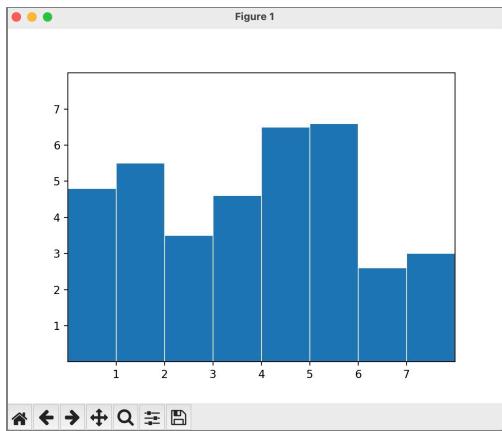
# plot
fig, ax = subplots()
bar(x, y, width=1, edgecolor="white", linewidth=0.7)

ax.set(xlim=(0, 8), xticks=arange(1, 8),
       ylim=(0, 8), yticks=arange(1, 8))

show()

```

ဘားချုတ်ကို ဒီလို ထုတ်ပေးပါတယ်။



ပုံ ၁၂

လိုက်ဘရီတွေဟာ ပရိုကရမ်တွေ တည်ဆောက်ရာမှာ အင်မတန်မှ အရေးပါတယ်။ ဖော်ပြထားတဲ့ မေ့သရစ်နဲ့ ဘားချုတ် ကုဒ်တွေကို (အခုတော့) နားလည်မှာ မဟုတ်သေးဘူးပေါ့။ ဒါပေမဲ့ သက်ဆိုင်ရာ လိုက် ဘရီတွေနဲ့ ဒီလိုကိစ္စတွေကို သိပ်မခက်ခဲဘဲ လုပ်လို့ရနိုင်တယ်ဆိုတာ မြင်မယ် ထင်ပါတယ်။ လိုက်ဘရီတွေ သာမရှိရင် ပရိုကရမ်တွေကို အခုထက် အဆပေါင်းများစွာ အချိန်ပေးပြီး ရှုပ်ရှုပ်တွေးတွေး ခက်ခက်ခဲ့ဘဲ တည်ဆောက်ကြရမှာပါ။

တံက်၊ ဓာတ်မန်နှင့် ဆင်းတာကို

ဂျိန်ပန်စာ၊ ပြင်သစ်စာ စတဲ့ လူ့ဘာသာစကားတွေဟာ စကားလုံးတွေ ဝါကျတွေနဲ့ ဖွဲ့စည်းထားသလို ပရိုကရမ်ကုဒ်တွေဟာလည်း စကားလုံးတွေ ဝါကျတွေနဲ့ ဖွဲ့စည်းထားတာပါပဲ။ Programming language တွေမှာ စကားလုံးတွေကို တံက် (token) လို့ခေါ်ပြီး ဝါကျတွေကိုတော့ စတိတ်မန် (statement) လို့ ခေါ်ပါတယ်။ ဝါကျတွေကို စကားလုံးတွေနဲ့ ဖွဲ့စည်းထားသလို စတိတ်မန်တွေကတော့ တံက်တွေနဲ့

ဖွဲ့စည်းထားတာပါ။ စတိတ်မန် ပုံစံတစ်မျိုးကို တွေ့ခဲ့ပြီးပါပြီ။ အဲဒါကတော့ ရှေ့စာမျက်နှာက အင်ပိုစတိတ်မန်ပဲ ဖြစ်ပါတယ်။

လူဘာသာစကားတွေမှာ ဖွဲ့စည်းတည်ဆောက်ပုံ စထရက်ချာရှိသလို programming language တွေမှာလည်း စထရက်ချာရှိဖို့ လိုအပ်တာပေါ့။ ဖွဲ့စည်းပုံ စထရက်ချာ မှန်/မမှန်ကို သဒ္ဓါစည်းမျဉ်တွေနဲ့ ထိန်းကွပ်ထားတာပါ။ ပရိုကရမ်ကုဒ် စထရက်ချာ မှန်/မမှန် ထိန်းကွပ်ပေးတဲ့ သဒ္ဓါစည်းမျဉ်တွေကိုတော့ ဆင်းတက်စ် (syntax) လိုခေါ်တယ်။

မြန်မာလိုရေးရင် မြန်မာသဒ္ဓါကို လိုက်နာရသလို Python နဲ့ ရေးရင် Python ဆင်းတက်စ်ကို လိုက်နာရမှာပေါ့။ မြန်မာသဒ္ဓါမှားရင် ဖတ်တဲ့သူက သည်းခံနားလည် ပေးပေါ့ ဆင်းတက်စ်မှားရင်တော့ Python က လုံးဝ လက်ခံမှာ မဟုတ်ပါဘူး။ ဆင်းတက်စ် စည်းကမ်းတွေဟာ ပိုပြီး တင်းကျပ်တယ်။ လွှဲချော်လို့ မရဘူး။ ဆင်းတက်စ်မှားနေတဲ့ ပရိုကရမ်ကို Python က run ခွင့် ပြုမှုမဟုတ်ဘဲ အမှားနဲ့ သက်ဆိုင်တဲ့ အယ်ရာမက်ဆွဲချုပ်တွေ ပြုပေးမှာပါ။ ဖြစ်လေ့ရှိတဲ့ ဆင်းတက်စ်အမှားတွေကို မကြာခင် တွေ ရပါမယ်။

Keywords

from, import, def , if စတာတွေဟာ keyword တွေဖြစ်တယ်။ Python ရေးတဲ့အခါ သူ့နေရာနဲ့ သူ အဓိပါယ်ကိုယ်စိန့် အသုံးပြုရတဲ့ စကားလုံးတွေဖြစ်တယ်။ from နဲ့ import ကို လိုက်ဘရှိ အင်ပိုလုပ်ဖို့ သုံးတယ်။ def ကို ဖန်ရှင် သတ်မှတ်တဲ့အခါ သုံးတယ်။ Python က သတ်မှတ်ထားတဲ့ နေရာတွေက လွှဲလို့ အခြားကိစ္စတော့အတွက် keyword တွေကို အသုံးပြုလို့ မရပါဘူး။ ဒါကြောင့် keyword တွေကို reserved word လိုလည်း ခေါ်တယ်။

main ဖန်ရှင်

‘Meet Karel’ ပရိုကရပ် အင်ပိုစတိတ်မန် အပြီးမှာ တွေ့ရတာကတော့ main ဖန်ရှင်သတ်မှတ်ချက်ပါ။ ကြည့်ရအဆင်ပြအောင် သူ့ချည်းသီးသန်း တစ်ဖြတ် ပြန်ပြပေးထားပါတယ်။

```
def main():
    """Karel code goes here!"""
    move()
    move()
    move()
    pick_beeper()
    turn_left()
    move()
    move()

    turn_left()
    turn_left()
    turn_left()

    move()
    put_beeper()
# End of main
```

ဖန်ရှင် (function) ဆိုတာ ကိစ္စတစ်ခု လုပ်ဆောင်ပေးဖို့အတွက် ယူနစ်တစ်ခုအနေနဲ့ ဖွဲ့စည်းထားတဲ့ ပရီဂရမ်ကုဒ် အစုအဝေးတစ်ခုပါပဲ။ ဖန်ရှင်ကို အသုံးပြုတဲ့အခါ ငါးရဲ့ လုပ်ငန်းတာဝန်အတိုင်း ဖန်ရှင် က လုပ်ဆောင်ပေးမှု ဖြစ်တယ်။ ဖန်ရှင်အသုံးပြုတာကို ‘ဖန်ရှင်ကောလ်’ (function call) လုပ်တာ လို့ ပြောတယ်။

main ဖန်ရှင်သတ်မှတ်ချက်ကို အပိုင်းနှစ်ပိုင်းခဲ့ ကြည့်နိုင်တယ်။ ပထမတစ်ပိုင်း

def main():

ကို ဖန်ရှင်ခေါင်းစီး (function header) လို့ခေါ်တယ်။ ဖန်ရှင်ခေါင်းစီးမှာ ဖန်ရှင်နံမည်နဲ့ ဖန်ရှင်ပါရာ မိတေတွေကို ပိုက်ကိုင်းထဲမှာ သတ်မှတ်ပေးရပြီး colon ‘:’ နဲ့ အဆုံးသတ်တယ်။ ဥပမာ x, y ပါရာ မိတေ နဲ့ myfun ဖန်ရှင် အတွက်

def myfun(x, y):

ပါရာမိတေမပါရောင်လည်း ပိုက်ကိုင်းအလွတ် တစ်နဲ့ ‘()’ တော့ပါရမယ်။ main ဖန်ရှင်မှာ ပါရာမိတေ မပါဘူး။ ပါရာမိတေတွေအကြောင်း နောက်ပိုင်းအခန်းတွေမှာ အသေးစိတ် လေ့လာရမှာပါ။ ကားခဲ့လ်ပရိုက်ရမ် တွေမှာ ပါရာမိတေအကြောင်း သိဖို့မလိုအသေးပါဘူး။ ပါရာမိတေ မလိုတဲ့ ဖန်ရှင်တွေပဲ တွေ့ရမှာပါ။

ဖန်ရှင်ခေါင်းစည်းအောက် ဒုတိယပိုင်းကတော့ main ဖန်ရှင် ကုဒ်ဘလောက် (code block) ပါ။ ကုဒ်ဘလောက် ဆိုတာ ကုဒ်တွေကို အုပ်စုတစ်စု အဖြစ် ဖွဲ့စည်းထားတာကို ဆုံးလိုတာပါ။ ဘလောက်လို့ လည်း ခေါ်တယ်။ ဘလောက်တစ်ခုမှာ ပါဝင်တဲ့ ကုဒ်လိုင်းတွေကို ညာဘက် ခွာရေးရပါမယ်။ အင်ဒန်ထုတ် (indent) လုပ်တာလို့ ခေါ်တယ်။ တက်သာကိုးတစ်ချက် (သို့) စပေါ်လေးခုစာ ခွာလေ့ရှိတယ်။ ဘော်ဒီ ပထမတစ်ကြောင်း

||||“Karel code goes here!||||

ကို docstring လို့ ခေါ်တယ်။ quote သုံးခုတဲ့ ‘“”’ တစ်နဲ့ကား ညာပ်ရေးတဲ့ ကွန်းမန်တစ်မျိုးလို့ ယူဆ နိုင်တယ်။ ဖန်ရှင်နဲ့ ပါတ်သက်တဲ့ ရှင်းလင်းဖော်ပြုချက်တွေ ရေးဖို့အတွက် သုံးတာပါ။

Docstring အောက်မှာ တွေ့ရတာကတော့ ကားခဲ့လ်ကွန်းမန်းတွေဆိုတာ သိပါလိမ့်မယ်။ ကားခဲ့လ် ကွန်းမန်းတွေဟာ stanfordkarel လိုက်ဘရဲ့ ဖန်ရှင်တွေပါ။ တနည်းအားဖြင့် stanfordkarel လိုက် ဘရီမှာ ကားခဲ့လ်ကွန်းမန်းတွေအတွက် ဖန်ရှင်သတ်မှတ်ချက်တွေ ပါဝင်တယ်။ ဖန်ရှင်တစ်ခုကို အသုံးပြုဖို့ အတွက် အဲဒီဖန်ရှင်ကို ခေါ်ပြုပါတယ်။ ဒါကို ဖန်ရှင်ကောလ် (function call) လုပ်တယ်လို့ ပြောတယ်။ ကားခဲ့လ်ကို ဘယ်ဘက်လုပ်လို့စေချင်ရင် turn_left ဖန်ရှင်ကောလ် လုပ်ရပါမယ်။ ဘိပါကောက်ခိုင်းချင် ရင် pick_beeper ဖန်ရှင်ကောလ် လုပ်ရပါမယ်။ ဖန်ရှင်ကောလ် လုပ်တဲ့ ပုံစံက

turn_left()

pick_beeper()

စသည်ဖြင့် ဖြစ်တယ်။

Python မှာ အင်ဒန်ထုတ်ကို ဖြစ်ကတတ်ဆန်း လုပ်လို့မရဘူး။ ဘေးမျဉ်းကနေ ခွာတဲ့ အကွဲအဝေး မ ညီတာနဲ့ ဆင်းတက်စံအမှား ဖြစ်တယ်။ မလိုတဲ့နောရာမှာလည်း ခွာရေးလို့ မရဘူး။ ခေါင်းစီးကို ဘေး မျဉ်းနဲ့ ခွာထားကြည့်ပါ။ အယ်ရာဖြစ်တာကို တွေ့ရမယ်။ အင်ပိုစတိတ်မန့်လည်း ဘေးမျဉ်းနဲ့ ကွာနေ လို့မရဘူး။ အခြား language တွေမှားလည်း အင်ဒန်ထုတ်လုပ် ရေးကြပေမဲ့ Python မှာလောက် မ တင်းကျပ်ဘူး။ အင်ဒန်ထုတ်မလုပ်လည်း ဆင်းတက်စံမှားတာ မဖြစ်ဘူး။

ကားရဲလ်ပရိုကရမ်တစ်ခုမှာ main ဟာ အထူးတာဝန်တစ်ခု လုပ်ဆောင်ပေးရတယ်။ အဲဒါကတော့ ပရိုကရမ်ဝင်းဒုးမှာ Run Program ခလုတ် (ပုံ ၁၅ မှာကြည့်ပါ) နိုပ်လိုက်ရင် တဲ့ပြန် လုပ်ဆောင်ပေးရတာပါ။ ဒါကြောင့် ကွန်မန်းတွေဟာ အဲဒီခလုတ် နိုပ်တော့မှုပဲ စအလုပ်လုပ်တာ ဖြစ်တယ်။

Entry Point

‘Meet Karel’ ပရိုကရမ်မှာ main ဖန်ရှင်နောက် အောက်ဆုံးအပိုင်းဟာ ပရိုကရမ် run တဲ့အခါ ပထမဆုံး စတင်လုပ်ဆောင်ပေးရမဲ့ ဖန်ရှင်ကို ဖော်ပြပေးတာပါ။ အန်ထရိုပိုင့် (entry point) လိုခေါ်တယ်။

```
if __name__ == "__main__":
    run_karel_program("meet_karel")
```

run_karel_program ဖန်ရှင်ဟာ ကားရဲပရိုကရမ် တစ်ခုအတွက် အန်ထရိုပိုင့် ဖြစ်တယ်။ ပရိုကရမ် တက်လာတဲ့ တစ်ပါတည်း ခေါ်တင်ချင်တဲ့ ကဲ့ကို ဒီဖန်ရှင်မှာ ထည့်ပေးတယ်။ meet_karel.w ကဲ့ကို စစျော်ခေါ်တင်ထားချင်ရင် "meet_karel" ထည့်ပေးရမယ်။ ကဲ့ဖိုင်မရှိရင် အယ်ရာတက်ပြီး ပရိုကရမ်ပွင့်လာမှာ မဟုတ်ဘူး။ ကဲ့မထည့်ပေးထားဘဲ ဒီလို

```
run_karel_program()
```

ဆိုရင် 8×8 အရွယ် ကဲ့ကို တင်ပေးပါတယ်။

ကားရဲလ်ကဲ့တစ်ခုကို လိုချင်တဲ့ပဲစဲ ဒီဇိုင်းဆွဲပြီး ဖိုင်နဲ့ သိမ်းထားရတာပါ။ ကဲ့ပုံစံချုတဲ့ ပရိုကရမ်လည်း ရှိတယ်။ ကဲ့ဖိုင်တွေက .w ဖိုင် အိပ်စာန်းရှင်းနဲ့ ဖြစ်တယ်။ ဒီစာအုပ်မှာပါတဲ့ ဥပမာတွေလေ့ကျင့်ခန်းတွေ အားလုံးအတွက် လိုအပ်တဲ့ ကဲ့တွေတွေကို အဆင်သင့်ပေးထားမှာပါ။ ကိုယ်ဟာကို လုပ်ဖို့ မလိုဘူး။ စိတ်ဝင်ထားရင် စမ်းကြည့်လိုရအောင် စာမျက်နှာ (၂၂၁) နောက်ဆက်တဲ့ (၁) မှာ အကျဉ်းဖော်ပြပေးထားပါတယ်။

၁.၃ ကားရဲလ် ပရိုကရမ် run ခြင်း

လိုအပ်တဲ့ဆော်ပဲတွေ ထည့်သွင်းနည်းကို စာမျက်နှာ (၂၂၁) နောက်ဆက်တဲ့ (၁) မှာ တစ်ဆင့်ချင်း ဖော်ပြပေးထားပါတယ်။ အခုက Python ပရိုကရမ်တစ်ခုကို အရိုးရှင်းဆုံး (လွယ်တယ်လို့ မဆိုလို) run လိုရတဲ့ နည်းကိုဖော်ပြပေးမှာပါ။ သဘောတရားပိုင်း နားလည်ဖို့ အထောက်အပံ့ဖြစ်မယ်။ အခုနည်းလမ်းကို အကြမ်းဖျော်း နားလည်အောင် ဖတ်ပြီးမှ နောက်ဆက်တဲ့ (၁) ကို ဖတ်စေချင်ပါတယ်။

မိုက်ခရိုဆော့ဖို့ ဝင်းဒုးမှာ Notepad ၊ အက်ပဲလ် မက်ခံအက်စဲမှာTextEdit စတဲ့ တက်စဲ အယ်ဒီတာတစ်ခုခုနဲ့ ပရိုကရမ်ကုံးရေးလိုရတယ်။ ကုံးဖိုင်ကို .py အိပ်စာန်းရှင်းနဲ့ သိမ်းရပါပယ်။ ပလိုန်းတက်စဲ (plain text) ဖိုင် ပါပဲ။ Python ကုံးဖိုင်မို့လို .txt အစား .py နဲ့ သိမ်းတာပါ။ Python ဖိုင်နံမည်ကို စာလုံးအသေးနှုပ် ပေးတဲ့ ထိုးစံရှိတယ်။ စပေါ်နေရာမှာ _ (underscore) သုံးတဲ့ ထုံးစံရှိတယ်။ ဒါကြောင့် ‘Meet Karel’ ပရိုကရမ်ကုံးရေးလိုက် meet_karel.py ဖိုင်မှာ သိမ်းသင့်တယ်။

Python နဲ့ ရေးထားတဲ့ ပရိုကရမ်ကို run မယဆိုရင် Python ဆော်ပဲရမှာပါ။ ဒီဆော်ပဲ အင်စတောလ် လုပ်နည်းကို နောက်ဆက်တဲ့ (၁) စာမျက်နှာ (၂၃၃) မှာ ဖော်ပြပေးထားပါတယ်။ Python ကုံးတွေကို ကွန်ပျူးတာက တိုက်ရှိက် နားမလည်ပါဘူး။ Python ဆော်ပဲတော့ Python ကုံးတွေကို တိုက်ရှိက် နားလည်ပြီး ကွန်ပျူးတာပေါ်မှာ run လိုရအောင် ကြားခံဆောင်ရွက်ပေးတဲ့ ဆော်ပဲလို ယေဘုယျအားဖြင့် ယူဆနိုင်တယ်။

Python ထည့်ပြီးရင် stanfordkarel လိုက်ဘရှိကို အောက်ပါကွန်မန်းနဲ့ အင်စတောလ် လုပ်ရပါမယ်။ အင်တာန်က်ပေါ်ကနေ ဒေါင်းလုဒ် လုပ်ရတာမျိုလို ကွန်နှုက်ရှင်ရှိရမယ်။

```
pip install stanfordkarel
```

ကားရဲလ်ပရိုကရမှာ ခေါ်တင်ချင်တဲ့ ကဗျာဖိုင်တွေလည်း ရှိရပါမယ်။ `meet_karel.w` ကဗျာဖိုင်က `meet_karel.zip` ဖိုင် `worlds` ဖိုဒါထဲမှာ ရှိပါတယ်။ <http://tinyurl.com/3mmmm9c7j> လင့်ကနေ `meet_karel.zip` ဖိုင်ကို ဒေါင်းလုပ်လုပ်ပါ။ ဒါ zip ဖိုင်ထဲက `worlds` ဖိုဒါကို `meet_karel.py` ဖိုင်ရှိတဲ့ နေရာမှာ ကော်ပိုက်းထည့်ပါ။ ဝင်းဒီးမှာ Command Prompt မက်ခဲ့အိုအက်စ်မှာ Terminal ဖွင့်ပြီး `cd` ကွန်မန်းနဲ့ ကုဒ်ဖိုင်ထားတဲ့ ဖိုဒါထဲကို သွားပြီး အောက်ပါအတိုင်း `python` ကွန်မန်းနဲ့ ကုဒ်ဖိုင်ကို run ပေးရပါမယ် (လာမဲ့စာမျက်နှာမှာ နှုန်းပြထားတာ ကြည့်ပါ)

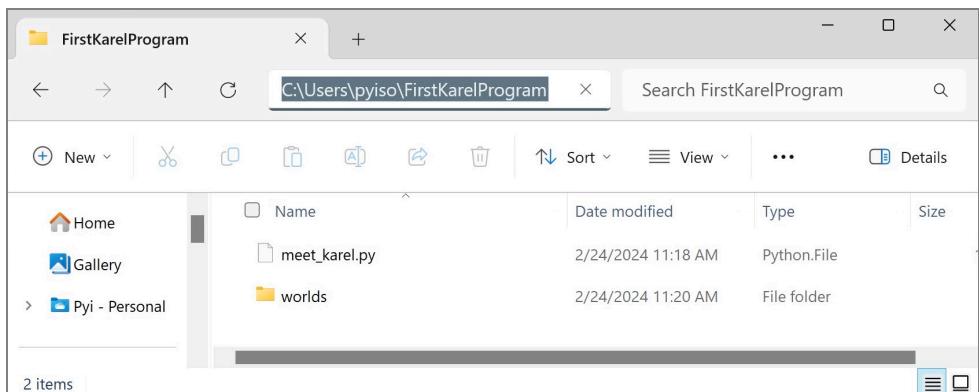
```
python meet_karel.py
```

ကုဒ်ရေးထားတာ ဆင်းတက်စ်အမှား မရှိဘူးဆိုရင် ကားရဲပရိုကရမ် ပွင့်လာမှာပါ။

အခုဖော်ပြုတဲ့က ကားရဲလ်ပရိုကရမ်တစ်ခု run ဖို့ မဖြစ်မနေလုပ်ရမဲ့ အနည်းဆုံးလိုအပ်ချက်ပါ။ Python ဆော်ဖို့ ရှိရမယ်။ `stanfordkarel` လိုက်ဘရဲ့ အင်စတောလ် လုပ်ရမယ်။ ကဗျာဖိုင်ပါတဲ့ `worlds` ဖိုဒါရှိရမယ်။ `.py` ဖိုင် တစ်ခုနဲ့ ပရိုကရမ်ကုဒ်ကို သိမ်းရမယ်။ `worlds` ဖိုဒါနဲ့ ကုဒ်ဖိုင်ကို တစ်နေရာ တည်းမှာ ထားရမယ်။ ပြီးရင် ကွန်မန်းလိုင်းမှာ

```
python your_karel_program.py
```

run ရှုပါပဲ။ `C:\Users\pyiso\FirstKarelProgram` ဖိုဒါထဲမှာ ကုဒ်ဖိုင်နဲ့ `worlds` ဖိုဒါကို ထားပြီး ဘယ်လို run ရလဲ နှုန်းပြထားတာကို ကြည့်ပါ။



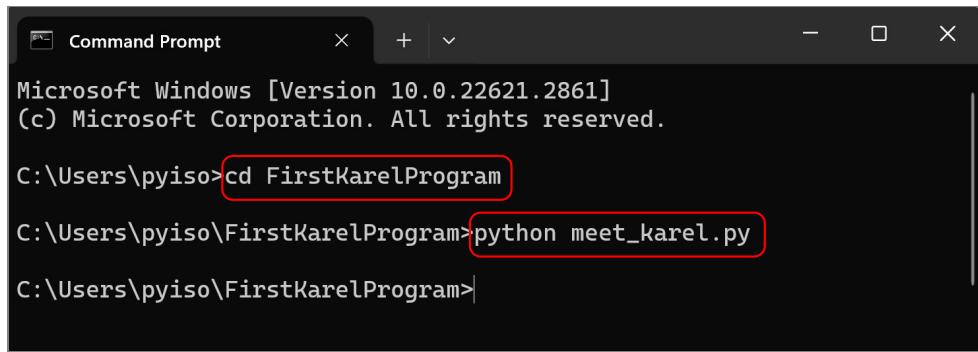
ပုံ ၁.၃

၁.၄ Python အင်စတောလ်လုပ်တဲ့အခါ ဘတွေပါလဲ

Python အင်စတောလ်လုပ်တယ်လို့ ယော်ယျု ပြောပေမဲ့ အင်စတောလ်လုပ်တဲ့အခါ တကယ်တမ်းက ပရိုကရမ်တစ်ခုတည်း ထည့်သွဲးပေးသွားတာ မဟုတ်ပါဘူး။ Python *interpreter* ပရိုကရမ်၊ Python *standard library* နဲ့ အခြားလိုအပ်တဲ့ ပရိုကရမ်တွေကို ထည့်ပေးသွားတာပါ။

Python Interpreter

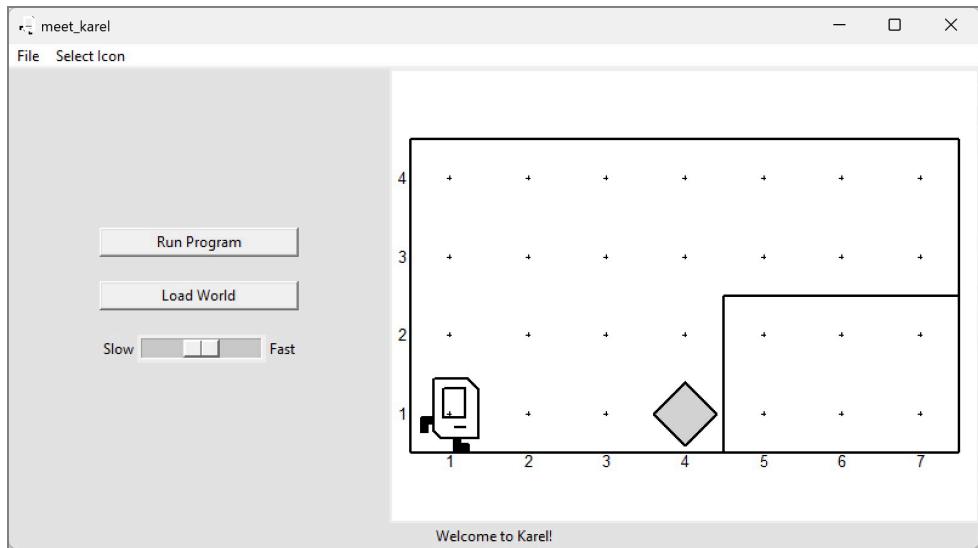
အင်တာပရ်တာ ဆိုတာ ပရိုကရမ်ကုဒ်တွေကို ဖတ်ပြီး ပရိုကရမ်ကုဒ်ထဲက ညွှန်ကြားချက်တွေအတိုင်း လုပ်ဆောင်ပေးတဲ့ ပရိုကရမ်ပါ။ Python အင်တာပရ်တာကတော့ Python နဲ့ရေးထားတဲ့ ကုဒ်တွေ ကို ဖတ်နိုင်တဲ့အပြင် ညွှန်ကြားထားတဲ့အတိုင်းလည်း လုပ်ဆောင်ပေးနိုင်ပါတယ်။ Python ကုဒ်တွေကို



```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pyiso>cd FirstKarelProgram
C:\Users\pyiso\FirstKarelProgram>python meet_karel.py
C:\Users\pyiso\FirstKarelProgram>
```

ပုံ ၁.၄



ပုံ ၁.၅

ကွန်ပျိုတာက တိုက်ရှိက် နားလည်တာ မဟုတ်ပါဘူး။ အင်တာပရက်တာကသာ တိုက်ရှိက်နားလည်တာပါ။ ငြင်းက တစ်ဆင့်ခံ၍ Python ကြုံတွေကို ကွန်ပျိုတာပေါ်မှ run ပေးရတာဖြစ်တယ်။ အင်တာပရက်တာကို python ကွန်မန်နဲ့ အသုံးပြု run ရပါတယ်။

`python meet_karel.py`

ဒီလို run တဲ့အခါ အင်တာပရက်တာက `meet_karel.py` ဖိုင်ထဲက ကုဒ်တွေကို ဖတ်ပြီး ပရိုကရမည့်နှင့်ကြားချက်တွေအတိုင်း လုပ်ဆောင်ပေးတာဖြစ်ပါတယ်။

Python Standard Library

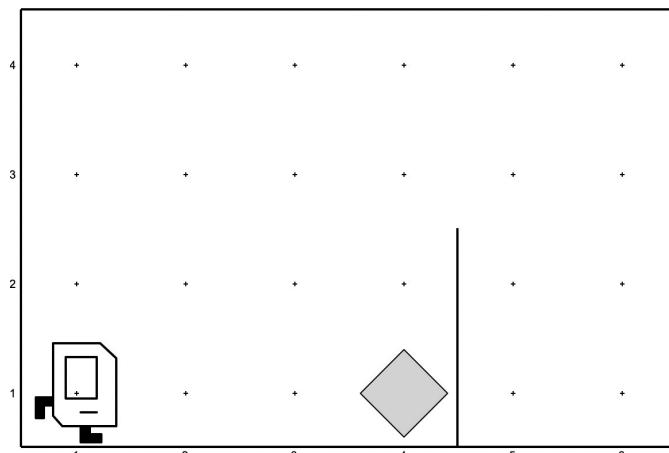
Python အင်စတောလုပ်တဲ့အခါ standard library လည်း တစ်ပါတည်း ထည့်သွေးပေးသွားတယ်။ Standard library မှာ ကွဲပြား၊ ပြားနားတဲ့ ရည်ရွယ်ချက်အမျိုးမျိုးအတွက် ကွန်ပိုးနှင့်တွေ့ အမြှေး ပါဝင်ပါတယ်။ Standard library ကွန်ပိုးနှင့်တွေ့ကို နောက်ပိုင်းမှာ အသုံးပြုကြရမှာပါ။ ငြင်းတိုက် အလျဉ်းသင့်သလို ဖော်ပြပေးသွားပါမယ်။

Standard အပြင် အဗြားရရှိနိုင်တဲ့ လိုက်ဘရှိတွေလည်း အများအဗြားရှိပါတယ်။ pip ဟာ လိုက်ဘရှိတွေ အင်စတောလ်လုပ်ဖို့ အသုံးပြုတဲ့ ပရိုကရမ်ဘစ်ခုပါပဲ။ Python documentation မှာတော့ pip ကို Python package တွေ အင်စတောလ် လုပ်ဖို့ အသုံးအများဆုံး tool (ထောက်ကူပြုပစ္စည်း) လို့ ဆိုထားပါတယ်။ Programming language တွေနဲ့ပါတ်သက်ပြီး မော်ချိုး (module)၊ ပက်ကော်ချုံ (package)၊ လိုက်ဘရှိ (library) စေတဲ့ အသုံးအနှစ်နဲ့တော့ တွေ့ရကြားရပါတယ်။ ဒီစကားလုံးတော့ အခေါ်အဝေါ်တွေရဲ့ အဓိပ္ပာယ်သတ်မှတ်ချက်ကလည်း သက်ဆိုက်ရာ programming language အလိုက် ကွားလေ့ရှိတယ်။ ဘိုင်နာအနေနဲ့ ဒါနဲ့ပါတ်သက်ပြီး သိပ်ပြီးခေါင်းရှုပ်ခံစရာ မလိုသေးပါဘူး။ ပရိုကရမ်ကုပ် သိမ်းဆည်း ဖွဲ့စည်း ထားသို့ ဖြန့်ဖြူးတဲ့ ကိစ္စတွေနဲ့ သက်ဆိုင်တဲ့ အခေါ်အဝေါ်တွေလို့ သိထားရင် လုံးလောက်ပါပြီ။ နောက်ပိုင်းမှာ ဒီအခေါ်အဝေါ်တွေနဲ့ သက်ဆိုင်တဲ့ အဓိပ္ပာယ်သတ်မှတ်ချက်တွေကို အလုပ်းသင့်ရင် သင့်သလို ဖော်ပြပေးပါမယ်။

၁.၅ Move Beeper to Other Side

ပရိုကရမ်းမင်း လေ့လာတဲ့ အခါ စာချည်းပဲ ဖတ်နေပြီး အမှန်တကယ် နားလည်သွားဖို့ဆိုတာ မဖြစ်နိုင်ပါဘူး။ လက်တွေ့ စမ်းသပ်ကြည့်၊ ရေးကြည့်မှုပဲ တကယ် နားလည်လာမယ်။ တကယ်လည်း ကျမ်းကျမ်းကျင်ကျင် ရေးတတ်လာမှာပါ။ ဒါကြောင့် လက်တွေ့ရေးကြည့်ပါ။ များများ လေ့ကျင့်ပါ။ ဥပမာတွေကိုလည်း နားလည် အောင် ဖတ်ပြီရင် မိမိဘာသာ အလွတ် ပြန်ရေးကြည့်ပါ။

```
if __name__ == "__main__":
    run_karel_program("move_beeper_to_other_side")
```



ပုံ ၁.၆

အခန်း J

ကားရဲလျှင့် တွန်ထရီးလ် စတိတ်မန်းများ

ကားရဲ့ပရိုဂရမ်တစ်ခု ဖွံ့ဖည်းတည်ဆောက်ပုဂ္ဂို ရှေ့အခန်းမှာ လေ့လာခဲ့ကြပြီး ကွန်ထရိုးလုံ စတိတ်မန် တွေ ဖြစ်ကြတဲ့ **for** loop, while loop, **if** နဲ့ **if...else** တို့ကို အခုခက်လက် လေ့လာကြပါမယ်။ ပရိုဂရမ်တစ်ခုကို run တဲ့အခါ ပရိုဂရမ်ကုတ်ထဲက ညွှန်ကြားချက်တွေအတိုင်း ကွန်ပျုံတာက လုပ်ဆောင်ပေးတာပါ။ ဒီလိုလုပ်ဆောင်ပေးတာကို အကွဲပော်ကြပါ။ (execute) လုပ်တယ်လို့ ခေါ်တယ်။ ကွန်ထရိုးလုံ စတိတ်မန် တွေဟာ ပရိုဂရမ်တစ်ခုခဲ့ execution flow ကို ထိန်းချုပ်ပေးတာ ဖြစ်တဲ့အတွက် control flow statements တော့လည်း ခေါ်ပါတယ်။

1.6 for loop

ကွန်လာရိုးလုပ်စတိတ်မနဲ့ တစ်မျိုးဖြစ်တဲ့ **for** loop ဟာ စတိတ်မနဲ့ တစ်ခု (သို့) စတိတ်မနဲ့ တစ်စုံကို သတ်မှတ်ထားတဲ့ အကြမ်အရေအတွက် ပြည့်အောင် ထပ်ခါထပ်ပါ ပြန်ကျေားပါတယ်။ move ကို နှစ် ဆယ့်ငါးကြိမ် ကျေားပေးဖို့ **for** loop နဲ့ အခြင်း

```
for i in range(25):  
    move()
```

ရေးရပါတယ်။ `put_beeper`, `move`, `turn_left` စတိတ်မနဲ့ သုံးကြောင်းတစ်စုကို အကြိမ်တစ်ရာ ကြောချင်ရင် ဒီလိပါ

```
for i in range(100):  
    put_beeper()  
    move()  
    turn_left()
```

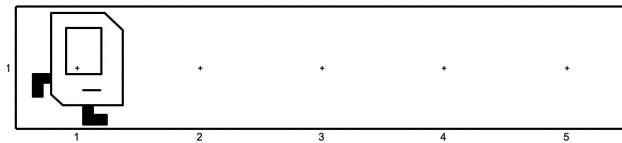
နှစ်ဆယ့်ငါးကြိမ်ကို $range(25)$, အကြိမ်တစ်ရာကို $range(100)$ စသည်ဖြင့် တွေ့ရတယ်။ ယခုအားဖြင့် အကြိမ်အရောတဲ့ N ကြိမ် ကျော်ချင်ရင်

```
for x in range(N):  
    statement1  
    statement2  
    statement3 etc.
```

ပုံစံနဲ့ သတ်မှတ်ရတာ။ N က အကြိမ်အရေအတွက်ကို ဖော်ပြတဲ့ ကိန်းပြည့်ကြန်းဖြစ်တယ်။ for loop ရေးတဲ့အခါ ကော်လံး : မကျွန်းခဲ့ဖို့ သတိပြုရပါမယ်။ x ဟာ ပေခိုရောလ်တစ်ခုဖြစ်ပြီး i , j , k စတဲ့ အကွားရာတစ်ခုနဲ့ ကိုယ်တားပြုလေ့ရှိတယ်။

ပြန်ကျော်စေချင်တဲ့ စတိတ်မန်တွေကို for ရဲ့ ညာဘက်ကို အင်ဒုံးထဲ လုပ်ပေးရပါမယ်။ အောက်ပါ အတိုင်းဆိုရင် put_beeper ကိုပဲ အကြိမ်တစ်ရာ လုပ်မှုပါ။ move နဲ့ turn_left ပြန်ကျော်မဲ့ မှာ မပါတော့ဘူး။

```
for i in range(100):
    put_beeper()
move()
turn_left()
```



ပုံ J.၁

ပုံ (J.၁) ကားရဲလ်ကဲ့မှာ ကွန်နာအားလုံးမှာ ဘိပါတစ်ခုစီ ချထားပေးရမယ်။ ကွန်နာဝါးခုရှိတာမျိုး လို့ စုစုပေါင်း ဘိပါဝါးခု ချထားရမှုပါ။ move ကတော့ လေးကြိမ်ပဲ လုပ်ရမယ် (ကားရဲလ်ရှေ့မှာ ကွန်နာ လေးခုပဲ ရှိတယ်)။ ဒီအတွက်ကို အခုလိုရေးနိုင်ပါတယ်။

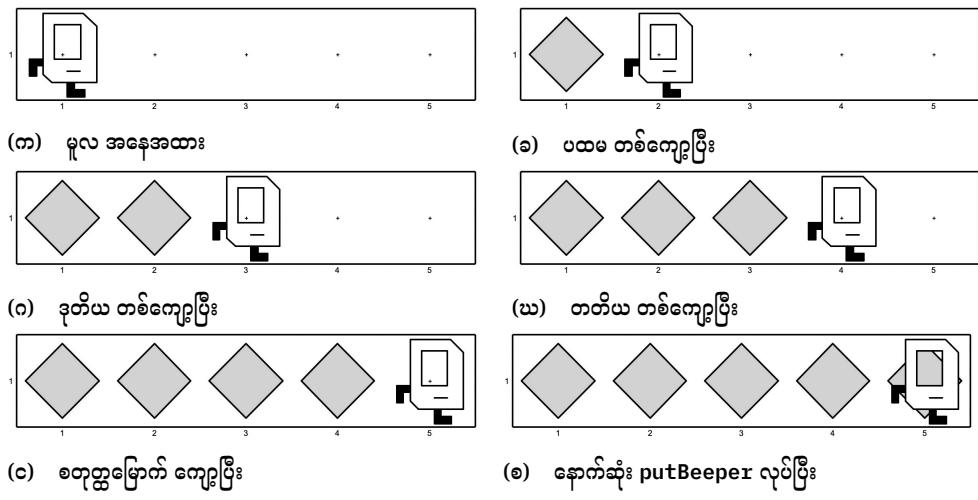
```
# File: make_row_of_5_beeper.py
from stanfordkarel import *

def main():
    for i in range(4):
        put_beeper()
        move()

    put_beeper()

if __name__ == "__main__":
    run_karel_program("make_row_of_5_beeper")
```

for loop ဟာ put_beeper နဲ့ move ကို လေးကြိမ်ကျော်ပေးမှုပါ။ တစ်ကြိမ်ပြီးတိုင်း ရှိနေမဲ့ အနေအထား တစ်ခုချင်းစီကို ပုံ (J.၂) မှာ တွေ့နိုင်ပါတယ်။ လေးကြိမ်ပေးမှုက နောက်ဆုံးတစ်ကျော်အပြီး ပုံ (J.၂) (c) မှာ ဘိပါတစ်ခုလုံးနေသေးတာ တွေ့ရမယ်။ for loop ပြီးတဲ့အခါ ပေါ်ပြုပါတယ်။ တစ်တန်းလုံးအပြည့်ဖြစ်သွားမှုပါ။



ဗို J-J

Off-by-one bug

ပြီးခဲ့တဲ့ ဥပမားမှု ဒါ၏ loop အပြီး put_beeper မလုပ်မိရင် ပရိုက်ရမ်ဟာ လိုချင်တဲ့ အတိုင်း မဖြစ်ပါဘူး။ ဘိပါတစ်ခုလိုနေမယ်။ ဒီလိုပြဿနာမျိုးဟာ အဖြစ်များတဲ့ အမှားဖြစ်ပြီး off-by-one bug လိုခေါ်ပါတယ်။ Loop သုံးတဲ့ အခါ ကြံးရလေ့ရှိတဲ့ ဖြစ်တတ်တဲ့ အမှား (bug) ဖြစ်တယ်။ သတ်မှတ်ထားတဲ့ အတိုင်း ဖြစ်သင့်တဲ့ အတိုင်း ပရိုက်ရမ်က အလုပ်မလုပ်ဘဲ ဖြစ်နေတဲ့ အမှားကို ကွန်ပျိုတာအသုံးအနှစ်းမှာ ပေးပို့လိုပါတယ်။ Bug ကိုလိုကြောပြီး မှန်အောင်ပြင်ပေးတာကိုတော့ debug လုပ်တယ်လိုခေါ်ပါတယ်။

ဘိပါတစ်ခုကျွန်းတဲ့ အမှားမျိုးဟာ off-by-one bug သာကေတစ်ခုမျှသာ ဖြစ်တယ်။ ခုနှစ်နဲ့ ဆယ့်သုံးကြား ကဏ္နား ခုနှစ်လုံးရှိပါတယ် (ခုနှစ်နဲ့ ဆယ့်သုံးအပါဝင် ဆိုလျှင်)။ $13 - 7 = 6$ လိုတွက်မိရင် တစ်လုံး လိုနေပါလိမ့်မယ်။ သုံးပေါ်း တစ်တိုင် အလျားပေသုံးဆယ်ရှိ တပြောင့်တည်း ခြေစည်းရှိးတစ်ခု အတွက် တိုင်စုစုပေါင်း တစ်ဆယ့်တစ်တိုင် လိုပါမယ်။ $30 \div 3 = 10$ လို တွက်ရင် မမှန်ပါဘူး။ တစ်ခု လိုတာ အပြင် တစ်ခုပုံးနေတာလည်း ဖြစ်တတ်တယ်။ စောနက ခြေစည်းရှိးမှာပဲ ထရံအချုပ်အရေအတွက် ကတော့ ကိုးချုပ်ဖြစ်ရမှာပါ။ $30 \div 3 = 10$ လို တွက်ရင် တစ်ခုပုံးနေပါမယ်။ ခုနှစ်နဲ့ ဆယ့်သုံးကို ထည့်မစည်းစားရင် ကြားမှာ ကဏ္နားဝါးလုံးရှိတာပါ။ $13 - 7 = 6$ လိုတွက်မိရင် တစ်လုံး ပို့နေပါလိမ့်မယ်။ ဒီ ဥပမာ အားလုံးဟာ အခြေခံသောတရားအားဖြင့် သိပ်မက္ခာ့ခြားတဲ့ off-by-one bug ပုံစံကွဲအမျိုးမျိုး ဖြစ်တယ်။

‘Make Row of Five Beepers’ ခုတိယ ဗားရှင်း

ပရိုက်ရမ်ရေးတဲ့ အခါ ပရိုက်ရမ်မာတွေ တစ်ယောက်နဲ့ တစ်ယောက် စဉ်းစားပဲ စဉ်းစားနည်း ဖြေရှင်းနည်း ထပ်တူကျေလေ့မရှိပါဘူး။ ဘိပါဝါးခဲ့ အတန်းလိုက် ချေပေးတဲ့ ပရိုက်ရမ်ကိုပဲ နောက်ဟာရှင်းတစ်မျိုးနဲ့ ရေးနိုင်ပါတယ်။

```
def main():
    put_beeper()
    for i in range(4):
        move()
        put_beeper()
```

for loop မစခင် put_beeper လုပ်ထားတာ၊ move နဲ့ put_beeper အစဉ်ပြောင်းသွားတာ (ပထမ ဗားရှုံးနဲ့ ပြောင်းပြန်ဖြစ်နေတာ) သတိထားကြည့်ပါ။

J.၂ အင်ဒန်ထုတ်လုပ်ခြင်းနှင့် ကုဒ်စာတရာ်ချာ

Python ဟာ အင်ဒန်ထုတ်လုပ်ခြင်းဖြင့် ပရိုဂရမ်ကုဒ် ဖွဲ့စည်းပဲ စထရက်ချာကို ဖော်ပြတဲ့ programming language ဖြစ်ပါတယ်။ Python ကုဒ်တွေကို ဘလောက် (block) တွေနဲ့ ဖွဲ့စည်းထားတယ်လို့ ရှုမြင် နိုင်တယ်။ ဘလောက်ဆိုတာ ကုဒ်တွေကို အုပ်စုတစ်စု အဖြစ် ဖွဲ့စည်းထားတာကို ဆိုလိုတာပါ။

```
def main():
    put_beeper()
    for i in range(4):
        move()
        put_beeper()
    pick_beeper()
    turn_left()
```

အခါ Python ကုဒ်မှာ အင်ဒန်ထုတ်လုပ်ထားဘဲ ဘယ်ဘက်စွန်း ကပ်ရေးထားတဲ့ **def main():** (ဖန် ရှင်ခေါင်းစည်း) ဟာ အပေါ်ခံးအဆင့် (top level) ဖြစ်တယ်လို့ ယူဆတယ်။ ဖန်ရှင်ခေါင်းစည်းအောက် အင်ဒန်ထုတ်လုပ်ထားတဲ့ ကုဒ်လိုင်းအားလုံးဟာ **main** ဖန်ရှင် ဘလောက်ထဲမှာ အကျိုးဝင်တယ်။ **pick_beeper** အထိပါတယ်။ **put_beeper** (ပထမတစ်ခု)၊ **for loop** နဲ့ **pick_beeper** တို့ဟာ ပထမအဆင့် အင်ဒန်ထုတ်ဖြစ်တယ်။

တစ်ခါ **for** အောက်မှာ ဒုတိယတစ်ဆင့် အင်ဒန်ထုတ်လုပ်ထားတဲ့ ကုဒ်လိုင်းအားလုံး **for** ဘလောက် ထဲမှာ အကျိုးဝင်တယ်။ **move** နဲ့ **put_beeper** ပါဝင်တယ်။ **pick_beeper** ကတော့ ပထမအဆင့် ပြန် ဖြစ်သွားတဲ့အတွက် **for** ဘလောက်ထဲမှာမပါဘူး။

အောက်ဆုံး **turn_left()** ကလည်း အင်ဒန်ထုတ်လုပ် မထားတဲ့အတွက် အပေါ်ခံးအဆင့်ပဲ။ **def main():** နဲ့ အဆင့်တူတာပေါ့။ **main** ဖန်ရှင်ဘလောက် အပြင်ဘက်မှာလို့ ယူဆရမယ်။

အင်ဒန်ထုတ်လုပ်ထားတဲ့ အဆင့်ကို ကြည့်ပြီး ဘလောက်တွေရဲ့ ဖွဲ့စည်းပဲကို ကွက်ကွက်ကွင်းကွင်း ထင်း ကနဲ့ မြင်နိုင်တယ်။ အပေါ်ကကုဒ်ကို ကြည့်တာနဲ့ **main** ဖန်ရှင်ထဲမှာ **for loop** ရှိတယ်။ **for loop** ထဲမှာ **move** နဲ့ **put_beeper** ပါဝင်တယ်ဆိုတာ သိသာတယ်။ **pick_beeper** ဟာ **for loop** အပြင်မှာ၊ **turn_left** ဟာ **main** အပြင်မှာ ဆိုတာကို အင်ဒန်ထုတ်လုပ်ထားတဲ့ အဆင့်က ဖော်ပြန်တယ်။

J.၃ while loop

အခြေအနေတစ်ရုပ် မှုန်နေသူ၏ စတိတ်မန်တွေကို တစ်ကြိမ်ပြီးတစ်ကြိမ် ပြန်ကျော့ လုပ်ဆောင်စေချင် ရင် **while** loop ကို အသုံးပြုပါတယ်။ **for** နဲ့ **while** loop နှစ်ခုလုံးက ပြန်ကျော့ပေးတာ ဖြစ်ပေါ့။ **for** ကို အကြိမ်အရေအတွက် အတိအကျသိတဲ့ကိစ္စမျိုးမှာ သုံးလေ့ရှိပြီး **while** ကိုတော့ ဘယ်နှစ်ကြိမ် လဲ ကိုတွေက်လို့မရတဲ့ အခြေအနေ အပေါ်မှုတည်ပြီး လုပ်ဆောင်ပေးရမဲ့ အကြိမ်အရေအတွက် ကွားဤော် နိုင်တဲ့ ကိစ္စမျိုးတွေမှာ သုံးလေ့ရှိတယ်။ ဥပမာတစ်ခုနဲ့ ကြည့်ရင် ပိုနားလည်ပါလိမ့်မယ်။

(၁) လမ်းပေါ်မှာ ဘိပါတစ်ခုရှိမယ်။ ဘိပါ ဘယ်ကွန်နာမှာရှိမှာလဲ၊ လမ်းဘယ်လောက်အရှည်ပြစ်မ လဲ ကိုတင်မသိဘူးလို့ ယူဆပါ။ နုံမှာကမ္မာ တစ်ခုကို ပဲ့ပါး (၂၃) မှာ တွေ့နိုင်တယ်။ ဘိပါကို သွားပြီး ကောက်ခိုင်းရပါမယ်။ အလားတူ မည်သည့်ကမ္မာအတွက်မဆို ဘိပါကောက်ပေးနိုင်ရမှာ ဖြစ်တယ်။ ဘိပါရှိ



ဗိုလ်ချုပ်

မူကွန်နာကို ကြိုတင်မသိထားတဲ့အတွက် ဘယ်နှစ်ကြိမ် move လုပ်ခိုင်းရမှုလဲ မသိဖြစ်နေတယ်။ အကြိမ် အရေအတွက် မသိတဲ့အတွက် for loop နဲ့ အဆင်မပြေတော့ပါဘူး။

ဘိပါမရှိသ၍ တစ်ကြိမ်ပြီးတစ်ကြိမ် move လုပ်ခိုင်းမယ်ဆိုရင် နောက်ဆုံးမှာ ဘိပါရှိတဲ့ကွန်နာကို ရောက်သွားမှာ သေချာပါတယ်။ အဲဒီလို လုပ်ခိုင်းဖို့ရာအတွက် while loop နဲ့ ရေးထားတာကို အခုလုံး တွေ့ရုပ်မယ်။

```
while no_beeper_present():
    move()
```

ကားရဲလ်ဟာ အခြေခံကွန်မန်း လေးခုအပြင် လက်ရှိ ကွန်နာမှာ ဘိပါရှိ/မရှိ၊ ရှေ့မှာ နံရုပ်တန်/မနေ့၊ အရောက်ကို ပျောက်နှုန်းမထား/မထား စတဲ့ သူနဲ့ (သို့) သူ့ရဲကွန်းနဲ့ သက်ဆိုင်တဲ့ အခြေအနေတစ်ရပ်ရပ် မှန်/မမှန် စစ်နိုင်ပါတယ်။ no_beeper_present က လက်ရှိ သူရှိနေတဲ့ ကွန်နာမှာ ‘ဘိပါ မရှိဘူးလား’ ဆိုတဲ့ အခြေအနေ စစ်ခိုင်းတာပါ။ လက်ရှိကွန်နာမှာ ဘိပါ ‘မရှိ’ ရင် ဒီအခြေအနေက မှန်တယ်လို့ ယူဆ ရမှုပေါ့။ အကယ်၍ ‘ရှိ’ နေရင်တော့ မှားတယ်လို့ ယူဆရမယ်။ အမှန် (သို့) အမှား ရလဒ်ထွက်မဲ့ ဒီလို မျိုး အခြေအနေစစ် ကွန်မန်းတွေကို ကွန်ဒီရှင် (condition) လို့ခေါ်ပါတယ်။

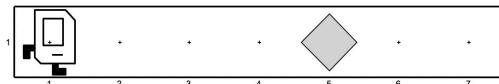
while loop အလုပ်လုပ်ပဲက ဒီလိုပါ။ no_beeper_present ကွန်ဒီရှင် စစ်ပါတယ်။ မှန်ရင် move လုပ်ပါတယ်။ ပြီးရင် ကွန်ဒီရှင်ပြန်စစ်တယ်။ မှန်ရင် move ကို နောက်ထပ်တစ်ကြိမ် ထပ်ကျော်ပါတယ်။ ကွန်ဒီရှင်စစ်လိုက်၊ မှန်ရင် တစ်ခါထပ်ကျော်လိုက်၊ ကွန်ဒီရှင်ပြန်စစ်လိုက်၊ မှန်ရင် နောက်တစ်ခါ ထပ်ကျော်လိုက်၊ ... ဒီအတိုင်းဆက်ပြီး ကွန်ဒီရှင်မှန်နေသ၍ while loop က move ကို ပြန်ကျော်ပေးမှုပါ။ ကွန်ဒီရှင် စစ်လိုက်လို့ မှားသွားပြီဆိုရင်တော့ ထပ်မကျော်တော့ဘဲ ရပ်သွားမှာဖြစ်တယ်။ while loop တစ်ကြိမ်ကျော်ပြီးတိုင်း ရှိနေမဲ့ အနေအထား တစ်ခုချင်းကို ပုံ (J.4) မှာ လေ့လာကြည့်ပါ။ လေးကြိမ်ပြောက် ကျော်ပြီးနောက် ကွန်ဒီရှင်စစ်လိုက်တဲ့အခါ မှားနေပြီ။ ဒီအခါ while loop က ထပ်မကျော်တော့ဘူး။ ဒီလို loop က ပြန်ကျော်နေတာ ရပ်သွားရင် loop ကနေ ထွက်တယ် (loop exits) လို့ ပြောလေ့ရှိတယ်။

အပြည့်အစုံ ဖော်ပြပေးထားတဲ့ ပရိုဂုဏ်ကို လေ့လာကြည့်ပါ။

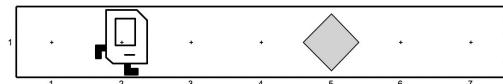
```
# File: go_pick_beeper.py
from stanfordkarel import *
```

```
def main():
    while no_beeper_present():
        move()

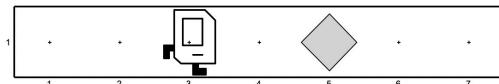
        pick_beeper()
```



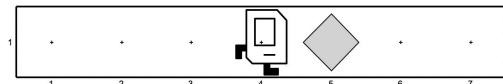
(က) မူလ အနေအထား



(e) පත්‍ර තෙවැනු ප්‍රිඩ්:



(က) ဒုတိယ တစ်ကျော်ပီး



(ബി) താൽക്കരി പിംബി



(c) മത്തുള്ളഭാവം ക്രൂഡിംഗ്

१९

```
if __name__ == "__main__":
    run_karel_program("go_pick_beeper")
```

ကားရဲလ် ကွန်ဒီရှင်များ

beepers_present ကွန်ဒီရှင် ကျတော့ လက်ရှိကွန်နာမှ ဘိပါရှိရင် အမှန်၊ မရှိရင် အမှား ရလဒ်ထဲက် တယ်။ no_beeper_present နဲ့ ဆန့်ကျက်သာက်ပေါ့။ ကားရဲလ် ကွန်ဒီရှင်တွေအားလုံးကို တေဘာ်လဲ (၂.၃) မှာ ကြည့်ပါ။ ပုံမှန်စစ်တာနဲ့ အငြင်းပုံစစ်တာ နှစ်မျိုးကို ယူပြုပြထားပါတယ်။

ကွန်ဒိရိုင်	သန်ကျင်ဘက် ကွန်ဒိရိုင်	စစ်ပေးသည့် အခြေအနေ
front_is_clear	front_is_blocked	ရှေ့မှာ နံရုံကပ်လျက် ရှိမရှိ
left_is_clear	left_is_blocked	ဘယ်ဘက်မှာ နံရုံကပ်လျက် ရှိမရှိ
right_is_clear	right_is_blocked	ညာဘက်မှာ နံရုံကပ်လျက် ရှိမရှိ
beepers_present	no_beeper_present	လက်ရှိကွုန်နာဥာ ဘိပါရှိမရှိ
beepers_in_bag	no_beeper_in_bag	ကားရုံလုံ၏ ဘိပါအပိုလ် ဘိပါရှိမရှိ
facing_north	not_facing_north	အရွှေဘက် မျက်နှာမှုလျက် ရှိမရှိ
facing_east	not_facing_east	အနောက်ဘက် မျက်နှာမှုလျက် ရှိမရှိ
facing_west	not_facing_west	တောင်ဘက် မျက်နှာမှုလျက် ရှိမရှိ
facing_south	not_facing_south	မြောက်ဘက် မျက်နှာမှုလျက် ရှိမရှိ

တေဘလ် J.C ကားရဲလ် စိန်နိုင်သည့် ကွန်ဒီဂိုင်များ

while loop සඳහා තැක්කීම්

while loop ରେ:ତେ ପୁଣ୍ଡିକ ଯେବୁବୁକୁଆହାଃଫ୍ରିଣ୍ ତିଲିପି॥

```
while condition :
    statement1
    statement2
    statement3 etc.
```

condition က ကားရဲလ်ကွန်ဒါရိုင် တစ်ခုဖြစ်တယ်။ ကော်လံ : မကျွန်ခဲ့အောင် ဂရိုစိုက်ပါ။ condition မှန်နေသေးသ၍ ပြန်ကျော့စေချင်တဲ့ စတိတ်မန်တွေကို while အောက်မှာ အင်ဒန်ထုတ်လုပ်ထား ရပါမယ်။ ရှုမှုရှင်းနေသ၍ move လုပ်တဲ့ while loop ကို အခုလို

```
while front_is_clear():
    move()
```

ရေးပါတယ်။

‘Make Beeper Row’ ဥပမာ

‘Make Row of Five Beepers’ ဥပမာမှာ လမ်းခွဲအလျားဟာ မပြောင်းလဲဘူး ယူဆတာမို့လို့ for loop ကို အသုံးပြုတာ ဆိုလျှင်ပါတယ်။ လမ်းခွဲအရှည်ကို ကြိုးမသိထားဘူး၊ တစ်လမ်းလုံး ဘိပါတွေ ဖြန့်ထားပေးရမယ်ဆုံးရင် while နဲ့ ရေးရမှာပါ။

```
# File: make_beeper_row.py
from stanfordkarel import *

def main():
    while front_is_clear():
        put_beeper()
        move()

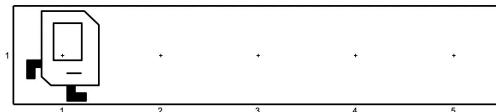
    put_beeper()

if __name__ == "__main__":
    run_karel_program()
```

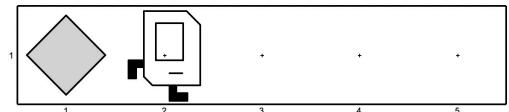
တစ်ကြိမ်ကျော့ပြီးတိုင်း ရှိနေမဲ့ အနေအထားကို ပုံ (၂၅) မှာ ကြည့်ပါ။ လေးကြိမ်မြောက်အပြီး front_is_clear စစ်တဲ့အခါ မှုးနေပြီဖြစ်လို့ ထပ်မကျော့တော့ဘဲ loop ကနေ ထွက်သွားမယ်။ ဒီအခါမှာ ဘိပါတစ်ခု လိုနေသေးတယ်။ put_beeper ထပ်လုပ်ရမယ်။ တစ်ခါပဲ လုပ်ရမှာပါ။ ဒါကြောင့် while loop ထဲမပါအောင် ပထမအဆင့် အင်ဒန်ထုတ်ပဲဖြစ်ရမယ်။

J.၄ if စတိတ်မန်

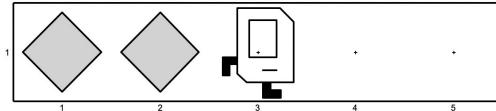
တစ်ခု (သို့) တစ်ခုထက်ပို့တဲ့ စတိတ်မန်တွေကို အခြေအနေတစ်ရပ် မှန်တော့မှဲပဲ လုပ်ဆောင်စေချင်တဲ့ အခါ if ကို အသုံးပြုနိုင်တယ်။ ဥပမာ



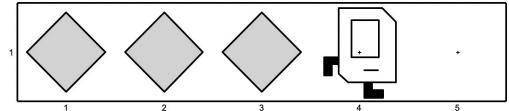
(က) မူလ အနေအထား



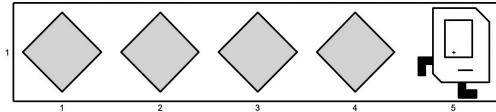
(ခ) ပထမ တစ်ကျော်ပြီး



(ဂ) ဒုတိယ တစ်ကျော်ပြီး



(ဃ) တတိယ တစ်ကျော်ပြီး

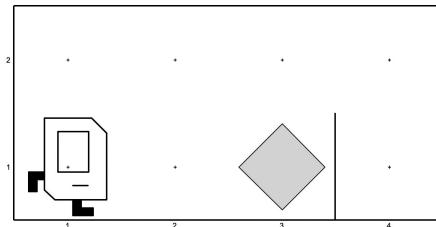


(င) စတုထွောက် ကျော်ပြီး

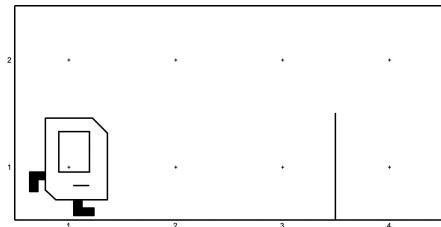
ပုံ J.၅

```
if beepers_present():
    pick_beeper()
```

if စတိတ်မန်ဟာ beepers_present ကွန်ဒိရှင် မှန်တော့မှုပဲ pick_beeper လုပ်မှာပါ။ မှားရင် မလုပ်ပါဘူး။
ပုံမှာတွေရတဲ့ ကဗျာနှစ်ခုထဲက တစ်ခုမှာ ကားရဲလှုံးနေမယ် ဆိုပါစို့။ ဘိပါရှိတဲ့ ကဗျာဆိုရင် ဘိပါကို



(က)



(ခ)

ပုံ J.၆

နံရံအခြားဘက် အောက်ခြေကို ရွှေ့ပေးရမယ်။ မရှိတဲ့ကဗျာဆိုရင် နံရံ ဒီဘက် အောက်ခြေမှာပဲ ရပ်နေရမယ်။ ပရိုက်မဲ့က ကဗျာနှစ်ခုလုံးမှာ မှန်အောင် အလုပ် လုပ်နိုင်ရပါမယ်။ ဒီအတွက် if စတိတ်မန် သုံးထားတာ လေ့လာကြည့်ပါ။

```
# File: move_beeper_to_other_side_if_any.py
from stanfordkarel import *
```

```
def main():
    move()
    move()
```

J.๒

```
if beepers_present():
    pick_beeper()
    turn_left()
    move()
    # turn_right
    turn_left()
    turn_left()
    turn_left()
    move()
    # turn_right
    turn_left()
    turn_left()
    turn_left()
    move()
    put_beeper()
    turn_left()

if __name__ == "__main__":
    run_karel_program("mbtos1")
```

if အောက်က စတိတ်မန့်တွေကို အင်ဒန်ထဲပေါ်ထားတာ သတိပြုပါ။ အဲဒီ စတိတ်မန့်အားလုံး beepers_present ဖြစ်မှ လုပ်ဆောင်မှာ ဖြစ်တယ်။ ဘိပါမရှိတဲ့ ကမ္ဘာမှာ စမ်းကြည့်ဖို့အတွက် Load World ခလုတ်နိုင်ပြီး ခေါ်တင်ပါ။ ဒါမှာဟုတ် ပရိုဂရမ်ကုဒ်မှာ "mbtos2" လိုပြင်ပြီး ပြန် run ပါ။
if စတိတ်မန့် ယေဘုယျစထရက်ချုပ်ကို အခုလိုတွေ့ရတယ်။

```
if condition :
    statement1
    statement2
    statement3 etc.
```

condition က front_is_clear, beepers_present စတဲ့ ကားရဲလ် ကွန်ဒါရှင် တစ်ခုခုဖြစ်မယ်။
တေဘာ်လ် (၂.၃) မှာ ကားရဲလ်ကို စစ်ခိုင်းလို့ရတဲ့ ကွန်ဒါရှင်အားလုံး ပြထားပါတယ်။

J.၃ if...else စတိတ်မန့်

အခြေအနေတစ်ရပ် မှန်တဲ့အခါ လုပ်ဆောင်ရမှာနဲ့ မှားတဲ့အခါ လုပ်ဆောင်ရမှာ မတူကဲပြားနေတဲ့အခါ if...else ကို သုံးပါတယ်။ ရှေ့က if စတိတ်မန့် ဥပမာ ပုံ (၂.၆) မှာ ဘိပါမရှိရင် နိုဂါးမူလနေရာ ကို ပြန်လာခိုင်းချင်တယ်ဆိုပါစို့။ if...else နဲ့ အခုလို ရေးရပါမယ်။

```
from stanfordkarel import *
```

```
def main():
    move()
    move()
```

```

if beepers_present():
    pick_beeper()
    turn_left()
    move()
    # turn_right
    turn_left()
    turn_left()
    turn_left()
    move()
    # turn_right
    turn_left()
    turn_left()
    turn_left()
    move()
    put_beeper()
    turn_left()

else:
    turn_left()
    turn_left()
    move()
    move()
    turn_left()
    turn_left()

```



```

if __name__ == "__main__":
    run_karel_program("mbtos2")

```

if...else စတိတ်မန့်က ကွန်ဒီရှင်မှန်ရင် if ဘလောက်ကိုလုပ်ဆောင်ပေးပြီး ကွန်ဒီရှင်မှားရင်
တော့ else ဘလောက်ကို လုပ်ဆောင်ပေးဘာပါ။ ယေဘုယျပုံစံက ဒီလိုပါ

```

if condition :
    statement1a
    statement2a
    statement3a etc.

else:
    statement1b
    statement2b
    statement3b etc.

```

J.၆ Nested ကွန်ထရီးလ် စတိတ်မန်များ

ကွန်ထရီးလ် စတိတ်မန် ဘလောက်ထဲမှာ ကွန်ထရီးလ် စတိတ်မန် ရှိလိုက်ယူ။ Nested ကွန်ထရီးလ်
စတိတ်မန်လို့ ခေါ်ပါတယ်။

Nested for loop

ဘိပါသုံးချခုလိုက် ရှေ့တိုးလိုက် လုပ်တာကို လေးကြိမ်ကျော်ချင်ရင် for loop နဲ့ အခုလို ရေးရမယ်ဆိုတာ သိခဲ့ပြီးပါပြီ။

```
for i in range(4):
    put_beeper()
    put_beeper()
    put_beeper()
    move()
```

ဒါ for loop ထဲက put_beeper သုံးကြောင်းကိုလည်း နောက်ထပ် for တစ်ခုနဲ့ ရေးလို ရရှိမှာပေါ့။ ဒီလို ဖြစ်သွားမှုပါ

```
for i in range(4):
    for j in range(3):
        put_beeper()
        move()
```

for loop နှစ်ခု ဆင့်ရေးထားလို nested for loop လိုခေါ်ပါတယ်။ Loop တစ်ခုချင်းကို သီးခြားရည်ညွှန်းချင်တဲ့အခါ အပြင် for loop နဲ့ အတွင်း for loop လို ပြောလေ့ရှိတယ်။ အင်ဒန်ထု လုပ်ထားပုံအရ အပြင် for loop ဘလောက်ထဲမှာ အတွင်း for loop နဲ့ move ပါဝင်တယ်။ အတွင်း for loop ဘလောက်ထဲမှာတော့ put_beeper ပဲပါတယ်။ move မပါ ပါဘူး။ အပြင် for loop မှာ ပေရီရောကဲလို i ဆိုရင် အတွင်းမှာ i မဖြစ်ရပါဘူး။ j, ဒါမှုမဟုတ် k မတူတာ တစ်ခုခုဖြစ်ရပါမယ်။

အတန်းလိုက် ကွန်နာ ငါးခုမှာ တစ်ကွန်နာ ဘိပါ (၂၂) ခုစီ ချထားပေးဖို့ nested loop နဲ့ ရေးထားတာကို လေ့လာကြည့်ပါ။

```
# File: row_of_beeper_piles.py
from stanfordkarel import *

def main():
    for i in range(4):
        for j in range(25):
            put_beeper()
            move()

    for i in range(25):
        put_beeper()

if __name__ == "__main__":
    run_karel_program("5x1")
```

Nested for and while

ကဗျာပါတ်လည် ဘိပါခြီးစည်းရှိုး ခတ်ပေးဖို့အတွက် for နဲ့ while nest လုပ်ထားတာကို လေ့လာ ကြည့်ပါ။ (၁, ၁) ကွန်နာမှာ အရှေ့ဘက်လည့် အနေအထားကနေ စမယ်လို့ ယူဆပါ။ ကဗျာအရွယ်အစား အမျိုးမျိုးကို ခြိုစည်းရှိုး ခတ်ပေးနိုင်ပါမယ်။

တောင်ဘက် နံရုံတလျောက် ဘိပါတွေချွေားမယ် (နောက်ဆုံး ကွန်နာမှာ ဘိပါမချွဲ ချိန်ထားမယ်)။ ပြီးရင် အရှေ့ဘက် နံရုံအတွက် အဆင်သင့်ဖြစ်အောင် ဘယ်ဘက်လှည့်မယ်။ ပုံ (၂.၇) (က) နဲ့ (ခ) ကို ကြည့်ပါ။ အရှေ့ဗြို့မြောက် နဲ့ အနောက်ဘက် နံရုံတွေအတွက်လည့် ဒီအတိုင်း လုပ်သွားရှုပါပဲ။ ပုံ (၂.၇) (က)၊ (ယ)၊ (ခ) အသီးသီးကို ကြည့်ပါ။

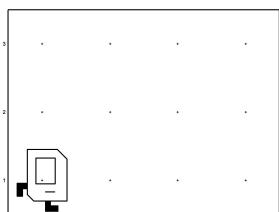
နံရုံတစ်ဘက် အတွက် ဆိုရင် အခုလို

```
while front_is_clear():
    put_beeper()
    move()
    turn_left()
```

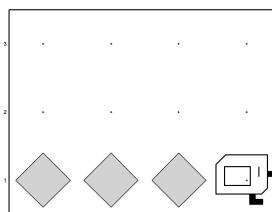
ဖြစ်မယ်။ (while ဘလောက်မှာ turn_left မပါတာ ဂရုပြုပါ)။ နံရုံလေးဘက်အတွက် လေးကြိမ် လုပ် ရမှာဆိုတော့ for နဲ့ အခုလို

```
# File: beeper_fence.py
for i in range(4):
    while front_is_clear():
        put_beeper()
        move()
        turn_left()
```

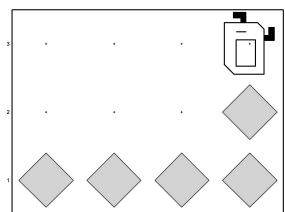
ရေးရပါမယ်။



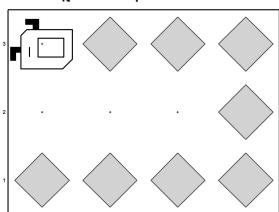
(က) မူလ အနေအထား



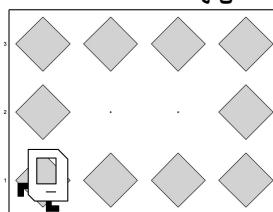
(ခ) ပထမ တစ်ကျော်ပြီး



(ဂ) ခုတိယ တစ်ကျော်ပြီး



(ယ) တတိယ တစ်ကျော်ပြီး



(ဇ) စုတို့မြောက် ကျော်ပြီး

ပုံ ၂.၇

ပရီဂရမ် တစ်ခုလုံး အတွက်ဆိုရင် အခုလိုပါ

```

# File: beeper_fence.py
from stanfordkarel import *

def main():
    for i in range(4):
        while front_is_clear():
            put_beeper()
            move()
            turn_left()

if __name__ == "__main__":
    run_karel_program("4x3")

```

Nested while and if

while နဲ့ if nest လုပ်လိုလည်း ရတာပေါ့။ လမ်းပေါ်မှာ ဘိပါတွေက ကြံရာကျပ်း ရှိနေမယ်။ ဘိပါတွေကို အမိုက်တွေလို ယူဆပြီး ရှင်းပေးရပါမယ်။ လမ်းအလျားကို ကြံမသိဘူး၊ ကွန်နာတစ်ခုမှာ ဘိပါတစ်ခုထက် ပိုမရှိနိုင်ဘူးလို ယူဆပါ။ လမ်းအဆုံးထိ while loop နဲ့ ရွှေ့ခိုင်းလို ရမယ်။ ရောက်တဲ့ ကွန်နာတိုင်းမှာ ဘိပါရှိရင် ကောက်ခိုင်းရပါမယ်။ if သုံးရမယ်။

```

from stanfordkarel import *

def main():
    while front_is_clear():
        if beepers_present():
            pick_beeper()
            move()

        if beepers_present():
            pick_beeper()

if __name__ == "__main__":
    run_karel_program("clean_the_street")

```

if စတိတ်မန့် နဲ့ move ကို while loop ရဲ့ ဘလောက်ထဲမှာ ထည့်ပြီး ရှုမှုရှင်းနေသူ၏ ပြန်ကျော့ခိုင်းထားတာပါ။ if စတိတ်မန့်က ဘိပါရှိတော့မှုပဲ ကောက်ပေးဖို့အတွက်။

ပြီးခဲ့တဲ့ဟာနဲ့ အလားတူတဲ့ နောက်ထပ် ဥပမာ တစ်ခုကတော့ လမ်းတစ်လျှောက် ကွန်နာတွေမှာ ဘိပါရှိရင် ကောက်ခိုင်းပြီး မရှိရင် တစ်ခုချေပေးရပါမယ်။ တန်ည်းအားဖြင့် ကွန်နာတစ်ခုချင်းရဲ့ ဘိပါရှိ/မရှိ အခြေအနေကို ဆန့်ကျင်တက် ပြောင်းတာပေါ့။

```
# File: toggle_beeper.py
from stanfordkarel import *

def main():
    while front_is_clear():
        if beepers_present():
            pick_beeper()
        else:
            put_beeper()
        move()

    if beepers_present():
        pick_beeper()

if __name__ == "__main__":
    run_karel_program("toggle_beeper")
```

if...else သုံးထားတယ်။ ကျွန်ုတာအားလုံး လမ်းရှင်းတဲ့ ပရိုကရမ်နဲ့ တူတူပဲ။

Nested if

အခြေအနေ တစ်ရပ် မှန်တော့မှုပဲ လုပ်ချင်ရင် if ကို သုံးတယ်။ အခြေအနေ 'နှစ်ရပ်' လုံးနဲ့ ကိုက်ညီမှ လုပ်ဆောင်စေချင်တဲ့အခါ nested if သုံးနိုင်ပါတယ်။ ညာဘက်လည်းရှင်း လက်ရှိကွန်နာမှာလည်း ဘို့မရှိမှ ဘိပါတစ်ခု ချထားခိုင်းမယ်ဆိုပါစို့။ အခုလိုရေးနိုင်ပါတယ်

```
if right_is_clear():
    if no_beeper_present():
        put_beeper()
```

ညာဘက်မှာ ရှင်းနေမှ အပြင် if က အတွင်း if ကို လုပ်ဆောင်စေမှာပါ။ အတွင်း if ကလည်း ဘိပါမရှိမဲ့ ပဲ put_beeper လုပ်မှာပါ။ ညာဘက်မှာ ပိတ်နေရင်သော်လည်းကောင်း ဘိပါရှိနေရင်သော်လည်းကောင်း pick_beeper လုပ်မှာ မဟုတ်ပါဘူး။ right_is_clear နဲ့ no_beeper_present အခြေအနေ နှစ်ခုလုံး မှန်မှုပဲ လုပ်မှာပါ။

ပုံ (၂.၈) (က) မှာ ဒုတိယ လမ်းတစ်လျှောက် လမ်းပြင်တဲ့အလုပ်ကို ကားရဲလှုပိုကို တာဝန်ပေးတယ် လို့ ယူဆပါ။ ဘိပါရော ညာဘက်နဲ့ရုံးပါ မရှိတဲ့ ကွန်နာတွေက လမ်းအခြေအနေ တော်တော်ဆိုးနေတဲ့ နေရာ တွေ။ ဒီလိုနေရာတွေမှာ ဘိပါတစ်ခု ဖြည့်ပြီး လမ်းပြင်ပေးရမှာပါ။

```
# File: repair_street.py
from stanfordkarel import *

def main():
    while front_is_clear():
        if right_is_clear():
            if no_beeper_present():
```

```

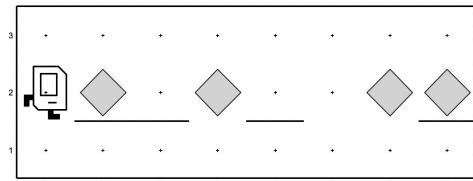
        put_beeper()
move()

if right_is_clear():
    if no_beeper_present():
        put_beeper()

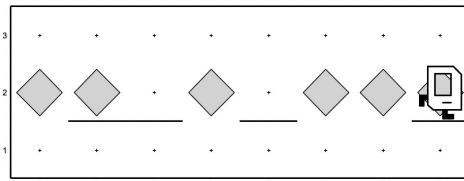
if __name__ == "__main__":
    run_karel_program("repair_street")

```

while ဘလောက်ထဲမှ nested if နဲ့ move ပါဝင်တယ်။ nested if က ညာဘက်လည်ရှင်း ဘိပါလည်းမရှိမှ လက်ရှိကွန်နာမှာ ဘိပါတစ်ခု ချခိုင်းထားတာပါ။ while ထဲမှာ if၊ အဲဒီ if ထဲမှာမှ နောက်ထပ် if တစ်ခု ဆင့်ထားတဲ့အတွက် သုံးဆင့် nest လုပ်ထားတာပါ။ အင်ဒန်ထု လုပ်ထားတော့ ဂရုစိက်ကြည့်ဖို့ လိုတယ်။ ဘယ်ဘလောက်ထဲမှာ ဘာရှိနေလဲဆိုတာ မြင်အောင် လုပ်ရမှာပါ။ ကိုယ်တိုင် ရေးရင်လည်း အင်ဒန်ထု ဂရုစိက်ပြီး လုပ်ရပါမယ်။ လမ်းပြင်ပြီး အခြေအနေကို ညာဘက် ပုံ (J.7) (ခ) တွင် ကြည့်ပါ။



(က) မူလ အနေအထား



(ခ) လမ်းပြင်ပြီး အနေအထား

ပုံ J.7

အခန်း ၃

ဖန်ရှင်များ (Functions)

ဖန်ရှင် (function) တွေဟာ ပရီဂရမ်းမင်းမှာ အရေးကြီးဆုံး အခြေခံသဘောတရားတစ်ခု ဖြစ်တယ်။ ဖန်ရှင်ဆိုတာ ဘာလဲ၊ ဘာကြောင့် အရေးပါရတာလဲ၊ ဖန်ရှင်တွေကို ပရီဂရမ် ဒီဇိုင်းပြုလုပ် ရေးသားတဲ့ အခါ ဘယ်လိုအသုံးချုပ်တဲ့ စတာတွေကို ဒီအခန်းမှာ လေ့လာကြပါမယ်။

၃.၁ ဖန်ရှင် သတ်မှတ်ခြင်း

ညာဘက် လှည့်ခိုင်းချင်တိုင်း turn_left သုံးခါရေးနေရတာ ရေရှည်အဆင်မပြေပါဘူး။ turn_right လို့ပဲ တိုက်ရိုက် ရေးလို့ရရင် ပိုပြီးတော့ အဆင်ပြေမှုပါ။ ဒီလို လိုအပ်ချက်မျိုးကို ဖြည့်ဆည်း ပေးဖို့အတွက် ဟာ ဖန်ရှင်တွေရဲ့ အဓိက ရည်ရွယ်ချက်တွေထဲက တစ်ခုဖြစ်တယ်။ turn_right ဖန်ရှင်ကို အခုလို သတ်မှတ်နိုင်ပါတယ်။

```
def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

ဒီလို သတ်မှတ်ထားပြီးရင် ညာဘက်လှည့်ချင်တဲ့အခါ

```
turn_right()
```

လို တိုက်ရိုက်ပြေလို ရသွားမှာ ဖြစ်ပါတယ်။ turn_left သုံးခါ ရေးဖို့ မလိုတော့ပါဘူး။

ဖန်ရှင်တစ်ခု သတ်မှတ်တယ် (defining a function) ဆိုတာ ကိစ္စတစ်ခု ဖြေရှင်းဆောင်ရွက်ဖို့ အတွက် စတိတ်မန့်တွေကို ယူနစ်တစ်ခုအဖြစ် ဖွဲ့စည်းထားလိုက်တာပါပဲ။ ငှါးယူနစ်အတွက် အမည်တစ်ခု ကိုလည်း သတ်မှတ်ပေးတယ်။

ဖန်ရှင် သတ်မှတ်မယ် ဆိုရင် def keyword သုံးရပါတယ်။ အထက်ပါ turn_right ဖန်ရှင် သတ်မှတ်ချက် (function definition) မှာ

```
def turn_right():
```

ကို ဖန်ရှင် ဟက်ဒေါ (function header) လို ဆော်တယ် (မြန်မာလို ဆိုရင်တော့ ဖန်ရှင် ခေါင်းစည်း ပေါ့)။ turn_right က ဖန်ရှင် အမည်။ ပါရာမီတာပါတဲ့ ဖန်ရှင်ဆိုရင် ပိုက်ကွင်းထဲမှာ ပါရာမီတာတွေ

သတ်မှတ်ရတယ်။ ဥပမာ (x, y)။ ပါရေးမီတာ မပါရင်တော့ () ပဲဖြစ်မယ်။ ကားရဲလ်မှာ ဖန်ရှင်အားလုံးဟာ ပါရေးမီတာ မပါတဲ့အတွက် () ပဲ ဖြစ်မှာပါ။ ဖန်ရှင်ဟက်ဒီ လိုင်းအဆုံးမှာ ကော်လံ ‘:’ ထည့်ပေးဖို့ လိုပါတယ်။

ဖန်ရှင် ဟက်ဒီအောက် အင်ဒန်ထုပ်ထားတဲ့ လိုင်းအားလုံးဟာ ငှင်းဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ကုံးဘေးကြီးတွေတဲ့ ဖြစ်တယ်။ အခု turn_right ဖန်ရှင် ဘလောက်မှာ turn_left သုံးကြိုးမြှင့်ပါတယ်။

| turn_right()

လုပ်ခိုင်းတာက (သတ်မှတ်ထားတဲ့) ဖန်ရှင်ကို အသုံးပြုတာ ဖြစ်တယ်။ ဒီအခါမှာ ငှင်းဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဖန်ရှင်ကို အသုံးပြုတာကို ဖန်ရှင်ကောလ် (function call) လုပ်တယ်လို့ ပြောပါတယ်။ (မြန်မာလို့တော့ ‘ဖန်ရှင်ခေါ်’ တယ်လို့ ပြောတာပေါ့)။

ဖန်ရှင်သတ်မှတ်တာနဲ့ ဖန်ရှင်ကောလ် လုပ်တဲ့ပုံစံကို အောက်ပါအတိုင်း ယော့ယျအားဖြင့် တွေ့ရပါမယ်။ Python ထုံးစံအရ ဖန်ရှင်နံမည်မှာ စာလုံးအသေးကိုပဲ သုံးလေ့ရှိတယ်။ စကားလုံး နှစ်ခုနဲ့ အထက်ဆိုရင် ကြားမှာ underscore (_) ခြားပေးလေ ရှိတယ်။

def name_of_function():
 statement₁
 statement₂
 statement₃ etc.

| name_of_function()

ဖန်ရှင်နံမည် အဓိပ္ပာယ် အရေးကြီးပါတယ်

ဘရေးတာပဲဖြစ်ဖြစ် ပရိုဂရမ်ကုဒ် ရေးတာပဲဖြစ်ဖြစ် စိတ်ထ တွေးတဲ့အတိုင်း စဉ်းစားတဲ့အတိုင်း ပေါ်လုပ်အောင် ဖော်ပြနိုင်တာဟာ အားသာချက်တစ်ခုပါပဲ။ ဖန်ရှင် သတ်မှတ်ထားခြင်း အားဖြင့် ညာဘက်လှည့်ခိုင်းရင် turn_right ဘိပါ နှစ်ဆယ့်ငါးခု ချရင် put_25_beeper တိုက်ရိုက် ဖော်ပြုလို့ ရတာဟာ အရေးပါတဲ့ ကိစ္စဖြစ်ပါတယ်။ ဘာသာစကားတစ်ခုရဲ့ ဖော်ပြနိုင်စွား ‘အား’ (expressive power) ကို ထပ်လောင်းအားဖြည့်ပေးတာလို့ ဆိုရမှာပါ။

ဖန်ရှင်လုပ်ဆောင်ပေးတဲ့ ကိစ္စကို သိသာစေမဲ့ နားလည်ရလွှာယ်မဲ့ နံမည်ပျိုး ဂရုစိုက်ရွေးချယ်တာက လည်း အရေးပါပါတယ်။ ကားရဲလ်ပရိုဂရမ်တွေ့မှာ အမိန့်ပေးခိုင်းစေတဲ့ ပုံစံနဲ့ ဖန်ရှင်နံမည်ပေးလေ့ရှိတယ်။ ဥပမာ turn_north, pick_all_beeper ။

ဖန်ရှင်သတ်မှတ်ချက်နဲ့ ဖန်ရှင်ကောလ် ဥပမာ တရာ့ကို လေ့လာကြည့်ပါ။ ဘိပါ နှစ်ဆယ့်ငါးခု ချပေးတဲ့ put_25_beeper ဖန်ရှင်ပါ

def put_25_beeper():
 for i in range(25):
 put_beep()

put_25_beeper()

ဒါကတော့ ကွန်နာတစ်ခုမှာ ရှိတဲ့ ဘိပါအားလုံးကောက်ပေးတဲ့ ဖန်ရှင်ဖြစ်ပါတယ်

```

def pick_all_beeper():
    while beepers_present():
        pick_beeper()

pick_all_beeper()

```

ဖန်ရှင် အသုံးပြုတဲ့အခါ ကိစ္စတစ်ခုကို ပိုပြီး အလယ်တကူ လုပ်လိုက်သွားတယ်။ ဘိပါအားလုံး ကောက် မယ်ဆုံးရင် pick_all_beeper ဖန်ရှင်ခေါ်လိုက်ရမဲ့။ ဘိပါ နှစ်ဆယ့်ငါးခု ချချင်ရင် sum_25_beeper ဖန်ရှင်ခေါ်လိုက်ရမဲ့။ ဖန်ရှင်တစ်ခုကို အသုံးပြုတဲ့အခါ အဲဒီဖန်ရှင်ကို ဘယ်လိုသတ်မှတ်ထားလဲ ပြန် စဉ်းစားနေဖို့ မလိုပါဘူး။ ဥပမာ ...

အခန်း (၁) မှာ လိုက်ဘရီဆိုတာ ဘာလဲ အကျဉ်း ဖော်ပြခဲ့တယ်။ မေ့ထရစ် $A \times B$ မြှောက်လဒ် $A \times B$ ကို numpy လိုက်ဘရီ ဖန်ရှင် matmul နဲ့ အဖြော်ရွှေ့ပါတယ်။

```
result = matmul(A, B)
```

matmul ဖန်ရှင်ကို လိုက်ဘရီ ထုတ်လုပ်တဲ့ ပညာရှင်တွေက ရေးပေးထားတာ။ အဲဒီဖန်ရှင်ကို ဘယ်ပုံ ဘယ်နည်း သတ်မှတ်ထားတယ်ဆိုတာ အသုံးပြုသူအတွက် အရေးမကြီးဘူး။ မေ့ထရစ်တွေကို ဒီဖန်ရှင်နဲ့ မြှောက်လိုက်ရတယ်ဆိုတာ သိရင် သုံးလိုရနေတာပါပဲ။ အခြား မေ့ထရစ် လုပ်ထုံးလုပ်နည်း ဖန်ရှင်တွေလည်း numpy လိုက်ဘရီမှာ ပါဝင်ပါတယ်။ မိမိ လုပ်ချင်တဲ့ မေ့ထရစ် လုပ်ထုံးလုပ်နည်းအတွက် ဖန်ရှင်ကို သိရင် (လိုက်ဘရီ documentation တ်ပြီး ရှာလိုရတယ်) လိုအပ်တဲ့အခါ ခေါ်သုံးလိုက်ရမဲ့ပါပဲ။ ပရှိကရမ် တွေ တည်ဆောက်ရာမှာ လိုက်ဘရီတွေ မရှိမဖြစ်လိုအပ်တယ်။ ဖန်ရှင်တွေဟာ လိုက်ဘရီတွေရဲ့ အမိက အစိတ်အပိုင်းတွေ ပဲဖြစ်ပါတယ်။

ဖန်ရှင် အခြေခံအုတ်ချုပ်များ

pick_all_beeper ဖန်ရှင်ကို အခြေခံအုတ်ချုပ်သွေ့ယူ အသုံးပြုပြီး အခြားဖန်ရှင်တွေကို သတ်မှတ်နိုင်ပါတယ်။ လမ်းတစ်လျှောက် ဘိပါ အားလုံး ရှင်းပေးမဲ့ clean_the_street ဖန်ရှင်ကို လေ့လာကြည့်ပါ။

```

def clean_the_street():
    while front_is_clear():
        pick_all_beeper()
        move()

pick_all_beeper()

```

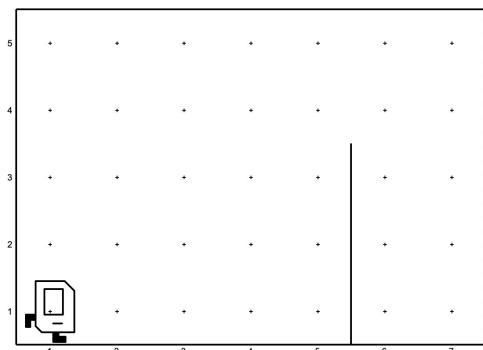
လမ်းတစ်လမ်းလုံး ရှင်းဖို့အတွက် ကွန်နာတစ်ခုက ဘိပါအားလုံး ကောက်ပေးတဲ့ pick_all_beeper ကို အခြေခံ အုတ်ချုပ်သွေ့ယူ အသုံးပြုထားတာပါ။

ရိုးရှင်းတဲ့ အခြေခံ ဖန်ရှင်လေးတွေကနေ ပိုပြီး ရှုပ်ထွေးခက်ခဲတဲ့ ကိစ္စတွေ ဖြေရှင်း ဆောင်ရွက်ပေး နိုင်တဲ့ ဖန်ရှင်တွေကို တစ်ဆင့်ပြီး တစ်ဆင့် တည်ဆောက်ယူလိုရတဲ့ သဘောကို တွေ့ရှုပါတယ်။ ကားရဲလ် ကမ္ဘာထဲက ရှိသမှု ဘိပါအားလုံး ရှင်းပေးမဲ့ clean_the_world ဖန်ရှင် သတ်မှတ်မယ် ဆိုပါစို့။ clean_the_street ကို အခြေခံအုတ်ချုပ်သွေ့ယူ ဆက်လက် အသုံးပြုနိုင်မှာ ဖြစ်တယ်။

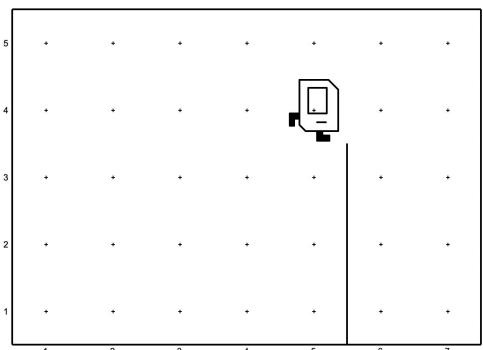
၃.၂ ပရီကွန်ဒီရှင်နှင့် ပို့ခိုကွန်ဒီရှင်

ပရီကွန်ဒီရှင် (precondition) နဲ့ (postcondition) ဟာ ဖန်ရှင်နဲ့ ပါတ်သက်ပြီး ကယာနကာ နားလည်ဖို့ လိုအပ်တဲ့ အရေးကြီးတဲ့ သဘောတရားနှစ်ခုပါ။ ဖန်ရှင် မလုပ်ဆောင်မဲ့ ကြိုတင်ရှိနေရမဲ့ အခြေအနေကို ပရီကွန်ဒီရှင်လို ခေါ်ပြီး လုပ်ဆောင်ပြီး ရှိရမဲ့ အခြေအနေကို ပိုစ်ကွန်ဒီရှင်လို ခေါ်ပါတယ်။ သတ်မှတ်ထားတဲ့ ပရီကွန်ဒီရှင်နဲ့ ကိုက်ညီမှသာ ဖန်ရှင်တစ်ခုဟာ သူလုပ်ဆောင်ပေးရမဲ့ ကိစ္စကို မှန်ကန်အောင် ဆောင်ရွက်ပေးနိုင်မှာပါ။ ဖန်ရှင် အသုံးပြုတဲ့ အခါမှာရော တည်ဆောက်တဲ့ အခါမှာပါ ပရီကွန်ဒီရှင် ပိုစ်ကွန်ဒီရှင်တွေ အပေါ် အခြေခံပြီး တိတိကျကျဖြည့်တားဖို့ ပစာနက်ပါတယ်။

ပုံ ၃.၁ (က) မှ (ခ) အနေအထားသို့ ကားရဲလ်က တိုင်ထိပ်အရောက် တက်သားရပါမယ်။ တိုင်အကွား အဝေး၊ အမြှင့် အမျိုးမျိုးနဲ့ အလားတူ ကွဲ့တွေမှာလည်း အလုပ်လုပ်ရပါမယ်။ (နံရုံကို တိုင်ဟု ယူဆပါ)။ တိုင်အောက်ခြေကိုသွားတာနဲ့ တိုင်ထိပ်တက်တာ အလုပ်နှစ်ခု ပါဝင်တယ်လို့ မြင်နိုင်တယ်။ ဒီအတွက် ဖန်



(က) မတိုင်မဲ့ အခြေအနေ



(ခ) ပြီးဆောက် အခြေအနေ

ပုံ ၃.၁ တိုင်ထိပ်သို့

ရှင်နှစ်ခု သတ်မှတ်ပေးပါမယ်။

```
def go_to_pole():
    while front_is_clear():
        move()
```

```
def ascend_pole():
    while right_is_blocked():
        move()
```

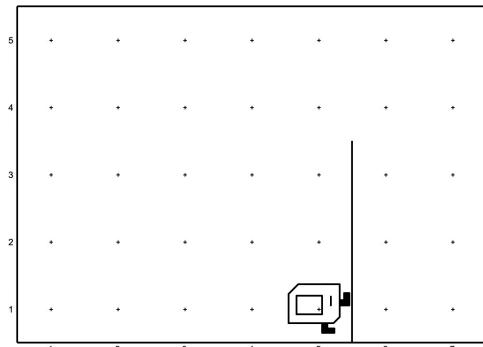
အထက်ပါ ဖန်ရှင်နှစ်ခုနဲ့ go_to_top ကို ဆက်လက် သတ်မှတ်ပါမယ်

```
def go_to_top():
    go_to_pole()
    turn_left()
    ascend_pole()
    turn_right()
```

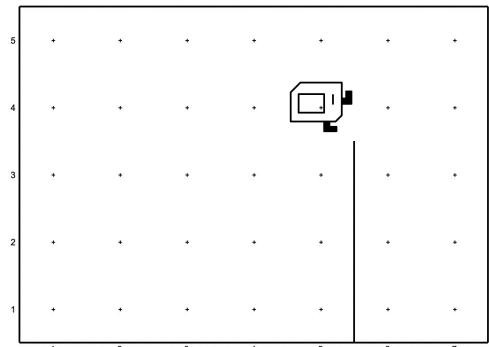
go_to_pole အပြီးမှာ ကားရဲလ်ဟာ တိုင်ခြေမှာ အရှေ့ဘက်မျက်နှာမှပြီး ရှိနေမှာပါ။ ascend_pole က တိုင်ခြေမှာ ကားရဲလ် အပေါ်ဘက်ကို မျက်နှာမှာတဲ့ အနေအထားကနေ စရပါမယ်။ go_to_pole ပြီး ရင် ascend_pole အတွက် အသင့်အနေအထားဖြစ်အောင် turn_left လုပ်ပေးရပါမယ်။

ဖန်ရှင် စတင်မလုပ်ဆောင်မီ ကြိုတင်ရှိနေရမဲ့ အခြေအနေကို ပရီကွန်ဒီရှင်လို့ ပြောခဲ့ပါတယ်။ ဖန်ရှင် သတ်မှတ်တဲ့အခါ ပရီကွန်ဒီရှင်ကို တိတိကျကျ စဉ်းစားဖို့ လိုပါတယ်။ ဖန်ရှင်အသုံးပြုတဲ့အခါမှာလည်း သတ်မှတ်ထားတဲ့ ပရီကွန်ဒီရှင် အတိုင်းကိုက်ညီဖို့ လိုတယ်။ `ascend_pole` ပရီကွန်ဒီရှင်းဟာ တိုင်ခြုံမှာ ကားခဲ့လဲ အပေါ်ဘက်ကို မျက်နှာမူတဲ့ အနေအထား ဖြစ်ပါတယ်။ ပုံ ၃.၂ (က) ကို ကြည့်ပါ။

ဖန်ရှင် လုပ်ဆောင်အပြီးမှာ ရှိနေရမဲ့ အခြေအနေကို ပိုစ်ကွန်ဒီရှင်လို့ ဖော်ပြုခဲ့တယ်။ ဖန်ရှင်တစ်ခု ဟာ ပရီကွန်ဒီရှင်နဲ့ ကိုက်ညီတဲ့ အနေအထားကနေ စတင်ရင် ပိုစ်ကွန်ဒီရှင်နဲ့ ကိုက်ညီအောင်လုပ်ဆောင်ပေးပြီး အဆုံးသတ်ရမှာပါ။ ပုံ ၃.၂ (ခ) မှာ `ascend_pole` ပိုစ်ကွန်ဒီရှင်ကို တော်နိုင်ပါတယ်။



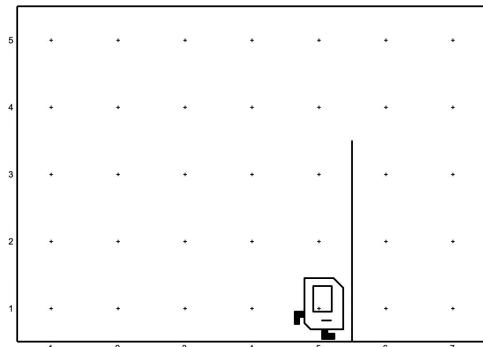
(က)



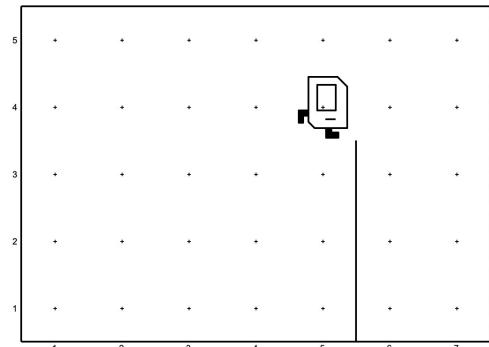
(ခ)

ပုံ ၃.၂ `ascend_pole` ပရီကွန်ဒီရှင်းနှင့် ပိုစ်ကွန်ဒီရှင်

ဖန်ရှင် ပရီကွန်ဒီရှင် ပိုစ်ကွန်ဒီရှင်ကို သင့်တော်သလို သတ်မှတ်နိုင်ပါတယ်။ ပုံ (၃.၃) တွင် `ascend_pole` အတွက် အခြား ရွေးချယ်နှင့် ပရီနဲ့ ပိုစ်ကွန်ဒီရှင်ကို ကြည့်ပါ။



(က)



(ခ)

ပုံ ၃.၃ `ascend_pole` ပရီနဲ့ ပိုစ်ကွန်ဒီရှင်

အထက်ပါ ပရီနဲ့ ပိုစ်ကွန်ဒီရှင်အရ `ascend_pole` ကို အခုလို

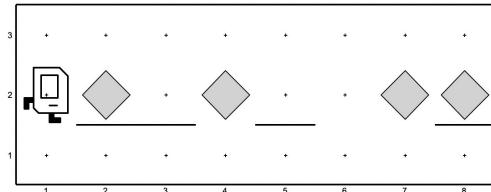
```
def ascend_pole():
    turn_left()
    while right_is_blocked():
        move()
        turn_right()
```

သတ်မှတ်ရမှာပါ။ go_to_top ကလည်း ဒီလို ဖြစ်သွားမယ်

```
def go_to_top():
    go_to_pole()
    ascend_pole()
```

၃.၃ ဖန်ရှင်များဖြင့် abstraction များ တည်ဆောက်ခြင်း

အခန်း (၂) တမျက်နာ (၂၆) မှ လမ်းပြင်တဲ့ ပရိုကရမ်မှာ ကွန်နာတစ်ခုဟာ ဘိပါလည်းမရှိ၊ အောက်ဘက် ကပ်လျက် နံရံလည်းမရှိရင် ဘိပါတစ်ခု ချထားပေးရပါတယ်။ ဒီကိုစွဲ ဆောင်ရွက်ပေးဖို့အတွက် ဖန်ရှင်တစ်ခု သတ်မှတ်နိုင်ပါတယ်။



ပုံ ၃.၆

```
def repair_corner():
    if right_is_clear():
        if no beepers_present():
            put_beepers()
```

အခြေအနှစ်ခုစလုံး မှန်တွေ့မှ ဘိပါချပေးဖို့ nested if သုံးထားတာက နားလည်ရ အတန်အသင့် ခက်ခဲ့နိုင်ပါတယ်။ ဒါပေမဲ့ repair_corner ကို အသုံးပြုတဲ့အခါ ဘယ်လိုပေးထားလဲ စဉ်းစားစရာ မလိုပါဘူး။ ဘိပါလည်းမရှိ၊ ညာဘက် နံရံလည်းမရှိတဲ့ ကွန်နာမှာ ဘိပါချချင်ရင် repair_corner ဖန်ရှင်နဲ့လုပ်လို့ရတယ်ခို့တာ သိထားရင် ထူးလို့ရပြီ။

ဖန်ရှင် သတ်မှတ်တဲ့အခါ ဆောင်ရွက်ပေးစေချင်တဲ့ ကိစ္စကို 'ဘယ်လို လုပ်ရမှာလဲ' အသေးစိတ် စဉ်းစားရမှာဖြစ်ပေမဲ့ အသုံးပြုတဲ့အခါမှာတော့ ဒီလိုအသေးစိတ်တွေ့ကို ထပ်ပြီး စဉ်းစားဖို့ မလိုတော့ပါဘူး။ ဖန်ရှင် 'ဘာလုပ်ပေးတာလဲ' ကပဲ အရေးကြီးတယ်။ 'ဘယ်လို' တည်ဆောက်ထားလဲ သိစရာမလိုဘဲ 'ဘာ' လုပ်ပေးလဲ သိရှုနဲ့ အသုံးပြုလို့ရတော့ကို abstraction လုပ်တယ်လို့ခေါ်ပါတယ်။ Abstraction လုပ်ခြင်းအတွက် ဖန်ရှင်တွေဟာ အခိုက အကျခုံး အခြေခံ အုတ်ချပ်တွေပါပဲ။

Abstraction လုပ်ထားလိုက်ခြင်းအားဖြင့် ရှုပ်ရှုပ်ထွေးထွေးတွေ ထပ်ခါထပ်ပါ ခေါင်းလျှပ်ခဲ့ စဉ်းစားနေဖို့ မလိုအပ်တော့ဘဲ ကိစ္စတစ်ခုကို အလုယ်တကူ လုပ်ဆောင်လို့ရသွားတယ်။ ကုပ်တွေဖတ်ရတာလည်းပိုရှင်းပြီး နားလည်ရ လွယ်ကူသွားတယ်။ ဒါကြောင့် ပရိုကရမ်တွေ တည်ဆောက်ရမှာ abstraction လုပ်ခြင်းဟာ အရေးပါပါတယ်။ အရေးကြီးဆုံးလို ဆိုရင်လည်း မမှားဘူး။ ကြီးမားရှုပ်ထွေးတဲ့ ဆော်ဖဲ့တွေကို လေ့လာကြည့်ရင် abstraction ပေါင်းများစွာနဲ့ တစ်ဆင့်ပြီးတစ်ဆင့်၊ တစ်လွှာပြီးတစ်လွှာ တည်ဆောက်ထားတယ်ဆိုတာ တွေ့ရမှာပါ။

repair_corner ဟာ အနိမ့်ဆုံးအလွှာက အခြေခံ abstraction တစ်ခု ဖြစ်တယ် ဆိုပါစို့။ ငြင်းကို အခြေခံပြီး တစ်ဆင့်ပိုမြင့်တဲ့ အလွှာအတွက် abstraction တွေကို တည်ဆောက်ယူနိုင်ပါတယ်။ ဥပမာ

```

def repair_street():
    while front_is_clear():
        repair_corner()
        repair_corner()
    
```

ဒီအလွှာထက် နောက်ထပ် တစ်ဆင့်ပိုမြင့်တဲ့ abstraction တွေကိုလည်း ဆက်လက် တည်ဆောက်ယူနိုင်ပါတယ်။ ဥပမာ repair_world ကို တည်ဆောက်ရမှာ repair_street ကို အခြေခံအစိတ်အပိုင်း အဖြစ် အသုံးပြန်ပါတယ်။ ဒီလို abstraction အလွှာတွေ အဆင့်ဆင့်နဲ့ ပရိုက်မဲ့ တည်ဆောက်နိုင် တွေကို ဆက်လက်လေ့လာကြရပါမယ်။

၃.၄ Bottom-up Programming

ကြီးကျယ်ခမ်းနားတဲ့ အဆောက်အအုံကြီးတွေ၊ လျှို့ဝှက်ဆန်းကြယ်တဲ့ သဘာဝဖြစ်စဉ်တွေ၊ အုံဖွယ်သိပုံနှင့် နည်းပညာ ဆန်းသစ်တိတင်မှုတွေ စတုအရာတွေအားလုံးဟာ ရုံးရှင်းတဲ့ အစိတ်အပိုင်းလေးတွေနဲ့ ဖွံ့ဖြိုးထားတာပါပဲ။ ဒါကြောင့် အရွယ်အစား ကြီးမား ရှုပ်ထွေးတဲ့ ပရိုက်မဲ့တွေကိုလည်း ရုံးရှင်းတဲ့ အစိတ်အပိုင်း လေးတွေနဲ့ အခြေပြု ဖွံ့ဖြိုးထားတည်ဆောက်သင့်တယ်လို့ ယူဆမယ်ဆိုရင် ကျိုးကြောင်းဆီလျှော်တယ်ပဲ ဆိုရမှာပါ။

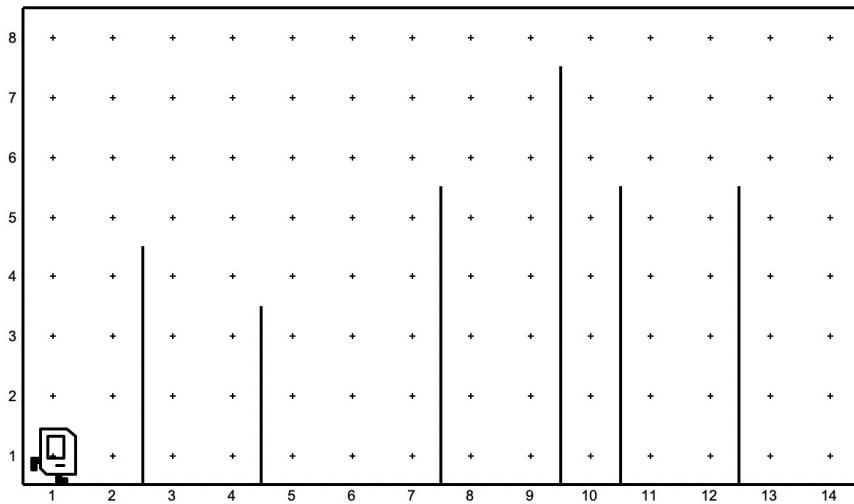
ခက်ခဲရှုပ်ထွေးတဲ့ ကိစ္စတစ်ခုကို ဖြေရှင်းတဲ့အခါ အပိုင်းတွေခဲ့ပြီး တစ်ပိုင်းချင်းကို သီး၌ဗြားဖြေရှင်းလေ့ရှိတယ်။ အပိုင်းတွေခဲ့လိုက်ခြင်းအားဖြင့် တစ်ပိုင်းစီဟာ မူလဖြေရှင်းရမဲ့ ကိစ္စလောက် မခက်ခဲတော့ ပါဘူး။ အရွယ်အစားအားဖြင့်လည်း နိဂုံထက် ငယ်သွားမယ်။ အပိုင်း တစ်ပိုင်းချင်းစီကို ဖြေရှင်းတဲ့အခါ မှာလည်း အလားတူပဲ လုပ်ဆောင်နိုင်တယ်။ အပိုင်းတစ်ပိုင်းကို သူ့ထက်သေးငယ်တဲ့ အပိုင်းတွေအဖြစ် ထပ်မံ့ထဲတော်နိုင်ပါတယ်။ ဒီတိုင်း တစ်ဆင့်ပြီးတစ်ဆင့် လုပ်သွားမယ်ဆိုရင် နောက်ဆုံးမှာ အလွယ်တကူ ဖြေရှင်းလို့ရတဲ့ အစိတ်အပိုင်းလေးတွေ ဖြစ်သွားရမှာပါပဲ။ *Bottom-up programming* ဆိုတာကတော့ ပရိုက်မဲ့ရေး ကိစ္စတစ်ခုကို ဖြေရှင်းရမှာ ရုံးရှင်းသထက်ရုံးရှင်း၊ သေးငယ်သထက်သေးငယ်တဲ့ အပိုင်း လေးတွေဖြစ်လာအောင် အဆင့်ဆင့်ပိုင်းခဲ့ပြီး အောက်ခြေအရှုံးရှင်းဆုံးအလွှာကနေ အပေါ်ကိုတစ်ဆင့်ချင်းတက် ဖြေရှင်းတဲ့နည်း ဖြစ်တယ်။ လေ့လာကြည့်ကြရအောင်။

ပုံ (၃.၅) ကမ္ဘာမှာ ကားရဲ့လှု တန်းကော်ပြီးပြုပါမယ်။ ထောင်လိုက်နံရုံတွေကို တန်းတွေလို့ယူဆပါ။ တန်းတွေဟာ အနိမ့်အမြင့် အမျိုးမျိုးဖြစ်နိုင်ပါတယ်။ ကမ္ဘာရဲ့ အပေါ်ဘက် နံရုံကို ထိတဲ့အထိတော့ မြင့်လို့မရပါဘူး။ ဒါမှ တန်းကိုကော်ပြီး အခြားဘက်ကို သွားလို့ရမှာပါ။ (1, 1) ကွန်နာကနေ တာစထွက်မှုဖြစ်ပြီး (14, 1) မှာ ပန်းဝင်ရမှာပါ။

တန်းကော်ပြီးတဲ့ ကိစ္စမှာ ပါဝင်တဲ့ အပိုင်းတစ်ခုက တန်းတစ်ခုကို တန်းတစ်ခု ကော်တာကို ထပ်ခဲ့ကြည့်ရင် အပေါ်ဘက်တော့ အောက်ဆင်းတာ နှစ်ပိုင်းရှုံးမယ်။ အခုလို နိဂုံမူလကိစ္စတစ်ခုကိုနေ တစ်ဆင့်ထက်တစ်ဆင့် ပိုသေးငယ်တဲ့ အပိုင်းလေးတွေဖြစ်လာအောင် ခွဲထုတ်တာကို *problem decomposition* လုပ်တယ်လို့ ခေါ်ပါတယ်။

Bottom-up programming နည်းနဲ့ ပရိုက်မဲ့ရေးတဲ့အခါ problem decomposition အရှင်းဆုံး လုပ်ရတယ်။ ပြီးရင် အောက်ဆုံးအလွှာမှာ အသေးဆုံးအပိုင်းလေးတွေကနေ စတင်ဖြေရှင်းတယ်။ ပြီး တဲ့အခါ အဲဒီ အသေးဆုံး အပိုင်းလေးတွေကို အခြေခံအစိတ်အပိုင်းအဖြစ် အသုံးပြုပြီး အောက်ဆုံးအလွှာထက် တစ်ဆင့်မြှင့်တဲ့ အပေါ်အလွှာက အပိုင်းတွေကို ဆက်လက်ဖြေရှင်းတယ်။ ဒီအလွှာက အပိုင်းတွေကို အခြေခံအစိတ်အပိုင်းအဖြစ် အသုံးပြုပြီး နောက်ထပ်တစ်ဆင့် ပိုမြင့်တဲ့အလွှာက အပိုင်းတွေကို ဆက်လက်ဖြေရှင်းမှာဖြစ်တယ်။ ဒီလိုမျိုး အောက်ကနေ အပေါ်ကို တစ်လွှာပြီးတစ်လွှာ တက်သွားပြီး ဖြေရှင်းတဲ့ နည်းစနစ်ဟာ bottom-up programming ပါပဲ။

တန်းကော်ပြီးပဲမှာ Problem Decomposition လုပ်ထားတာကို တစ်လွှာချင်း ခွဲကြည့်မယ်ဆို



ပို ၃၅

ရင် အခုလိုတွေ့ရမှာ ဖြစ်ပါတယ်။

- တန်းကျော်ပြေးပြိုင်ပွဲဝင်ခြင်း
 - ▶ တန်းတစ်ခုကျော်ခြင်း
 - ▶ အပေါ်တက်ခြင်း
 - ▶ အောက်ဆင်းခြင်း

Bottom-up နည်းအရ အောက်ဆုံးအလွှာ အပေါ်တက်၊ အောက်ဆင်း ကိစ္စကို ပထမဆုံး ဖြော်ပါမယ်။ အပိုင်းတစ်ခုအတွက် ဖန်ရှင်တစ်ခု သတ်မှတ်ရပါမယ်။

```

def ascend():
    """
    တန်းနဲ့လွှတ်တဲ့အထိ အပေါ်ဘို့ တက်သည်
    Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ
    Postcondition: တန်းထိပ်အပေါ် ဘယ်ဘက်ကွန်နာမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ
    """
    turn_left()
    while right_is_blocked():
        move()
        turn_right()

def descend():
    """
    တန်းအောက်သို့ ပြန်ဆင်းသည်
    Precondition: တန်းထိပ်အပေါ် ညာဘက်ကွန်နာမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ
    Postcondition: တန်းညာဘက် ခြေရင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ
    """
    turn_right()
    while front_is_clear():

```

```
move()
turn_left()
```

အထက်ပါ ဖန်ရှင်နှစ်ခုကို အသုံးပြုပြီး အပေါ်တစ်ဆင့် ပိုမြင့်တဲ့ အလွှာက တန်းတစ်ခုကျော်တာကို ဆက်လက်ဖြောင်းရပါမယ်။ `ascend` နဲ့ `descend` ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေအရ ၂move လုပ်ပေးဖို့လိုပါ တယ်။

```
def jump_over_hurdle():
```

```
    """

```

တန်းတစ်ခုကို ကျော်သည်

Precondition: တန်းသယ်ဘက် ခြေရှင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရှင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ

```
    """

```

```
ascend()
```

```
move()
```

```
descend()
```

ဒီဖန်ရှင်နဲ့ နောက်တစ်ဆင့်ကို ဆက်လက်တည်ဆောက်ရပါမယ်။ ရှေ့မှာရှင်းနေလျှင် ရှုံးတိုးမယ်၊ မရှင်းလျှင် တန်းကိုကျော်ရမယ်။ ပန်းဝင်ဖို့အတွက်ခိုလျှင် ဒီအလုပ်ကို ဆယ့်သုံးခါလုပ်ပေးရုံပါပဲ။

```
def compete_hurdle_race():
```

```
    """

```

တန်းကျော်ပြီးပြုင့်ပဲ ပန်းဝင်သည်ထိ ယုံးပြိုင်သည်

Precondition: (၁,၁) ကွန်နာတွင် အရှေ့ဘက်မျက်နှာမူနေ

Postcondition: (၁၄,၁) ကွန်နာတွင် အရှေ့ဘက်မျက်နှာမူနေ

```
    """

```

```
for i in range(13):
```

```
    if front_is_clear():
```

```
        move()
```

```
    else:
```

```
        jump_over_hurdle()
```

ပရီဂရမ် အစအဆုံးကို လေ့လာကြည့်ပါ။ ဖန်ရှင်တစ်ခုချင်း ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေကို တိတိကျကျ မြင်အောင် ရရှိကိုကြည့်ဖို့လည်း အရေးကြီးတယ်။ ဖန်ရှင် docstring မှာ ငါးဖန်ရှင် ပရီနဲ့ ပိုစ် ကွန်ဒါရှင် ကို တိတိကျကျ ဖော်ပြထားတာဟာ အလေ့အထကောင်း တစ်ခုပါ။ ဖန်ရှင်တိုင်းမှာ ပါသင့်တယ်။

```
# File: hurdle_jumping.py
```

```
from stanfordkarel import *
```

```
def main():
```

```
    """Hurdle Jumping Program"""

```

```
    compete_hurdle_race()
```

```
def ascend():
```

```
    """

```

တန်းနဲ့လွတ်တဲ့အထိ အပေါ်သို့ တက်သည်

Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရေးဘက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းထိပ်အပေါ် ဘယ်ဘက်ကွန်နာမှာ အရေးဘက်မျက်နှာမူပြီး ရှိနေ

„„„

turn_left()

while right_is_blocked():

move()

turn_right()

descend():

„„„

တန်းအောက်သို့ ပြန်ဆင်သည်

Precondition: တန်းထိပ်အပေါ် ညာဘက်ကွန်နာမှာ အရေးဘက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရင်းမှာ အရေးဘက်မျက်နှာမူပြီး ရှိနေ

„„„

turn_right()

while front_is_clear():

move()

turn_left()

jump_over_hurdle():

„„„

တန်းတစ်ခုကို ကျော်သည်

Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရေးဘက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရင်းမှာ အရေးဘက်မျက်နှာမူပြီး ရှိနေ

„„„

ascend()

move()

descend()

compete_hurdle_race():

„„„

တန်းကျော်ပြီးပြိုင်ပဲ ပန်းဝင်သည်ထိ ယုံးပြိုင်သည်

Precondition: (၁,၁) ကွန်နာတွင် အရေးဘက်မျက်နှာမူနေ

Postcondition: (၁၇,၁) ကွန်နာတွင် အရေးဘက်မျက်နှာမူနေ

„„„

for i in range(13):

if front_is_clear():

move()

else:

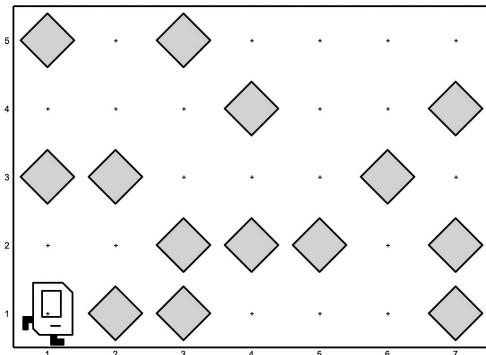
jump_over_hurdle()

```
def turn_right():
    for i in range(3):
        turn_left()

if __name__ == "__main__":
    run_karel_program("hurdle_jumping")
```

အမိန်သိမ်းတဲ့ ကားရဲလ်

အခုတ်စံခါမှာ ကားပေါ်က အမိုက်တော်ရှင်းပေးပါမယ်။ ပုံ (၃၆) နဗုံးနာ ကမ္မာမှာ ဖြောကျပန်း (random) ပြန့်ကျေနေတဲ့ ဘိပါတွေကို အမိုက်လို ယူဆပါ။ ကမ္မာအရွယ်အစား အမျိုးမျိုးအတွက် အမိုက်ရှင်းပေးတဲ့ ပရိုဂရမ်တစ်ခု ရေးပေးရမှာပါ။

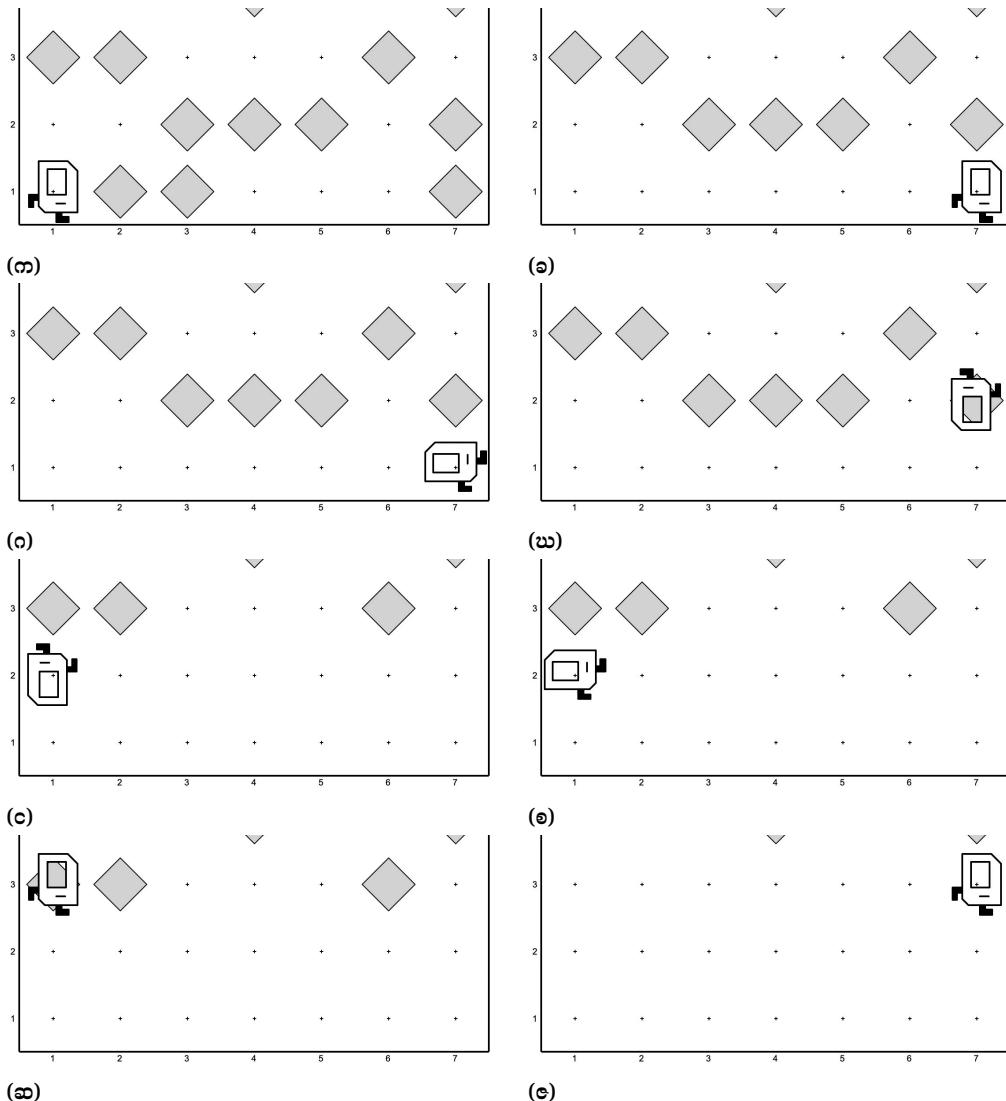


१८

ဒီကိစ္စကို ဖြေရှင်းဖိုက နည်းလမ်းတစ်ခုတည်း ရှိတာ မဟုတ်ပါဘူး။ နည်းလမ်းတစ်ခုက ဒီလိုပါ။ (၁) လမ်းကနေ စပြီး ရှင်းတယ် စာမျက်နှာ ၄၀ ပုံ (၃.၇) (က) နှင့် (ခ) တွင် ကြည့်ပါ။ ပြီးရင် မြေက်ဘက် လှည့်ထားမယ် ပုံ (၃.၇) (ဂ)။ ရှင်းနေသေးတယ်ဆိုရင် ဒုတိယလမ်းကို တက် အနောက်ဘက် မျက်နှာမူ ထားမယ် ပုံ (၃.၇) (ဃ)။ ဒုတိယလမ်းကို ဆက်ရှင်းပြီးတော့လည်း မြေက်ဘက်ကိုပဲ မျက်နှာမူထား ပါမယ် ပုံ (၃.၇) (၁)၊ (၁)။ ရှင်းနေသေးရင် တတိယလမ်းကို ကူး၊ အရှေ့ဘက်မျက်နှာ မူထားမယ် ပုံ (၃.၇) (ဆ)။ ဒီအတိုင်း တစ်လမ်းပြီးတစ်လမ်း ရှင်းသွားမှုဖြစ်တယ်။ လမ်းတစ်လမ်းရှင်းပြီး ပြောက်ဘက် မျက်နှာမူလို ပိတ်နေရင် နောက်ထပ် လမ်းမရှိတော့ဘူး။ လမ်းအားလုံး ရှင်းပြီးသွားပြီ။ စာမျက်နှာ ၄၁ ပုံ (၃.၈) (က)၊ (ခ)၊ (ဂ) တိုကို ကြည့်ပါ။ Problem decomposition လုပ်ကြည့်ရင် အောက်ပါ အတိုင်း တော်ဆုံးပါတယ်။

- ကုမ္ပဏီတစ်ခုလုံး အမိုက်တွေ့ရှင်း (clean world)
 - ▶ လမ်းတစ်လမ်း အမိုက်ရှင်း (clean street)
 - ▶ ကွုန်နာတစ်ခု အမိုက်ရှင်း (clean corner)
 - ▶ မြေကာ်ဘက်လှည့် (turn north)
 - ▶ လမ်းပြောင်း/နောက်တစ်လမ်းကူး (change street)

လမ်းတစ်လမ်း ရှင်းတဲ့ ကိစ္စကို ထပ်ခဲ့ကြည့်ရင် ကုန်နာတစ်ခုရှင်းတာကို တွေ့ရမှုပါ။ Bottom-up နည်းအရ အောက်ဆုံးအလွှာက ကုန်နာတစ်ခု အမိုက်ရှင်းတဲ့ ကိစ္စအတွက် ဖုန်ရှင်ကို ပထမဆုံး သတ်မှတ်



፳፻፭

ରପିମର୍ଯ୍ୟ॥

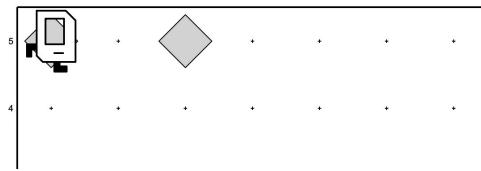
```

def clean_corner():

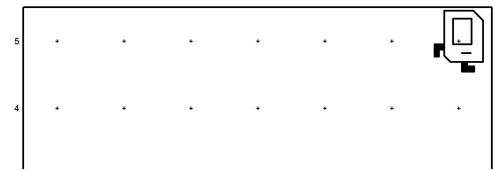
    """
    ကွန်နာတစ်ခုမှာ ဘိပါရှင်းပေးသည်
    Precondition: ကွန်နာတစ်ခုမှာ ကားရဲ့လ် ရှိနေမယ်။ အများဆုံး ဘိပါတစ်ခု ရှိနိုင်တယ်။
    Postcondition: ဘိပါ မရှိတဲ့ ကွန်နာဖြစ်ရမယ်
    """
    if beepers_present():
        pick beeper()

```

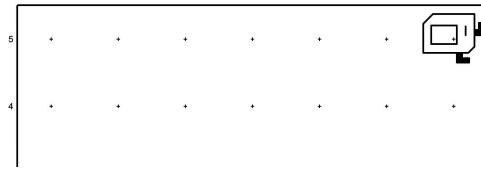
`clean_corner` ກົດລັບຕົວລັບຕົວຕະຫຼາດ



(a)



(b)



(c)

ပုံ ၃.၈

```
def clean_street():
```

```
    """

```

လမ်းတလျောက် ဘိပါအားလုံးရှင်းပေးသည်

Precondition: လမ်းအစွမ်းတစ်ဖက်မှာ အခြားဘက်စွန်းကို မျက်နှာမျှ၍ ရှိမယ်

Postcondition: လမ်းပေါ်ဘိပါအားလုံး ရှင်းပြီး အခြားဘက်စွန်း ရောက်နေမယ်

```
    """

```

```
    while front_is_clear():

```

```
        clean_corner()

```

```
        move()

```

```
        clean_corner()
```

မြောက်ဘက်လှည့်တဲ့ ဖန်ရှင်

```
def turn_north():

```

```
    while not_facing_north():

```

```
        turn_left()
```

while loop နဲ့ မြောက်ဘက်ကို မျက်နှာမူ မနောသ၍ ဘယ်ဘက်လှည့်ခိုင်းထားတာ သတိထားကြည့်ပါ။
မြောက်ဘက် မျက်နှာမူနေတဲ့ အနေအထားမှာ ရပ်သွားမှာ ဖြစ်တယ်။

လမ်းတစ်လမ်းရှင်းပြီး နောက်တစ်လမ်းကူးတဲ့ ဖန်ရှင်

```
def change_street():

```

```
    """

```

လမ်းတစ်လမ်း၏ အစွမ်းတစ်ဖက်မှာ အပေါ်လမ်းသို့ ကူးသည်

Precondition: လမ်းအစွမ်းတစ်ဖက်မှာ ကပ်လျက် နဲ့ရုံကို မျက်နှာမျှ၍ ရှိမယ်

Postcondition: အပေါ်တစ်လမ်းကူးပြီး အခြားဘက်စွန်းကို မျက်နှာမျှ၍ ရှိမယ်

```
    """

```

```
    move()

```

```
    if right_is_blocked():

```

```
        turn_left()

```

```
    else:

```

```
        turn_right()
```

တစ်လမ်းချင်း အမိုက်အကုန် ရှင်းတဲ့ ဖန်ရှင်ကို သတ်မှတ်လို့ ရပါပြီ

```
def clean_world():
    """
    လမ်းအားလုံးမှ ဘို့ပါတွေ ရှင်းပေး
    Precondition: (၁,၁) ကွန်စာမှာ အရှေ့ဘက် မျက်နှာမျှ၍ ရှိနေမယ်
    Postcondition: လမ်းအားလုံး အမိုက်ရှင်းပြီး ဖြစ်မယ်
    """

    clean_street()
    turn_north()
    while front_is_clear():
        change_street()
        clean_street()
        turn_north()

# File: clean_world.py
from stanfordkarel import *

def main():
    """Karel clean the world"""
    clean_world()

def clean_world():
    clean_street()
    turn_north()
    while front_is_clear():
        change_street()
        clean_street()
        turn_north()

def clean_corner():
    if beepers_present():
        pick_beeper()

def clean_street():
    while front_is_clear():
        clean_corner()
        move()
        clean_corner()
```

ပရိုကရမ် အစအဆုံး ဆက်လက်လေ့လာကြည့်ပါ။ (ပရိုနဲ့ ပိုစ် ကွန်အိုးရှင်တွေကို ချုပ်ထားတယ်။ နေရာ သက်သာအောင်ပါ။ အပြည့်အစုံ ရေးတဲ့အခါ မလိုတော့တာ မဟုတ်ဘူး။)

```

def turn_north():
    while not_facing_north():
        turn_left()

def change_street():
    move()
    if right_is_blocked():
        turn_left()
    else:
        turn_right()

def turn_right():
    for i in range(3):
        turn_left()

if __name__ == "__main__":
    run_karel_program("clean_world")

```

၃.၅ Top-down Programming

Top-down programming ဟာ အပေါ်ဆုံးအလွှာမှ စ၍ တည်ဆောက်တဲ့နည်း ဖြစ်ပါတယ်။ Bottom-up မှာလို အောက်မှ အထက် မတက်ဘဲ အထက်မှအောက် တစ်လွှာပြီးတစ်လွှာ အဆင့်ဆင့် တည်ဆောက် သွားမှာပါ။ နည်းလမ်းနှစ်ခုလုံးမှာ problem decomposition လုပ်ရမှာဖြစ်ပေမဲ့ လုပ်ငန်းစဉ် ကွားခြင်းနှင့်တယ်။ Top-down နည်းအတွက် problem decomposition လုပ်တဲ့အခါ bottom-up မှာ လို အောက်ဆုံးအလွှာထိ တစ်ခါတည်း ခွဲခြမ်းစိတ်ဖြေစရာမလိုဘူး။ တစ်ဆင့်ချင်းပဲ ခွဲရပါတယ်။ ပြီးခဲ့တဲ့ ဥပမာ အမှိုက်သိမ်းတဲ့ ကိစ္စကို ခွဲခြမ်းစိတ်ဖြေမယ်ဆိုရင် အခါလို

- ကဲ့ကဲ့တစ်ခုလုံး အမှိုက်တွေရှင်း (clean world)
 - ▶ လမ်းတစ်လမ်း အမှိုက်ရှင်း (clean street)
 - ▶ မြောက်ဘက်လှည့် (turn north)
 - ▶ လမ်းပြောင်း/နောက်တစ်လမ်းကူး (change street)

ဖြစ်မှာပါ။ စာမျက်နှာ (၃၉) က ခွဲခြမ်းစိတ်ဖြေတာနဲ့ တူတယ်ဆိုပေမဲ့ သတိပြုရမှာက လမ်းတစ်လမ်း ရှင်းတဲ့ ကိစ္စကို လောလောဆယ် ထပ်ပြီး အသေးစိတ် မခွဲသေးဘူး။ ဒီအဆင့်မှာပဲ ရပ်တားတယ်။

ပြီးတဲ့အခါ အပေါ်ဆုံး အဆင့်ကို စပြီးဖြေရှင်းတယ်။ Top-down နည်းရဲ့ အမိက ထူးခြားချက်က လက်ရှိဖြေရှင်းပဲ ကိစ္စရဲ့ အောက်အလွှာကို 'ဖြေရှင်းစိုင်ပြီး' ဖြစ်တယ်လို့ မှတ်ယူရတာပါ။ တစ်နည်းအားဖြင့် clean_world ဖန်ရှင်း သတ်မှတ်တဲ့အခါ clean_street, turn_north, change_street ဖန်ရှင်းတွေ ရှိထားပြီးဖြစ်တယ်လို့ ယူဆရမှာပါ။ ဖန်ရှင်းတွေ အမှန်တကယ် မရှိသေးဘဲ ရှိပြီးဖြစ်တယ်လို့ ယူဆရတာဖြစ်တယ်။

```

def clean_world():
    clean_street()
    turn_north()
    while front_is_clear():
        change_street()
        clean_street()
        turn_north()

```

ဒီနေရာမှ ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေ တိတိကျကျ သတ်မှတ်ထားဖို့ အလွန်အရေးပါတယ်။ clean_street, turn_north, change_street ဖန်ရှင်တစ်ခုချင်းခဲ့ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေ တိတိကျကျ ရှိထားမှပဲ clean_world ကို မှန်ကုန်အောင် ရေးဖို့ ဖြစ်နိုင်မှာပါ။ ဥပမာအားဖြင့် clean_street ပိုစ်ကွန်ဒါရှင်ကသာ မြောက်ဘက်လှည့် အနေအထားနဲ့ အဆုံးသတ်မယ်ဆိုရင် clean_world ကို အခုလို ရေးပါလိမ့်မယ်။

```

def clean_world():
    clean_street()
    while front_is_clear():
        change_street()
        clean_street()

```

အပေါ်ဆုံးအလွှာ ဖြေရှင်းပြီးသွားရင် အောက်အလွှာကို တစ်ဆင့် ဆင်းပြီး ဖြေရှင်းပါတယ်။ အောက်ပါ ကိစ္စရပ်

- လမ်းတစ်လမ်း အမှိုက်ရှင်း (clean street)
- ဖြောက်ဘက်လှည့် (turn north)
- လမ်းပြောင်း/နောက်တစ်လမ်းကူး (change street)

သုံးခုပါဝင်တယ်။ အပေါ်ဆုံးကို ပထမအလွှာလို ယူဆရင် ဒါသည် ဒုတိယ အလွှာပါ။ တစ်ခုချင်း လိုအပ်သလို ထပ်မံခွဲမြေးစိတ်ဖြာ ရုပါမယ်။ ဒါပေမဲ့ နောက်ထပ်တစ်ဆင့်ပဲ ခွဲခြမ်းစိတ်ဖြာ ရမှာပါ။ ဆိုလိုတာက ဒုတိယအလွှာကို ဖြေရှင်းတဲ့အခါ တတိယအလွှာထိပဲ ခွဲဖို့ စဉ်းစားတယ်။ တတိယ အလွှာကို ဘယ်လို ခွဲမလဲ ဆက်မစဉ်းစားသေးဘူး။ လမ်းတစ်လမ်း ရှင်းတဲ့ကိစ္စရှင်း ဖြေရှင်းတဲ့အခါ အခုလို

- လမ်းတစ်လမ်း အမှိုက်ရှင်း (clean street)
 - ▶ ကွန်နာတစ်ခု အမှိုက်ရှင်း (clean corner)

ဒီကွန်ပိုစ် (decompose) လုပ်နိုင်တယ်။ ကွန်နာတစ်ခု ရှင်းတာက လွှယ်တဲ့အတွက် ထပ်ခွဲစရာတော့ မလိုဘူး။ အကယ်၍ ထပ်ခွဲဖို့ လိုတဲ့ ကိစ္စဖြစ်ခဲ့ရင်လည်း top-down နည်းအရ လောလောဆယ် အနေနဲ့ ဒီအဆင့်မှာပဲ မခဲ့သေးဘဲ ရုပ်ထားရပါမယ်။ clean_street ဖန်ရှင်အတွက် clean_corner ရှိထားပြီး ဖြစ်တယ်လို့ မှတ်ယူရမှာပါ။

```

def clean_street():
    while front_is_clear():
        clean_corner()
        clean_corner()

```

ဒုတိယအလွှာမှာ ပါဝင်တဲ့ မြောက်ဘက်လှည့်တာနဲ့ လမ်းကူးတာကို ဆက်လက်ဖြေရှင်းပါမယ်။ နှစ်ခုလုံး အတန်အသင့် ရိုးရှင်းတဲ့အတွက် ထပ်မခွဲတော့ဘူး။ (လိုအပ်ရင်တော့ အောက်တစ်ဆင့် ထပ်ပြီး ဒီကွန်ပိုစ် လုပ်ရမှာပါ)။

```

def turn_north():
    while not_facing_north():
        turn_left()

def change_street():
    move()
    if right_is_blocked():
        turn_left()
    else:
        turn_right()

```

ဒုတိယတစ်ဆင့် ဖြေရှင်းပြီးသွားပါပြီ။

ညာဘက်လှည့်တာကို လမ်းပြောင်းတဲ့ကိစ္စရဲ့ အခဲ့လို ယူဆကောင်း ယူဆနိုင်ပါတယ်။ ဒါပေမဲ့ အရမ်း အခြေခံကျတဲ့အတွက် နိုပါပြီး turn_left နဲ့ အဆင့်တူလို ယူဆရင် ပိုပြီး ကျိုးကြောင်းဆီလော် မယ်လို ယူဆတယ်။ ဒီလိုဖန်ရှင်မျိုးတွေဟာ ဘုံး (common) သုံး ဖြစ်လေ့ ရှိတယ်။ အားဖြစ်ရှင်တွေ (နှစ်ခုနဲ့အထက်) ကနေ ခေါ်သုံးရလေ့ ရှိတယ်။ ကားရဲ့ပရိုကရမ် အားလုံးလိုလိုမှာလည်း လိုအပ်လေ့ ရှိတယ်။ ဒီသဘောအရ turn_north ကိုလည်း turn_left, turn_right တို့လို အခြေခံ အကျ ဆုံး ဘုံးဖန်ရှင်အနေနဲ့ ယူဆမယ်ဆုံးရင်လည်း မမှားပါဘူး။

(ယူဆချက်ကို ဖော်ပြတာသာဖြစ်ပါတယ်။ ဘယ်လိုမှ မှန်တယ် ဘယ်လိုဆုံးရင်ဖြင့် မှားတယ် မဆိုလို ပါ။ မိမိကိုယ်တိုင် စဉ်းစားဆင်ခြင် သုံးသပ်ပြီးမှ ဖြစ်သင့်တယ်ထင်တာကို ဆုံးဖြတ်လိုပါတယ်။)

အောက်တစ်လွှာ ဆက်ဆင်းရပါမယ်။ ဒုတိယ အဆင့်ကို ဒီကွန်ပို့စ် လုပ်လိုက်တာတော့ဟာ တတိယ အလွှာမှာ ပါဝင်တယ်။ ကွန်နာတစ်ခု ရှုံးတဲ့ကိစ္စပဲ ရှိပါတယ်။ ရှိုးရှင်းတဲ့အတွက် နောက်တစ်ဆင့် ထပ်မံ့တော့ တစ်ခါတည်း ဖန်ရှင်သတ်မှတ်ပါမယ်။

```

def clean_corner():
    if beepers_present():
        pick_beeper()

```

အပေါ်ဆုံးကနေ တစ်ဆင့်ပြီးတစ်ဆင့် ဖြေရှင်းလာတာ တတိယအလွှာမှာ ထပ်မံ့တော့တဲ့အတွက် ဒီမှာ ပဲ ပြီးသွားပြီဖြစ်တယ်။ အကယ်၍ ထပ်ခဲ့ထားတာ ရှိတယ်ဆုံးရင် အောက်တစ်ဆင့် ထပ်ဆင်းပြီး ပြီးခဲ့တဲ့ တစ်ဆင့်ချင်းမှာ လုပ်ခဲ့သလိုပဲ ဆက်သွားရမှာပါ။ ပရိုကရမ် run မယဆုံးရင် ကျွန်ုံးနေသေးတဲ့ turn_right ။ main နဲ့ အန်ထရှိပိုင် ဖြည့်ရုံပါပဲ

```

def main():
    clean_world()

if __name__ == "__main__":
    run_karel_program("clean_world")

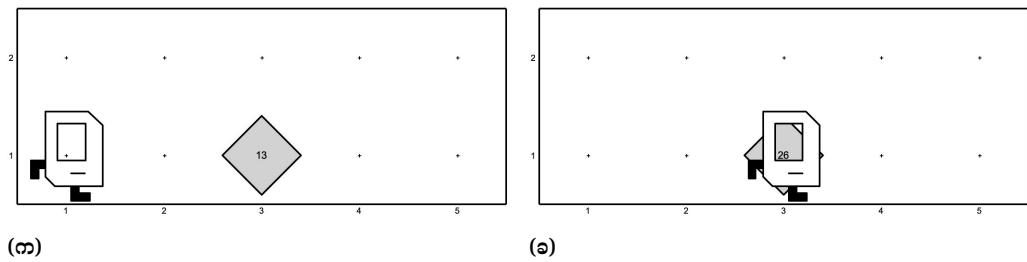
```

(turn_right ဖန်ရှင်ကို ထပ်မဖော်ပြတော့ပါ)

ခုတိယ top-down ဥပမာ (Double Beeper Pile)

အမှန်တကယ် မရှိသေးတဲ့ ဖန်ရှင်တွေကို ရှိထားပြီးဖြစ်တယ်လို့ ယူဆထားရတာဟာ top-down နည်းရဲ့ အခိုက် သော့ချက်ဖြစ်သလို ပရိုဂရမ်းမင်း စလေ့လာသူတွေအတွက် နားလည်ရ အခက်ဆုံး သဘောတရား တစ်ခုဆိုရင်လည်း မမှားဘူး။ အောက်တစ်ဆင့်အလွှာ ဖန်ရှင်တွေ ‘ဘာလုပ်ပေးလဲ’၊ ပရိုနဲ့ ပိုစ် ကွန်ဒီရှင် တွေ ဘယ်လို့ဖြစ်သင့်လဲ တိတိကျကျ စိတ်ကူးပုံဖော်ထားပြီး လက်ရှိဖန်ရှင်ကို ဘယ်လို့ရေးမလဲ စဉ်းစားရဲ့ တာ ဦးနှောက်ခြောက်စရာ ဖြစ်နေတာပါ။ ဥပမာ နောက်တစ်ခုလောက် ထပ်ကြည့်ရင် ပိုပြီး သဘောပေါက် သွားမယ် ထင်ပါတယ်။

အခုပုဂ္ဂိုလ်မှာ ကားရဲ့က ကွန်နာတစ်ခုမှာ စုပုံထားတဲ့ ဘိပါတွေကို နှစ်ဆဖြစ်အောင် လုပ်ပေးရ ပါမယ်။ ပုံ (၃.၉) တွင်ကြည့်ပါ။ နိုင် ဆယ့်သုံးကနေ နှစ်ဆယ့်ခြောက်ခု ဖြစ်သွားပါတယ်။ ကားရဲ့က ကိန်းကောင်းတွေအကြောင်း နားမလည်သလို ရော့က်တာလည်း မလုပ်တတ်ပါဘူး။ ဒါကြောင့် ဘိပါနှစ် ဆွားဖို့ အခြားနည်းလမ်းရှာရပါမယ်။



ပုံ ၃.၉

ဘိပါပုံ (အစုအပံ့ကို ဆိုလို) ထဲကနေ ဘိပါတစ်ခုကောက်လိုက်၊ ရှေ့ကွန်နာမှာ နှစ်ခုချထားလိုက် ဆက်တိုက် လုပ်မယ်ဆိုရင် နိုင်ဘိပါတွေ ကွန်သွားတဲ့အခါ နှစ်ဆရှိတဲ့ ဘိပါပုံတစ်ခု (ရှေ့ကွန်နာမှာ) ရ မှာပါ။ တစ်ခါတည်း ဘိပါအားလုံး ကောက်လို့ မရပါဘူး။ ကားရဲ့က ဘယ်နှစ်ခု ကောက်ထားလဲ မမှတ်ပို့ တဲ့ အတွက်ကြောင့်ပါ။ တစ်ခုကောက်လိုက် ရှေ့ကွန်နာမှာ နှစ်ခုပြန်ချထားလိုက် လုပ်ရပါမယ်။

ရှေ့ကွန်နာမှာ နိုင်ဘိပါပုံရဲ့ နှစ်ဆဖြစ်အောင် လုပ်လိုရတဲ့နည်းတော့ စဉ်းစားလိုပြီး။ ပြီးတဲ့အခါ အဲဒီ ဘိပါတွေကို နိုင်ဘိပါပုံနေရာကို ရွှေ့လုပ်ပါပဲ။ ဒီကွန်ပိုစ်လုပ်ကြည့်ရင်

- ဘိပါပုံသွား (go to beeper pile)
- ဘိပါပုံကို (မူလနေရာတွင်) နှစ်ဆလုပ် (double beeper pile)
 - ▶ ဘိပါပုံကို ရှေ့ကွန်နာမှာ နှစ်ဆလုပ်
 - ▶ နိုင်ကွန်နာသို့ ဘိပါပုံ ပြန်ရွှေ့

ဘိပါပုံဆိုကို သွားတာက လွှယ်တယ်

```
def go_to_beeper_pile():
    while no_beeper_present():
        move()
```

နိုင်နေရာမှာ ဘိပါပုံ နှစ်ဆလုပ်တဲ့ ကိစ္စကို နောက်တစ်ဆင့် ထပ်ခွဲထားတယ်။ ရှေ့ကွန်နာမှာ နှစ်ဆ ပုံတာနဲ့ နိုင်နေရာ ပြန်ရွှေ့တာ ကိစ္စနှစ်ခု ပါဝင်တယ်။ ဒီအတွက် ဖန်ရှင်နှစ်ခု ရှိတယ်လို့ မှတ်ယူရပါမယ်။ ငါးတို့ လုပ်ဆောင်ချက်က ဘာလဲ ငါးတို့ရဲ့ ပရိုနဲ့ ပိုစ် ကွန်ဒီရှင်တွေ ဘယ်လို့ဖြစ်သင့်လဲ တိတိကျကျ စဉ်းစားထားရမှာပါ။ ဒီအတွက်ကို စိတ်ထဲမှာပဲ လုပ်လိုရသလို အောက်ပါအတိုင်း ဗလာဖန်ရှင် (empty

function) သတ်မှတ်ပြီး docstring နဲ့ တိတိကျကျ ချရေးထားတာဟာလည်း ပရီဂရမ်မာ အများစုံ လုပ်လေ့လုပ်ထိုတဲ့ အလေ့အထတစ်ခုပါ။

```
def double_beeper_pile_next():
    """
    ဘိပါပုံကို ရွှေ့ကွန်နာတွင် နှစ်ဆဖြစ်အောင် ရွှေ့ပေးသည်
    Precondition: ဘိပါပုံပေါ်ပွဲတွင် အရောက် မျက်နှာမူ၍ ရှိနေမယ်
    Postcondition: နှစ်ဆဘိပါပုံပေါ်ပွဲတွင် အရောက် မျက်နှာမူ၍
    """
    pass

def move_beeper_pile_next():
    """
    ဘိပါပုံကို ရွှေ့ကွန်နာသို့ ရွှေ့ပေးသည်
    Precondition: ဘိပါပုံပေါ်ပွဲတွင် အနောက်ဘက်မျက်နှာမူ ရှိနေမယ်
    Postcondition: ရွှေ့ကို ရွှေ့ပြီးဘိပါပုံပေါ်ပွဲတွင် အနောက်ဘက်မျက်နှာမူ ရှိနေမယ်
    """
    pass
```

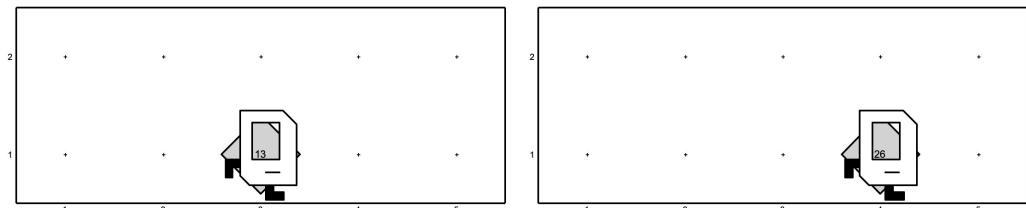
Python မှာ ဗလာဖန်ရှင်ဆိုပေမဲ့ စတိတ်မန့်တစ်ခုတော့ ပါရမယ်။ မဟုတ်ရင် ဆင်းတက်စ် အမှားဖြစ်မှပါ။ ဒါကြောင့် pass စတိတ်မန့်ကို ယာယီအနေနဲ့ သုံးလေ့ရှိတယ်။ ဒေါ်မိစတိတ်မန့် (dummy statement) လို့ ခေါ်ပါတယ်။ ဖန်ရှင်နှစ်ခုလဲ ပရီနဲ့ ပိုစ် ကွန်ဒီရှင်တွေကို ပုံးပို့ (၃.၁၀) နဲ့ (၃.၁၁) မှာ တွေ့နိုင်ပါတယ်။ နှစ်ခုလဲး မျက်နှာမူတဲ့ဘက် မပြောင်းတာကို ကရာပြုပါ။ ဘိပါပုံက မူလနေရာမဟုတ်ဘဲ ရှေ့ကိုရောက်သွားတာကိုလည်း ကရာပြုပါ။

မူလနေရာမှာပဲ ဘိပါနှစ်ဆဖြစ်အောင် လုပ်တဲ့ ဖန်ရှင်က ဒီလိုဖြစ်ပါမယ်

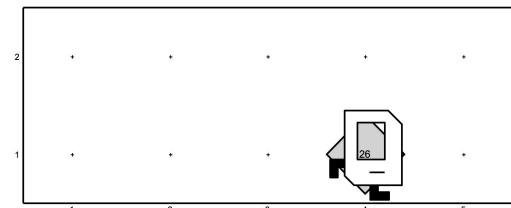
```
def double_beeper_pile():
    """
    ဘိပါပုံကို မူလနေရာတွင်ပင် နှစ်ဆတိုးပေးသည်
    Precondition: ဘိပါပုံပေါ်ပွဲတွင် အရောက်မျက်နှာမူလျက် ရှိနေ
    Postcondition: နှစ်ဆတိုးပြီး ဘိပါပုံပွဲတွင် မူလအတိုင်း အရောက်မျက်နှာမူလျက် ရှိနေ
    """
    double_beeper_pile_next()
    turn_left()
    turn_left()
    move_beeper_pile_next()
    turn_left()
    turn_left()
```

ပထမဖန်ရှင်ပြီး ဒုတိယဖန်ရှင်အတွက် အဆင်သင့်ဖြစ်အောင် ဘယ်ဘက်နှစ်ခါ လှည့်ရပါမယ်။ နောက်ဆုံးမှာလည်း အရောဘက် ပြန်လှည့်ပေးရမယ်။ မဟုတ်ရင် အခုဖန်ရှင်ရဲ့ သတ်မှတ်ထားတဲ့ ပိုစ်ကွန်ဒီရှင်နဲ့ မကိုက်ညီဘဲ အနောက်ဘက်လှည့် အနေအထားဖြစ်နေမှာ။

အထက်ပါဖန်ရှင်မှာ ဘယ် နှစ်ခါလှည့်ကို နှစ်ကြိမ်လုပ်ထားပါတယ်။ ဆန်ကျင်ဘက် အရပ်ကို လှည့်ဖို့အတွက် ရည်ရွယ်တာပါ။ ဒီအတွက် turn_around ဖန်ရှင် ရှိသင့်ပါတယ်။ ရှိမယ်ဆိုရင်

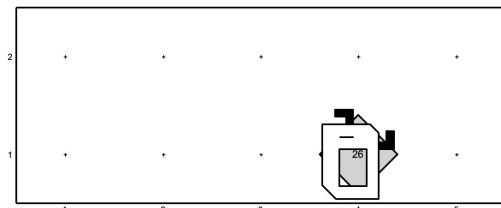


(က)

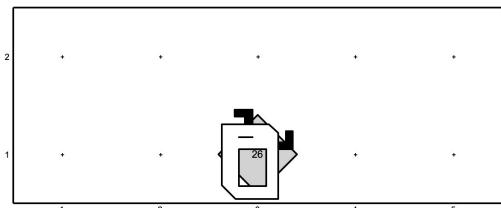


(ခ)

ပို့ ၃.၀၀



(က)



(ခ)

ပို့ ၃.၀၀

```
def double_beeper_pile():
    """
    ဘိပို့ကို မူလနေရာတွင်ပင် နှစ်ဆတိုးပေးသည်
    Precondition: ဘိပို့ပေါ်တွင် အရှေ့ဘက်မျက်နှာမူလျက် ရှိစေ
    Postcondition: နှစ်ဆတိုးပြီး ဘိပို့တွင် မူလအတိုင်း အရှေ့ဘက်မျက်နှာမူလျက် ရှိစေ
    """
    double_beeper_pile_next()
    turn_around()
    move_beeper_pile_next()
    turn_around()
```

ဒီနေရာမှာ မှတ်သားသင့်တာတစ်ခုက ဒီကွန်ပို့စွဲ လုပ်တဲ့အခါ တစ်ခါတည်းနဲ့ ပြီးပြည့်စုတဲ့ ရလဒ် ဖြစ် ချင်မှု ဖြစ်မှုပါ။ တစ်ခါတစ်ရုံ ပရိုဂရမ် ရေးနေရင်း စိတ်ကူးသစ် သီးမဟုတ် ပို့ကောင်းတဲ့နည်းလမ်း ခေါင်းထဲ ပေါ်လာတတ်ပါတယ်။ တဖြည်းဖြည့်း သဘောပေါက်လာတတ်တယ်။ ဒီအခါမှာ ဒီကွန်ပို့စွဲ လုပ် တာကို လိုအပ်သလို အလိုက်သင့် ပြောင်းပေးနိုင်ပါတယ်။ အထက်က ဥပမာမှာ turn_around လိုအပ် မယ်ဆိုတာ ကြို့မသိခဲ့ပါဘူး။

double_beeper_pile_next နဲ့ move_beeper_pile_next အတွက် ဆက်လက်စဉ်းစားပါ မယ်။ အတန်အသင့် လွယ်ကူးမယ် ယူဆတဲ့အတွက် နောက်တစ်ဆင့် ထပ်မံ့ထပ်ဘူး။

```
def double_beeper_pile_next():
    """
    ဘိပို့ကို ရေးကွန်နာတွင် နှစ်ဆဖြစ်အောင် ရွှေ့ပေးသည်
    Precondition: ဘိပို့ပေါ်တွင် အရှေ့ဘက် မျက်နှာမူလျက် ရှိနေမယ်
    Postcondition: နှစ်ဆဘိပို့ပေါ်တွင် အရှေ့ဘက် မျက်နှာမူလျက် ရှိနေမယ်
```

```

    """
    while beepers_present():
        pick_beeper()
        move()
        put_beeper()
        put_beeper()
        turn_around()
        move()
        turn_around()

    move()

def move_beeper_pile_next():
    """
    ဘိပိုကို ရွှေ့ကွန်နာသို့ ရွှေ့ပေးသည်
    Precondition: ဘိပိုပေါ်တွင် အနောက်ဘက်မျက်နှာမှ ရှိနေမယ်
    Postcondition: ရွှေ့ကို ရွှေ့ပြီးဘိပိုပေါ်တွင် အနောက်ဘက်မျက်နှာမှ ရှိနေမယ်
    """

    while beepers_present():
        pick_beeper()
        move()
        put_beeper()
        turn_around()
        move()
        turn_around()

    move()

```

အခုနောက်ဆုံး turn_around, main နဲ့ အန်ထရိပိုင့် ဖြည့်စီပွဲ ကျန်ပါတယ်။ ပရိုဂရမ် အစအဆုံး လေ့လာကြည့်ပါ။

```

# File: double_beeper_pile.py
from stanfordkarel import *

def main():
    """Karel doubles the number of beepers in a beeper pile"""
    go_to_beeper_pile()
    double_beeper_pile()

def double_beeper_pile():
    double_beeper_pile_next()
    turn_around()
    move_beeper_pile_next()
    turn_around()

```

```
def go_to_beeper_pile():
    while no_beeper_pile():
        move()

def double_beeper_pile_next():
    while beepers_pile():
        pick_beeper()
        move()
        put_beeper()
        put_beeper()
        turn_around()
        move()
        turn_around()

    move()

def move_beeper_pile_next():
    while beepers_pile():
        pick_beeper()
        move()
        put_beeper()
        turn_around()
        move()
        turn_around()

    move()

def turn_around():
    turn_left()
    turn_left()

if __name__ == "__main__":
    run_karel_program("double_beeper_pile")
```

အခန်း ၄

ကားရဲလ်နှင့် ရီကားဆိုစ် ဖန်ရှင်များ

ဖန်ရှင်တစ်ခုကနေ ‘အဗြား’ ဖန်ရှင်တွေ ခေါ်သုံးတာကို အခန်း (၃) မှာ တွေ့ခဲ့ပြီးပါပြီ။ ဒါပေမဲ့ ဖန်ရှင်တစ်ခုက ငှုံးကိုယ်ငှုံး ပြန်ခေါ်ထားတာကိုတော့ မကြိုးသေးပါဘူး။ ဖန်ရှင်တွေဟာ ဆေ့ဖို့ အဆောက်အအီး တည်ဆောက်ရာမှာ မရှိမဖြစ်တဲ့ အခြေခံအုတ်ချုပ်တွေလို ဆိုရှုမှပါ။ ငှုံးကိုယ်တိုင်ကို ပြန်လည်အသုံးပြု၍ ဖန်ရှင်အုတ်ချုပ်တစ်ခု ဖန်တီးလို ရနိုင်ပါမလား။ ဒီမေးခွန်းဟာ ထူးဆန်းကောင်း ထူးဆန်းနေပါလိမ့်မယ်။ အခြေခံကျပြီး စိတ်ဝင်စားစရာကောင်းတဲ့ ဖို့လော်ဆော်ဖီ မေးခွန်းလည်း ဖြစ်တယ်။

ဖန်ရှင် သတ်မှတ်ချက်ထဲမှာ ငှုံးဖန်ရှင်ကိုယ်တိုင်ကို ပြန်ခေါ်လို ရပါတယ်။ ရီကားဆိုစ် ဖန်ရှင် (*recursive function*) လို ခေါ်တယ်။ ရီကားဆိုစ် ဖန်ရှင်တွေဟာ ဘီဂိုင်နာ ပရိုကရမ်မာအတွက် နားလည်ဖို့ ခက်ခဲတဲ့ သဘောတရားအဖြစ် ယူဆကြတာကြောင့် စာအုပ် အတော်များများမှာ နောက်ကျပြီး ဖော်ပြလေ့ရှိတယ်။ တကယ်က လူများစု ထင်/ပြောသလို နားမလည်နိုင်လောက်အောင် ရှုပ်တွေး ခက်ခဲတဲ့ သဘောတရား မဟုတ်ပါဘူး။ သာမန်လူအားလုံး နားလည်နိုင်ပါတယ်။ ဒါကြောင့် တော့စီးစီး အခုပဲ မိတ်ဆက်ပေးလိုက်ပါတယ်။ အကယ်၍ နားမလည်ခဲ့ရင်လည်း ပြသာမရှိပါဘူး။ အကြမ်းဖျော်းလောက် ဖတ်ကြည့်ပြီး နောက်လာမဲ့ အခန်းတွေကို ကျော်သွားနိုင်ပါတယ်။ နောင်တစ်ခို့ကျော် ပြန်လာဖတ်ပေါ့။

၄.၁ ရီကားဆိုစ် ဖန်ရှင် ဘယ်လို အလုပ်လုပ်လဲ

ရီကားဆိုစ် ဖန်ရှင် ဥပမာတစ်ခုကို လေ့လာကြည့်ပါမယ်။ အောက်ဖော်ပြပါ ဖန်ရှင်ဟာ ငှုံးကိုယ်တိုင်ကို ငှုံး ပြန်ခေါ်ထားတာ ကရှုပြုကြည့်ပါ။ *recursive call* လို ကွန်းမန်ရေးထားတဲ့ လိုင်းမှပါ။

```
def make_beeper_row():
    if front_is_clear():
        put_beeper()
        move()
        make_beeper_row() # recursive call
    else:
        put_beeper()
```

ဒီဖန်ရှင်ကို ခေါ်လိုက်ရင် ဘာဆက်ဖြစ်မလဲဆိုတာ စိတ်ဝင်စားစရာပါ။ ဖန်ရှင်မစတင်မဲ့ အနေအထား ကို ပဲ (၄.၁) မှာ ကြည့်ပါ။ ဖန်ရှင်ကို ကန်းဦး စခေါ်လိုက်တာမို့လို ဒီ ရည်ညွှန်းပါမယ်။

```
# initial call
make_beeper_row()
```



ပို ၄.၁

ဖန်ရှင် စတင် လုပ်ဆောင်ပါမယ်။ ရှေ့မှာရှင်းနေတဲ့ အတွက် if ဘလောက်ကို လုပ်မှာပါ

```
put_beeper()
move()
make_beeper_row() # recursive call
```

ဘိပါချာ ရှေ့တိုး ပုံ (၄.၂) (က) ကြည့်ပါ။ ပြီးရင် သူကိုယ်သူ ပြန်ခေါ်ထားတယ်။ ဒါဟာ ပထမဆုံး တစ်ကြိမ်ပါ။ ဖန်ရှင်ခေါ်ရင် ဖြစ်မြေအတိုင်းပဲ ဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို ဆောက်ရွက်တာပေါ့။ ဒီတော့ make_beeper_row ဖန်ရှင်ဘလောက်ကိုပဲ တစ်ခါထပ်လုပ်မှာပါ။ ရှေ့မှာ ရှင်းနေတဲ့အတွက် if အပိုင်းကို လုပ်တယ်

```
put_beeper()
move()
make_beeper_row() # recursive call
```

ဘိပါချာ ရှေ့တိုး ပုံ (၄.၂) (ခ) ပြီး သူကိုယ်သူ ပြန်ခေါ်ထဲကိစ္စ တစ်ခါထပ်ဖြစ်ပြန်တယ်။ ဒါနဲ့ဆို နှစ်ကြိမ်။ ဖန်ရှင်ဘလောက် အလုပ် ပြန်လုပ်မယ်။ ရှေ့မှာရှင်းတယ်၊ if ကိုပဲ ထပ်လုပ်

```
put_beeper()
move()
make_beeper_row() # recursive call
```

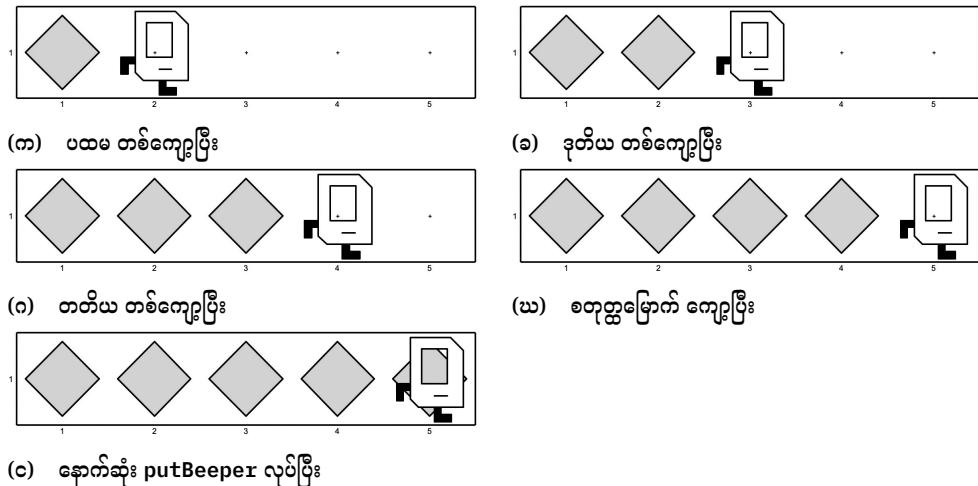
ပုံ (၄.၂) (ဂ) နေရာရောက်ပြီး သူကိုယ်သူ ထပ်ခေါ်ထားပြန်တယ်။ သုံးကြိမ်ရှိပြီ။ ဒီတစ်ခါလည်း if အပိုင်းပဲ ထပ်လုပ်

```
put_beeper()
move()
make_beeper_row() # recursive call
```

ရှေ့တိုးပြီးရင် နံရံပိတ်နေပြီ ပုံ (၄.၂) (ဃ)။ သူကိုယ်သူ ခေါ်တယ်။ ရှေ့မှာ ပိတ်နေတဲ့အတွက် else အပိုင်းကို လုပ်မှာပါ။

```
put_beeper()
```

သူကိုယ်သူ ပြန်ခေါ်တဲ့ကန္တ ထပ်မဖြစ်တော့ဘူး။ ဒီမှာပဲ ပြီးဆုံးသားတယ်။ ရုံကားဆစ်ဖုံး ဖန်ရှင် အလုပ် လုပ်ပုံ အခြေခံသဘောတရားက ဒါပါပဲ။ loop တွေ မသုံးဘဲ ပြန်ကျော်နေတဲ့ သဘောကို ရုံကားဆစ်ဖုံး ဖန်ရှင်မှာ တွေ့ရပါတယ်။ ဖန်ရှင်က သူကိုယ်သူ (သို့ ကိုယ့်ကိုကိုယ်) ပြန်ခေါ်တာကို recursive call လို



ပုံ ၄၂

ခေါ်ပါတယ်။ ဒုက္ခလာများရှင်းအရ recursive function တွေမှာ recursive call အနည်းဆုံး တစ်ခု ပါ ရမှာ ဖြစ်ပါတယ်။

```
def make_beeper_row():
    if front_is_clear():
        put_beeper()
        move()
        make_beeper_row()
    else:
        put_beeper()
make_beeper_row()
```

ရိုက္ခားဆစ်ဖောက် ဖြစ်တာကို စိတ်ကုံးပံ့ဖော်ကြည့်ဖို့ ပြထားတာပါ။ အပိုင်ဆုံး မြားအနက်က ကန် ဦး ဖန်ရှင်းကောလ် စတင်တာဖြစ်ပေါ်တာကို ဖော်ပြထာ်။ ရိုက္ခားဆစ်ဖောက် မဟုတ်သေးဘူး။ မြား အပြာက ရိုက္ခားဆစ်ဖောက်ကြောင့် ဖန်ရှင်းအစ ပြန်ရောက်သွားတာကို ပြထာ်။ ပထမနဲ့ ဒုတိယ ရိုက္ခားဆစ်ဖောက် နှစ်ခုအတွက်ပြထားတာပါ။ ရွှေ့က ဥပမာအတွက် မြားအပြာ လေးခု ရိုရှုံးပါ (ရိုက္ခားဆစ်ဖောက် လေးကြိမ်အတွက်)။ မြားအပြာ နောက်ထပ် နှစ်ခုရှိတယ် မှတ်ယူပါ (ပုံမှာထပ်ထည့်ရင် ကြပ်ညပ်ပြီး ကြည့်ရှုရှုပ်လို့ မဆွဲပြတာ)။ လေးကြိမ်မြားကို ရိုက္ခားဆစ်ဖောက် ထပ်မဖြစ်တော့ဘူး (if အပိုင်းကို မလုပ်တော့တဲ့အတွက်)။ ဘို့ပါချုပြုး ဖန်ရှင်းကောလ် စခဲ့တဲ့နေရာကို ပြန်ရောက် သွားမယ် (မြားအနဲ့)။ ဘယ်လို့ ပြန်ရောက်သွားတာလဲ ဆက်ကြည့်ရအောင်။

နောက်ဆုံး ရိုက္ခားဆစ်ဖောက်မှ ပြန်လုပ်ခြင်း

နောက်ဆုံး ရိုက္ခားဆစ်ဖောက်ကနေ မူလ ဖန်ရှင်၏ခဲ့တဲ့ နေရာကို ဘယ်လိုပြန်ရောက်သွားတာလဲ။ ဒီ ကိစ္စနားလည်ဖို့ ဖန်ရှင် return ပြန်ခြင်းအကြောင်း အရင်ကြည့်ရပါမယ်။ ဖန်ရှင်းကောလ် လုပ်ဆောင် တဲ့အခါ အဲဒီဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘလောက်ဆီကို ခုန်ကျော် ရောက်ရှိသွားမှုပါ။ ဖန်ရှင်းဘလောက်ကို လုပ်ဆောင်ပြီး ခေါ်ခဲ့တဲ့ နေရာကို ပြန်လည်ရောက်ရှိသွားမှု ဖြစ်တယ်။ ဒီဖြစ်စဉ်ကို ဖန်ရှင် return ပြန် တယ်လို့ ပြောပါတယ်။

```

def main():
    turn_right() ←
    move()
    turn_right()
    move()

def turn_right():
    turn_left() ←
    turn_left()
    turn_left() →

```

ပထမ turn_right ကောလ် လုပ်ဆောင်တဲ့အခါ ကောလ်လုပ်တဲ့ နေရာကနေ turn_right ဖန်ရှင်ထဲ
ကို jump လုပ်ပြီး ရောက်သွားတယ်။ မြှားအနက်နဲ့ ပြထားတယ်။ ဖန်ရှင်ဘေးလောက် လုပ်ဆောင်ပြီးတဲ့
အခါ ခေါ်ခဲ့တဲ့နေရာ main ဖန်ရှင်ထဲ ပြန်ရောက်သွားတယ် (မြှားအနီး)။ ဒုတိယ turn_right လည်း
ထိုနည်းတူစွာပဲ ဖြစ်ပါတယ်။

```

def main():
    turn_right()
    move()
    turn_right() ←
    move()

def turn_right():
    turn_left() ←
    turn_left()
    turn_left() →

```

နှစ်ဆင့် သုံးဆင့် ဖန်ရှင်ကောလ်တွေမှာလည်း ဒီသဘောတရား အတိုင်းပါပဲ။ အောက်ဖော်ပြပါ ပရီ
ဂရမ်ကုဒ်ကို ကြည့်ပါ။ main ဖန်ရှင်ထဲကနေ do_tricks ဆိုကို ရောက်သွားမယ်။ do_tricks ထဲက
နေ put_two ထဲကို ရောက်သွားမယ်။

```

def main():
    do_tricks() ←
    move()

def do_tricks():
    move() ←
    put_two() ←
    turn_left()
    move() →

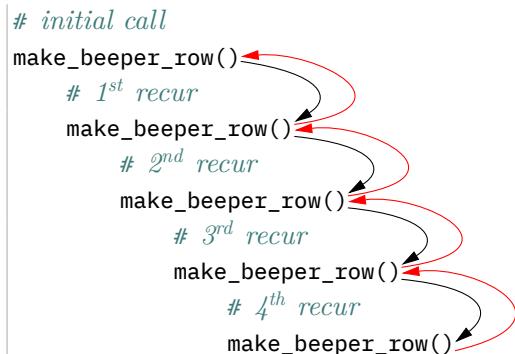
def put_two():
    put_beeper() ←
    put_beeper() →

```

put_two ပြီးသွားတဲ့အခါ do_tricks ထဲ ပြန်ရောက်သွားမယ်။ turn_left, move ဆက်လုပ်ပြီး do_tricks ခေါ်ခဲ့တဲ့နေရာ main ထဲ ပြန်ရောက်သွားမယ်။ နောက်ဆုံး main ထဲက move ကို ဆက်လုပ်ပါတယ်။

ဖန်ရှင် အဆင့်ဆင့် ခေါ်ထားတဲ့အခါ နောက်ဆုံးခေါ်တဲ့ ဖန်ရှင်က အရင်ဆုံး return ပြန်ပါတယ်။ main ကနေ do_tricks ကိုခေါ်။ do_tricks ကနေ put_two ကိုခေါ်ထားရင် put_two ကနေ do_tricks ဆိုကို အရင် return ပြန်တယ်။ ပြီးတော့မှာ do_tricks ကနေ main ကို ပြန်ရောက်မှာပါ။ ဒီသဘောအရ put_two return မပြန်မချင်း do_tricks ဖန်ရှင်မပြီးသေးဘူး။ put_two ကနေ ပြန်လာပြီးမှ ကျွန်တဲ့ turn_left, move ဆက်လုပ်တယ်။ ပြီးတော့မှာ do_tricks ဖန်ရှင်က return ပြန်ပါတယ်။

ရိုကားဆစ်စ် ဖန်ရှင်ကောလွှာတွေ ဘယ်လို့ return ပြန်လဲ။ ရှေ့ကလို မြှားတွေနဲ့ ဆွဲပြလို့ ရပေမဲ့ ကြည့်ရတာ ရှုပ်ရှက်ခတ်နေမှာပါ။ အခုလို မြင်ကြည့်ရင် ပိုရှင်းပါတယ်။



မြှားအနက်တွေက ဖန်ရှင်ကောလွှာတွေ တစ်ဆင့်ပြီးတစ်ဆင့် ဖြစ်တာကို ပြတာပါ။ အထက်မှအောက် အစီအစဉ်အတိုင်း ဖြစ်ပါတယ်။ လေးကြိမ်မြှာက်မှာ နောက်ထပ် ရိုကားဆစ်စ်ကောလွှာတွေက အရင်ဆုံး return စပြန်ပါတယ် (အောက်ဆုံး မြှားအနိန့် ပြထား)။ ဒီအခါ တတိယ ရိုကားဆစ်စ်ကောလွှာတွေက အရင်ဆုံး return စပြန်ပါတယ် (အောက်ဆုံး မြှားအနိန့် ပြထား)။ ဒီအတိုင်း တစ်ဆင့်ပြီးတစ်ဆင့် အထက်ကို return ပြန်ပြီး နောက်ဆုံးမှာ ပထမဆုံး ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားမှာပါ။ (အခုပြထားတာကို တကယ့် Python ကုဒ် အနေနဲ့ မယူဆရပါ။ ဖြစ်စဉ် နားလည်အောင် ပြခြင်းသာဖြစ်ပါတယ်)။ နဲ့ပြင်ထားတဲ့ make_beeper_row ဗားရှင်းမှာ return ပြန်တဲ့ ဖြစ်စဉ်ကို ကြည့်ရအောင်။

| make_beeper_row()

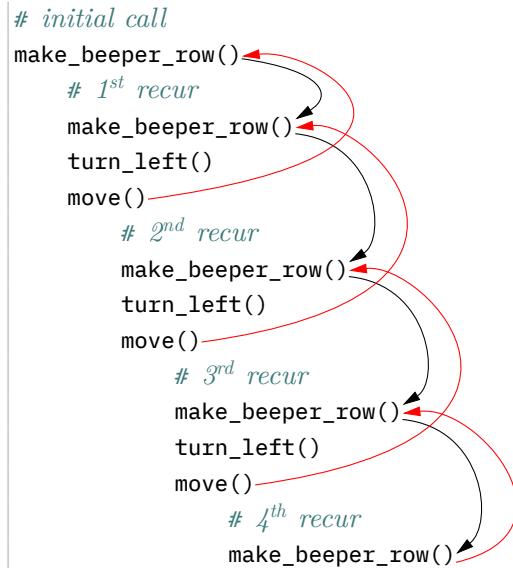
```

def make_beeper_row():
    if front_is_clear():
        put_beeper()
        move()
        make_beeper_row()
        turn_left()
        move()
    else:
        put_beeper()

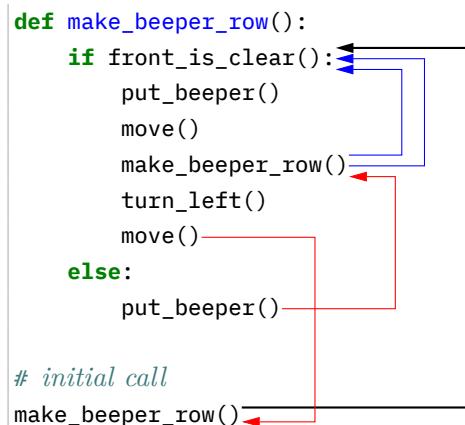
```

if အပိုင်း ရိုကားဆစ်စ်ကောလွှာပြီး turn_left နဲ့ move ထပ်ဖြည့်ထားတာ။ ရိုကားဆစ်စ်ကောလွှာတွေက အပိုင်း ရိုကားဆစ်စ်ကောလွှာတွေက ပြန်လာပြီးမှ ဒီနှစ်ခု ဆက်လုပ်မှာပါ။ နောက်ဆုံး ရိုကားဆစ်စ်ကောလွှာတွေက အရင်ဆုံး return စပြန်ပါတယ်။ ဒီအတိုင်း တစ်ဆင့်ပြီးတစ်ဆင့် အထက်ကို return ပြန်ပြီး နောက်ဆုံးမှာ ပထမဆုံး ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားမှာပါ။ (အခုပြထားတာကို တကယ့် Python ကုဒ် အနေနဲ့ မယူဆရပါ။ ဖြစ်စဉ် နားလည်အောင် ပြခြင်းသာဖြစ်ပါတယ်)။ နဲ့ပြင်ထားတဲ့ make_beeper_row ဗားရှင်းမှာ return ပြန်တဲ့ ဖြစ်စဉ်ကို ကြည့်ရအောင်။

တော့မှ တတိယအောက် `turn_left` နဲ့ `move` ကို လုပ်ဆောင်မှာပါ။ ပြီးမှ တတိယကောလ် `return` ပြန် တယ်။ ဒီအတိုင်း အထက်ကို တက်သွားပြီး ကျွန်ုန်းနေသေးတဲ့ စတိတ်မနဲ့တွေကို လုပ်ဆောင်ပါတယ်။ ပထမ ဆုံးကောလ်အောက် ကျွန်ုန်းနေတာတွေ နောက်ဆုံးကျွမ်းပြီးမှာပါ။



ရိုကားဆစ်စကေလ် နှစ်ခါပဲ ဖြစ်မယ်ဆိုရင် အောက်ပါအတိုင်း မြင်ကြည့်လို့ ရပါတယ်။ မြှားအပြာ နှစ်ခါက ရိုကားဆစ်စကေလ်ဖြစ်တာ။ ဒုတိယကောလ်က ဘိပါချုပြီး (else အပိုင်း) အရင် `return` မယ်။ အထက်ကို ညွှန်တဲ့ မြှားအနီးကို ကြည့်ပါ။ ဘယ်လူညွှန်တဲ့ ရှေ့တိုးပြီး ပထမ ကောလ်က နောက်မှ initial call လုပ်ခဲ့တဲ့ဆို ပြန်ရောက်တာ။ အောက်ကိုညွှန်တဲ့ မြှားအနီးကို ကြည့်ပါ။



ရိုကားဆစ်စ ဖန်ရှင်အကြောင်း လေ့လာတဲ့အခါ ဘိုင်နာအများစုံ ကြော်ကြော်လာက် နားလည်ဖို့ အတွက် အခက်အခဲဆုံးတစ်ခါက `return` ပြန်တဲ့ သဘောတရားပါပဲ။ များများစဉ်းစား၊ များများလေ့ကျင့် ရင် ဒီအခက်အခဲ ကျော်ဖြတ်နိုင်မှာပါ။ `if...else` အပြီးမှာ `put_beeper` လေးတစ်ခဲ့ပဲ ထပ်ဖြည့်လိုက် ရင် ဘယ်လိုဖြစ်မလဲ။

```

# File: make_beeper_row2.py
def make_beeper_row():

```

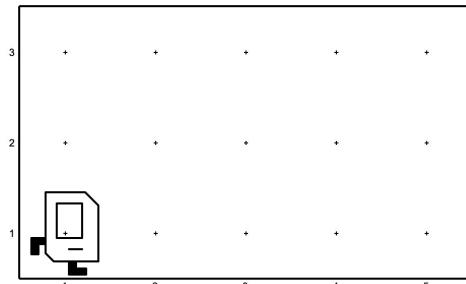
```

if front_is_clear():
    put_beeper()
    move()
    make_beeper_row()
    turn_left()
    move()
else:
    put_beeper()
    put_beeper()

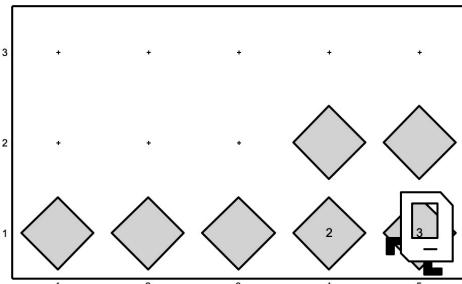
# initial call
make_beeper_row()

```

ပုံ (၄၃) (က) နဲ့ (ခ) က မတိုင်မိနဲ့ ပြီးနောက် အခြေအနေပါ။ နောက်ဆုံးမှာ ဘိပါလေးခုကို ပါတ်လည် ဘယ်လိုချသွားလဲ စဉ်းစားကြည့်ပါ။ ရှေ့မှောဖော်ပြခဲ့သလို ဖန်ရှင်ကောလ်တွေ တစ်ဆင့်ပြီးတစ်ဆင့် ဖြစ်ပုံနဲ့ return ဖြစ်ပုံကို မြှားဆွဲကြည့်ပါ။



(က)



(ခ)

ပုံ ၄၃

၄.၂ ရီကားဆစ်ဖုနည်းဖြင့် ပုံစံဖြေရှင်းခြင်း

ရှေ့စက်ရှင်မှာ လေ့လာခဲ့တာက ရီကားဆစ်ဖုနည်း ဘယ်လို အလုပ်လုပ်လဲပဲ ရိုပါသေးတယ်။ တစ်နည်းအားဖြင့် မကြန်စ်မေး (mechanism) ကို လေ့လာတာပါ။ အခုက်ခုမှုတစ်ခါ ရီကားဆစ်ဖုနည်းတွေနဲ့ ပုံစံတွေ ဘယ်လိုဖြေရှင်းမလဲ ဆက်လက်လေ့လာပါမယ်။ ‘ရီကားဆစ်ဖုနည်းတွေကို စဉ်းစားခြင်း’ (thinking recursively) သို့မဟုတ် ‘ရီကားဆစ်ဖုနည်းဖြင့် ပုံစံဖြေရှင်းခြင်း’ (solving problems recursively) ကို လေ့လာမှာပါ။

ရီကားဆစ် စဉ်းစားခြင်း ဥပမာ (၁)

ကွန်နာမှုရှိတဲ့ ဘိပါတွေအားလုံး ကောက်မယ်ဆိုပါစို့။ ဘိပါတစ်ခုနဲ့ အထက်ရှိရှိတယ်။ ဘိပါမရှိတာလည်း ဖြစ်နိုင်တယ် ယူဆပါ။ ဒီအတွက် ရီကားဆစ်ဖုနည်း သတ်မှတ်ပါမယ်။

```

def pick_all beepers():
    ...
    # to do soon

```

ဖြေရှင်းမဲ့ ကိစ္စတစ်ခုကို ငြင်းကိုယ်ဝိုင်နဲ့ ပုံပန်းသဏ္ဌာန်တူပြီး အရွယ်အစားအားဖြင့် တစ်ဆင့်ထက် တစ်ဆင့် သေးငယ်တဲ့ ကိစ္စတွေအဖြစ် ခဲ့မြေးကြည့်ပါတယ်။ ဥပမာ ဘိပါဝါးခုရှိတဲ့ ကိစ္စကို ဘိပါလေးခု သုံးခု၊ နှစ်ခု နဲ့ တစ်ခု ရှိတဲ့ ကိစ္စတွေအဖြစ် ခဲ့ပြီး ပြင်ကြည့်ရမှာပါ။ ဒီကိစ္စမှာ ဘိပါအရေအတွက်ဟာ အရွယ်အစားပဲ။ လေးခုရှိတဲ့ ကိစ္စဟာ ငါးခုရှိတဲ့ ကိစ္စထက် အရွယ်အားဖြင့် တစ်ဆင့်ငယ်တာပေါ့။ ဘိပါမရှိ တာလည်း ဖြစ်နိုင်တော့ သူည်ဘိပါဟာ အငယ်ဆုံးဖြစ်တယ်လို့ ယူဆနိုင်တယ်။

`pick_all beepers` ဖန်ရှင်ဟာ လက်ရှိဖြေရှင်မဲ့ အရွယ်အစားထက် တစ်ဆင့်ငယ်တဲ့ ကိစ္စကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူရပါမယ်။ လက်ရှိဖြေရှင်းမဲ့ ကိစ္စက ဘိပါဝါးခု ကောက်ရမယ်ဆုံးရင် ဘိပါလေးခု ကောက်နိုင်ပြီးသားလို့ ယူဆရမှာပါ။ n ဘိပါရှိတယ် ဆုံးရင် $n - 1$ ဘိပါကို ကောက်နိုင်ပြီးသားလို့ ယူဆရမယ်။ Top-down နည်းမှာလည်း ဒီလိုပဲ မရှိသေးဘဲ မလုပ်နိုင်သေးဘဲ ရှိတယ် လုပ်နိုင်တယ် မှတ်ယူပြီး စဉ်းစားဖြေရှင်းတာကို ပြန်အောင်မှတ်ရမှာပါ။ ရှိကားဆစ်ဖို့ စဉ်းစားတဲ့အခါမှာ လက်ရှိသတ်မှတ်နေတဲ့ ဖန်ရှင်ကိုယ်တိုင်ကို ရှိပြီးဖြစ်တယ်လို့ သဘောထားရတာ။

ပြီးတဲ့အခါ လက်ရှိကိစ္စကနေ သူ့ထက်တစ်ဆင့်ငယ်တဲ့ ကိစ္စဖြစ်သွားအောင် ဘာလုပ်ရမလဲ စဉ်းစားရတယ်။ ဘိပါဝါးခုကနေ လေးခုဖြစ်အောင် ဘိပါတစ်ခု ကောက်ရမှာပေါ့။ ယျေဘုံယျေပြောရင် n ဘိပါရှိရင် ဆုံးရင် $n - 1$ ဘိပါဖြစ်သွားအောင် ဘာလုပ်ရမလဲ စဉ်းစားတာ။

လက်ရှိဖန်ရှင်ဟာ $n - 1$ ဘိပါကို ကောက်နိုင်ပြီးသားလို့ ယူဆထားတယ်။ n ဘိပါရှိရင် ဘိပါတစ်ခု ကောက်လိုက်ရင် $n - 1$ ဘိပါဖြစ်သွားမယ်။ ကျွန်ုန်နေတဲ့ $n - 1$ ဘိပါကောက်ဖို့ လက်ရှိဖန်ရှင်ကိုပဲ ပြန်ခေါ်လိုက်မှာပေါ့။

```
# only partially done
def pick_all beepers():
    ...
    # to pick (n) beepers
    pick_beeper()
    pick_all beepers() # assuming it can pick (n - 1) beepers
    ...
```

ရှိကားဆစ်ဖို့ စဉ်းစားတတ်ဖို့ ဒီအဆင့်က အခရာအကျဆုံးပဲ။ တစ်ဆင့်ငယ်တဲ့ ကိစ္စ $n - 1$ ကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူပြီး လက်ရှိကစ္စ n ကို ဘယ်လို့ ဖြေရှင်းမလဲ စဉ်းစားသွားတာ။

ဖန်ရှင်သတ်မှတ်ချက်က မပြီးသေးပါဘူး။ အသေးငယ်ဆုံး ကိစ္စကို ချင်းချက်အနေနဲ့ စဉ်းစားရမယ်။ အသေးငယ်ဆုံးကိစ္စက ဘိပါမရှိတာ (သူည်) ဖြစ်တယ်။ ဘိပါမရှိရင် ဘာမှုလုပ်စရာမလိုဘူး။ n နဲ့ $n - 1$ ဘိပါအတွက် အထက်ပါအတိုင်း စဉ်းစားတဲ့အခါ $n \neq 0$ လို့ ယူဆရမှာပါ။ ဒါကြောင့် ဘိပါရှိမှာပဲ လုပ်အောင် အခုလိုဖြစ်ရပါမယ်

```
# finished
def pick_all beepers():
    if beepers_present():
        pick_beeper()
        pick_all beepers()
```

ရှိကားဆစ်ဖန်ရှင် မှန်မမှန် စ်ဆေးခြင်း

ရှိကားဆစ်ဖန်ရှင် တက်စ် (test) လုပ်ရင် အသေးဆုံးကိစ္စကနေ စရာတယ်။ ပြီးခဲ့တဲ့ ဖန်ရှင်အတွက် အသေးဆုံးက သူည်ပါ။ ဘိပါမရှိရင် ဖန်ရှင်က မှန်ရဲလား အရင်ဆုံး စစ်ကြည့်ပါမယ်။

```
# assume no beeper
pick_all_beeper()
```

if ဘလောက် မလုပ်ဆောင်ဘဲ return ဖြစ်သွားမှာပါ (ရီကားဆစ်စ်ကောလ် မဖြစ်လိုက်ဘူး)။ သူည့်ဘို့ အတွက် ဖြစ်သင့်တဲ့ အတိုင်း ဖြစ်ပါတယ်။ ဘိပါတစ်ခုပဲ ရှိရင်ရော ဘယ်လိုဖြစ်မလဲ။ ဘိပါကောက်တယ် သူညာဘို့ ဖြစ်သွားပြီး ရီကားဆစ်စ်ကောလ် ဖြစ်မယ်။ တစ်ကြိမ်ပဲ ဖြစ်မယ်။ if ဘလောက် အလုပ်မလုပ် ဘဲ return ပြန်မယ်။

```
# initial call
pick_all_beeper()
  # 1st recur
  pick_all_beeper()
```

သုံးခုရှိရင် ရီကားဆစ်စ်ကောလ် နှစ်ခါဖြစ်ပြီးမှ return ပြန်မှာပါ။

```
# initial call
pick_all_beeper()
  # 1st recur
  pick_all_beeper()
    # 2nd recur
    pick_all_beeper()
```

မျှော်လင့်ထားသလို အလုပ်လုပ်နေပါတယ်။ အထက်ပါအတိုင်း စစ်ကြည့်သွားရင် ဘိပါ သုံး၊ လေး၊ ငါး၊ ... ခဲ့တွေအတွက်လည်း တောက်လျှောက်မှန်နေမှာပါ။ ဘာကြောင့် ပြောနိုင်ရတာလဲ။ အခုလို စဉ်းစား ကြည့်နိုင်ပါတယ်။ ရှေ့မှာ စစ်ကြည့်တာ $n = 0, n = 1$ အတွက် မှန်တာ သေချာပြီ။ $n = 2$ အတွက် စစ်မယ်ဆိုပါစို့

```
# start with n = 2
if beepers_present():
    pick_beeper() # after this n = 1
    pick_all_beeper() # works correctly for n = 1
```

$n = 2$ အတွက်မှန်ရင် $n = 3$ အတွက်လည်း ဆက်ပြီး မှန်နေမှာပါ

```
# start with n = 3
if beepers_present():
    pick_beeper() # after this n = 2
    pick_all_beeper() # already works for n = 2
```

$n = 3$ အတွက်မှန်ရင် $n = 4$ အတွက်လည်း မှန်ပြီပေါ့

```
# start with n = 4
if beepers_present():
    pick_beeper() # after this n = 3
    pick_all_beeper() # already works for n = 3
```

ဒီတိုင်းဆက်သွားရင် သူညာအပါအဝင် မည့်သည့် အပေါင်းကိန်းပြည့် n အတွက်မဆို (non-negative integer) မှန်တယ်ဆိုတာ မြင်နိုင်ပါတယ်။

ရိုကားဆုံး စဉ်းစားခြင်း ဥပမာ (၂)

ဒုတိယ ဥပမာအနေနဲ့ အခန်း (၃) က အမှိုက်ရှင်းတဲ့ ဥပမာကို ရိုကားဆုံးဖြစ်နည်းနဲ့ ဖြေရှင်းကြည့်ရအောင်။ လမ်းတစ်လမ်းရှင်းဖို့ ဘယ်လိုစဉ်းစားမလဲ။

ဖြေရှင်းမဲ့ ကိုစွဲတော့ အဖြစ် ခွဲခြမ်းကြည့်ရမှာပါ။

လမ်းတစ်ခုရဲ့ အရှည်ကို အရွယ်အစားလို့ ယူဆနိုင်တယ်။ တစ်နည်းအားဖြင့် လမ်းတစ်လျှောက် ကွန်နာအရေအတွက်ဟာ အခါဖြေရှင်းမဲ့ ကိုစွဲရဲ့ အရွယ်အစားပဲ။ ကွန်နာတစ်ခုတော့ အနည်းဆုံး ရှိရမယ်။ ဒါ ကြောင့် အသေးဆုံးက $n = 1$ ဖြစ်တယ်။

clean_street ဖန်ရှင်ဟာ လက်ရှိဖြေရှင်းမဲ့ အရွယ်အစားထက် တစ်ဆင့်ယောက် ကိုစွဲကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူရပါမယ်။

အခါကိုစွဲအတွက် ကွန်နာငါးခဲ့ ရှင်းမယ်ဆိုရင် clean_street ဖန်ရှင်ဟာ ကွန်နာလေးခဲ့နဲ့ လမ်းကိုရှင်းနိုင်တယ်လို့ ယူဆရမယ်။ n ကွန်နာရှိတဲ့ လမ်းအတွက် $n - 1$ ကွန်နာကို ရှင်းနိုင်ပြီးသားလို့ ယူဆရမှာ ဖြစ်ပါတယ်။

လက်ရှိအရွယ်အစားကို သုထက်တစ်ဆင့်ယောက် ကိုစွဲဖြစ်သွားအောင် ဘာလုပ်ရမလဲ စဉ်းစားရတယ်။ ကွန်နာငါးခဲ့ လမ်းကိုရှင်းတဲ့ အခါ ကွန်နာ လေးခဲ့ ရှင်းစရာကျွန်အောင် ဘာလုပ်မလဲ။ ယော်ယျာပြာရင် n ကွန်နာဆိုရင် $n - 1$ ကွန်နာ ရှင်းဖို့လို့တော့အောင် ဘာလုပ်မလဲ။

လမ်းတစ်လမ်းရှင်းတဲ့ အခါ လမ်းအစား သို့မဟုတ် လမ်းအခုံးမှာ အြားသာက်စွန်းကို မျက်နှာမူထားတယ်။ ရှိတစ်ကွန်နာ ရွှေလိုက်ရင် ရှင်းစရာ ကွန်နာတစ်ခဲ့ လျော့သွားမှာပါ။

တစ်ဆင့်ယောက် ကိုစွဲ $n - 1$ ကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူပြီး လက်ရှိကွဲ n ကို ဘယ်လို့ ဖြေရှင်းမလဲ စဉ်းစားရပါမယ်။

ကွန်နာငါးခဲ့ရှင်းမယ်ဆိုရင် လေးခုကို ရှင်းနိုင်ပြီးသား မှတ်ယူထားရမယ်။ ဘိပါရှိရင်ကောက်၊ ရွှေတစ်ကွန်နာ ရွှေထားလိုက်ရင် ရှင်းစရာ လေးခုပဲကျွန်မယ်။ ဒီလေးခုကို လက်ရှိသတ်မှတ်နေတဲ့ ဖန်ရှင်းနဲ့ ရှင်းလိုက်ရုပဲပေါ့။

```
def clean_street():
    ...
    if beepers_present():
        pick_beeper()
    move()
    clean_street() # assuming already works for n - 1
    ...

```

အသေးငယ်ဆုံး ကိုစွဲကို ချင်းချက်အနေနဲ့ စဉ်းစားရမယ်။ ကွန်နာတစ်ခုဟာ အသေးငယ်ဆုံးကိုစွဲ ဖြစ်တယ်။ ဒီကိုစွဲကို ဘယ်လို့ ဖြေရှင်းမလဲ။

ကွန်နာတစ်ခုပဲ ရှိတယ်ဆိုရင် ဘိပါရှိရင် ကောက်လိုက်ရုပဲပေါ့။ $n \neq n - 1$ ကွန်နာ အတွက် အထက်ပါ အတိုင်း စဉ်းစားတဲ့ အခါ $n > 1$ လို့ ယူဆရမှာပါ။ ရှိမှာရှင်းနေရင် $n > 1$ မို့လို့ မရှင်းတော့ဘူးဆိုရင်

$n = 1$ ဖြစ်နေပြီ။

```
def clean_street():
    if front_is_clear():          # ရှေ့မှာ ကွန်နာတွေ ရှိနေသေးရင်
        if beepers_present():
            pick_beeper()
            move()
            clean_street()
    else:                         # နောက်ဆုံးကွန်နာဆိုရင်
        if beepers_present():
            pick_beeper()
```

အသေးဆုံးကနေစပြီး ဖန်ရှင် အလုပ်လုပ်တာ မှန်/မမှန် စိစစ်ပါ။ ကွန်နာတစ်ခုပဲ ရှိတဲ့လမ်းဆိုရင် စစချင်းပဲ ရှေ့မှာ ပိတ်နေမှုပါ။ else အပိုင်း အလုပ်လုပ်မယ်။ ဘိပါရှိရင် ကောက်တယ်။ ကွန်နာနှစ်ခု ရှိတယ်ဆိုရင် စစချင်းရှေ့မှာ နံရုံမရှိဘူး။ if အပိုင်း အလုပ်လုပ်မယ်။ ဘိပါရှိရင် ကောက်တယ် နံရုံပိတ်သွားပြီ။ ရိုကားဆုံးကောလ် ဖြစ်တယ်။ else အပိုင်းလုပ်ပြီးတာနဲ့ return စဖြစ်တယ်။

ဒီအတိုင်းဆက်စစ်သွားရင် တစ်ခုထက်ပိုတဲ့ ကွန်နာတွေအတွက်လည်း မှန်အောင် အလုပ်လုပ်နေမယ်ဆုံးတာ သက်သေပြန်ပါတယ်။ စာရေးသူ အတော့အကြံအရ အသေးဆုံးနဲ့ သူထက်ကြီးတာ နှစ်ခုသုံးခဲ့လောက်ထိ မှန်တယ်ဆိုရင် နောက်ဟာတွေအတွက် မှားစရာ အကြောင်းမရှိတော့ဘူး။

ရိုကားဆုံးနည်းနဲ့ လမ်းတစ်လမ်း ရှင်းလို့ရပါပြီ။ ကားရဲလ်ကမ္ဘာတစ်ခုလုံး ရှင်းဖို့ ရိုကားဆုံးဆုံးပါမယ်။

- လမ်းအရေအတွက်ဟာ အရွယ်အစားလို့ ယူဆနိုင်တယ်။ အသေးဆုံးက လမ်းတစ်လမ်းပါ။
- ငါးလမ်းရှိရင် ကျိုန်တဲ့လေးလမ်းကို ရှင်းနိုင်ပြီးသား မှတ်ယူရမယ်။
- (၁) လမ်းရှင်းပြီး အဆုံးမှာ အပေါ်လမ်း ကူးလိုက်ရင် လေးလမ်းပဲကျွမ်းမယ် (လမ်းအရေအတွက် n ရှိရာကနေ $n - 1$ ဖြစ်အောင် ဘာလုပ်မလဲ စဉ်းစားတာ)။
- တစ်ဆင့်ယော့တဲ့ ကိစ္စ $n - 1$ ကို ဖြဖော်းနိုင်ပြီးသားလို့ မှတ်ယူပြီး လက်ရှိကစ္စ n ကို ဘယ်လို့ ဖြဖော်းမလဲ စဉ်းစားရပါမယ်။

```
def clean_world():
    ...
    clean_street()
    turn_north()
    change_street()
    clean_world()
    ...
```

တစ်လမ်းရှင်းပြီး နောက်တစ်လမ်းကို ကူးလိုက်ရင် ဆက်ရှင်းဖို့ လမ်း အရေအတွက် $n - 1$ ကျွမ်းမယ် (ငါးလမ်းရှိတာကို တစ်လမ်းရှင်းပြီး အပေါ်လမ်းကူးလိုက်ရင် ရှင်းစရာ လမ်း လေးခုပဲ ကျွမ်းမယ်)။ ကျိုန်တဲ့ $n - 1$ လမ်းကို ရှင်းဖို့ လက်ရှိ clean_world ဖန်ရှင်းကိုပဲ ပြန်ခေါ်တယ်။

- အသေးငယ်ဆုံး ကိစ္စကို ချွင်းချက်အနေနဲ့ စဉ်းစားရမယ်။ လမ်းတစ်လမ်းပဲရှိတာက အသေးငယ်ဆုံး။ လမ်းတစ်လမ်းရှင်းပြီး မြောက်ဘက်လုညွှာအပြီး ပိတ်နေပြီးဆိုရင်တော့ နောက်ထပ် ဆက်ပြီး ရှင်းစရာ လမ်းမရှိတော့ဘူး။ အပေါ်ဆင့်က လမ်းကူးတာနဲ့ ကျိုန်တဲ့လမ်းတွေကို ရှင်းတဲ့ကိစ္စကို မြောက်ဘက်လုညွှာပြီးတဲ့အခါ ရှေ့မှာရှင်းနေမှု လုပ်ရမှုပါ။ ဒီတော့ အခုလို

```

def clean_world():
    clean_street()
    turn_north()
    if front_is_clear():
        change_street()
        clean_world()

```

ဖြစ်ရမယ်။

တစ်လမ်း၊ နှစ်လမ်း၊ သုံးလမ်း အသေးဆုံး ကိစ္စတွေ မှန်/မမှန် စိစစ်ကြည့်ပါ။ ပရိဂရမ် အစအဆုံး ဖော်ပြ ပေးထားပါတယ်။ လေ့လာကြည့်ပါ။

```

# File: clean_world_recur1.py
from stanfordkarel import *

def main():
    clean_world()

def clean_world():
    clean_street()
    turn_north()
    if front_is_clear():
        change_street()
        clean_world()

def clean_street():
    if front_is_clear():          # ရှေ့မှာ ကွန်နာတွေ ရှိနေသေးရင်
        if beepers_present():
            pick_beeper()
            move()
            clean_street()
    else:                      # နောက်ဆုံးကွန်နာဆိုရင်
        if beepers_present():
            pick_beeper()

def change_street():
    move()
    if right_is_blocked():
        turn_left()
    else:
        turn_right()

def turn_right():
    turn_left()
    turn_left()
    turn_left()

```

```
def turn_north():
    while not_facing_north():
        turn_left()

if __name__ == "__main__":
    run_karel_program("clean_world")

# File: checkerboard_recur.py
from stanfordkarel import *

def main():
    mk_checkerboard()

def mk_checkerboard():
    mk_checker_row()
    turn_north()
    if front_is_clear():
        if beepers_present():
            switch_row()
            mk_checker_row2()
        else:
            switch_row()
            mk_checker_row()
        turn_north()
        if front_is_clear():
            switch_row()
            mk_checkerboard()

def mk_checker_row():
    put_beeper()
    if front_is_clear():
        move()
    if front_is_clear():
        move()
        mk_checker_row()

def mk_checker_row2():
    if front_is_clear():
        move()
        put_beeper()
    if front_is_clear():
        move()
        mk_checker_row2()

def switch_row():
```

```
move()  
if right_is_blocked():  
    turn_left()  
else:  
    turn_right()  
  
def turn_right():  
    turn_left()  
    turn_left()  
    turn_left()  
  
def turn_north():  
    while not_facing_north():  
        turn_left()  
  
if __name__ == "__main__":  
    run_karel_program("4x5")
```

အခန်း ၅

ဒေတာများနှင့် ဖန်ရှင်များ

ရှုပိုင်း ကားရဲလ်အခန်း လေးခုမှာ လေ့လာခဲ့ကြတဲ့ အကြောင်းအရာတွေဟာ ပရိုကရမ်မင်း ဘာသာရပ်ရဲ့ အခြေခံအကျခုံး ပင်မထောက်တိုင် သဘောတရားတွေလို့ ဆိုရမှာပါ။ ဒီသဘောတရားတွေ မကျေည်် ဘဲ ပရိုကရမ်ရေးလို့ မရပါဘူး။ ကွန်ဒီဇိုင်နယ်တွေဖြစ်တဲ့ if, if...else ပြန်ကျော်ခြင်းအတွက် for နဲ့ while loop၊ ဖန်ရှင်တွေ၊ top-down, bottom-up ပရိုကရမ်းမင်း၊ ရိုကားရှင်းနဲ့ ရိုကားဆစ်ပ် စဉ်းစား ခြင်း စတာတွေနဲ့ ပရိုကရမ်းမင်း ပုံစံတွေ ဖြေရှင်းနိုင်ရင် ပရိုကရမ်မာလောကား ပထမ တစ်ထိ တက်လှမ်း အောင်ပြု ပြောနိုင်ပါတယ်။ ဒီသဘောတရားတွေကို ဘိုင်နာတွေ အရိုးရှင်းဆုံးနည်းနဲ့ နားလည်အောင် လေ့ကျင့်လို့ရအောင် ကားရဲလ်က ထောက်ကူပေးတာပါ။ စက်ရုပ်လေး ကားရဲလ်ကို နှုတ်ဆက်ခဲ့ပြီး အခု ဆက်လက်လေ့လာကြမှာကတော့ ဒေတာ၊ အိပ်စံပရက်ရှင်းနဲ့ ဗော်ရောဘဲလ်တွေ အကြောင်းပါ။

ကွန်ပျိုးတာတွေဟာ အချက်အလက် (ဒေတာ) အမျိုးမျိုးကို ကိုင်တွယ်ဆောင်ရွက်ပေးနိုင်တယ်။ ကိုန်း ကဏ္န်းတွေအပြင် စာသား၊ ရုပ်သံ စတာတွေကိုပါ လက်ခံ အလုပ်လုပ်ပေးနိုင်တယ်။ ဒီလိုလုပ်ဆောင်နိုင်စွမ်း ဟာ ကွန်ပျိုးတာတွေကို နယ်ပယ်ပေါင်းစုံမှာ တွင်တွင်ကျယ်ကျယ် အသုံးချလာရခြင်းရဲ့ အဓိက အကြောင်း အရင်း ဆိုရင်လည်း မမှားဘူး။

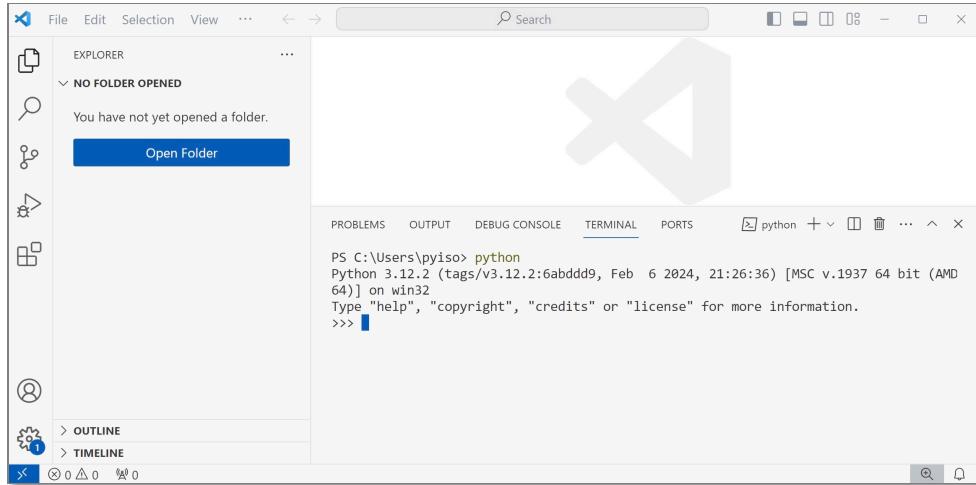
ဒေတာ အမျိုးအစား အများအပြားရှိပေမဲ့ အခြေခံအကျခုံးက ကိုန်းဂဏန်းတွေပါ။ ကွန်ပျိုးတာတွေ ကို စတင်တိထွင်ဖို့ ကြိုးစားလာကြတဲ့ အဓိက အကြောင်းအရင်းကလည်း ကဏ္န်းသချာ အတွက်အချက် တွေကို လုပ်ဆောင်ရာမှာ လူတွေကို အကောအညီ ပေးဖို့အတွက်ပဲလို့ ဆိုရိုင်ပါတယ်။ ဒါအပြင် စာသား၊ ရုပ်သံ စတဲ့ အခြားဒေတာ အမျိုးအစားတွေကို ကွန်ပျိုးတာထဲမှာ ဖော်ပြုသိမ်းဆည်းထားဖို့အတွက် ကိုန်းဂဏန်းတွေကိုပဲ အသုံးပြုထားတယ်ဆိုတာ နောက်ပိုင်းမှာ နားလည်သိမြင် လာပါလိမ့်မယ်။ ဒါကြောင့်လည်း ယနေ့ခေတ် ကွန်ပျိုးတာတွေကို အစိုက်တယ် ကွန်ပျိုးတာတွေလို့ ခေါ်တာဖြစ်တယ်။ ကိုန်းဂဏန်းကို အခြေခံပြီး အလုပ်လုပ်တဲ့ ကွန်ပျိုးတာတွေပေါ့။

၅.၁ ကိန်းဂဏန်းများ

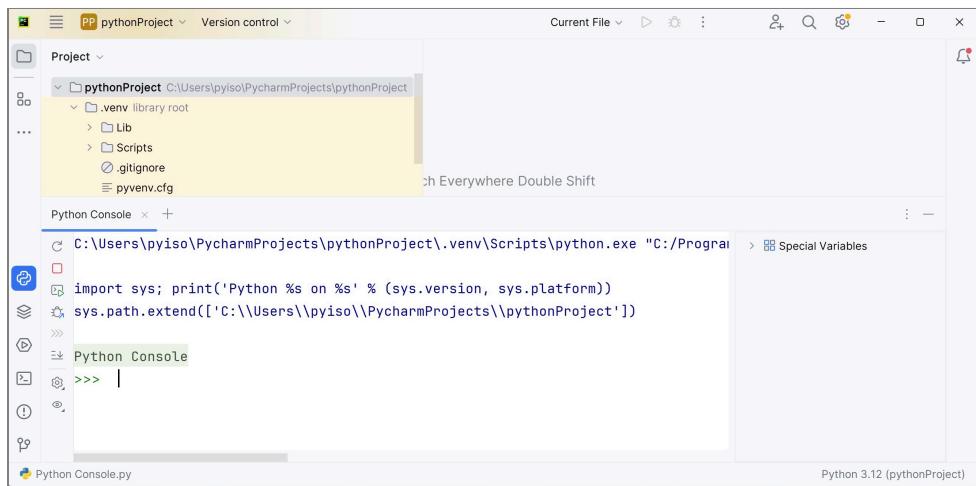
Python အင်တာပရက်တာနဲ့ Python ကုဒ်တွေကို run လိုရတဲ့နည်း နှစ်ခုရှိပါတယ်။ တစ်နည်းက ကုဒ်တွေကို .py ဖိုင်နဲ့သိမ်းပြီး python ကွန်မန်းနဲ့ run တာပါ။ ဒါက ကားရဲလ် ပရိုကရမ်တွေမှာ သုံးခဲ့တဲ့နည်း၊ နောက်တစ်နည်းကတော့ ကုဒ်တစ်ကြောင်းချင်း ရိုက်ထည့်ပြီး run တဲ့နည်းပါ။ ဒီနည်းလမ်းက interactive mode နဲ့ အင်တာပရက်တာကို အသုံးပြုတာပါ။ တစ်နည်းအားဖြင့် ကိုယ်က ကုဒ်တစ်ကြောင်းချင်း ရိုက်ထည့်ပေးပြီး အင်တာပရက်တာကလည်း အဲဒီတစ်ကြောင်းချင်းကို run ပြီး ရလဒ်ကို ပြပေးပါတယ်။ အခုအခန်းအတွက် interactive mode နဲ့ သုံးပါမယ်။ ကုဒ်တစ်ကြောင်းချင်း စမ်းသပ်

ကြည့်ရတာ လွယ်တဲ့အတွက်ကြောင့်ပါ။

င်းဒီ Command Prompt သိမဟုတ် မက်ခံအိအက်စ် Terminal မှ python ကွန်မန်း run ပြီး interactive mode ကို ဝင်နိုင်ပါတယ်။ VS Code မှာပဲ သုံးချင်လည်း ရတယ်။ View မိန္ဒာမှု Terminal ကိုဖွံ့ဖြိုး (Ctrl + ` ရှေ့ကတ်ကီးနဲ့ ဖွံ့ဖြိုးလည်းရတယ်) ပြီး python ကွန်းမန်း run ရဲ့။ PyCharm မှာဆိုရင် Python Console အိုင်ကွန်နိုပ်ပြီး ဝင်ရပါမယ်။ ပုံ (၅.၁)၊ (၅.၂) တွင်ကြည့်ပါ။



ပုံ ၅.၁ VS Code Python Console



ပုံ ၅.၂ PyCharm Python Console

2 + 5 ထည့်ပြီး Enter ကီးနိုပ်ပါ။ အင်တာပရက်တာက ရလဒ် 7 နဲ့ တုံးပြန် လုပ်ဆောင်ပေးပါလိမ့်မယ်။ ဒီလို အင်တာပရက်တာက လုပ်ဆောင်ပေးတာကို evaluate လုပ်တယ်လို့ ပြောတယ်။

```
>>> 2 + 5
7
```

အောက်ပါအတိုင်း တစ်ခုပြီးတစ်ခု ဆက်လက်စမ်းသပ်ကြည့်ပါ။

```
>>> 2 + 2
4
>>> 3 * 3
9
>>> 4 - 2
2
>>> 5/2
2.5
```

$3 \times 3 \text{ ကို } 3 * 3$ လို ရိုက်ထည့်ပေးရပြီး $5 \div 2$ အတွက် $5 / 2$ လို ရေးရတာကို သတိပြုမှုမှာပါ။ Programming language အများစုံမှာ * (asterisk) အမြောက်သကောက်အဖြစ် အသုံးပြုပြီး / (forward slash) ကို အစားသကောက်အနေနဲ့ အသုံးပြုလေ့ရှိတယ်။

Values and Types

တန်ဖိုးတိုင်းဟာ တိုက်ပ် (type) တန်မျိုးမျိုးမှာ ပါဝင်ပါတယ်။ -3, 0, 2 စတဲ့ တန်ဖိုးတွေဟာ int (integer ရဲ အတိုက်ပ်) တိုက်ပ်ဖြစ်ပြီး -3.0, 0.1, 3.3333 စတာတွေက float တိုက်ပ် ဖြစ်ပါတယ်။ ဒဿ်မကိန်းတွေကို ကွန်ပျူးတာနဲ့ ဖော်ပြန့် floating point လိုပေါ်တဲ့ နည်းစနစ်ကို အသုံးပြုတယ်။ ဒဿ်မကိန်း အတွက်အချက်တွေကိုလည်း ဒီနည်းစနစ်ကို အခြေခံပြီး ကွန်ပျူးတာက လုပ်ဆောင်တာပါ။ ဒေါက်တွေနဲ့ floating point ဟာ ဒဿ်မကိန်းတွေကို ဖော်ပြန့်နဲ့ ဒဿ်မကိန်း အတွက်အချက်တွေ လုပ်ဆောင်ဖို့ တိုထွင်ထားတဲ့ နည်းစနစ်တစ်ခုလုံး ဆုနိုင်ပါတယ်။ ဒီစနစ်ကို အခြေခံထားတဲ့ ဒဿ်မကိန်းတွေကို programming language တွေမှာ float တိုက်ပ်လို ခေါ်တာပါ။

float တိုက်ပ်ဟာ လိုသလောက် တိုကျလိုမရတဲ့ သဘောရှိတယ်။ အောက်ပါအတိုင်း စမ်းကြည့်ရင် 0.3 နဲ့ 1.0 ရသင့်တာ ဖြစ်ပေမဲ့ အတိအကျ အဖြော်ထွေက်ပါဘူး။

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
0.9999999999999999
```

ကွဲပြားချက်က မဆိုစလောက် သေးငယ်တယ် ဆုံးပေမဲ့ ဒီအချက်ကို ပရုပြုမိဖို့ လိုပါတယ်။ Floating point စနစ်ဟာ လုံးဝကြီးတိုကျဖို့ မလိုအပ်တဲ့ (တန်ည်းအားဖြင့် မဆိုစလောက် သေးငယ်တဲ့ ကွာဟာချက်ကို လက်ခံနိုင်တဲ့) ကိန်းကြောင်းအတွက်အချက် ကိစ္စတွေအတွက် ရည်ရွယ်တာပါ။ သိပ္ပါနဲ့ နည်းပညာဆိုင်ရာ တိုင်းတာ တွေက်ချက်မှုတွေအတွက် အသုံးပြုလေ့ရှိတယ်။ ဒဿ်မကိန်းတွေ လုံးဝအတိအကျ ဖြစ်ဖို့ လိုအပ်တဲ့ ကိစ္စမျိုးတွေ (ဥပမာအားဖြင့် ငွေကြေးကိစ္စ အတွက်အချက်) မှာ အသုံးမပြုသင့်ပါဘူး။ ဆယ်ပြားကို 0.1 နဲ့ဖော်ပြရင် ဆယ်ပြားစေ ဆယ်စွေားတော့ တစ်ကျပ် ဖြစ်ကို ဖြစ်သင့်ပြီး 0.9999999999999999 မဖြစ်သင့်ဘူး။ ဒီလိုကိစ္စမျိုးတွေအတွက် Python မှာ Decimal ကို အသုံးပြုနိုင်ပါတယ်။ လောလောဆယ် ကိန်းကြောင်းတွေနဲ့ ပါတ်သက်ပြီး စကတည်းက သိထားသင့်တာတချို့ကို ကြိုးကြော် အကြောင်း မကြေခင် လေ့လာမှုပါ။

```
>>> from decimal import *
>>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1')
Decimal('0.3')
```

int တိုက်ပ် ဖော်ပြနိုင်တဲ့ အကြီးဆုံး အပေါင်းကိန်းပြည့် သို့မဟုတ် အငယ်ဆုံး အနှုတ်ကိန်းပြည့် တန်ဖိုးကို ကန့်သတ်ထားတာ မရှိဘူး။ သိဒ္ဓရီအရ ကြိုးလို့/ငယ်လို့ ရပါတယ်။ လက်တွေ

မှာတော့ အသုံးပြုတဲ့ ကွန်ပျူးတာစနစ်ပေါ် မူတည်ပြီး ကန့်သတ်ချက်ရှုမှုပါ။ ဒီစာရေးနေတဲ့ ကွန်ပျူးတာ မှာ အောက်ပါကိန်းပြည့်တွေကို အသာလေး တွက်ချက်ပေးနိုင်ပါတယ်။ ဒို့ထက်အများကြီး ကြိုးတဲ့ဟာတွေ ကိုလည်း ကိုင်တယ်တွက်ချက်ရှုနိုင်အုံမှုပါ။ တော်ရုံကိစ္စဓတွေအတွက် ကွန်ပျူးတာစနစ်တစ်ခုရဲ့ လက်တွေ ကန့်သတ်ချက်ကို ကျော်လန်သားဖို့ မလုပ်ပါဘူး။

float ကတော့ int နဲ့ မတူဘဲ အကြီးဆုံး/အငယ်ဆုံး ကန်သတ်ချက်ရှိပါတယ်။ ဒီကန်သတ်ချက် ကလည်း ကွန်ပူးတာစနစ်ပေါ် မူတည်ပါတယ်။ တစ်ကိုယ်ရော့သုံး ကွန်ပူးတာ အများစုအတွက် 1×10^{400} (၁ နောက် သုည အလုံး ၄၀၀) သို့မဟုတ် -1×10^{400} (အနှစ် ၁ နောက် သုည အလုံး ၄၀၀) ဟာ ကန်သတ်ချက်ကျော်လွန်ပါတယ်။

```
>>> 1e400
inf
>>> -1e400
-inf
>>> 1e-400
0.0
>>> -1e-400
-0.0
```

သသုတကိန်းကို e အမှတ်အသားနဲ့ရေးထားတာပါ။ အကွဲရာ e နောက် လိုက်တဲ့ ကိန်းပြည့်ကဏ္ဍးကို 10 ထပ်ကိန်းလို မှတ်ယူရပါတယ်။ e3 က 1×10^3 , e-3 က 1×10^{-3} ပါ။ အကွဲရာ E အားဖြူနဲ့လည်း ရေး နိုင်တယ်။ ထပ်ကိန်း ကြီးလုန်းတဲ့အတွက် ကန်သတ်ချက် ကျော်လွန်ရင် inf (အနှုတ်ကိန်းဆုံးရင် -inf) ရပါမယ်။ Infinity ကို ဆိုလိုတာပါ။ (၁) မပြည့်တဲ့ ပမာဏ သေးငယ်လွန်းတဲ့ ကဏ္ဍးတွေကိုလည်း အနီး စပ်ဆုံး သုညအနေနဲ့ ယူပါတယ်။ အနီးစပ်ဆုံးယူတဲ့အခါ အပေါင်း/အနှုတ်ကိုတော့ ခွဲခြားပေးပါတယ်။ e အမှတ်အသားနဲ့ရေးထားတဲ့ နောက်ထပ် ပုံမှန်လေး နစ်ခုပါ

```
7.34767309e22    # mass of the moon in kg  
9.1093837015e-31 # mass of an electron in kg
```

ကဏ္နားအလုံးအရေအတွက် များရင် 100,500 (တစ်သိန်းငါးရာ)၊ 1,500,000 (တစ်သိန်းငါးသိန်း) စသဖြင့် သုံးလုံးတစ်ဖြတ် ကော်မာခံရေးလေ့ရှိတယ်။ Python မှာတော့ ကော်မာအား _ (under-score) နဲ့ သုံးလုံးတစ်ဖြတ် ဘားရေးဝိုင်ပါတယ်။

```
>>> 1_500_000 + 100_500
1600500
>>> 200_000.33 + 3_800_000.22
4000000.5500000003
```

တိက်ပိမတ္တသည့် ကိန်းဂဏေန်း အိပ်စ်ပရက်ရင်များ

အိပ်စ်ပရက်ရှင်တွေမှ ပါဝင်တဲ့ တိုက်ပဲ တစ်မျိုးတည်း ဖြစ်ရမယ် မရှိပါ။ တိုက်ပဲမတဲ့ တဲ့ ကိန်းကဏ္ဍးတွေ အိပ်စ်ပရက်ရှင်တစ်ခါးမှ ရော်ပြီး ပါဝင်နိုင်ပါတယ်။

```
>>> 5 - 2.0
3.0
>>> 5 - 2
3
>>> 3 * 2.0
6.0
>>> 3 * 2
6
```

int နဲ့ float ရောနေရင် အပိုစ်ပရက်ရှင် ရလဒ်သည် float တိုက်ပဲ ဖြစ်မှာပါ။ အစား (division) မှာတော့ အင်တိဂျာအချင်းချင်း စားတဲ့အခါမှာလည်း ရလဒ်က float ဖြစ်ပါမယ်။

```
>>> 9/3
3.0
>>> 9.12/3.3
2.7636363636363637
>>> 88/3
29.333333333333332
>>> 1/3
0.3333333333333333
>>>
```

အင်တိဂျာ ဒီပီးရှင်း၊ မော်ဒူးလို နှင့် ထပ်ကိန်းတင်ခြင်း

အကယ်၍ သသယကိန်းမထွက်ဘဲ ကိန်းပြည့် လိုချင်ရင် // ကို သုံးရပါမယ်။ ဒီအခါ အကြွင်းကိုဖယ်ပြီး စားလတ်ကိုပဲ ကိန်းပြည့်အနေနဲ့ ရမှာပါ။

```
>>> 9//3
3
>>> 12//5
2
>>> 3//5
0
```

သချာမှာ ဒီလိုမျိုး အစားကို အင်တိဂျာ ဒီပီးရှင်း (integer division) လိုခေါ်ပါတယ်။ အကြွင်းရှာမယ ဆုံးရင် % အော်ပရိတ်တာ ရှိပါတယ်။ % ကို မော်ဒူးလို (modulo) အော်ပရိတ်တာလို ခေါ်တယ်။ remainder အော်ပရိတ်တာလိုလည်း ခေါ်တယ်။

```
>>> 7 % 5
2
>>> 100 % 10
0
```

အင်တိဂျာ ဒီပီးရှင်းနဲ့ မော်ဒူးလိုကို အနှုတ်ကိန်းတွေနဲ့ သုံးမယ်ဆုံးရင် သတိပြုပါ။ စားလတ် အနှုတ် ကိန်း ဖြစ်ရင် // က ပိုင်ယတဲ့ အနှုတ်ကိန်းကို အနီးစပ်ဆုံး ယူမှာပါ။ တစ်နည်းအားဖြင့် round down လုပ်တာ ဖြစ်တယ်။

```

>>> -12 // -10
1
>>> -12 // 10
-2
>>> 12 // -10
-2
>>> -31 // 10
-4
>>> -35 // 10
-4
>>> -38 // 10
-4

```

-2 နဲ့ -4 ထွက်တာ သတိပြုပါ။ အဖြေအတိအကျက -1.2 ကို အနီးစပ်ဆုံး သူ့ထက်ပိုင်ယွဲ -2 ကို အနီးစပ်ဆုံး ယူတယ်။ $-3.1, -3.5, -3.8$ တို့ကိုလည်း အနီးစပ်ဆုံး -4 ယူတာပါ။

မော်ဒူးလို အော်ပရိတ်တာ % သုံးတဲ့အခါ ရလဒ်ဟာ စားကိန်းရဲ့ sign အပေါ် မူတည်တယ်။ (အပေါင်း အနှုတ်ကို ဆိုလိုတာပါ)။

```

>>> -17 % 10
3
>>> 17 % -10
-3

```

မော်ဒူးလိုနဲ့ အင်တိဂျာ ဒီပီးရှင်း အော်ပရိတ်တာ နှစ်ခုက အောက်ပါညီမျှခြင်းအရ ဆက်စပ်နေတာပါ။ စားကိန်း $B \neq 0$ ဖြစ်ပါတယ်။

$$B * (A//B) + A\%B = A$$

ဒါ့ကြောင့် $B = 10, A = -17$ ဖြစ်လျှင်

$$\begin{aligned}
 B * (A//B) + A\%B &= A \\
 10 * (-17//10) + -17\%10 &= -17 \\
 -20 + -17\%10 &= -17 \\
 -17\%10 &= -17 + 20 \\
 -17\%10 &= 3
 \end{aligned}$$

အကယ်၍ $B = -10, A = 17$ ဖြစ်လျှင်

$$\begin{aligned}
 B * (A//B) + A\%B &= A \\
 -10 * (17// - 10) + 17\% - 10 &= 17 \\
 20 + 17\% - 10 &= 17 \\
 17\% - 10 &= 17 - 20 \\
 17\% - 10 &= -3
 \end{aligned}$$

အထက်ပါ ညီမျှခြင်းဟာ ကိန်းပြည့်တွေအတွက်ပဲ မှန်တာပါ။ // နဲ့ % ကို အသေမကိန်းတွေနဲ့လည်း သုံးလိုရပေမဲ့ ရလဒ်တွေက အထက်ပါ ညီမျှခြင်းကို ပြောည့်စေမှာ မဟုတ်ပါဘူး။ float တိုက်ပ်ဟာ

```

>>> 9.9 // 3.3
3.0
>>> 9.9 % 3.3
8.881784197001252e-16
>>> 9.9 / 3.3
3.0000000000000004
>>> 3.5 / 0.1
35.0
>>> 3.5 // 0.1
34.0
>>> 3.5 % 0.1
0.09999999999999981

```

ထပ်ကိန်းတင် (exponentiation) ဖို့ အတွက် အော်ပရိတ်တာက $\star\star$ ပါ။ 2^4 နဲ့ $(3.3)^3$ ကို အခဲ့လို တွက်ပါတယ်။

```

>>> 2 ** 4
16
>>> 3.3 ** 3
35.937

```

သချုပ်ရှင်များ

ကိန်းဂဏ်းတွေအကြောင်း လေ့လာလက်စနဲ့ math လိုက်ဘရဲ့ သချုပ်ရှင်တရီးကိုလည်း တစ်ခါတည်း ဆက်ကြည့်လိုက်ရအောင်။ အဓိကက သချုပ်ရှင်ဆိုတာထက် ဖန်ရှင် အခြော့အသုံးပြုပုံကို စပြီးလေ့လာ မှုပါ။ math လိုက်ဘရဲ့က Python မှာ တစ်ခါတည်း ထည့်ထားပေးပြီးသား (built-in) လိုက်ဘရဲ့ပါ။ အင်စတောလ်လုပ်စရာ မလိုဘဲ အင်ပို့လုပ်ပြီး သုံးလို့ရတယ်။

```
>>> from math import *
```

အင်ပို့လုပ်ပြီးရင် math လိုက်ဘရဲ့ဖန်ရှင်တွေကို သုံးလို့ရပါပြီ။ ကိန်းတစ်ခုခဲ့ နှစ်ထပ်ကိန်းရင်းကို sqrt, သုံးထပ်ကိန်းရင်းကို cbrt ဖန်ရှင်နဲ့ ရှာနိုင်ပါတယ်။

```

>>> cbrt(27)
3.0
>>> sqrt(81)
9.0

```

သချုပ်ရှင်အားလုံးဟာ input တန်ဖိုးတစ်ခု သို့မဟုတ် တစ်ခုထက်ပို၍ လက်ခံပြီး output တန်ဖိုး တစ်ခု ပြန်ထုတ်ပေးပါတယ်။ 27 နဲ့ 81 ဟာ input ဖြစ်ပြီး 3.0 နဲ့ 9.0 က output ဖြစ်တယ်။

```

>>> gcd(2406, 654)
6
>>> gcd(2406, 654, 354)
6
>>> gcd(2406)
2406

```

အကြိုးဆုံးဘုံဆွဲကိန်းကို gcd ဖန်ရှင်နဲ့ ရှာတာပါ။ အင်တိဂျာ input တစ်ခုနဲ့အထက် လက်ခံတဲ့ ဖန်ရှင်ဖြစ်တယ်။ input ကတန်းအားလုံးကို တားလို့ပြတ်တဲ့ အကြိုးဆုံးကိန်းကို ရှာပေးတယ်။ ကိန်းပြည့်မဟုတ်တာ ထည့်ရင် အယ်ရာဖြစ်ပါတယ်။

```
>>> gcd(2.4, 4.8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
```

လေ့ကရစ်သမ်း ထရိရိကိုမေတာရဲ့ ဖန်ရှင်တွေလည်းပါတယ်။ $\log_{10}(x)$, $\sin(x)$, $\cos(x)$ တို့ကို ဥပမာပြထားတာ ကြည့်ပါ။

```
>>> log10(1000)
3.0
>>> sin(pi/2) # pi/2 radians = 90 degrees
1.0
>>> sin(pi/4) ** 2 + cos(pi/4) ** 2
1.0
```

၅.၂ ‘တိုက်ပ်’ ဆိုတာ ဘာလဲ

Programming language အားလုံးမှာ တိုက်ပ် သို့မဟုတ် အော်တိုက်ပ် သဘောတရား ပါရှိပါတယ်။ int နဲ့ float တိုက်ပ်မှာ ပါဝင်တဲ့ ကိန်းပြည့်တွေနဲ့ ဒသမကိန်းတွေကို မိတ်ဆက်ပြီးတဲ့အခါ ‘တိုက်ပ်’ ဆိုတာဘာလဲ တို့တိုကျကျ ရှုံးပြလို့ရပါပြီ။ တိုက်ပ်တစ်ခုဟာ

- တန်ဖိုးတွေပါဝင်တဲ့ အစု (set) တစ်ခု နဲ့
- ငှုံးတန်ဖိုးများအပေါ်တွင် အသုံးချိန်တဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစုတစ်ခုဖြစ်ပါတယ်။ ဥပမာ int တိုက်ပ်ကို ကြည့်ရင် ကိန်းပြည့်တွေ ပါဝင်တဲ့ အစုနဲ့ ကိန်းပြည့်တွေအပေါ်မှာ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစု

```
{..., -3, -2, -1, 0, 1, 2, 3, ...}
{+, -, *, /, //, %, **, ...}
```

ဖြစ်ပါတယ်။ float တိုက်ပ်ကတော့ ကိန်းစစ် (real numbers) တွေပါဝင်တဲ့ အစုနဲ့ ငှုံးတို့အပေါ်မှာ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစုတို့ ဖြစ်ပါတယ်။ int နဲ့ float တိုက်ပ်မှာ အော်ပရေးရှင်းတွေ တူတူဖြစ်နေတာ တွေ့ရမှာပါ။ ဒီလိုအမြဲဖြစ်မယ်လို့ မယူဆရပါဘူး။ တိုက်ပ် မတူတဲ့အခါ အသုံးချလို့ ရနိုင်တဲ့ အော်ပရေးရှင်းတွေ ကွာမြေးနိုင်ပါတယ်။ ဥပမာ str တိုက်ပ် အော်ပရေးရှင်းတွေက int တို့ float တို့နဲ့ မတူပါဘူး။ str က string ရဲ့ အတိုကောက်ဖြစ်ပြီး စာသားတွေအတွက် အသုံးပြုပါတယ်။ မကြာခေါ် လေ့လာကြမှာပါ။

အပေါက် အော်ပရေးရှင်း အစုမှာ အစက်သုံးစက် ... ကို သတိပြုပါ။ ဆုံးလိုတာက အခြား အော်ပရေးရှင်းတွေ ဒီအစုမှာ ပါဝင်ပါသေးတယ်။ int နဲ့ float တွေအတွက် ဖန်ရှင်တွေကိုလည်း ဒီအစုမှာ ပါဝင်တယ်လို့ ယူဆရမှာပါ။

```
>>> from math import *
>>> sqrt(2.0)
```

```
1.4142135623730951
```

```
>>> abs(-5)
```

```
5
```

ဥပမာအနေနဲ့ sqrt နဲ့ abs ဖန်ရှင် အသုံးချုပ်ပါ။ နှစ်ထပ်ကိန်းရှင်းနဲ့ ပကတိတန်ဖိုး ရှာပေးပါတယ်။ လိုအပ်ရင် ကိုယ်ပိုင်ဖန်ရှင်တွေ သတ်မှတ်ပြီး တိုက်ပ်တစ်ခုခဲ့ အော်ပရေးရှင်းတွေကို ဖြည့်စက်တိုးခဲ့နိုင် ပါတယ်။

အော်ပရေးရှင်းနဲ့ အော်ပရိတ်တာ ရောထွေးစရာ ရှိပါတယ်။ $+, -, *, /, //, \%, **$ စဲတဲ့ သက်ကောင်တွေကို အော်ပရိတ်တာလို့ ခေါ်ပါတယ်။ အော်ပရေးရှင်း လုပ်ဆောင်ဖို့အတွက် အသုံးပြုတဲ့ သက်ကောင်တွေကို အော်ပရိတ်တာလို့ ခေါ်တာပါ။ ဥပမာ “* သက်ကောင်ဟာ အမြှောက်အော်ပရေးရှင်း လုပ်ဆောင်ဖို့ သတ်မှတ်ထားတဲ့ အော်ပရိတ်တာ” လို့ ပြောတယ်။ အမြှောက် ‘အော်ပရေးရှင်း’ ကျတော့ မြှောက်တဲ့အလုပ် ဆောက်ရွက်တာကို ဆိုလိုတာ။

၅.၃ ဖော်ရောဲလ်များ

ဖော်ရောဲလ်ဆိုတာ တန်ဖိုးတစ်ခုကို ကိုယ်စားပြုတဲ့ နံမည်ပါပဲ။ နံမည်နဲ့ ငြင်းကိုယ်စားပြုတဲ့ တန်ဖိုး တဲ့ဖက်ပေးဖို့ အဆိုင်းမနဲ့ (assignment) စတိတ်မနဲ့ကို သုံးရပါတယ်။

```
>>> age = 12
```

```
>>> weight = 35.5
```

age နဲ့ weight ဟာ ဖော်ရောဲလ်တွေ ဖြစ်ပါတယ်။ ညီမြှော်ခြင်းသက်ကောင် (=) ကတော့ အဆိုင်းမနဲ့ အော်ပရိတ်တာပါ။ ဖော်ရောဲလ်နဲ့ တန်ဖိုး တွဲဖက်ပေးတဲ့ အော်ပရိတ်တာ ဖြစ်တယ်။ ဖော်ရောဲလ်နံမည်ကို variable identifier လိုလည်း ခေါ်ပါတယ်။ Identifier က နည်းပညာ အခေါအဝေါပဲ။ Variable name က သာမန်လူ နားလည်တဲ့ နည်းနဲ့ ပြောတာပါ။ ဖော်ရောဲလ် တစ်ခုချင်း ထည့်ကြည့်ရင် ငြင်းကိုယ်စားပြုတဲ့ တန်ဖိုးကို ပြန်ထုတ်ပေးတာ တွေ့ရှုမှုပါ။

```
>>> age
```

```
12
```

```
>>> weight
```

```
35.5
```

အိပ်စ်ပရက်ရှင်တောက ဖော်ရောဲလ်တွေနဲ့ ဖြစ်နိုင်ပါတယ်။ အိပ်စ်ပရက်ရှင် တွက်ချက်ရှင် ဖော်ရောဲလ် တန်ဖိုးနဲ့ အစားထိုး တွက်ချက်တယ်လို့ ယူဆရမှုပါ။ ဥပမာ

```
>>> age + 1
```

```
13
```

```
>>> weight / 2
```

```
17.75
```

ဖော်ရောဲလ်တစ်ခုကို အိပ်စ်ပရက်ရှင်ရလဒ်နဲ့ အဆိုင်းမနဲ့ လုပ်လို့ရပါတယ်။ rect_area ကို အောက်တွင် ကြည့်ပါ။ အလျား အနဲ့ မြှောက်လဒ်ကို အဆိုင်းမနဲ့ လုပ်ထားတာ တွေ့ရှုပါမယ်။

```
>>> rect_width = 22.5
```

```
>>> rect_length = 10
```

```
>>> rect_area = rect_width * rect_length
>>> rect_area
225.0
```

အဆိုင်းမန် ခတိတ်မန်

ပေရီရေဘဲလ်တစ်ခုဟာ အချိန်တစ်ချိန်မှာ တန်ဖိုးတစ်ခုကိုပဲ ကိုယ်စားပြနိုင်တယ်။ ဒါပေမဲ့ အချိန်ကာလပေါ် မူတည်ပြီး တန်ဖိုးပြောင်းနိုင်တယ်။ (တစ်ချိန်တည်းမှာ တန်ဖိုးနှစ်ခု မဖြစ်နိုင်ဘူး)။ ဥပမာ x တန်ဖိုးဟာ ပထမ 10 ပါ။ ဒုတိယ အဆိုင်းမန်လုပ်ပြီးတဲ့ အချိန်မှာ အဲဒီ x ကပဲ 1000 ဖြစ်နေမှုပါ။

```
>>> x = 10
>>> x
10
>>> x = 1000
>>> x
1000
```

၆.၄ စာသားများ

စာသား (text) ဟာ အသုံးအများဆုံး ဆက်သွယ်ဆောင်ရွက်ရေး ကြားခံနယ်တစ်ခုပါ။ ဝက်ဘ်ဆိုက် စာမျက်နှာ၊ အီးဘေးလုံး၊ အီးဘွတ်ခုနဲ့ အီးလက်ထရွန်းနှစ် စာရွက်စာတမ်း (e-documents) စတာတွေမှာ ရုပ်သံတွေ အသုံးပြုလာကြပေမဲ့ စာသား အမိကဖြစ်နေဆဲပါပဲ။ ဆိုရှယ်မီဒီယာ၊ ဂိမ်းနဲ့ အခြားအပ်ပဲတွေ ဟာလည်း စာသားနဲ့ မကင်းနိုင်ကြပါဘူး။ ဒါကြောင့် ပရိုက်များမင်းအတွက် စာသားဟာ ဘယ်လောက်ထိ အရေးပါကြောင်း အများကြီးပြောစရာ လိုမယ်မထင်ပါဘူး။

ပရိုက်များမင်းမှာ စာသားကို *string* လိုက်ပြီး ကာရိုက်တာ (character) တွေနဲ့ စီတန်းစွဲစည်းထားတယ်။ ကာရိုက်တာဆိုတာ အခြေခံ သတင်းအချက်အလက် ယူနစ်တစ်ခုပါပဲ။ အကွဲရာ၊ ဂဏန်း (digit)၊ သက်ဗျာ၊ သို့မဟုတ် ကွန်ထရှုံးလုံကုဒ် တစ်ခုခဲ့ ဖြစ်နိုင်ပါတယ်။ ဥပမာ A, B, C, \$, @, #, 1, 3, _ စသည်ဖြင့် Double quotes ("") တစ်စုံကြား ညုပ်ရေးထားတဲ့ ကာရိုက်တာတွေ အသီအတန်းလိုက်ကို စာသားအနေနဲ့ ယူဆတယ်။ Python မှာ စာသားရဲ့ တိုက်ပဲဟာ *str* ဖြစ်တယ်။ *string* ကို အတိုကောက် ယူထားတာပါ။

```
>>> "Hello, World!"
'Hello, World!'
```

သို့မဟုတ် " အစား single quotes('') တစ်စုံလည်း သုံးနိုင်ပါတယ်။

```
>>> 'Hello, World!'
'Hello, World!'
```

စာသားတစ်ခုမှာ ပါဝင်တဲ့ ကာရိုက်တာ အရေအတွက်ကို *len* ဖန်ရှင်နဲ့ စစ်ကြည့်နိုင်ပါတယ်။ ကာရိုက်တာ တစ်လုံးမှ မပါတဲ့ "" (သို့ '') ကို empty string လို့ ခေါ်ပါတယ်။

```
>>> len("Hello, World!")
13
>>> long_sentence = "This is a long sentence nobody wants to read."
```

```

>>> len(long_sentence)
45
>>> len("")
0
>>> len(" ") # contain a single space
1

```

str တိုက်ပဲခဲ့ အခြေခံကျတဲ့ အော်ပရေးရှင်းတစ်ခုက စာသားတစ်ခုနဲ့ တစ်ခု ဆက်တာပါ။ + အော်ပရိတ်တာနဲ့ စာသားတွေကို ဆက်နိုင်ပါတယ်။

```

>>> "Yangon " + "and " + "Mandalay"
'Yangon and Mandalay'

```

စာသားအချင်းချင်းပဲ ဆက်လိုပါတယ်။ စာသားနဲ့ ကိန်းကဏ္ဍး ဆက်လိုမရပါဘူး။ အောက်ပါအတိုင်း စမ်းကြည့်တဲ့အခါ str နဲ့ float ဆက်လိုမရဘူးလို့ အယ်ရှုမက်ဆွဲချုပ် ကျလာမှပါ။

```

>>> from math import *
>>> pi
3.141592653589793
>>> "The value of π is " + pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "float") to str

```

str ဖန်ရှင်က ကိန်းကဏ္ဍးတစ်ခုကနေ စာသားကို ထုတ်ပေးပါတယ်။ မူရှင်းကိန်းကဏ္ဍးကို စာသားဖြစ်အောင် ပြောင်းလိုက်တာ မဟုတ်ပါဘူး။ ကိန်းကဏ္ဍး တန်ဖိုးကနေ ငြင်းကိုဖော်ပြတဲ့ စာသားကို ဖန်ရှင်က ပြန်ထုတ်ပေးတာပါ။

```

>>> str(pi)
'3.141592653589793'

```

ထွက်လာတဲ့ တန်ဖိုးပော စာသားဖြစ်တဲ့အတွက် single quote ပါနေတာ သတိပြုပါ။ pi တန်ဖိုးနဲ့ စာသားအောင်လို့ ဆက်ရပါမယ်။

```

>>> "The value of π is " + str(pi)
'The value of π is 3.141592653589793'

```

str ဖန်ရှင်နဲ့ စာသားရအောင် အရင်လုပ်ပြီးမှ ဆက်ထားတာပါ။
စာသားကနေ ကိန်းကဏ္ဍး လိုချင်ရင် int နဲ့ float ဖန်ရှင် သုံးနိုင်ပါတယ်။

```

>>> int('1024')
1024
>>> int('1024') * 2
2048
>>> float('2.4') * 3
7.199999999999999

```

ကဏ္ဍးပြောင်းလို့မရတဲ့ စာသားဖြစ်နေရင် အယ်ရှုဖြစ်မှပါ။

```

>>> int('1a24')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1a24'
>>> int('12.3')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '12.3'

```

'12.3' မှာ အသေမ ပါနေတာကြောင့် int ပြောင်းလို့ မရတဲ့အတွက် အယ်ရာတက်တာပါ။
စာသားကို * အော်ပရိတ်တာနဲ့ ပွားယူလို့ရတယ်။

```

>>> 'hello' * 3
'hellohellohello'

```

'hello' သုံးခါ ဆက်လိုက်တာပါ။ string နဲ့ အကြိမ်အရေအတွက် ကိန်းပြည့် ဖြစ်ရပါမယ်။ သူည် သို့မဟုတ် အနှစ်ကိန်း ဖြစ်နေရင် empty string ရမှာပါ။

```

>>> 'World' * -3
 ''
>>> 'Hello' * 0
 ''

```

စာသားနဲ့ ဂဏန်း ဖလှယ်လို့ရပါတယ်

```

>>> 3 * 'Hello'
'HelloHelloHello'

```

string ပေါ်ရောဲလ်များ

ပေါ်ရောဲလ်တွေကို string တန်ဖိုးတွေအတွက်လည်း အသုံးပြုနိုင်တယ်။ အောက်ပါ အိပ်ပရက်ရှင်တော်ကို နားလည်းပြုနိုင်မလား ကြိုးစားကြည့်ပါ။ ပေါ်ရောဲလ် တစ်ခုချင်းကို သူ့ရဲ့တန်ဖိုးနဲ့ အစားထိုးပြီး +, * အော်ပရိတ်တော့ အလုပ်လုပ်ပုံနဲ့ ဆက်စပ်စဉ်းစားရင် ဘာကြောင့် အခုလို့ အဖြတွက်လဲ ခန့်မျန်းနိုင်မှာပါ။ သိပ်ခက်ခက်ခဲ့ခဲ့ မဟုတ်ပါဘူး။

```

>>> name = 'Kathy'
>>> first_part = 'Hello'
>>> second_part = 'How are you doing?'
>>> first_part + ', ' + name + '. ' + second_part
'Hello, Kathy. How are you doing?'
>>> (first_part + ', ') * 3 + name
'Hello, Hello, Hello, Kathy'

```

Escape Character and Escape Sequence

String တစ်ခု ရေးတဲ့အခါ ပုံမှန်အားဖြင့် လိုချင်တဲ့ စာသားအတိုင်း ကီးဘုံးကနေ ကာရက်တာ တစ်လုံး ချင်း ရိုက်ရုံးပါပဲ့။ စပ်ယူရှုံး ကာရက်တာ တချို့ကိုတော့ ကီးဘုံးကနေ တိုက်ရိုက် ရိုက်ထည့်လို့ မရဘဲ သိ

မြားနည်းလမ်းတစ်ခုနဲ့ ရေးပေးရပါတယ်။ ဥပမာ စာသားထဲမှာ tab ကာရက်တာအတွက် \t နဲ့ newline အတွက် \n ရေးရမှုပါ။ ကိုဘုံးကနေ tab ကို၊ enter/return ကို နိုပ်ပြီး တိုက်ရှိက်ထည့်လို့မရပါဘူး။ သီးပြားအစိုးပိုယ် တစ်ခုအတွက် \ နဲ့စတဲ့ ကာရက်တာအတွဲလိုက်ကို *escape sequence* လို့ခေါ်ပြီး \ ကိုတော့ *escape character* လို့ ခေါ်ပါတယ်။

```
>>> two_lines = "Line 100\nLine 101"
>>> two_lines
'Line 100\nLine 101'
>>> tabs_eg = "Line 1\t\t1,000,000\nLine 1000\t10,000"
>>> tabs_eg
'Line 1\t\t1,000,000\nLine 1000\t10,000'
```

Escape sequence တွေကို **bold** ဖော်နဲ့ ပြထားပါတယ်။ Python ကွန်ဆိုးလုံး \t နဲ့ \n ကို အရှိအတိုင်း ပြနေပါတယ်။ ဒါပေမဲ့ အခုလို စမ်းကြည့်ရင် သိသာပါလိမ့်မယ်။

```
>>> print(two_lines)
Line 100
Line 101
>>> print(tabs_eg)
Line 1      1,000,000
Line 1000    10,000
```

Double quotes တစ်စုံနဲ့ စာသားထဲမှာ " ပါနေရင် \ " လို့ရေးရပါမယ်။ Single quote တစ်စုံနဲ့ စာသားထဲမှာ ' ပါနေရင်လည်း \ ' လို့ရေးရပါမယ်။

```
>>> 'I\'ll tell you the truth'
"I'll tell you the truth"
>>>
>>> 'I'll tell you the truth'
File "<stdin>", line 1
    'I'll tell you the truth'
    ^
SyntaxError: invalid syntax
```

```
>>> "He said, \"I am very tired\""
'He said, "I am very tired"'
>>>
>>> "He said, "I am very tired\""
File "<stdin>", line 1
    "He said, "I am very tired\""
    ^
SyntaxError: invalid syntax
```

Double quotes နဲ့ စာသားထဲက single quote ထို့မဟုတ် single quote နဲ့ စာသားထဲက double quotes ထို့ရင်တော့ \ မလိုပါဘူး။

```

>>> "I'll tell you the truth"
'I'll tell you the truth'
>>> 'He said, "I am tired"'
'He said, "I am tired"'

```

နှစ်မျိုးလုံး ပါနေရင်တော့ တစ်မျိုးက \ ပါရပါမယ်။ ကျွန်တဲ့တစ်မျိုးကတော့ ပါရင်လည်းရာ မပါလည်း
ပြသောမရှိဘူး။ အောက်ပါတို့ကို ဂရုစိုက် လေ့လာကြည့်ပါ။

```

>>> 'He asked, "Don\'t you like?"'
'He asked, "Don\'t you like?"'
>>> "He asked, \"Don't you like?\""
'He asked, "Don\'t you like?"'
>>> "He asked, \"Don\\'t you like?\""
'He asked, "Don\\'t you like?"'
>>> 'He asked, \"Don\\'t you like?\"'
'He asked, "Don\\'t you like?"'

```

Escape Sequence	အဓိပ္ပာတ်
\'	single quote
\"	double quote
\\\	backslash
\t	tab
\n	newline
\r	carriage return

တေဘတ် ၅.၁ Python Escape Sequences

၅.၅ အိပ်စ်ပရက်ရှင်များ

တိုက်ပဲ ဆိတာဘာလ အကြမ်းဖျဉ်း ရှင်းပြခဲ့ ပြီးပါပြီ။ ကိန်းကဏ္ဍးနဲ့ စာသား တိုက်ပဲ တချို့ကိုလည်း
လေ့လာခဲ့ပြီးပြီ။ တကယ်တော့ အိပ်စ်ပရက်ရှင် (expression) ဆိတာလည်း အသစ်အဆန်း မဟုတ်ပါ
ဘူး။ တန်ဖိုးတစ်ခုဟာ အရှိုးရှင်းဆုံး အိပ်စ်ပရက်ရှင်လို့ ဆိုခိုင်ပါတယ်။ "Hello", 2.3 စတဲ့ တန်ဖိုးတွေ
ဟာ အိပ်စ်ပရက်ရှင်တွေပါပဲ။ ဗောဓါရေဘဲလဲဟာလည်း တန်ဖိုးကို ကိုယ်စားပြုတဲ့အတွက် အိပ်စ်ပရက်ရှင်
လို့ ယူဆရမှာပါ။

ရုံးရှင်းတဲ့ အိပ်စ်ပရက်ရှင်တွေကနေတစ်ဆင့် ပေါင်းစပ် အိပ်စ်ပရက်ရှင် (compound expression)
တွေ ဖွဲ့စည်းတည်ဆောက် ယူနိုင်ပါတယ်။

```

>>> 2 + 5
7
>>> (3 + 2) * (2 / 5)
2.0
>>> 'Hello, ' * 3 + 'World'
'Hello, Hello, Hello, World'

```

အိပ်စ်ပရက်ရှင်ကို ရှုထောင့်အမျိုးမျိုးကင်း အမိပိုယ်ဖွင့်ဆိုကြတာ တွေ့ရပါတယ်။ “အိပ်စ်ပရက်ရှင် ဆိုတာ တန်ဖိုးတစ်ခု ပြန်ပေးတဲ့ အော်ပရေးရှင်း အတွဲအဆက်ဖြစ်တယ်” လို့ဆိုရင် အတိုင်းအတာတစ်ခုအထိ တိတိကျကျရှိပြီး နားလည်ရလည်း လွယ်ပါတယ်။ တချို့စာအုပ်တွေမှာတော့ အိပ်စ်ပရက်ရှင်ကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်လို့ သတ်မှတ်ပါတယ်။

ပထမအမိပိုယ်အရ အိပ်စ်ပရက်ရှင်း စတိတ်မန့် မတူဘူးလို့ ယူဆပါတယ်။ ဒီရှုထောင့်က ကြည့်ရင် စတိတ်မန့်ဟာလည်း အော်ပရေးရှင်း အတွဲအဆက်ဖြစ်ပေမဲ့ တန်ဖိုးပြန်မပေးဘူး။ အိပ်စ်ပရက်ရှင်ကတော့ တန်ဖိုးပြန်ပေးရမှာပါ။ $3 * 2$ ကို လုပ်ဆောင်တဲ့အခါ 6 ရပါတယ်။ ဒါကြောင့် $3 * 2$ ဟာ အိပ်စ်ပရက်ရှင်ဖြစ်တယ်။ `result = 3 * 2` က စတိတ်မန့်ဖြစ်တယ်။ အဆိုင်းမန့်ဟာ ဖေရှိရောဘဲလိုကို တန်ဖိုးတစ်ခုနဲ့ တွဲဖက်ပေးတာ။ တန်ဖိုးပြန်မပေးဘူး။

ဒုတိယအမိပိုယ်အရ အိပ်စ်ပရက်ရှင်သည်လည်း စတိတ်မန့်ပဲ။ စတိတ်မန့်တွေကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်နဲ့ ပြန်မပေးတဲ့ စတိတ်မန့် အုပ်စုနှစ်စု ခွဲခြားတယ်။ တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်တွေကို အိပ်စ်ပရက်ရှင် သို့မဟုတ် အိပ်စ်ပရက်ရှင်စတိတ်မန့်လို့ ဒုတိယအမိပိုယ် သတ်မှတ်ချက်အရ ခေါ်ဘာပါ။

တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်ကောလ်တွေကိုလည်း အိပ်စ်ပရက်ရှင်လို့ ယူဆပါတယ်။ ဥပမာ `sqrt(9)` က 3.0 ရပါတယ်။

```
>>> from math import *
>>> sqrt(9)
3.0

print(3) ကတော့ အိပ်စ်ပရက်ရှင် မဟုတ်ပါဘူး။ အခုလို စမ်းကြည့်ရင် 3 ထုတ်ပေးတဲ့အတွက် အိပ်စ်ပရက်ရှင်လို့ ထင်စရာ အကြောင်းရှုပါတယ်။
```

```
>>> print(3)
3

ဒါပေမဲ့ ဒါဟာ print ဖန်ရှင်က output ထုတ်ပေးတာပါ။ တန်ဖိုးပြန်ပေးတာ မဟုတ်ပါဘူး။ တန်ဖိုးပြန်ရတယ်ဆိုရင် အခြားအိပ်စ်ပရက်ရှင်တစ်ခုမှာ တန်ဖိုးအနေနဲ့ သုံးလို့ရ ရမယ်။ ဥပမာ print(3) + 2 ကို စမ်းကြည့်ပါ။
```

```
>>> print(3) + 2
3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

၅.၆ ဘူလီယန် တိုက်ပန့် ဘူလီယန် အိပ်စ်ပရက်ရှင်

Python မှာ `bool` တိုက်ပေးတာ ဘူလီယန် (boolean) တိုက်ပိုကို ဆိုလိုတာဖြစ်ပြီး `True` နဲ့ `False` တန်ဖိုး နှစ်ခုပဲ ပါဝင်တယ်။ မှန်ခြင်း/မှားခြင်း၊ ရှိခြင်း/မရှိခြင်း၊ ဖြစ်ခြင်း/မဖြစ်ခြင်း စတဲ့အချက်အလက် မျိုးတွေကို ဖော်ပြန် (boolean) တိုက်ပိုကို အသုံးပြန်ပါတယ်။

```
is_winter = True
has_four_legs = False
is_adult = True
```

တန်ဖိုးရှာတဲ့အခါ ဘူလီယန်တိုက်ပ် ရလာမဲ့ အိပ်စ်ပရက်ရှင်တွေကို ဘူလီယန် အိပ်စ်ပရက်ရှင်လို့ ခေါ်တယ်။ အောက်ပါတို့ကို ကြည့်ပါ

```
>>> 'abc' == 'abc'
True
>>> 'Apple' < 'Opple'
True
>>> 'Opple' < 'Apple'
False
>>> 5 == 5
True
>>> 2 < 5.0
True
>>> 2.0 <= 2.0
True
>>> 2 != 2
False
>>> 2 != 3
True
```

<, >, <=, >=, ==, != စတဲ့ သကောက်တတွေဟာ comparison operator တွေပါ။ Relational operator တွေလိုလည်း ခေါ်ကြတယ်။ အလယ်တန်းသချုပ်မှာ သင်ခဲ့ရတဲ့ <, >, ≤, ≥, =, ≠ စာတွေနဲ့ အမိပိုယ်တူပါတယ်။ ညီ/မညီ စစ်ချင်ရင် ညီမျှခြင်းသကောက်တန်းခါ == နဲ့ စစ်ရပါမယ်။ Comparison operator တွေကို အောက်တိုက်ပ် အမျိုးမျိုးနဲ့ တွဲဖက် အသုံးပြုခိုင်တာကို တွေ့ရမှာပါ။ တန်ဖိုးတစ်ခုနဲ့ တစ်ခု နှိမ်းယူဉ်စိုင်ခြင်းဟာ အရေးပါတဲ့ ကိစ္စဖြစ်ပါတယ်။

Python မှာ True == 1, False == 0 ဖြစ်တယ်လို့ သတ်မှတ်ပါတယ်။

```
>>> True == 1
True
>>> False == 0
True
>>> True == 5
False
```

ဘူလီယန် အိပ်စ်ပရက်ရှင်တွေနဲ့ ပါတ်သက်ပြီး နောက်ထပ် သိထားရမဲ့ အော်ပရိတ်တာ သုံးခေါ်တော့ and, or, not တို့ပဲ ဖြစ်ပါတယ်။ ငှင်းတို့ကို ဘူလီယန်အော်ပရိတ်တာလို့ ခေါ်ပြီး ဘူလီယန်တန်ဖိုး (သုံး) ဘူလီယန် အိပ်စ်ပရက်ရှင်တွေနဲ့ တွဲဖက်အသုံးပြုရပါတယ်။

ဘူလီယန်တန်ဖိုး/အိပ်စ်ပရက်ရှင် နှစ်ခုမှာ နှစ်ခုလုံး မှန်/မမှန် စစ်ကြည့်ချင်တဲ့အခါ and အော်ပရိတ်တာ သုံးရပါမယ်။ နှစ်ခုလုံး အမှန်ဖြစ်မှု True ရပါတယ်။

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
```

```
>>> False and False
False
```

ကိန်းကဏ္ဍးတစ်ခုက တစ်နဲ့ တစ်ဆယ်ကြား ရှိ/မရှိ အခုလိုစစ်နိုင်ပါတယ်

```
>>> x = 5
>>> x > 1 and x < 10
True
>>> y = 11
>>> y > 1 and y < 10
False
```

တစ်ထက်လည်းကြီး၊ တစ်ဆယ်ထက်လည်း ငယ်တယ်ဆိုရင် တစ်နဲ့တစ်ဆယ်ကြား ဖြစ်ပါတယ်။ (Python မှာ $1 < x < 10$ နဲ့ စစ်လိုလည်း ရတယ်)။ ယဉ်မောင်းလိုင်စင် ဖြစ် အသက် ဆယ့်ရှစ်နှစ်နဲ့ အထက် ဖြစ်ရမယ်၊ မှတ်ပုံတင်လည်း ရှိရမယ်ဆိုပါစို့။ ဘူလီယန် အပ်စ်ပရက်ရှင်နဲ့ အခုလို ဖော်ပြနိုင်တယ်

```
>>> is_18 = True
>>> has_nric = True
>>> is_18 and has_nric
True
>>> is_18 = False
>>> has_nric = True
>>> is_18 and has_nric
False
```

(အခါ ဥပမာမှာ True/False တွေ ပုံသေထည့်ထားပေမဲ့ နောက်ပိုင်းမှာ ပရိုဂရမ် input ပေါ် မှတည် ပြီး တွက်ချက်လုပ်ဆောင်တာတွေ တွေ့ရမှာပါ)။

ဘူလီယန်တန်ဖိုး/အပ်စ်ပရက်ရှင် နှစ်ခုအနက် အနည်းဆုံး တစ်ခု မှုန်/မမှုန် or နဲ့ စစ်နိုင်တယ်။ နှစ် ခုလုံး အမှုးဖြစ်မှ False ပါပါတယ်။ တစ်ခုမှုန်တာနဲ့ အမှုန်ထွက်ပါတယ်။

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

ကိန်းတစ်ခုဟာ တစ်နဲ့ တစ်ဆယ်ရဲ့ ပြင်ပမှာရှိလား အခုလို စစ်နိုင်တယ်

```
>>> x = 11
>>> x < 1 or x > 10
True
>>> y = -2
>>> y < 1 or y > 10
True
```

```

>>> z = 5
>>> z < 1 or z > 10
False

```

တစ်ထက်ငယ်ရင် (သို့) တစ်ဆယ်ထက်ကြီးရင် တစ်နဲ့တစ်ဆယ်ကြားမှာ မဟုတ်လိုပါ။ (Python မှာ တန်ဖိုးနှစ်ခုကြားမှာရှိလား သို့မဟုတ် တန်ဖိုးနှစ်ခု ပြင်ပမှာရှိလား စစ်လို့ရတဲ့နည်း တစ်ခုမကရှိပါတယ်။ အခုန်ည်းလမ်းက and နဲ့ or ကို အသုံးချဖို့ ဥပမာတချို့ကို ပြခြင်းသာဖြစ်တယ်။)

ကုမ္ပဏီ တစ်ခုက အလုပ်ခေါ်တဲ့အခါ ဘဲရ (သို့) ဒီပါလိုမနဲ့ လုပ်သက် (၂) နှစ်အထက် အနည်းဆုံး ရှိသူ ဖြစ်ရမယ် ဆိုပါစို့။ သတ်မှတ် အရည်အချင်း ပြည့်မီ/မမီ အခုလို့ ဖော်ပြနိုင်ပါတယ်

```

>>> is_graduate = False
>>> has_2yrs_exp = True
>>> has_diploma =True
>>> is_graduate or (has_diploma and has_2yrs_exp)
True

```

တန်ဖိုးနှစ်ခု ကြား (သို့) ပြင်ပမှာ ရှိ/မရှိ စစ်တဲ့အခါ အဲဒီတန်ဖိုးနှစ်ခု အပါအဝင်လား (inclusive)၊ ဒါမှုမဟုတ် မပါဝင်ဘူးလား (exclusive) ဂရိစိုက်ဖို့ လိုပါတယ်။ တစ်နဲ့ တစ်ဆယ်ကြား (ငှုံးတို့ အပါအဝင်) ဆိုရင် အခုလို့ စစ်ရမှာပါ

```

>>> y = 1
>>> y >= 1 and y <= 10
True

```

not အော်ပရိတ်တာကတော့ True/False တန်ဖိုးကို ပြောင်းပြန် ဖြစ်စေတယ်။

```

>>> not True
False
>>> not False
True

```

တစ်နဲ့ တစ်ဆယ် ပြင်ပ (ငှုံးတို့မပါ) မှာ ရှိ/မရှိကို အခုလို့လည်း စစ်လို့ရတယ်

```

not (x >= 1 and x <= 10)

```

‘ငါးရာအောက်’ လို့ ပြောတာဟာ ‘ငါးရာနှင့်အထက် မဟုတ်’ လို့ ပြောတနဲ့ အဓိပါယ်တူတူပါပဲ။

```

>>> z = 499
>>> z < 500
True
>>> not (z >= 500)
True
>>> w = 500
>>> w < 500
False
>>> not (w >= 500)
False

```

not အော်ပရိတ်တာ သုံးခြင်းအားဖြင့် အကြောင်းအရာတစ်ခုကိုပဲ အဓိပါယ်အားဖြင့် ညီမျှတဲ့ အိပ်စံပရ်ရှင် အမျိုးမျိုးနဲ့ ဖော်ပြနိုင်ရလာတာကို တော်းခိုင်တယ်။

အခန်း ၆

အော်ဂျက်များ

“အော်ဂျက် (object) ဆိုတာဘာလ” ရှုထောင့် အမျိုးမျိုးကနေ ရှင်းပြနိုင်ပါတယ်။ အပြည့်စုံဆုံး၊ အမှန်ကန်ဆုံး ဥပမာ သို့မဟုတ် အဓိပ္ပာယ်ဖွင့်ဆိုချက် ဆိုတာ မရှိပါဘူး။ သူနည်း သူ့ဟန်နဲ့ မှန်ကန်ကြတေပါပဲ။ ဒီအခြားများတော့ အော်ဂျက်ဆိုတာ ဘာလဲ၊ ဘယ်လိမ့်မျိုးလဲ ခံစားလို့ရအောင်နဲ့ အခြေခံ အသုံးချက်တိရုံး လောက်ပဲ အဓိကထား လေ့လာကြမှာပါ။ လက်ရှိအခြေအနေနဲ့ သင့်တော်မဲ့ အဓိပ္ပာယ်ဖွင့်ဆိုချက် တချို့ကို လည်း ဖော်ပြပေးသွားမှာပါ။ သိထားသင့်တဲ့ ဘာသာရပ်ဆိုင်ရာ စကားလုံး အသုံးအနှစ်းတွေကိုလည်း မိတ်ဆက်ပါမယ်။

ဆော်စဲ အော်ဂျက်တွေဟာ အပြင်မှာ တကယ်ရှိတဲ့ အရာတွေရော တကယ်မရှိဘဲ စိတ်ကူးသက်သက်ဖြစ်တဲ့ ဒုံးခိုင်ဒီယာတွေကိုပါ ပရိုက်မဲ့တဲ့မှာ ထင်ဟပ် ဖော်ပြတယ်။ ဥပမာ ကေသို့ဘက်အကောင့်၊ $\frac{7}{13}$ (အပိုင်းကဏ်နဲ့ တစ်ခု၊) 1948-01-04 (မြန်မာပြည် လွှတ်လပ်ရေးရတဲ့နေ့၊) စနီပိုင်တဲ့ အနီရောင် တို့ယို တာကား စတာတွေကို အော်ဂျက်တွေနဲ့ ဖော်ပြနိုင်တယ်။

အော်ဂျက်များလည်း တို့က်ပ်သဘောတရား ရှိတယ်။ တို့က်ပ်တူတဲ့ အော်ဂျက်အားလုံး အော်ဂျက်အားလုံး စည်းထားပဲ တူတယ်။ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေလည်း တူပါတယ်။ ဘက်အကောင့် အော်ဂျက်အားလုံးဟာ လက်ကျော်ငွေနဲ့ အကောင့်နံပါတ် ပါရှိပြီး ငွေသွေး၊ ငွေထုတ်၊ ငွေလွှဲ အော်ပရေးရှင်းတွေ လုပ်ဆောင်လို့ ပါမယ်။

အော်ဂျက်တွေရဲ့ တို့က်ပ်နဲ့ နီးနီးစပ်စပ် ဆက်နှုယ်နေတာကတော့ ကလပ်စဲ (class) သဘောတရားပါ။ ကလပ်စကို တို့က်ပ်တူအော်ဂျက်တွေ ဖန်တီးဖို့အတွက် သတ်မှတ်ထားတဲ့ ပရိုက်မဲ့ကုဒ် အစုအဝေးလို့ ယော်ယျု ပြောနိုင်ပါတယ်။ Account ကလပ်စဲ date ကလပ်စဲ Fraction ကလပ်စဲ စသည်ဖြင့် အော်ဂျက် တို့က်ပ် တစ်မျိုးအတွက် ကလပ်စက်ခု ရှိမှာပါ။ တို့က်ပ်တူ အော်ဂျက်တွေ အားလုံးမှာ ပါဝင်မဲ့ အချက်အလက်တွေ၊ အော်ပရေးရှင်းတွေနဲ့ အော်ဂျက် ဖန်တီးယူတဲ့ ဖန်ရှင်တွေကို ကလပ်စဲ တစ်ခုနဲ့ သတ်မှတ်ရတာပါ။ ကလပ်စက်အော် အော်ဂျက်တွေ (တို့က်ပ် တူပါမယ်) ထုတ်ယူရတာ ဖြစ်တဲ့ အတွက် ကလပ်စကို အော်ဂျက် စက်ရှုလိုလည်း ဆိုနိုင်ပါတယ်။ ပုံစံတူ အော်ဂျက်တွေ ထုတ်ပေးတာ မို့လို့ ကလပ်စဲဆိုတာ အော်ဂျက်တည်ဆောက်တဲ့ blueprint သို့မဟုတ် template ပဲလို့ ယူဆတာ ဟာလည်း သဘာဝကျက်တယ် ဆုံးရမှာပါ။

အော်ဂျက်တွေကို အက်ဘ်စရက်ရှင်း (abstraction) အနေနဲ့လည်း ရှုမြင်နိုင်တယ်။ ဘယ်လို ဖန်တီး တည်ဆောက်ထားလဲ မသိဘဲ အသုံးပြုလိုရတဲ့ အရာအားလုံးကို အက်ဘ်စရက်ရှင်းလို့ ဆိုနိုင်တယ်။ အပြင်မှာသုံးကြတဲ့ ကား၊ တို့စွာ ကွန်ပူးတာ စတာတွေဟာ အက်ဘ်စရက်ရှင်းတွေ ဖြစ်တယ်။ ဖန်ရှင်တွေဟာလည်း အက်ဘ်စရက်ရှင်းတွေပါပဲ။ အော်ဂျက်တွေကတော်တော့ အော်ပရေးရှင်း တွဲဖက်ပေါင်းစပ်ထားတဲ့ အက်ဘ်စရက်ရှင်းတွေပါ။ အော်ဂျက်တွေကတော်ခုနဲ့ တွဲဆက်ထားတဲ့ အော်ပရေးရှင်းတွေဟာ

အဲဒီအော်ဂျက်ရဲ ဒေတာတွေကို အသုံးပြုတယ်။ အဲဒီအော်ဂျက်ရဲ ဒေတာအပေါ် သက်ရောက်မှု ရှိနိုင်တယ်။ အခြားအော်ဂျက်ရဲ ဒေတာကို မသုံးဘူး။ သက်ရောက်မှုလည်း မရှိစေဘူး။ အော်ဂျက် အတွင်းပိုင်း ဒေတာတွေ ဖွံ့ဖြည်းထားပဲနဲ့ တိုက်ပဲကို မသိဘဲ အော်ဂျက်ကို အသုံးပြုလိုရတယ်။ အော်ပရောင်းတွေဟာ တကယ်ကတော့ အော်ဂျက်ဒေတာ အသုံးပြုတဲ့ ဖန်ရှင်တွေပါပဲ။ ဒီဖန်ရှင်တွေ ဘယ်လိုရေးထားလဲ ဒေတာကို ဘယ်ပုံဘယ်နည်း အသုံးပြုတာလဲ သိစရာမလိုဘဲ အသုံးပြုလို ရပါတယ်။

ဖန်ရှင် အသင့်ရှိပြီးသား ဆိုရင် အသုံးပြုလို ရသလို ကလပ်စ် အသင့်ရှိပြီးသား ဆိုရင် အော်ဂျက် တွေ ဖန်တီးအသုံးပြုနိုင်ပါတယ်။ ဖန်ရှင်ရေးရတာ ခက်ခဲနိုင်ပါတယ်။ ရှိပြီးသား ဖန်ရှင်သုံးတာကတော့ မ ခက်ပါဘူး။ ဒီသဘောပါပဲ။ ကလပ်စ်သတ်မှတ်ရတာ၊ ဒီဇိုင်းလုပ်ရတာ ရှုပ်ထေး ခက်ခဲနိုင်ပါတယ်။ ရှိပြီးသား ကလပ်စ်ကနေ အော်ဂျက် ဖန်တီးအသုံးပြုရတာ မခက်ပါဘူး။ အသုံးပြုသူ လွှယ်ကူးအဆင်ပြေစေနိုင်တယ်။ တည်ဆောက်သူက အဓိကစုစုံစား ဖြေရှင်းရတာပါ။ သုံးစွဲသူအဆင့်ကနေ စတင်ပြီး တည်ဆောက်သူ ပရိုဂရမ့်မာ ဖြစ်လာအောင် တစ်ဆင့်ချင်း တက်လှမ်းစွဲဟာ အဓိကပန်းတိုင်ပါ။ အော်ဂျက်မိတ်ဆက် ခက်ရပ်၊ အခုပဲ လက်တွေ့စမ်းသပ် ကြည့်လိုက်ရအောင် ...

၆.၁ date, time and datetime

ဆော်ဝဲ အပ်ပလီကေးရှင်းတွေမှာ အချိန်နာရီ၊ နေ့ရက်တွေနဲ့ တွေ့ကျက်ဆုံးဖြတ်ရတာတွေ အမြတ်လိုပါတယ်။ ဒါကြောင့် အချိန်နဲ့သက်ဆိုင်တဲ့ အချက်အလက်တွေကို စနစ်တကျ ကိုင်တွေယ်ဖြေရှင်းတတ်ဖို့ လေ့လာထားရပါမယ်။ အချိန်နဲ့ နေ့ရက်အတွက် date, time, datetime ကလပ်စ်တွေ ထောက်ပဲပေးထားတဲ့ datetime လိုက်ဘာရီ အသုံးပြုပါမယ်။ date ကလပ်စ်ကနေ ဖန်တီးယူတဲ့ အော်ဂျက်တစ်ခုဟာ အနောက်တိုင်းပြုကွာဒိန် နေ့ရက်တစ်ရက်ကို ဖော်ပြုတယ်။ ခုနှစ်၊ လ၊ ရက် အချက်အလက် သုံးခုပါဝင်တယ်။ မြန်မာပြည် လွှတ်လပ်ရေးရဲ့တဲ့ နေ့ရက်ကို ဖော်ပြရင် အခုလိုပါ

```
>>> from datetime import *
>>> date(1948, 1, 4)
datetime.date(1948, 1, 4)
```

ဒုတိယလိုင်းက အော်ဂျက် ဖန်တီးတာပါ။ ဖန်ရှင်ခေါ်တာနဲ့ ပုံစံတူတာ တွေ့ရတယ်။ အော်ဂျက်ဖန်တီး ဒုတိယလိုင်းက အော်ဂျက် ဖန်တီးတာပါ။ ဒုတိယလိုင်းက အော်ဂျက် ဖန်တီးယူတဲ့ instantiation လိုပေါ်ပြီး ရရှိလာတဲ့ အော်ဂျက်ကို အဲဒီကလပ်စ်ရဲ့ instance လိုလည်း ခေါ်ပါတယ်။

```
>>> mmid = date(1948, 1, 4)
```

အော်ဂျက်ကို မေရီရောတဲ့လိုနဲ့ အဆိုင်းမန်လုပ်တာပါ။ အော်ဂျက်ကို mmid မေရီရောတဲ့လိုနဲ့ ရည်ညွှန်းအသုံးပြုလို ရှုံးဖြစ်တယ်။

```
>>> mmid.year
1948
```

Dot notation (. အမှတ်အသား) နဲ့ အော်ဂျက်ရဲ့ ခုနှစ်ကို ရယူထားတာပါ။ လနဲ့ ရက်ကိုလည်း အလားတူနည်းလမ်းနဲ့ ယူကြည့်နိုင်တယ်။

```
>>> mmid.month
1
>>> mmid.day
4
```

အော်ဂျက်တစ်ခုမှာ ပါဝင်တဲ့ ဒေတာကို *attribute* လို့ ခေါ်တယ်။ *Attribute* တွေဟာ အော်ဂျက် ရဲ့ လက်ရှိအခြေအနေ (*state*) ကိုဖော်ပြတယ်။ စတိတ် ပြောင်းလဲနိုင်တဲ့ အော်ဂျက်တွေကို *mutable object* လို့ ခေါ်တယ်။ စတိတ် မပြောင်းလဲနိုင်တဲ့ အော်ဂျက်တွေကို *immutable object* လို့ ခေါ်တယ်။ *date* အော်ဂျက်တွေဟာ *immutable object* တွေပါ။ ဆိုလိုတာက *attribute* တွေဖြစ်တဲ့ *year*, *month*, *day* တန်ဖိုးတွေ မပြောင်းလဲနိုင်ပါဘူး။

ခုနှစ်၊ လ၊ ရက် အတွက် *attribute* သုံးခဲ့ဟာ *date* အော်ဂျက် တစ်ခုစီတိုင်းအတွက် ကိုယ်ပိုင် ပါရှိမှုပါ။ ဆိုလိုတာက အောက်ပါ *usid* အော်ဂျက်ရဲ့ *attribute* တွေနဲ့ ခုနာ *mmid* ရဲ့ *attribute* တွေဟာ သီးခြားစီပါ။ နံမည်တူပေမဲ့ တစ်ခုနဲ့တစ်ခု မရောယ်ဘူး။

```
>>> usid = date(1776, 7, 4)
```

```
>>> usid.year
1776
>>> usid.month
7
>>> usid.day
4
```

အခုလို တန်ဖိုးပြန်ယူကြည့်ရင်လည်း ဖြစ်သင့်တဲ့ အတိုင်း သက်ဆိုင်ရာ အော်ဂျက်ရဲ့ *attribute* တန်ဖိုး တွေပဲ ပြန်ရတာပေါ့။ လွတ်လပ်ရေး ရဲ့တဲ့ နောက် ဘာနေ့ဖြစ်မလဲ

```
>>> usid.isoweekday()
4
>>> mmid.isoweekday()
7
```

ဖန်ရှင်တစ်ခုတည်းကို မတူညီတဲ့ အော်ဂျက်နှစ်ခုအပေါ်မှ အသုံးချုတာ ဖြစ်တယ်။ ဒေါ်ထိကိုပဲ သုံး တယ်။ ပထမတစ်ခုက *usid* အော်ဂျက်၊ နောက်တစ်ခုက *mmid* အော်ဂျက်အပေါ်မှ သုံးထားတာပါ။ အမေရိကန် လွတ်လပ်ရေးရဲ့တာ ကြောသာပတေးနေ့၊ မြန်မာကတော့ တန်ငံ့နေ့ပါ (တန်လဲ့က တစ်၊ တန်ငံ့နေ့က ခုနှစ်ပါ)။ *isoweekday* ဖန်ရှင်ကို အော်ဂျက်တစ်ခုအပေါ် အသုံးပြုတဲ့ အခါ ငင်းအော် ဂျက်နဲ့ သက်ဆိုင်တဲ့ ဒေတာနဲ့ ဖန်ရှင်က အလုပ်လုပ်သွားတာပါ။ ဒါကြောင့်လည်း *attribute* မတူတဲ့ အော်ဂျက်တွေအပေါ်မှ အသုံးချုတဲ့ အခါ မတူညီတဲ့ ရလဒ်တွေ ထွက်လာရတာပေါ့။ ‘ဒေတာနဲ့ အော်ပ ရေးရှင်း တွဲဖက်ထားတယ်’ ဆိုတာ ဒီသဘောတရားကို ဆိုလိုတာပါ။ အော်ဂျက်ဒေတာနဲ့ တွဲဖက်အလုပ် လုပ်တဲ့ ဖန်ရှင်တွေကို မက်သွဲ (method) လို့ ခေါ်တယ်။ နောက် မက်သွဲတစ်ခုက *isoformat* ပါ။ နေ့ရက်ကို စားသားအဖြစ် 'yyyy-mm-dd' ဖော်မတဲ့ ပြန်ပေးတယ်။

```
>>> usid.isoformat()
'1776-07-04'
>>> mmid.isoformat()
'1948-01-04'
```

date တစ်ခု ဖန်တီးတဲ့ အခါ ခုနှစ်၊ လ၊ ရက် နေရာ မှန်ဖို့ အရေးကြီးပါတယ်။ *date(1948, 4, 1)* လို့ ရောမိရင် လေးလပိုင်း တစ်ရက်နေ့ ဖြစ်သွားမှုပါ။ ဒါပေမဲ့ Python မှ အုပ္ပန်တွေကို နံမည်နဲ့ တွဲပြီး ထည့်ပေးလို့ရတယ်။

```
>>> mmid = date(day=4, year=1948, month=1)
```

ဒီနည်နဲ့ ဆိုရင်တော့ year, month, day ကြိုက်သလို အစီအစဉ်နဲ့ ထည့်လိုရမှာပါ။
replace မက်သင် ဘယ်လို အလုပ်လုပ်လဲ ကြည့်ရအောင်

```
>>> usid = date(1776, 7, 4)
```

```
>>> usid100 = usid.replace(year=1876)
```

နိုင် usid နေ့ရက်ရဲ့ ခုနှစ်ကို 1876 နဲ့ အစားထိုးထားတဲ့ အော့သံဂျက် အသစ်တစ်ခု ပြန်ရပါတယ်။ နိုင်ရက်စွဲက မပေါ်ပေါ်သွားဘူး (date အော့သံဂျက် ဟာ immutable ဖြစ်တာ သတိပြုပါ)။

```
>>> usid
```

```
datetime.date(1776, 7, 4)
```

```
>>> usid100
```

```
datetime.date(1876, 7, 4)
```

ခုနှစ်၊ လ၊ ရက် သုံးခုလုံး အစားထိုးချင်ရင်

```
>>> dt1 = date(2000,2,21)
```

```
>>> dt2 = dt1.replace(2010,10,10)
```

```
>>> dt3 = dt1.replace(day=20,month=12,year=2020)
```

အားကျန်း နံမည် ပပါရင် ခုနှစ်၊ လ၊ ရက် အစဉ်အတိုင်း ဖြစ်ရပါမယ်။ ရလဒ်တွေ ကြည့်ရင်

```
>>> dt1
```

```
datetime.date(2000, 2, 21)
```

```
>>> dt2
```

```
datetime.date(2010, 10, 10)
```

```
>>> dt3
```

```
datetime.date(2020, 12, 20)
```

ခုနှစ်၊ လ၊ ရက် တခုခု ချုပ်ထားကြည့်ပါ

```
>>> dt4 = dt1.replace(2020)
```

```
>>> dt5 = dt1.replace(2030,11)
```

ချုပ်ထားခဲ့တော့ နိုင်အတိုင်းရှိပါမယ်

```
>>> dt4
```

```
datetime.date(2020, 2, 21)
```

```
>>> dt5
```

```
datetime.date(2030, 11, 21)
```

ရက်တစ်ခုတည်း အစားထိုးမယ်ဆိုရင် နံမည်နဲ့တဲ့ နည်းကပဲ အဆင်ပြေပါမယ်

```
>>> dt6 = dt1.replace(day=28)
```

```
>>> dt6
```

```
datetime.date(2000, 2, 28)
```

နံမည်မပါရင် ခုနှစ်၊ လ၊ ရက် အစဉ်အတိုင်းဖြစ်တော့ကြောင့် ခုနှစ်ကို အစားထိုးမှာပါ

```
>>> dt7 = dt1.replace(28)
>>> dt7
datetime.date(28, 2, 21)
```

time and datetime

နေ့ရက်နဲ့ အချိန် တွဲရက်ကို datetime, ရက်စွဲမလိုဘဲ အချိန်ပဲဆိုရင် time သုံးပါတယ်

```
>>> t1 = time(10, 15, 20)
>>> t1.hour
10
>>> t1.minute
15
>>> t1.second
20
>>> mmid2 = datetime(1948,1,4,4,20)
>>> mmid3 = datetime(1948,1,4,4,20,0)
>>> mmid2.second
0
>>> mmid3.second
0
```

timedelta

အချိန်ကာလနဲ့ ပါတ်သက်ပြီး မဖြစ်မနေ သိထားသင့်တဲ့ နောက်ထပ်ကလပ်စ် တစ်ခုကတော့ ကြာချိန် (duration) ကို ဖော်ပြတဲ့ timedelta ကလည်ပါ။

```
>>> duration = timedelta(
...     days=50,
...     seconds=27,
...     microseconds=10,
...     milliseconds=29000,
...     minutes=5,
...     hours=8,
...     weeks=2
... )
>>> duration
datetime.timedelta(days=64, seconds=29156, microseconds=10)
```

(... က အော်တို့ ထည့်ပေးသွားတာ။ ကိုယ်တိုင် ရိုက်ထည့်စရာမလိုဘူး။ တစ်လိုင်းချင်း Enter ခေါက်သွားရဲ့ပဲ။ အပိတ်စိုက်ကွဲ့မှာ စတိတ်မနဲ့ ဆုံးတယ်ဆုံးတာ အင်တာပရက်တာက နားလည်တယ်။)

အော့သ်ဂျက် အသုံးပြုသူအနေနဲ့ အချိန်ကာလ ကြာမြင့်ချိန်ကို days, weeks, hours ... စတေတွေနဲ့ သတ်မှတ်လို့ရတယ်။ ငါးတို့ကို ရက်၊ စက်နှုန်း၊ မိုက်ခရီးစက်နှုန်း ဖွဲ့ပြီး အော့သ်ဂျက် အတွင်းပိုင်း days, seconds, microseconds attributes ဒေတာအနေနဲ့ သိမ်းမှာပါ။

```

>>> twoweeks_twomin = timedelta(weeks=1, days=7, minutes=2)
>>> twoweeks_twomin
datetime.timedelta(days=14, seconds=120)

```

ဖော်ပြခြားတဲ့ အော့ဘ်ဂျက်တွေနဲ့ အပေါင်း၊ အနှစ် အော်ပရေးရှင်းတွေ လုပ်လိုပါတယ်။ ဥပမာ
တရို့၊ လေ့လာကြည့်ပါ

```

>>> dt1 = datetime(2021, 2, 10, 23, 45, 43)
>>> dt2 = datetime(2022, 2, 10, 23, 44, 42)
>>> duration1 = dt2 - dt1
>>> duration1
datetime.timedelta(days=364, seconds=86339)

```

ဒီလိုစစ်ကြည့်ပါ

```

>>> dt3 = dt1 + duration1
>>> dt3
datetime.datetime(2022, 2, 10, 23, 44, 42)
>>> dt4 = dt2 - duration1
>>> dt4
datetime.datetime(2021, 2, 10, 23, 45, 43)
>>> dt1 == dt4
True
>>> dt2 == dt3
True

```

၆.၂ list

List ဆိတာ အိုက်တမ် item တွေ အတွဲလိုက် စုစည်းထားဖို့ အသုံးပြုတဲ့ စထရက်ချေတစ်မျိုး ဖြစ်တယ်။ Python မှာ list အော့ဘ်ဂျက်တွေကို item တွေ အတွဲလိုက် စုစည်းထားဖို့ သုံးတယ်။ ဘာအိုက်တမ်
မှ မပါတဲ့ list အသစ်တစ်ခု လိုချင်ရင် ဒီလို

```

>>> odds = list()

```

ဒဲ့ 1sit ထဲမှာ ပါတွေပါလဲ

```

>>> odds
[]

```

လေးထောင့်ကွောင်းနဲ့ list ကိုပြပေးတယ်။ လက်ရှိ list ထဲမှာ အိုက်တမ် မရှိသေးဘူး။ အိုက်တမ် တစ်
ခုချင်း ထည့်ချင်ရင် append မက်သဒ်ရှိတယ်

```

>>> odds.append(1)
>>> odds.append(3)
>>> odds.append(5)
>>> odds.append(7)

```

```
>>> odds
[1, 3, 5, 7]
```

ပါဝင်တဲ့ အိုက်တမ်တစ်ခုစီကို ကော်မားပြီး ပြတယ်။ နောက်ထပ် နည်းလမ်းတစ်ခုနဲ့လည်း list ဖန်တီးလို့ ရတယ်။ လေးထောင့်ကွင်း သုံးတဲ့နည်းပါ

```
>>> empty = []
>>> lst1 = [1,2,3,4,5,6]
>>> empty
[]
>>> lst1
[1, 2, 3, 4, 5, 6]
```

list ဟာ mutable ဖြစ်တယ်။ အိုက်တမ် နောက်တစ်ခု ထပ်ထည့်ကြည့်ပါ

```
>>> odds.append(9)
>>> odds
[1, 3, 5, 7, 9]
```

နှင့် အော့သ်ဂျက်မှာ အိုက်တမ်တစ်ခု ထပ်တိုးသွားတာ။ အော့သ်ဂျက်ရဲ့ စတိတ် ပြောင်းသွားတယ်။ အိုက်တမ်တစ်ခုကို ဖယ်ထုတ်ချင်ရင်

```
>>> odds.pop(0)
1
>>> odds
[3, 5, 7, 9]
```

list အိုက်တမ်တော့ရဲ့ တည်နေရာကို index လို့ခေါ်တယ်။ သူညျှနဲ့ စတယ်။ ဒုတိယက တစ်၊ တတိယက နှစ် စသည်ဖြင့် ဖြစ်မယ်။ အခါး odds မှာ အိုက်တမ် လေးခုရှုံးနေတယ်။ 7 ဖြူတ်ချင်ရင် index နံပါတ် 2 ကို ပေါ်လုပ်ရမှာ

```
>>> odds.pop(2)
7
>>> odds
[3, 5, 9]
```

ဖယ်လိုက်တဲ့ အိုက်တမ်တွေ နှင့်နေရာမှာ ပြန်ထည့်ချင်တယ်ဆိုပါစို့။ insert ရှိပါတယ်

```
>>> odds.insert(2, 7)
>>> odds
[3, 5, 7, 9]
>>> odds.insert(0,1)
>>> odds
[1, 3, 5, 7, 9]
```

အိုက်တမ် အစားထိုးတာ၊ index နဲ့ နေရာတစ်ခုက အိုက်တမ်ကို ပြန်ထုတ်ကြည့်တာကို လေးထောင့်ကွင်းနဲ့ ရေးနည်းလည်းရှိတယ်

```

>>> evens = [2,4,6,8,10,12]
>>> evens[0]
2
>>> evens[1]
4

```

pop နဲ့ မတူတာကို သတိပြုပါ။ pop က အိုက်တမ်ကို ဖယ်ထုတ်လိုက်တယ်။ စတိတ်ပြောင်းလဲစေတယ်။ အခုနည်းက မဖယ်ထုတ်ဘူး။ အိုက်တမ်ကိုပဲ ပြန်ပေးတာပါ။

```

>>> evens
[2, 4, 6, 8, 10, 12]

```

replace နဲ့ သဘောတရားတူတာကတော့

```

>>> evens[5] = 14
>>> evens
[2, 4, 6, 8, 10, 14]

```

နောက်ဆုံး နေရာ (index နံပါတ် 5) ကို 14 လဲထည့်လိုက်တာ။

ပေးရှိရောဲလ် နှစ်ခုက အော့ဘ်ဂျက်တစ်ခုတည်းကို ရည်ညွှန်းနေရင် သတိပြုသင့်တဲ့ ထူးခြားချက် တွေ ရှိလာပါတယ်။

```

>>> fruits = ['mango', 'apple', 'strawberry', 'kiwi']
>>> myfav = fruits

```

ဒီလိုဆိုရင် အော့ဘ်ဂျက် တစ်ခုတည်းကို fruits ကို myfav နဲ့ပါ သုံးလို့ရပါမယ်။ fruits နဲ့ အိုက် တမ်တစ်ခု ထပ်ထည့်ကြည့်မယ်

```

>>> fruits.append('orange')

```

myfav ပါ လိုက်ပြောင်းတာကို တွေ့ရမှာပါ

```

>>> myfav
['mango', 'apple', 'strawberry', 'kiwi', 'orange']

```

Mutable အော့ဘ်ဂျက်တွေမှ ဒီအချက်ကို သတိပြုစွဲ လိုပါတယ်။ ပေးရှိရောဲလ် တစ်ခုကနေ အော့ဘ်ဂျက် စတိတ်ကို ပြောင်းလဲဝဲအခါ အဲဒီအော့ဘ်ဂျက်ကို ရည်ညွှန်းတဲ့ အခြား ပေးရှိရောဲလ် အားလုံးက လည်း အပြောင်းအလဲကို မြင်ရှာ့ ဖြစ်တယ်။ Immutable ဆိုရင်တော့ စတိတ်မပြောင်းနိုင်တာပြောင့် ဒီလိုကိစ္စမျိုး စဉ်းစားစရာ မလိုဘူး။

အခန်း ၃

ကွန်ထရီးလ် စတိတ်မန္ဒများ

ကွန်ထရီးလ် စတိတ်မန္ဒတွေက ကားရဲလ်မှာ တွေ့တော့ တွေ့ခြဲ့ပြီးသားပါ။ ဒါပေမဲ့ ကားရဲလ်ပရိုကရမ်မင်း အတွက် လိုသလောက် အခြေခံကိုပဲ ကန့်သတ်ဖော်ပြခဲ့တာပါ။ ဒီအခန်းမှာ ပြည့်စုံအောင် အသေးစိတ် ဆက်လက် လေ့လာကြပါမယ်။ လက်တွေအသုံးချု ဥပမာတွေ လေ့ကျင့်ခန်းတွေ ဂရတစိုက် စီစဉ်ပေးထားတယ်။ စတိတ်မန္ဒ အသစ်တချို့လည်း တွေ့ရမယ်။ စာအုပ်တွေမှာ ဖော်ပြတာ သိပ်မတွေရပေမဲ့ ဘိုင်နာ အများစုံ အခက်အခဲတွေကြတဲ့ နေရာတွေ၊ တိတိကျကျ နားလည်ဖို့ လိုတဲ့ ပိုင့်တွေကိုလည်း အလေးပေးရှင်းပြထားတယ်။ အထူးခြားဆုံးကတော့ ရုပ်ပုံတွေဆုံးတာနဲ့ အန်နီမေးရှင်း အခြေခံကို Arcade ဂိမ်းလိုက်ဘရှိ အသုံးပြုပြီး စတင်မိတ်ဆက်ထားတာပဲ ဖြစ်တယ်။ စာသားတွေချည်းပဲထက် စိတ်လှပ်ရှားဖို့ ပုံကောင်းမယ်ထင်ပါတယ်။

၃.၁ if စတိတ်မန္ဒ

စားသောက်ဆိုင် တစ်ဆိုင်က ကျပ်ငွေ 50,000 နဲ့ အထက် သုံးတဲ့ ကတ်စတမ်းမာတွေကို 10% လျှော့ပေးပြီး ပရိုမိုးရှင်း လုပ်တယ် ဆိုပါစို့။ ကိုးဘုံးကနေ ကျသင့်ငွေ ရိုက်ထည့်ပေးရမယ်။ ဒစ်စကောင့် ရမဲ့ ကတ်စတမ်းမာအတွက်ပဲ ‘Get 10 % discount.’ ပြပေးပြီး လာရောက် စားသုံးတဲ့အတွက် ကျေးဇူးတင်ကြောင်း ‘Thanks for coming!’ ကိုတော့ ကတ်စတမ်းကို ပြပေးချင်ပါတယ်။

```
amt = float(input("Enter amount: "))

if amt >= 50_000:
    print("Get 10% discount.")
    print("Thanks for coming!")
```

amt > 50_000 ဘူးလိုယန် အိပ်စ်ပရက်ရှင် true ဖြစ်မှုပဲ if ဘလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဒစ်စကောင့် ဘယ်လောက်ရလဲရော ကျသင့်ငွေပါ ပြပေးမယ်ဆိုရင် ဒီလို

```
amt = float(input("Enter amount: "))

amt_to_pay = amt

if amt >= 50_000:
    discount = amt * 0.1
    print(f'Get 10% discount ({discount}).')
    amt_to_pay = amt - discount
```

```
print(f'Please pay: {amt_to_pay}'))
print('Thanks for coming!')
```

Python ဘားရှင်း 3.6 ကစွဲး f-string (formatted string) ခေါ်တဲ့ string အသစ်တစ်မျိုး ပါလာပါတယ်။ String ရှေ့မှာ f နဲ့ စရိတ် f-string လို့ သတ်မှတ်တယ်။ Single/double quote ရှေ့မှာ f ထည့်ပေးရတာပါ။

```
>>> f"Two plus three is {2 + 3}"
'Two plus three is 5'
>>> f'Two plus three is {2 + 3}'
'Two plus three is 5'
```

F-string နဲ့ဆိုရင် ဖော်ရောဘဲလ် (လို့) အပိုစ်ပရက်ရှင် တွေကို တွန့်ကွင်းထဲမှာ ထည့်ရေးလို့ရတယ်။ ငှါးတိုကို f-string က တန်ဖိုးရှာပြီး အစားထိုးပေးမှာပါ။ ဒါကြောင့် {2 + 3} က 5 ဖြစ်သွားတာပါ။ ရိုးရိုး string နဲ့ဆို အခုလုံး

```
>>> 'Two plus three is ' + str(2 + 3)
'Two plus three is 5'
```

ရေးနေရမယ်။ F-string နဲ့ဆို ပိုအဆင်ပြေတယ်။ နမူနာတရှို့ကို လေ့လာကြည့်ပါ

```
>>> x = 9
>>> y = 3
>>> f'2x + y = {2*x + y}'
'2x + y = 21'
>>> f'Times three hello {'hello' * 3}'
'Times three hello hellohellohello'
>>> f'Times three hello length is {len('hello' * 3)}'
'Times three hello length is 15'
```

ဒစ်စကောင့်ပေးပြီး ပရီမီးရှင်းလုပ်တဲ့အခါ သတ်မှတ်ပမာဏ မပြည့်သေးရင် ဘယ်လောက်ဖိုးထပ်သုံးတာနဲ့ ဒစ်စကောင့် ရမှာဖြစ်ပြောင့်ပြောပြီး ဆွဲဆောင်လေ့ရှိတယ်။ ဒစ်စကောင့်ရှုအောင် ဘယ်လောက်ထပ်သုံးရမလဲ ပရိုကရမ်က ပြပေးချင်တယ် ဆုံးပါစ္စား။ if...else သုံးနိုင်ပါတယ်။

```
from math import *

amt = float(input("Enter amount: "))
amt_to_pay = amt
if amt >= 50_000:
    discount = amt * 0.1
    print(f"Get 10% discount({discount}).")
    amt_to_pay = amt - discount
else:
    amt_req = ceil(50_000 - amt)
    print(f"Spend just {amt_req} to get 10% discount!")

print("Please pay: " + str(amt_to_pay))
print("Thanks for coming!")
```

`amt >= 50_000` မှန်ရင် if ဘလောက်၊ မှားရင် else ဘလောက် လုပ်ဆောင်မှာဖြစ်တယ်။
if နဲ့ if...else ထော်ယူပုံစံကို ကြည့်ရင် အခုလိုရှိပါတယ်

```
if test:  
    statement1  
    statement2  
    statement3  
    ...etc.
```

```
if test:  
    statement1a  
    statement2a  
    statement3a  
    ...etc.  
else:  
    statement1b  
    statement2b  
    statement3b  
    ...etc.
```

`test` ဟာ ဘူလိယန် အိပ်စ်ပရက်ရှင်း ဖြစ်ရပါမယ်။ (ကားရဲ့ ကုန်ဒီရှင်တွေဟာ ဘူလိယန်တန်ဖိုး ပြန်ပေးတဲ့ predicate မက်သခ်တွေပါ။ predicate မက်သခ်တွေကို ဘူလိယန် အိပ်စ်ပရက်ရှင်းလို့ ယူဆနိုင်တယ်။)

အခုခက်ကြည့်ကြမဲ့ if...elif...else ပုံစံကတော့ ရွှေ့ပိုင်းမှာ မတော့ဖူးသေးဘူး။ “Cascading if statement” လိုခေါ်တယ်။ အောက်ပါ ပေါ်မေးခွဲမှုတဲ့ ကနေ grading ထုတ်ပေးမယ် ဆိုပါစို့။

Score	Grade
90 ... 100	A
80 ... 89	B
70 ... 79	C
60 ... 69	D
(below 60) 0 ... 59	F

တေဘာ် ဂုံး Score and Grading

ကျောင်းသူး/သား နံမည်နဲ့ ရမှတ်ကို ထည့်ပေးရင် ပရှိကရမ်က အခုလို ပြပေးရပါမယ်။

```
Student name: Amy  
Score: 95  
Amy get grade A
```

ဒီပရှိကရမ် အတွက် cascading if သုံးထားတာ ကြည့်ပါ

```
stu_name = input("Student name: ")  
score = int(input("Score: "))
```

```

grade = 'F'
if 90 <= score <= 100:
    grade = 'A'
elif 80 <= score <= 89:
    grade = 'B'
elif 70 <= score <= 79:
    grade = 'C'
elif 60 <= score <= 69:
    grade = 'D'
elif 0 <= score <= 59:
    grade = 'F'
else:
    print(f'You entered {score}. Score must be between 0 and 100.')
print(f'{stu_name} get grade {grade}')

```

အပေါ်ဆုံး if ပြီးတဲ့အခါ အောက်မှာ elif တွေ အတဲ့လိုက် တွေ့ရပါမယ်။ နောက်ဆုံးမှာ else အပိုင်းကို တွေ့ရတယ် (ဒီအပိုင်းက optional ပဲ၊ မပါလိုလဲရတယ်။ ခေါ်နေ ရှင်းပြုပါမယ်)။ အလုပ်လုပ်ပုံက ဒီလို ... သက်ဆိုင်ရာ if (သို့) elif တွေရဲ့ ဘူလီယန် အိပ်စ်ပရက်ရှင် တစ်ခုချင်းကို အထက်အောက် အစဉ်အတိုင်း တန်ဖိုးရှာပါတယ်။ ပထမဆုံး True ဖြစ်တဲ့ အိပ်စ်ပရက်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို လုပ်ဆောင်ပေးမှာ ဖြစ်တယ်။ အားလုံး False ဖြစ်ရင်တော့ else ဘလောက်ကို လုပ်ဆောင်တယ်။

နောက်ဆုံး else အပိုင်းက မပါလိုလည်းရတယ်။ အပေါ်မှာ တစ်ခုမှာ True မဖြစ်တော့မှာ else ဘလောက်ကို လုပ်ဆောင်တယ်။ နောက်ဆုံးမှာ else အပိုင်းမပါဘူး၊ အပေါ်မှာလည်း ဘယ်တစ်ခုကမှ True မဖြစ်ဘူး ဆိုရင်တော့ လုပ်ဆောင်ပေးစရာ ဘလောက်လည်း မရှိဘူးပေါ့။ ဒီတော့ အားလုံး False ဖြစ်ခဲ့ရင် လုပ်ချင်တဲ့ကိစ္စ ရှိ/မရှိ အပေါ်မှာတည်ပြီး else အပိုင်း လှို/မလှိ ဆုံးဖြတ်ရတယ်။

အခု grading ပရိုကရမှာ ရမှတ်ပာ သုညနဲ့ တစ်ရာကြား ဖြစ်သင့်တယ်။ အကယ်၍ ထည့်ပေးတာမှားရင် မှားတယ်လို့ ပြပေးချင်တယ်။ ဥပမာ

```

Student name: Sandy
Score: 110
You entered 110. Score must be between 0 and 100.

```

သုညနဲ့ တစ်ရာအတွင်း မဟုတ်ရင် အပေါ်မှာ စစ်ထားတဲ့ ဘူလီယန် အိပ်စ်ပရက်ရှင်တွေ တစ်ခုမှာ မမှန်နိုင်ဘူး။ ဒါကြောင့် else အပိုင်းနဲ့ မှားထည့်ထားတယ်လို့ ပြပေးလိုက်တယ်။

အခုပရိုကရမှာ သိပ်စိတ်တိုင်းကျစရာ မကောင်းတဲ့ ပြသုနာတစ်ခုတွေ့ရပါတယ်။ အောက်ပါအတိုင်း စမ်းကြည့်ရင် Sandy က grade F ရတယ်လို့ ပြနေပါတယ်

```

Student name: Sandy
Score: 110
You entered 110. Score must be between 0 and 100.
Sandy get grade F.

```

အမှတ်ထည့်ပေးတာ မှားနေရင် grade ကို မပြပေးသင့်ပါဘူး။ ဒီလို ပြင်ရေးလိုက်မယ် ဆိုရင်

```

stu_name = input("Student name: ")
score = int(input("Score: "))

if 0 <= score <= 100:
    grade = 'F'
    if 90 <= score <= 100:
        grade = 'A'
    elif 80 <= score <= 89:
        grade = 'B'
    elif 70 <= score <= 69:
        grade = 'C'
    elif 60 <= score <= 59:
        grade = 'D'
    elif 0 <= score <= 59:
        grade = 'F'

    print(f'{stu_name} get grade {grade}.')
else:
    print(f'You entered {score}. Score must be between 0 and 100.')

```

ရုံးတဲ့ သုညနဲ့ တစ်ရာကြားဖြစ်မှ grading ထုတ်ပေးတဲ့ ကိစ္စလုပ်တယ် (if 0 <= score <= 100: နဲ့ စစ်ထားတာ)။ မဟုတ်ရင် ထည့်ထားတာ မှားနေတယ်ဆိုတာ else အပိုင်းက ပြေားမှာပါ။ အပြင် if ဘလောက်ထဲက cascading if အဆုံးမှာ else မပါတော့တာ သတိပြုပါ။ ဘာကြောင့်ပါလဲ ...

Cascading if မှ ဘူလိယန် အိပ်စံပရ်ရှင် တစ်ခုချင်းကို အထက်အောက် အစဉ်အတိုင်း တာနိုးရှာတယ်၊ 'ပထမဆုံး True ဖြစ်တဲ့ ဘလောက် တစ်ခုကိုပဲ လုပ်ဆောင်ပေးတယ်' ဆိုတဲ့အချက်ကို နားလည်ဖို့ အရေးကြီးတယ်။ အောက်ကိုရောက်လာတာဟာ အပေါ်မှာ မှားခဲ့လိုပဲ။ တစ်ခုမှန်ပြီဆိုတာနဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို လုပ်ဆောင်ပြီး cascading if တစ်တဲ့လုံး ပြီးဆုံးသွားမှာ ဖြစ်တယ်။ အောက်ပိုင်းက elif (သို့) else တွေကို မရောက်လာတော့ဘူး။ ဒီအကြောင်းကြောင့် grading အတွက် အခုလိုလည်းရေးလိုရတယ်

```

stu_name = input("Student name: ")
score = int(input("Score: "))

if 0 <= score <= 100:
    grade = 'F'
    if score >= 90:
        grade = 'A'
    elif score >= 80:
        grade = 'B'
    elif score >= 70:
        grade = 'C'
    elif score >= 60:
        grade = 'D'
    else:
        grade = 'F'

    print(f'{stu_name} get grade {grade}.')
else:

```

```
    print(f'You entered {score}. Score must be between 0 and 100.')
```

ပထမ elif ကို ရောက်လာရင် ကိုဆယ်အောက် ဖြစ်မှုတော့ သေချာတယ် (score >= 90 မဟုတ်လို့ ဒါ ကို ရောက်လာတာ)၊ ဒါကြောင့် ရှစ်ဆယ်နဲ့အထက် (score >= 80) ဖြစ်လား စစ်ရင်ရပြီ။ နောက်တစ်ဆင့် ကို ရောက်လာရင် ရှစ်ဆယ်အောက် မို့လို့ သေချာတယ်၊ score >= 70 ဖြစ်လားစစ်ရုံပဲ။ စသည်ဖြင့် အောက်အဆင့်တွေ အတွက်လည်း ထိန်ညှုံးတူစွာ စဉ်းစားနိုင်တယ်။

Cascading if မသုံးဘဲ if...else တွေနဲ့လည်း ရေးလိုတော့ ရပါတယ်။ Nesting လုပ်တာ တွေ အခမ်းများပြီး ဖတ်ရမလွယ်ကူတာ တွေ့ရမှာပါ။ Grading ပရိုဂရမ်ကို cascading if မသုံးဘဲ ရေးထားတာပါ။

```
stu_name = input("Student name: ")
score = int(input("Score: "))

if 0 <= score <= 100:
    grade = 'F'
    if score >= 90:
        grade = 'A'
    else:
        if score >= 80:
            grade = 'B'
        else:
            if score >= 70:
                grade = 'C'
            else:
                if score >= 60:
                    grade = 'D'
                else:
                    grade = 'F'
    print(f'{stu_name} get grade {grade}.')
else:
    print(f'You entered {score}. Score must be between 0 and 100.')
```

၃.J for Loop

Python for loop ဟာ အဆင့်မြင့် အက်ဘ်စရက်ရှင်း တစ်ခု ဖြစ်ပါတယ်။ စတိတ်မန့်တစ်စုကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ထပ်ခါထပ်ခါ လုပ်ဆောင်ဖို့ လိုတဲ့အခါ for loop ကို အသုံးပြုတယ်။ Loop ကို စတင် လုပ်ဆောင်တဲ့အချိန်မှာ ဘယ်နှစ်ကြိမ် ပြန်ကျောမလဲ အတိအကျ ကြိုသိရင် definite loop လို့ သတ်မှတ်တယ်။ for loop ဟာ definite loop ဖြစ်ပါတယ်။ ‘အဆင့်မြင့် အက်ဘ်စရက်ရှင်း’ လို့ ပြောရတာက list, dictionary, set, range စတဲ့ စထရက်ချုပ် အမျိုးမျိုးနဲ့ အသုံးပြုလို့ရတဲ့ အတွက်ကြောင့်ပါ။

for loop နဲ့ list ထဲက အိုက်တမ်တစ်ခုချင်း ထုတ်ယူအသုံးပြုနိုင်ပါတယ် ...

```
fruits = ['Orange', 'Kiwi', 'Banana', 'Papaya', 'Apple', 'Plum', 'Mango']

for item in fruits:
    print(item)
```

Output:

```
Orange
Kiwi
Banana
...
```

Loop တစ်ခေါက်ပြန်ကျော်တိုင်း `itm` မေရီရောင်တဲ့မှာ `list` အိုက်တမ်တစ်ခုချင်း အစဉ်အတိုင်း ထည့်
ပေးမှာပါ။ အိုက်တမ်တွေကို နံပါတ်စဉ်နဲ့ တွဲချင်ရင် `enumerate` လုပ်ပြီး အခုလို ထုတ်လိုခြာယ်

```
for idx, itm in enumerate(fruits, start=1):
    print(idx, itm)
```

Output:

```
1 Orange
2 Kiwi
3 Banana
...
```

နံပါတ်စဉ်ကို `idx`၊ အိုက်တမ်ကို `itm` နဲ့ ယူသုံးထားတာပါ။ `str` တစ်ခုထဲက ကာရ်ကာတစ်လုံးချင်း
လိုချင်ရင်လည်း ရတာပဲ

```
for ltr in 'This is a sentence written with full of emotion':
    print(ltr)
```

Output:

```
T
i
s
...
```

List နှစ်ခုရဲ့ cartesian product ၏ (ဖြစ်နိုင်တဲ့ အတွဲအားလုံး ရှာတာပါ)

```
colors = ['black', 'white']
sizes = ['S', 'M', 'L']
for color in colors:
    for size in sizes:
        print((color, size))
```

Output:

```
('black', 'S')
('black', 'M')
('black', 'L')
('white', 'S')
('white', 'M')
('white', 'L')
```

Dictionary နဲ့လည်း သုံးလို့ရတာပေါ့

```
scientist_birthdate = { 'Newton': date(1643, 1, 4),
                        'Darwin': date(1809, 2, 12),
                        'Turing': date(1912, 6, 23)}
for sci, bdt in scientist_birthdate.items():
    print(sci, bdt)
```

Output:

```
Newton 1643-01-04
Darwin 1809-02-12
Turing 1912-06-23
```

ဖော်ပြခဲ့တဲ့ ဥပမာတွေကို ကြည့်ခြင်းအားဖြင့် Python for loop ဗာ list, dictionary, string စတဲ့ စထရက်ချာအမျိုးမျိုးနဲ့ အလုပ်လုပ်နိုင်တာ တွေ့ရမှာပါ။ တစ်ခါကျော်တိုင်း အိုက်တမ်တစ်ခုကို loop ဖေရံရော့လဲထဲမှာ ထည့်ပေးထားတယ်။ အိုက်တမ်တွေအားလုံး ပြီးတဲ့အခါ for loop ရပ်သွားမှာ ဖြစ်တယ်။ ပြန်ကျော်တဲ့ အကြိမ်အရေအတွက်ဟာ အိုက်တမ်အရေအတွက်ပဲ ဖြစ်တယ်။

range ဖန်ရှင်နှင့် for loop

သူညကနေ တစ်ဆယ်ထိ အစဉ်အတိုင်း ရေတွက်ချင်ရင် နည်းလမ်းတစ်ခုက

```
for n in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    print(n)
```

ဒီလိုသာ ကဏ္န်းတစ်လုံးချင်း ရိုက်ထည့်ရရင် အဆင်မပြောဘူး။ ပိုကောင်းတဲ့ နည်းလမ်း ရှိရမှာပါ။ range ဖန်ရှင်ဟာ ဒီလိုနေရာမျိုးအတွက် အသင့်တော်ဆုံးပါပဲ။ range(0,11) က သူညကနေ တစ်ဆယ်ထိ အစဉ်အတိုင်း ထုတ်ပေးမဲ့ range အော့ဘ်ဂျက်ကို ပြန်ပေးတယ်။ ဒီကဏ္န်းတွေကို တစ်ခါတည်း ကြိုးထုတ်ထားတာ မဟုတ်ပါဘူး။ လိုအပ်မှ တစ်ခုချင်း ထုတ်ပေးတာပါ။ (ဒီအတွက်ကြောင့် range(0, 11) နဲ့ range(0, 1_000_000) နှစ်ခုလုံး မမိမ့်ရှိသုံးစွာဘာအရ သိပ်မကွာခြားပါဘူး)။

```
for i in range(0, 11):
    print(i)
```

2, 4, 6, ..., 12 စုံကိန်းတွေ လိုချင်ရင် range(2, 13, 2)၊ 5, 8, 11, ..., 98 လိုချင်ရင် range(5, 99, 3) စသည်ဖြင့် သုံးခုထည်ပြီး ဖန်ရှင်ခေါ်ပါမယ်။ အစ အဆုံးနဲ့ နောက်ခုံးတစ်ခုကတော့ ကိုန်းတန်းမှာပါတဲ့ ကပ်လျက်ကိန်းနှစ်ခုလဲ ကွာခြားချက်ပါ။

```
for i in range(2, 13, 2):
    print(i)

for i in range(5, 99, 3):
    print(i)
```

အစဂဏန်း မသတ်မှတ်ပေးရင် သူညလို့ ယူဆတယ်။ ဒါကြောင့် သူညကနေ တစ်ဆယ်ထိကို range(11) နဲ့ ယူလို့ရတယ်။ ကွာခြားချက်က အနှစ်ကိန်း ဖြစ်လို့ရတယ်

```

for i in range(10, 0, -1):
    print(i)

for i in range(0, -11, -2):
    print(i)

```

မှတ်ချက်။ ။ ကွားချက် တစ်မဟုတ်ရင် အစ အဆုံး ကွားချက် သုံးခုလုံး လိုပါမယ်။

for loop အသုံးချုပ်များ

စတိတ်မန်တစ်စုံကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ပြန်ကျောစို့ for loop ကို အသုံးပြုနိုင်ပါတယ်။ ကိုးဘုံးကနေ ထည့်ပေးတဲ့ ကဏ္ဍး ဆယ်လုံးကို ပေါင်းမယ်ဆိုပါစို့။

```

tot = 0
for i in range(10):
    val = float(input("?" ))
    tot += val

print(f"Total: {tot}")

```

ပရိုဂရမ် run တဲ့ အချိန်ကျတော့မှ အကြိမ်အရေအတွက် သတ်မှတ်လိုလည်း ရတယ်။ ဥပမာ

```

cnt = input("How many numbers you want to add? ")
tot = 0
for i in range(cnt):
    val = float(input("?" ))
    tot += val

print(f"Total: {tot}")

```

Loop တစ်ကျော်း တစ်ကျော် ဖေရီရောဘဲလ် တန်ဖိုးတွေ ဘယ်လောက်ဖြစ်နေလဲ အခုလို လိုက် ကြည့်ရှုနိုင်ပါတယ်။

```

cnt = input("How many numbers you want to add? ") # 4 ထည့်တယ် ယူဆပါ
tot = 0

1st iter:
i = 0
val = float(input("?" )) # 2 ထည့်တယ် ယူဆပါ
tot += val # 2 (လက်ရှိ tot တန်ဖိုး)

2nd iter:
i = 1
val = float(input("?" )) # 3 ထည့်တယ် ယူဆပါ
tot += val # 5 (လက်ရှိ tot တန်ဖိုး)

3rd iter:
i = 2

```

```

val = float(input(" ? "))      # 4 ထည့်တယ် ယူဆပါ
tot += val                      # 9 (လက်ရှိ tot တန်ဖိုး)

4th iter:
i = 3
val = float(input(" ? "))      # 11 ထည့်တယ် ယူဆပါ
tot += val                      # 20 (လက်ရှိ tot တန်ဖိုး)

print(f"Total: {tot}")          # 20 ထုတ်ပေးမှာပါ

```

အောက်ပါ nested list ထဲမှ list တစ်ခုစီ ပေါင်းလဒ်နဲ့ list အားလုံး စုစုပေါင်း (grand total) ထုတ်ပေးပါမယ်။

```

rows = [[1, 3, 5, 2],
        [2, 9, 3, 7],
        [4, 4, 8, 3],
        [6, 2, 7, 9]]

# File: sum_of_rows.py
grand_tot = 0
for row in rows:
    row_tot = 0
    for val in row:
        row_tot += val
    print('Row total: ' + str(row_tot))
    grand_tot += row_tot

print('Grand total: ' + str(grand_tot))

```

Nested `for` loop သုံးထားတယ်။ ပေါင်းလဒ်ကို ထည့်ထားဖို့ ဖေရီရောဘဲလ် နှစ်ခု သုံးထားတာ သတိထားကြည့်ပါ။ ဒီနေရာမှာ ဖေရီရောဘဲလ် စကုပ် (scope) သဘောတာရားကို နားလည့်ဖို့ လိုအပ်လာပါတယ်။ ဖေရီရောဘဲလ်တစ်ခုကို ဘယ်နေရာကနေ သုံးလိုရလဲဆိုတာဟာ ငြင်းဖေရီရောဘဲလ်ရဲ့ စကုပ်နဲ့ သက်ဆိုင်ပါတယ်။ `grand_tot` ဟာ top level ဖေရီရောဘဲလ် ဖြစ်တယ်။ ငြင်းကို ကြော်လွှာတဲ့ နေရာကစပြီး အောက်ပိုင်းတလျှောက်လုံး သုံးလိုရတယ်။ `grand_tot` ရဲ့ စကုပ်ဟာ ငြင်းကို ကြော်ထားတဲ့ ဖိုင်အဆုံးထိ ဖြစ်တယ်။ `rot_tot` ကတော့ block level ဖေရီရောဘဲလ်ပါ။ Block level ဖေရီရောဘဲလ်ကိုတော့ ငြင်းကို ကြော်ထားတဲ့ ဘလောက်အတွင်းမှာပဲ သုံးလိုရပါမယ်။ Block level ဖေရီရောဘဲလ်တစ်ခုရဲ့ စကုပ်ဟာ ငြင်းကို ကြော်ထားတဲ့ ဘလောက် အဆုံးထိ ဖြစ်တယ်။

`row` တစ်ခုချင်း ပေါင်းလဒ်ကို `grand_tot` မှာ ပေါင်းထည့်ပေးဖို့ လိုတယ်။ အောက်ခုံးမှာလည်း `grand_tot` ကို `print` ထုတ်ပေးရမယ်။ အကယ်၍ block level မှာ ထားလိုက်ရင် အောက်ဆုံးမှာ သုံးလိုရမှာ မဟုတ်တော့ဘူး။ `row` တစ်ခု ပေါင်းပြီးရင် `rot_tot` ကို ထုတ်ပေးဖို့ လိုတယ်။ ဒါကြောင့် ငြင်းကို အပြင်ဘလောက်မှာ ကြော်ရမယ်။ အတွင်းဘလောက်မှာဆုံးရင် အပြင်ကနေ သုံးလိုရမှာ မဟုတ်တော့ဘူး။ (အတွင်း `for` loop ဟာ အပြင် `for` loop ရဲ့ ဘလောက် အတွင်းမှာ ပါဝင်တဲ့အတွက် `rot_tot` ကို သုံးလိုရပါတယ်)။

Loop တစ်ကျော်ပြီး တစ်ကျော် ဖေရီရောဘဲလ် တန်ဖိုးတွေ ပြောင်းလဲသွားတာကို ပြထားတယ်။ တစ်ဆင့်ချင်း ဂရုစိုက်ပြီး လိုက်ကြည့်ပါ။ (အကြိမ်အရေအတွက် သိပ်မများအောင် အိုက်တမ် နည်းတဲ့

list နဲ့ ဥပမာ ပြထားတာပါ။

```
# ဒဲ list ဆဲ ကေနဲတွေ ပေါင်မှာ။
rows = [[1, 3, 5],
         [2, 4, 6]]
```

```
grand_tot = 0
```

အပြင်း for loop

1st iter:

```
row = [1, 3, 5]
```

```
row_tot = 0
```

အပောင်း for loop

1st iter:

```
val = 1
```

```
row_tot += val      # 1
```

2nd iter:

```
val = 3
```

```
row_tot += val      # 4
```

3rd iter:

```
val = 5
```

```
row_tot += val      # 9
```

```
print('Row total: ' + str(row_tot))
```

```
grand_tot += row_total # 9
```

အပြင်းလောက် တစ်ကျော်ပြီး row_tot စုံပါ။

အပြင်း for loop

2nd iter:

```
row = [2, 4, 6]
```

row_tot = 0 # ပေါင်မှာ တစ်ခါတယ်ကြိုးတယ်

အပောင်း for loop

1st iter:

```
val = 2
```

```
row_tot += val      # 2
```

2nd iter:

```
val = 4
```

```
row_tot += val      # 6
```

3rd iter:

```
val = 6
```

```

row_tot += val           # 12

print('Row total: ' + str(row_tot))
grand_tot += row_total # 21
# အပြင်ဘလေက နောက်တစ်ကျော်မီး row_tot စုပ် အဆုံး

print('Grand total: ' + str(grand_tot))

ခရစ်စမတ် သစ်ပင်လေး ဆွဲကြည့်ရအောင်။ တစ်တန်းမှာ စပေါ်ဘယ်နှစ်ခုပါလဲ ကြည့်ရလွယ်အောင်
မ လေးတွေနဲ့ ပြထားတယ်။

# File: christmas_tree.py
LEAF_ROWS = 8
TRUNK_ROWS = 3
# the width of the trunk
TRUNK_SZ = 3
# formula: LEAF_ROWS - 2
SPC_FOR_TRUNK = 6

for r in range(LEAF_ROWS):
    for i in range(LEAF_ROWS - 1 - r):
        print(' ', end='')
    for i in range(r * 2 + 1):
        print('*', end='')
    print()

for r in range(TRUNK_ROWS):
    for i in range(SPC_FOR_TRUNK):
        print(' ', end='')
    for i in range(TRUNK_SZ):
        print('*', end='')
    print()

```

```

***** *
***** ***
***** *****
***** *****
***** *****
***** *****
***** *****
***** *****
***** ***
***** ***
***** ***

```

အပေါ်ပိုင်း ဗြိုင်မှာ အတန်း ရှစ်ခု (LEAF_ROWS)၊ ပင်စည်မှာ သုံးခု (TRUNK_ROWS) သတ်မှတ်

ထားတယ်။ ပင်စည် အကျယ် (TRUNK_SZ) ကိုလည်း * သုံးခု ထားတယ်။ အပေါ်ဆုံးကို row နံပါတ် သုညာ ဒုတိယကို တစ် စသည်ဖြင့် ယူဆပါမယ်။ row နံပါတ်စဉ်ကို ၁ မေရီရောင်းက ဖော်ပြတယ်။ တိုက်ပုံမှာ စပေါ်အရေအတွက်နဲ့ row နံပါတ်စဉ်ဟာ (LEAF_ROWS - 1 - ၁) ဖော်မြှုလာနဲ့ ဆက်စပ် နေတယ်။ * အရေအတွက်ကတော့ နှစ်ခုစီတိုးသွားပြီး (၁ * ၂ + ၁) ဖြစ်တယ်။ ပင်စည်ပိုင်း စပေါ် အရေအတွက်ကတော့ တစ်တန်း ခြောက်ခုပါ (LEAF_ROWS - ၂) ။

၄.၃ Python Arcade Library

Python Arcade (ဝက်ဘ်ဆိုင် <https://api.arcade.academy>) ဟာ အရည်အသွေးကောင်းတဲ့ ထိပ်ဆုံး Python ဂိမ်းလိုက်ဘရှိတွေထဲက တစ်ခုဖြစ်တယ်။ Arcade အပြင် အသုံးများတဲ့ အခြားတစ်ခုက pygame (ဝက်ဘ်ဆိုင် <https://www.pygame.org>) ပါ။ ဒီစာအပ်မှာ Arcade ကို သုံးပါ မယ်။ Arcade ပိုကောင်းတယ်လို မဆိုလိုပါဘူး။ ဘိုင်နာတွေအတွက် ပိုသင့်တော်မယ် ယူဆတဲ့အတွက် အသုံးပြုတာပါ။ အောက်ပါအတိုင်း အင်စတောလ်လုပ်နိုင်ပါတယ်

```
| pip install arcade
```

Arcade နဲ့ ပုံစွဲဖို့အတွက် ကရပ်ဖစ် ဝင်းဒီးတစ်ခုကို အောက်ပါအတိုင်း ယူရပါတယ်။ Run လိုက် ရင် ပုံ (၇.၁) မှာ တွေ့ရတဲ့ နောက်ခံရောင်နဲ့ ဝင်းဒီးအလွတ် တစ်ခု တက်လာမှာပါ။

```
# File: arcade_starter.py
import arcade

arcade.open_window(300, 200, "Arcade Starter")

# Set the background color
arcade.set_background_color(arcade.color.PINK_PEARL)

# Get ready to draw
arcade.start_render()

# Finish drawing
arcade.finish_render()

# Keep the window up until someone closes it.
arcade.run()
```

အပေါ်ဆုံးမှာ arcade လိုက်ဘရှိ အင်ပိုလုပ်ထားတာပါ။ ရှေ့ပိုင်းမှာသုံးတဲ့ from lib import * ပုံစွဲ ကွားကြားတာက အခုနည်းနဲ့ အင်ပိုလုပ်ထားရင် လိုက်ဘရှိမှာ ပါတဲ့ အစိတ်အပိုင်းတွေကို ဖော်ထဲ အမှတ်အသားနဲ့ အသုံးပြုရပါမယ်။ ဥပမာ open_window ဖန်ရှင်ကို

```
| arcade.open_window(arguments)
```

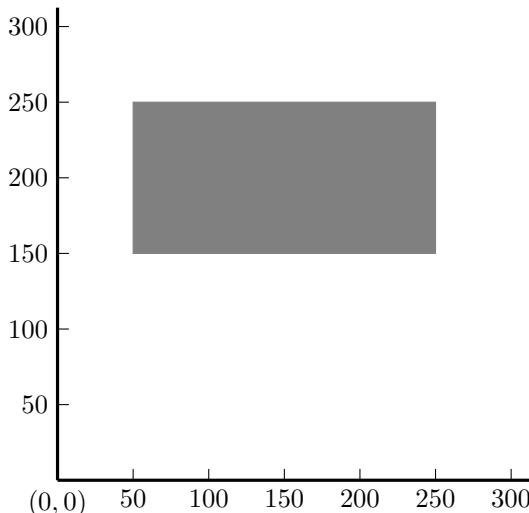
လိုက်ဘရှိ နံမည်နောက်မှာ (.) အမှတ်အသား ခံပြီး ခေါ်ရှုမှာပါ။ ဒီဖန်ရှင်မှာ လိုချင်တဲ့ဝင်းဒီး အကျယ် အမြင့်၊ ခေါင်းစည်းစာသား ထည့်ပေးထားတယ်။



ပုံ ၃.၁

ကိုဖြေခြင်းနှင့် ဝင်းဒီးပေါ်တွင် တည်နေရာ သတ်မှတ်ခြင်း

ဝင်းဒီး အောက် ဘယ်ဘက်ထောင့်စွန်းကို origin အမှတ် ($x = 0, y = 0$) အဖြစ် သတ်မှတ်ပါ တယ်။ ညာဘက်သွားရင် x တန်ဖိုး များလာမယ်။ အပေါ်ဘက်တက်ရင် y တန်ဖိုး တိုးလာမှာပါ။ အခြား ကွန်ပျူးတာ ကရပ်ဖော်စနစ်တွေမှာလိုပဲ x, y တန်ဖိုးကို ယူနစ်အားဖြင့် pixel နဲ့ ဖော်ပြတယ်။ ပုံ (၃.၂) မှာ rectangle ဘယ်ဘက်အောက် ထောင့်စွန်း (x, y) တန်ဖိုး (50, 150), ညာဘက် အပေါ်ထောင့်ဟာ (250, 250) ဖြစ်တယ်။



ပုံ ၃.၂

အောက်ပါ ဖန်ရှင်ကတော့ ဝင်းဒီးရဲ့ နောက်ခံရောင် သတ်မှတ်တာပါ။ လိုက်ဘရီရဲ့ color မော်ချိုး (module) မှာ အရောင်တန်ဖိုးတွေ အဆင်သင့် သတ်မှတ်ပေးထားတယ်။ (မော်ချိုးဆိုတာ လိုက်ဘရီရဲ့ အစိတ်အပိုင်းတစ်ခုလို့ အကြောင်းဖျက်း ယူဆနိုင်တယ်)။

```
arcade.set_background_color(arcade.color.PINK_PEARL)
```

အခုလို့ အင်ပို့လုပ်ထားရင် အရောင်တွေ သုံးရတာ ပို့အဆင်ပြေတယ်

```
import arcade
from arcade.color import *
```

```

...
arcade.set_background_color(PINK_PEARL)

PINK_PEARL, RED စသည်ဖြင့် အရောင်နံမည် တမ်းရေးလိုဂုဏ်။ ရှေ့မှာ arcade.color. ထည့်စိုး
မလိုတော့ဘူး။

ပုံဆွဲဖို့ အဆင်သင့်ဖြစ်အောင် start_render ခေါ်ပေးရတယ်။ ဆွဲပြီးရင်လည်း finish_render
ခေါ်ဖို့လိုတယ်။ ပုံဆွဲတဲ့ကိစ္စကို ငှုံးတိန္တစ်ခုကြားမှာ လုပ်ရမှာပါ။

arcade.start_render()
# call drawing functions here
...
...
arcade.finish_render()

arcade.run()

```

ဝင်းဒီးကို မပိတ်မချင်း ပေါ်နေအောင် run ဖန်ရှင် ခေါ်ပေးရတယ်။ မခေါ်ထားဘဲ ပရိုဂရမ်ကို run ရင် ဝင်းဒီးပွင့်လာပြီး ဖျက်ခနဲ့ပြန်ပိတ်သွားမှာပါ။ မကျွန်ခဲ့ဖို့ သတိပြုရပါမယ်။

Arcade မှာ ပါတဲ့ အခြေခံ ပုံဆွဲဖန်ရှင် တချို့ကို ဆက်ကြည့်ရအောင်။ ထောင့်မှန်စတုဂံဆွဲတဲ့ ဖန်ရှင်တွေထဲက နှစ်ခု သုံးပြထားတယ်။ နှစ်ခုလုံးက ထောင့်မှန်စတုဂံရဲ့ ဘယ်ဘက်အောက် ထောင့်စွန်းနဲ့ တည်နေရာကို သတ်မှတ်ပြီး အရွယ်အစားကို အကျယ်၊ အမြင့်နဲ့ သတ်မှတ်ပေးရတယ်။ draw_xywh_rectangle_filled က အတွင်းပိုင်း အရောင်နဲ့ ဆွဲပေးတယ်။ အနားတွေကိုပဲ ဆွဲချင်ရင် draw_xywh_rectangle_outline ဖန်ရှင်သုံးရပါမယ်။

```

# File: arcade_drawing.py
import arcade
from arcade.color import *

arcade.open_window(300, 200, "Drawing Example")

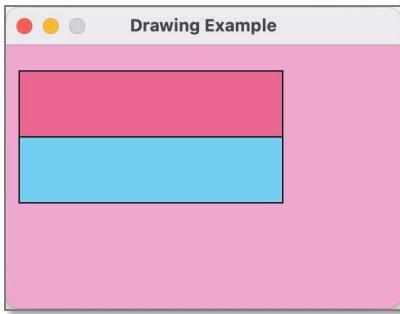
# Set the background color
arcade.set_background_color(PINK_PEARL)

# Get ready to draw
arcade.start_render()

# Lower rectangle (blue)
# Lower left corner (x, y) = (10, 80)
# width = 200, height = 50
arcade.draw_xywh_rectangle_filled(10, 80, 200, 50, BABY_BLUE)
arcade.draw_xywh_rectangle_outline(10, 80, 200, 50, BLACK)

# Upper rectangle (pink)
# Lower left corner (x, y) = (10, 130)
arcade.draw_xywh_rectangle_filled(10, 130, 200, 50, PALE_VIOLET_RED)
arcade.draw_xywh_rectangle_outline(10, 130, 200, 50, BLACK)

```



ပုံ ၃၃

```
# Finish drawing
arcade.finish_render()

# Keep the window up until someone closes it.
arcade.run()
```

အပေါ် ထောင့်မှန်စတုဂံပုံကို ဒီနှစ်ခုနဲ့

```
arcade.draw_xywh_rectangle_filled(10, 130, 200, 50, PALE_VIOLET_RED)
arcade.draw_xywh_rectangle_outline(10, 130, 200, 50, BLACK)
```

ဆွဲထားတာပါ 〔ပုံ (၇.၃)〕။ ပါရှာမိတာတွေက $x, y, width, height, color$ အစဉ်အတိုင်းပဲ။ အောက်က ထောင့်မှန်စတုဂံကို အခုလို

```
arcade.draw_xywh_rectangle_filled(10, 80, 200, 50, BABY_BLUE)
arcade.draw_xywh_rectangle_outline(10, 80, 200, 50, BLACK)
```

ဆွဲထားတာပါ။ ထောင့်မှန်စတုဂံ နှစ်ခုလုံး အကျယ် (၂၀၀)၊ အမြင့် (၅၀) pixels ရှိတယ်။

ကျားကွဲက်ခံ (သို့) စစ်တုရင်ခံ ဆွဲခြင်း

ကျားကွဲက် (သို့) စစ်တုရင်ခံ ပုံဖော်တဲ့ checkerboard ဥပမာကို အောက်မှာလေလာကြည်ပါ။ ပုံ (၇.၄)

```
# File: arcade_checkerboard.py
import arcade
from arcade.color import *

WIN_WIDTH = 600
WIN_HEIGHT = 420
arcade.open_window(WIN_WIDTH, WIN_HEIGHT, "Arcade Checkerboard")

arcade.set_background_color(WHITE_SMOKE)

arcade.start_render()
```

```

# width/height of the whole board
BOARD_SIZE = 400
COLS = 8
ROWS = 8
# size of each square of the board
SQ_SIZE = BOARD_SIZE // ROWS
# x of left side of the board
X_LFT = (WIN_WIDTH - BOARD_SIZE) // 2
# y of the bottom side of the board
Y_BTM = (WIN_HEIGHT - BOARD_SIZE) // 2

y = Y_BTM
for i in range(ROWS):
    # start x from the left again
    x = X_LFT
    for j in range(COLS):
        # if sum of row and col numbers is even
        if (i + j) % 2 == 0:
            arcade.draw_xywh_rectangle_filled(x, y, SQ_SIZE, SQ_SIZE,
                                              WOOD_BROWN)
        else:
            arcade.draw_xywh_rectangle_filled(x, y, SQ_SIZE, SQ_SIZE,
                                              BLACK)
        # move to right
        x += SQ_SIZE

    # update y after each row
    y += SQ_SIZE

arcade.finish_render()

arcade.run()

```

ပထမပိုင်းမှာ လိုအပ်တဲ့ constants တွေ သတ်မှတ်ထားတယ်။ ကျားကွက်တစ်ခုလုံး အရွယ်အစား ကို (၄၀၀) pixels ထားတယ်။ စတုရန်းပုံဆိတော့ အကျယ်/အမြင့် နှစ်ခုလုံး (၄၀၀)။ X_LFT က row တစ်ခုချင်းရဲ့ စုတ် x တန်ဖိုး၊ Y_BTM ကတော့ အောက်ဆုံး row y စုတ် တန်ဖိုး ဖြစ်ပါတယ်။

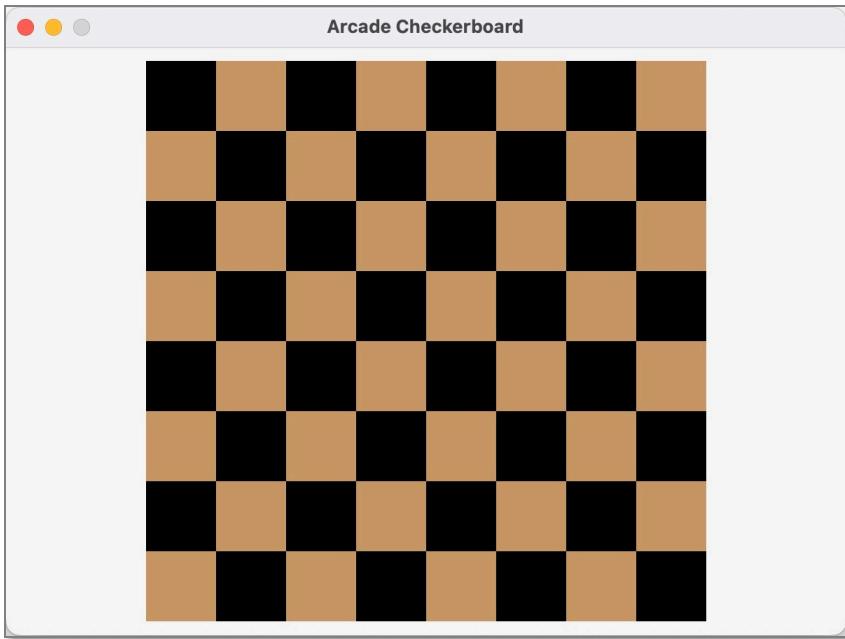
အတွင်း for loop က တစ်တန်းမှုပါတဲ့ ကျားကွက် (၈) ခုကို ဘယ်ကနေညာ တစ်ကွက်ချင်း ဆဲ သွားမှာဖြစ်ပြီး အပြင် for loop ကတော့ row တစ်ခုပြီး တစ်ခု အောက်ဆုံးကစပြီး ပုံဖော်ပေးမှုပါ။ Row တစ်တန်းမှာ ပါဝင်တဲ့ အကျက်တွေအားလုံး y တန်ဖိုး တူတယ်။ Row တစ်ခုပြီး နောက်တစ်ခု ကူးတဲ့အခါ လက်ရှိ row ရဲ့ y တန်ဖိုးကို ကျားကွက်တစ်ကွက်စာ ပေါင်းပေးရပါမယ်

```

y += SQ_SIZE

```

Row တစ်ခုပြီးမှ y တန်ဖိုးကို ပေါင်းပေးရမှာပါ။ ဒါကြောင့် ဒီစတိတ်မန့်က အတွင်း for loop အပြီးမှာ ဖြစ်ရပါမယ် (Indentation ဂရိစိက်ကြည့်ပါ)။ Row တစ်ခုအတွက် ကျားကွက်တစ်ကွက်ချင်းကို ဘယ်



ပုံ ၃.၄

ဘက်ကနေစပီး ဆွဲမှာပါ။ တစ်ကွဲက်ပြီး နောက်တစ်ကွဲက် ကူးတဲ့အခါ လက်ရှိ x တန်ဖိုးကို ကျားကွဲက် တစ်ကွဲက်စာ ပေါင်းပေးရပါမယ်။ ဒီအတွက်

```
| x += SQ_SIZE
```

ကို အတွင်း for loop တစ်ကျော့ပြီးတိုင်း လုပ်ဖို့လိုတယ်။ Row တစ်ခု ပြီးသွားပြီဆိုရင် နောက် row တစ်ခုက ဘယ်ဘက်ကနေ ပြန်စရုပါ။ ဒီတော့ အတွင်း for loop မစခင်မှာ

```
| x = X_LFT
```

ပြန် လုပ်ပေးရပါမယ်။

ကျားကွဲက်ပတ်တန် တစ်လျှည့်စီ အရောင်ခြယ်ဖို့ ဘယ်လိုလုပ်ရမလဲ။ တစ်ကွဲက်ချင်းရဲ့ row နံပါတ် နဲ့ column နံပါတ်ကို ပေါင်းကြည့်ပါ။ ပေါင်းလာ် စုံကေန်း ဖြစ်တဲ့ အကွဲက်တွေ အရောင်တူတယ်။ မ ဂောက်နဲ့ဖြစ်တဲ့ အကွဲက်အားလုံး အရောင်တူတယ်။ အခုလို

```
if (i + j) % 2 == 0:
    arcade.draw_xywh_rectangle_filled(x, y, SQ_SIZE, SQ_SIZE,
                                      WOOD_BROWN)
else:
    arcade.draw_xywh_rectangle_filled(x, y, SQ_SIZE, SQ_SIZE,
                                      BLACK)
```

အရောင် တလူည့်စီခြယ်တယ်။ ဒီလောက်ဆိုရင် ကျားကွဲက်ပရှိကရမ်ကို နားလည်မယ် ထင်ပါတယ်။

၃.၄ while loop

while loop သာ indefinite loop ဖြစ်ပါတယ်။ Loop ကို စတင် လုပ်အောင်တဲ့အချိန်မှာ ဘယ်နှစ်

ကြိမ် ပြန်ကျောမလဲ အတိအကျ ကြိုး ‘မသိ’ ရင် indefinite loop လို့ သတ်မှတ်တယ်။ တစ်နှုန်း တစ်ဆယ်ကြား (အပါအဝင်) ကိန်းပြည့်တစ်လုံးကို ကျပောင်း (random) ထုတ်ထားမယ်။ မမှန်မချင်း အဲဒိုက်နှုန်းကို ခန့်မှန်းပေးရမယ်။ ခန့်မှန်းတာ မှန်သွားရင် ဘယ်နှစ်ခါ ခန့်မှန်းရလဲနဲ့ မှန်တဲ့ကောင်းကို ပြပေးတဲ့ ပရီဂရမ်မှာ while loop အသုံးပြု ရေးထားတာ တွေ့ရပါမယ်။

```
from random import *

num = randint(1, 10)

guess = int(input('? '))
times = 1

while guess != num:
    guess = int(input('? '))
    times += 1

print(f'You get correctly after {times} guesses.')
print(f'The number is {num}.')
```

guess != num (ဘူလိုယန် အိပ်စ်ပရက်ရှင်) မဟားမချင်း loop က ပြန်ကျောပေးနေမှာပါ။ မူးတဲ့ အခါ ထပ်မကျော့တော့ဘဲ ရပ်သွားမှုဖြစ်တယ်။ ကျပောင်းကောင်းက ခုနှစ်ကျေတယ် ဆိုပါစို့။ တစ်၊ ကိုး၊ သုံး၊ တစ်ဆယ်နဲ့ ခုနှစ် တို့ကို အစဉ်အတိုင်း ခန့်မှန်း ထည့်ပေးမယ်ဆိုရင် တစ်ကျောချင်း လိုက်ကြည့်တဲ့အခါ အခုလို တွေ့ရမှာပါ။

```
num = randint(1, 10)                      # 7 ကျတယ် ယူဆပါ

guess = int(input('? '))                   # 1 ထည့်ပေးတယ်
times = 1

guess != num:                                # 1 != 7 က True
    1st iter:
        guess = int(input('? '))              # 9 ထည့်ပေးတယ် ယူဆပါ
        times += 1                            # 2 ဖြစ်သွားမယ်

guess != num:                                # 9 != 7 က True
    2nd iter:
        guess = int(input('? '))              # 3 ထည့်ပေးတယ် ယူဆပါ
        times += 1                            # 3 ဖြစ်သွားမယ်

guess != num:                                # 10 != 7 က True
    3rd iter:
        guess = int(input('? '))              # 10 ထည့်ပေးတယ် ယူဆပါ
        times += 1                            # 4 ဖြစ်သွားမယ်

guess != num:                                # 7 != 7 က False ဖြစ်သွားပြီ
    # ထပ်မကျော့တော့ဘူး

# loop အောက်က စတိတ်မန်တွေ ဆက်လုပ်ပါတယ်
```

```

print(f'You get correctly after {times} guesses.')
print(f'The number is {num}.')

```

Loop က ထပ်မကျော်တော့ဘဲ ရပ်သွားတာကို *loop exits* ဖြစ်သွားတယ်လို့ ပြောတယ်။ မြန်မာ လိုတော့ loop ကနေ ထွက်သွားတယ်ပေါ့။ အကယ်၍ စောဘောက ဥပမာမှာ ပထမဆုံးတစ်ခါမှာပဲ ခုနှစ် ထည့်လိုက်ရင်တော့ ဘလောက်ကို တစ်ခါမှ မလုပ်ဆောင်ဘဲ loop က ချက်ချင်းထွက်သွားမှာပါ။

Sentinel Controlled Loop

ကဏ္နားတွေ ဘယ်နှစ်လုံး ပေါင်းမှာလဲ ကြိုးသိရင် `for` loop နဲ့ အလုပ်ဖြစ်တယ်။ ကဏ္နား ဘယ်နှစ်လုံးရှိ လဲ ကြိုးသိထားဘဲ ရှိသလောက် တစ်လုံးချင်းရှိက်ထည့်ပြီး အားလုံးပေါင်းချင်တော့ `for` loop နဲ့ အဆင်မပြောဘူး (မရနိုင်ဘူးလို့ မဆိုလိုပါ မရမက လုပ်တဲ့နည်းတွေလည်း ရှိပါတယ်)။ ဒီလိုအခြေအနေ မျိုးမှ *sentinel controlled loop* ကို သုံးပါတယ်။

Sentinel controlled loop မှာ loop ကနေ ထွက်ဖို့အတွက် အသုံးပြုတဲ့ သီးသန့်တန်ဖိုးတစ်ခု ရှိရပါမယ်။ ဒီတန်ဖိုးကို *sentinel value* လို့ ခေါ်ဟယ်။ ကဏ္နားတွေပေါင်းတဲ့ ကိစ္စအတွက် *sentinel value* ရွေးချယ်သတဲ့မှတ်တဲ့အခါ ပေါင်းရမှုကဏ္နား မဖြစ်နိုင်တဲ့ တစ်ခုကို ရွေးရပါမယ်။ သူညာဟာ အပေါင်းထပ်တူရ ဂုဏ်သွေးရှိတဲ့အတွက် ငှင်းကို *sentinel value* အဖြစ် ထားနိုင်တယ်။

```

# File: add_nums_sentinel.py
SENTINEL = 0
total = 0

val = int(input('? '))
while val != SENTINEL:
    total += val
    val = int(input('? '))

print(f'Total is {total}')

```

ထည့်ပေးတဲ့ ကဏ္နားဟာ သူညာမဟုတ်မချင်း `total` မှာ ပေါင်းထည့်ပြီး နောက်ကဏ္နားတစ်ခု ထည့်ပေးဖို့ `input` တောင်းမှာပါ။ Sentinel တန်ဖိုး သူညာ ထည့်လိုက်ရင် loop က ထွက်သွားမယ်။ ပြီးတဲ့အခါ `total` ကို ပြေားမှာပါ။ စစ်ချင်းပဲ သူညာ ထည့်လိုက်ရင် loop က တစ်ခါမှ မကျော်ဘဲ ထွက်သွားမယ်။ `total` လည်း သူညာပဲ ထွက်ပါမယ်။ အခုပ်ရှိကရမ်ကို နောက်တစ်နည်း ရေးလို့ရပါတယ်။

```

# File: add_nums_sentinel2.py
SENTINEL = 0
total = 0

while True:
    val = int(input('? '))
    if val == SENTINEL:
        break
    total += val

print(f'Total is {total}')

```

while True: က ပုံမှန်ဆိုရင် infinite loop ဖြစ်ပါလိမ့်မယ်။ အမြဲမှန်နေတဲ့အတွက် ပြန်ကျော်တာ ဘယ်တော့မှ ရပ်မှာ မဟုတ်ဘူး။ ဒါလို့ဖြစ်နေတာကို ဖြတ်ချေပစ်ဖို့အတွက် break စတိတ်မန့်ကို သုံးထားပါတယ်။ if val == SENTINEL: (ထည့်ပေးတဲ့ တန်ဖိုးက သုညဆိုရင်) လက်ရှိ အလုပ်လုပ်နေတဲ့ loop ကို break လိုက်မှာဖြစ်တယ်။

အထက်ပါ sentinel loop ပုံစံနှစ်ခုကို ယူဉ်ကြည့်ရင် ပထမတစ်ခုမှာ input တန်ဖိုးထည့်ခိုင်းတာကို loop မစမိ တစ်ခါ ကြိုလုပ်ထားရတယ်။ နောက်တစ်ခုမှာတော့ အဲဒီလို့ ကြိုလုပ်ထားစရာမလိုဘူး။ Loop ဘလောက်ထဲက စတိတ်မန့်တစ်ခုကို အပြင်မှာထုတ်ထားရတဲ့အတွက် တချို့က ပထမပုံစံထက်ဒုတိယပုံစံက ကုဒ်စတိုင်လအားဖြင့် ပုံပြီး သပ်ရပ်ကြောရင်းတယ်လို့ ယူဆကြပါတယ်။

Result Controlled Loop

တစ်နှစ်ပါးရာခိုင်နှုန်း အတိုးနဲ့ ဘဏ်မှာ ဒေါ်လာတစ်ထောင် စုတေားတယ်။ တစ်နှစ်ပြည့်တိုင်း အတိုးအရင်းပေါင်းပြီး နောက်နှစ်အတွက် အတိုးတွေက်တယ်။ နှစ်ထပ်တိုး (yearly compound interest) လိုခေါ်ပါတယ်။ ဒီနည်းလမ်းနဲ့ အနည်းဆုံး ဒေါ်လာတစ်သန်း စုမ္ပါန်း (မီလျှောက်ဖြစ်ဖို့) ဘယ်နှစ်နှစ်တောင့်ရမလဲ။ သုံးနှစ်ပြည့်ရင် စုမ္ပါမဲ့ ငွေပမာဏ တွေက်ထားတာကို ကြည့်ပါ။

Year	Interest	Balance
1	$1000 \times 0.05 = 50$	$1000 + 50 = 1050$
2	$1050 \times 0.05 = 52.5$	$1050 + 52.5 = 1102.5$
3	$1102.5 \times 0.05 = 55.125$	$1102.5 + 55.125 = 1157.625$

တေဘဲလ ၃.၂ ဆုံးနှစ်စာ နှစ်ထပ်တိုး

ဒေါ်လာတစ်သန်း အနည်းဆုံးရဖို့ နှစ်ဘယ်လောက်ကြာမလဲ အခုလို ရှာနိုင်ပါတယ်

```
# File: one_million.py
balance = 1000
INT_RATE = 0.05
TARGET = 1_000_000
yr = 0

print(f"{'yr':>4s} {'interest':>10s} {'balance':>10s}")
while balance < TARGET:
    interest = balance * INT_RATE
    balance += interest
    yr += 1
    print(f'{yr:4d} {interest:10.2f} {balance:10.2f}')

print(f'You have to wait {yr} years!')
```

while loop ထဲမှာ တစ်နှစ်ကုန်တဲ့အခါ ရမဲ့ အတိုးနဲ့ အတိုးအရင်းပေါင်း တွေက်ထားပြီး yr ကိုလည်း တစ်နှစ် တိုးပေးတယ်။ yr, interest, balance ထုတ်ပြေပေးတယ် (မပြုလည်း ပြသောမရှိပါ။ တစ်နှစ် စာ တွေက်ထားတာ စစ်ကြည့်လို့ရအောင် ထုတ်ကြည့်တာပါ)။ တစ်သန်းမပြည့်မချင်း ပြန်ကျော်ပေးအောင် loop ကွန်ဒီရှင်ကို balance < TARGET နဲ့ စစ်ထားတယ်။ တစ်သန်းနဲ့ညီရှင် (သို့) တစ်သန်းကျော်သွားတာနဲ့ ကွန်ဒီရှင် False ဖြစ်သွားပြီး ထပ်မကျော်တော့ဘူး။ Loop တစ်ခါကျော့တိုင်း တစ်နှစ်စာ အတိုး

အရင်းပေါင်း balance ကို တွက်ချက်ပြီး loop ကနေ ဘယ်အချိန် ထွက်မလဲကလည်း အဲဒီရလဒ်အပေါ် မူတည်နေတာ တော့ရပါမယ်။ ဒီလို သဘောတရားရှိတဲ့ loop မျိုးကို result controlled loop လို့ ခေါ်ပါတယ်။ ကျော့မဲ့ အကြိမ်အရေအတွက်က loop ထဲမှာ တွက်ချက်ထားတဲ့ ရလဒ်ပေါ် မူတည်တယ်။

ပရိုဂုရမ် ပေါ်ပေါ် ကို အခုလို ကော်လံတစ်ခုစိတိကို အကျယ် တစ်သမတ်တည်းဖြစ်၊ စာသားတော့ ညာဘက်ကပ်ပြီး ညီနေအောင် f-strings ကို format spec လို့ခေါ်တဲ့ ဖော့မတ်သတ်မှတ်တဲ့ နည်းစနစ် ကို သုံးထားတာပါ။

```
yr    interest    balance
1      50.00    1050.00
2      52.50    1102.50
...
142    48602.79  1020658.53
You have to wait 142 years!
```

Format spec နဲ့ ပါတ်သက်ပြီး အကျယ်တစ်ခု မဖော်ပြတော့ဘဲ အခုလို ဖော့မတ်ရအောင် ဘယ်လိုလုပ်ထားလိုကိုပဲ ရှင်းပြပါမယ်။

```
f'{ 'yr':>4s} {'interest':>10s} {'balance':>10s}'
```

ကော်လံ ခေါင်းစည်း အတွက် f-string ပါ။ ဖော့မတ်လုပ်ရမဲ့ တန်ဖိုး/ပေရှိရော့လုပ်/အိပ်စ်ပရက်ရှင်ကဲ့ (ကော်လံ) ဘယ်ဘက်မှာ ရှိမယ်။ ညာဘက်မှာ format spec ရှိမယ်။ >4s က 'yr' အတွက် format spec ဖြစ်တယ်။ > က ညာဘက်ကပ်ဖို့ 4 က ကော်လံအကျယ်ကို ကာရက်တာ လေးလုံးသတ်မှတ်တာ။ s ကတော့ တန်ဖိုးကို string အနေနဲ့ ပြပေးပါလို့ ဆိုလိုတယ်။ ကျိုးနဲ့ ကော်လံနှစ်ခု အတွက် format spec ကို >10s သတ်မှတ်တယ်။ ကာရက်တာ ဆယ်လုံး ကော်လံအကျယ် သတ်မှတ်တယ်။

```
f'{yr:4d} {interest:10.2f} {balance:10.2f}'
```

ဒါကတော့ yr, interest, balance တန်ဖိုးတွေကို ပြပေးဖိုပါ။ ကော်လံအကျယ် 4, 10, 10 သတ်မှတ်တယ်။ yr ကို ကိန်းပြည့်ကဏ်းအနေနဲ့ ပြပေးအောင် d သုံးတယ်။ ကျိုးနဲ့နှစ်ခုကို ဒသေမနဲ့ ပြဖို့ f သုံးတယ်။ .2 ကတော့ ဒသေမနောက် ကဏ်းနှစ်လုံး ပြပေးခိုင်းတာ။ ကိန်းကဏ်းဆိုရင် သူ့နှင့်အတိုင်း ညာဘက်ကပ်ပေးတဲ့အတွက် > ထည့်ဖို့ မလိုဘူး။ ဘယ်ဘက်ကပ်ချင်ရင် < သုံးလို့ရတယ်။

၃.၅ လေ့ကျင့်ရန် ဥပမာဏ်း

ကွန်ထရိုးလ် စထရက်ချာတွေ အသုံးချုတ်လာအောင် များများလေ့ကျင့်၊ များများစဉ်းစား၊ များများရေးကြည့် ရမှာပါ။ ဘယ်အတတ်ပညာမဆို များများလေ့ကျင့်မှ ကျွမ်းကျင်လာနိုင်မှာပါ။ ဒီသဘောအရ ပရိုဂုရမ်းမင်း ပညာရပ်ဟာလည်း ချင်းချက်မဖြစ်နိုင်ပါဘူး။

အောက်ပါ ဥပမာတစ်ခုချင်းကို ပုံစံနားလည်အောင်ဖတ်ပြီး ကိုယ်တိုင်စဉ်းစား ရေးကြည့်ဖို့ လေးလေးနှက်နှက် အကြိုပြုပါတယ်။ ရေးပြီးသွားရင် ပုံစံမှာ ဖော်ပြထားချက်နဲ့ အညီ အလုပ်လုပ်/မလုပ်ဟာကွက်မရှိအောင် ဘယ်လုံးစိတ်ဆေး စိစစ်မလဲ မိမိဘာသာ စဉ်းစားကြည့်ပါ။

Internet Delicatessen

Online ကနေ အစားအသောက်တွေ delivery ပိုပေးဖို့ အမှားလက်ခံတဲ့ ဆိုင်လေးတစ်ဆိုင်အတွက် ပရိုဂုရမ်ရေးပေးရပါမယ်။ အော်ဒါမှာတဲ့အခါ မှာမဲ့ အစားအသောက် နံမည်၊ ရွေးနှုန်းနဲ့ အိပ်စ်ပရက်စ် delivery-

ery ယူမယူ ပရိုဂရမ်မှာ ထည့်ပေးရပါမယ်။ ပရိုဂရမ်က အောင်ဒါနဲ့ စုစုပေါင်းကျသင့်ငွေကို ထုတ်ပေးရပါမယ်။ တစ်သောင်းနဲ့အထက်မှာရင် delivery ဖိုး ပေးစရာမလိုပါ။ တစ်သောင်းအောက်ဆိုရင်တော့ နှစ်ရာ ပေးရပါမယ်။ အိပ်စ်ပရက်စ် delivery ယူမယ်ဆိုရင် သုံးရာအပို ထပ်ပေးရပါမယ်။

Enter the item: Tuna Salad

Enter the price: 4500

Express delivery (0==no, 1==yes): 1

Invoice:

Tuna Salad 4500

delivery 500

total 5000

```
itm_name = input('Item name: ')
price = int(input('Price: '))
is_ex_deli = int(input('Express delivery (0==no, 1==yes): '))

tot_deli_fee = 0
if price >= 10_000 and is_ex_deli == 1:
    tot_deli_fee = 300
elif price >= 10_000 and is_ex_deli == 0:
    tot_deli_fee = 0
elif price < 10_000 and is_ex_deli == 1:
    tot_deli_fee = 200 + 300
elif price < 10_000 and is_ex_deli == 0:
    tot_deli_fee = 200
else:
    print("You may have wrong value for express deli.")

tot_cost = price + tot_deli_fee

print("Invoice: ")
print(f'{itm_name:<10s} {price:8.2f}')
print(f'{delivery:<10s} {tot_deli_fee:8.2f}')
print(f'{total:<10s} {tot_cost:8.2f}')
```

အိပ်စ်ပရက်စ် delivery ယူ/မယူ တစ် (သို့) သူည် ထည့်ပေးရမှာပါ။ တစ်/သူည် မဟုတ်တဲ့ ကဏ္န်းတစ်ခု မှားထည့်မိရင် if...elif ကွန်ဒါရင်တွေ တစ်ခုမှ True ဖြစ်မှာ မဟုတ်ပါဘူး။ မှားထည့်ထားတယ်လို့ သတိပေးဖို့ else အပိုင်းမှာ လုပ်ထားတယ်။ မှန်/မမှန် စိစစ်တဲ့အခါ အောက်ပါပေါ်ယေားမှ ဖြစ်နိုင်ခြေ အားလုံး ခြိုင်မံအောင် စစ်သုံးတယ်။ အိပ်စ်ပရက်စ် delivery အတွက် input ကဏ္န်း မှားထည့်ရင် သတိပေးတာကိုလည်း စစ်သင့်တယ်။ တစ်သောင်းဖိုး ဝယ်တာကို သုံးဖျိုးစစ်ထားတာ လေ့လာကြည့်ပါ။ တစ်သောင်း မပြည့်တာနဲ့ တစ်သောင်းကျကျဖိုး အတွက်လည်း အလားတူ စစ်ကြည့်လို့ အားလုံးမှန်တယ်ဆိုရင် ဒီပရိုဂရမ်မှာ ပေးပါနိုင်ခြေ လုံးဝမရှိသလာက် ဖြစ်သွားပါပြီ။

10,000 MMK and above?	Express Delivery?
Yes	Yes
Yes	No
No	Yes
No	No

တေဘာ် ၃၃ သုံးနှစ်စာ နှစ်ထပ်တိုး

Test Output:

```
Item name: Salad
Price: 10000
Express delivery (0==no, 1==yes): 1
Invoice:
Salad      10000.00
delivery    300.00
total      10300.00
```

```
Item name: Salad
Price: 10000
Express delivery (0==no, 1==yes): 0
Invoice:
Salad      10000.00
delivery    0.00
total      10000.00
```

```
Item name: Salad
Price: 10000
Express delivery (0==no, 1==yes): 2
You may have wrong value for express deli.
Invoice:
Salad      10000.00
delivery    0.00
total      10000.00
```

ပရိုကရမ်တစ်ခုရဲ့ လိုအပ်ချက်ဟာ မကြာခကာ ပြောင်းလဲသွားလေ့ရှိတယ်။ အခုပရိုကရမ်မှာ ဈေးနှစ်း သတ်မှတ်ချက်တွေ ပြောင်းလဲနိုင်တယ်။ အနည်းဆုံး တစ်သောင်းခွဲဝယ်မှ ပို့ခဲ့ free ရမှာဖြစ်ပြီး တစ်သောင်းခွဲအောက်ဆိုရင် ပို့ခဲ့သွားလဲသတ်မှတ်လိုက်တဲ့အားဖြင့် အိပ်စ်ပရက်စ် delivery ကလည်း တစ်ရာရွေးထပ်တက်သွားတယ် ဆိုပါစို့။ တကယ့်လက်တွေမှာလည်း ဒါမျိုးဖြစ်လေ့ရှိပါတယ်။ ဒီအတွက်ကို ပရိုကရမ်ကို ပြင်ပေးရပါမယ်။ ဆိုင်ပိုင်ရှင်ကလည်း ဈေးနှစ်းမကြာခကာ ပြောင်းဖို့လိုနိုင်ပြောင်း ပြောလာပါတယ်။ နောင်လည်း ထပ်ပြင်ပေးဖို့လိုမဲ့ သဘောပါ။ လွှာလွှာယ်ကူကူ ပြင်ပေးနိုင်ရင် အကောင်းဆုံးပါ။ ပြင်ဆင်တဲ့အခါ မှားနိုင်ခေါ်လည်းဖို့လည်း အရေးကြီးပါတယ်။ ခုရေးထားတဲ့အတိုင်းဆိုရင် ပြီးခဲ့တဲ့ ပရိုကရမ်မှာ ပြင်မယ်ဆိုရင် 10_000 ကို လေးနေရာ၊ ပုံမှန်ပို့ခဲ့ 200 နဲ့ အိပ်စ်ပရက်စ်အပိုပြောင်း 300 စတာတွေကို နှစ်နေရာစီ လိုက်ပြင်ရပါမယ်။ အလုပ်ရှုပ်တဲ့အားဖြင့် ပြင်ဖို့ကျန်ခဲ့တာလို့ မှားတာလည်း

ပြီးခဲ့တဲ့ ပရိုကရမ်မှာ ပြင်မယ်ဆိုရင် 10_000 ကို လေးနေရာ၊ ပုံမှန်ပို့ခဲ့ 200 နဲ့ အိပ်စ်ပရက်စ်အပိုပြောင်း 300 စတာတွေကို နှစ်နေရာစီ လိုက်ပြင်ရပါမယ်။ အလုပ်ရှုပ်တဲ့အားဖြင့် ပြင်ဖို့ကျန်ခဲ့တာလို့ မှားတာလည်း

ဖြစ်နိုင်တယ်။ “Find and Replace” လုပ်မှာပေါ့ပို့ စောဒကတက်စရာ ရှိပါတယ်။ အခုလို ကုဒ်လိုင်း နည်းနည်းပရှိတဲ့ ပရိုကရမ်အသေးလေးမှာ အဆင်ပြောင်းပေ့ ပို့ချုပ်ထွေးပြီး ကုဒ်လိုင်းတွေများတဲ့ ပရိုကရမ် မျိုးတွေမှာ “Find and Replace” လုပ်ရင် မပြင်သင့်တာတွေကိုပါ မရည်ရွယ်ပဲ ပြင်မိသွားတာဖြစ် တတ်ပါတယ်။ ပရိုကရမ်ရေးတဲ့အခါ ကျင့်သုံးရမဲ့ အလေ့အထကောင်းတစ်ခုက ကုဒ်ထဲမှာ ဒီတိုင်းချရေးထား တဲ့ တန်ဖိုးတွေ (literal constants) တွေကို နာမည်ပေးထားတာပါ။ အပေါ်ကာ ပရိုကရမ်မှာ literal constants တွေချည်း သုံးထားတယ်။ နံမည်ပေးထားတဲ့ constants (named constants) တွေ အဖြစ် ပြောင်းရေးသင့်တယ်။

```

FREE_DELI_AMT = 15_000
DELI_FEE = 350
EXP_DELI_FEE = 400

itm_name = input('Item name: ')
price = int(input('Price: '))
is_exp_deli = int(input('Express delivery (0==no, 1==yes): '))

tot_deli_fee = 0

if price >= FREE_DELI_AMT and is_exp_deli == 1:
    tot_deli_fee = EXP_DELI_FEE
elif price >= FREE_DELI_AMT and is_exp_deli == 0:
    tot_deli_fee = 0
elif price < FREE_DELI_AMT and is_exp_deli == 1:
    tot_deli_fee = DELI_FEE + EXP_DELI_FEE
elif price < FREE_DELI_AMT and is_exp_deli == 0:
    tot_deli_fee = DELI_FEE
else:
    print("You may have wrong value for express deli.")

tot_cost = price + tot_deli_fee
print("Invoice: ")
print(f'%-10s %8.2f' % (itm_name, price))
print(f'%-10s %8.2f' % ('delivery', tot_deli_fee))
print(f'%-10s %8.2f' % ('total', tot_cost))

```

ဒီပရိုကရမ်ကို cascading if မသုံးဘဲ ရှိုးရိုး if နဲ့ ရေးလိုလည်း ရတယ်။

```

FREE_DELI_AMT = 15_000
DELI_FEE = 350
EXP_DELI_FEE = 400

itm_name = input('Item name: ')
price = int(input('Price: '))
is_exp_deli = int(input('Express delivery (0==no, 1==yes): '))

tot_deli_fee = 0

```

```

if price < FREE_DELI_AMT:
    tot_deli_fee += DELI_FEE

if is_express_deli == 1:
    tot_deli_fee += EXP_DELI_FEE

if not (is_express_deli == 0 or is_express_deli == 1):
    print("You may have wrong value for express deli.")

tot_cost = price + tot_deli_fee

print("Invoice: ")
print(f'%-10s %8.2f' % (item_name, price))
print(f'%-10s %8.2f' % ('delivery', tot_deli_fee))
print(f'%-10s %8.2f' % ('total', tot_cost))

```

ပထမ if က delivery အ ပေးဖို့ လိုတယ်ဆိုရင် tot_deli_fee မှာ DELI_FEE ပေါင်းထည့်ပေးတယ်။ ဒုတိယ if က အိပ်စံပရ်စ် delivery ယူမယ်ဆိုရင် tot_deli_fee မှာ EXP_DELI_FEE ထပ်ပေါင်းထည့်တယ်။ အောက်ဆုံး if ကတော့ အိပ်စံပရ်စ် delivery အတွက် တစ်နဲ့ သုည မဟုတ်တာ ထည့်မိရင် သတိပေးစာသား ပြပေးတယ်။

ဒီပုဂ္ဂရမ်နဲ့ ဒီမတိုင်ခင် သူရှေ့က ပရိုဂ္ဂရမ်က ဖြစ်နိုင်ခြေအားလုံးအတွက်တော့ ရလဒ် တူတူမထွက်ပါဘူး။ အခြေအနေ တစ်ခုကလွှဲလို့ ကျွန်တော့အတွက်တော့ ရလဒ်တူပါတယ်။ တစ်ဆောင်းငါးထောင်ထက်ငယ်တဲ့ တန်ဖိုးနဲ့ အိပ်စံပရ်စ် delivery အတွက် ကဏ္နား လွှဲထည့်ကြည့်ပါ။ ဥပမာ

```

Item name: salad
Price: 12000
Express delivery (0==no, 1==yes): 2
You may have wrong value for express deli.

Invoice:
salad      12000.00
delivery    0.00
total      12000.00

```

```

Item name: salad
Price: 12000
Express delivery (0==no, 1==yes): 2
You may have wrong value for express deli.

Invoice:
salad      12000.00
delivery    350.00
total      12350.00

```

နောက်ဆုံး ပရိုဂ္ဂရမ်က အိပ်စံပရ်စ် delivery အတွက် မပေါင်းထည့်ပေမဲ့ ပုံမှန် delivery အ သုံးရှုင်းဆယ်ကိုတော့ ထည့်ပေါင်းသွားတာ တွေ့ရမယ်။ မှားထည့်တာကိုပဲ သတိပေးစာသားပြပေးတာ၊ ထည့်မတွက်သွားတာက ပို့ပြီး သဘာဝကျေတယ်လို့ ယူဆရမှာပါ။

ပြင်ပကန် ထည့်ပေးတဲ့အခါ မဖြစ်သင့်တဲ့ input တန်ဖိုးတွေ ဝင်မလာအောင် စိစစ်တာကို input validation လို့ ခေါ်တယ်။ တကယ့် လက်တွေ အသုံးချ ပရိုကရမ်တွေမှာ input validation လုပ်ထားဖို့ အရေးကြီးပေမဲ့ ဘီကင်နာအဆင့် လေ့လာတဲ့ ဥပမာတွေမှာတော့ လေ့လာရင်းကိစ္စကန် လမ်းကြောင်းမချုပ်သွားအောင် ဆင်ခြင်ရမှာဖြစ်ပြီး သင့်တော်ရဲ့ ဆက်စပ်ရှင်းပြပါမယ်။

တာရာ ပရိုကရာ

ကားသီးလေထိုးတာဟာ ကားရဲ့ စွမ်းဆောင်ရည်ရော အန္တရာယ်ကင်းဖို့အတွက်ပါ ပစာနျကျပါတယ်။ ကားတစ်စီးအတွက် အကြံပြုထားတဲ့ တာယာပရိုကရာ (recommended pressure) အပေါ် မူတည်ပြီး လေ ဘယ်လောက်တင်းလို့ရလဲ၊ လျှော့လို့ရလဲ ရှိပါတယ်။ ဥပမာ recommended pressure က 35 psi (pounds per square inch) ဖြစ်ရင် အလွန်ဆုံး 31.5 psi ထိ လေလျှော့လို့ရပါတယ်။ လေပို့တင်းမယ်ဆိုရင်လည်း အလွန်အလွန်ဆုံး 44 psi အထိ ရဲပါနိုင်ပါတယ်။

ရှေ့တာယာနှစ်လုံး ပရိုကရာအနီးစပ်ဆုံးတူသင့်ပြီး 3 psi အထိ ကွာဟာလိုရတယ်။ ကွာဟာချက်က 3 psi ထက်တော့ မများသင့်ဘူး။ နောက်တာယာနှစ်လုံးလည်း ထိုနည်းတူစွာပဲ ဖြစ်တယ်။ ကားမော်ဒယ် အလိုက် recommended pressure ကွာဗြားပေမဲ့ အိမ်စီးကားအများစုအတွက် 35 psi ဖြစ်တယ်လို့ ယူဆပြီး တာယာပရိုကရာ အိုကောမကေ စစ်ပေးတဲ့ ပရိုကရမ် ရေးပေးရပါမယ်။ Input အနေနဲ့ တာယာတစ်ခုချင်းအတွက် ပရိုကရာ psi တန်ဖိုး ထည့်ပေးရမှာပါ။ ထည့်ပေးလိုက်တဲ့ တာယာပရိုကရာ 31.5 psi ထက်နည်းနေရင် သို့မဟုတ် 44 psi ထက်များနေတာနဲ့ သတ်မှတ်ဘောင် မဝင် (out of range) ဖြစ်နေကြောင်း သတိပေးရပါမယ်။ တာယာအားလုံးရဲ့ ပရိုကရာတွေ သတ်မှတ်ဘောင်အတွင်း ဝင်တယ်။ ရှေ့တာယာနှစ်လုံး ကွာဟာချက်၊ နောက်တာယာနှစ်လုံး ကွာဟာချက်တွေ ခွင့်ပြုလိုရတာထက် မပို့ဘူးဆိုရင် လေထိုးထားတာ အိုကေတယ်။ တာယာတစ်လုံး out of range ဖြစ်နေတာနဲ့ လေထိုးထားတာ မအိုကေဘူး ပြပေးရပါမယ်။

```

MIN_ALLOWABLE = 31.5
MAX_ALLOWABLE = 44.0
WARNING = 'Waring: Pressure is out of range!'
LEFT_RIGHT_DIFF_ALLOWABLE = 3.0

is_out_of_range = False

front_left = float(input("Front left pressure: "))
if not (MIN_ALLOWABLE <= front_left <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

front_right = float(input("Front right pressure: "))
if not (MIN_ALLOWABLE <= front_right <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

rear_left = float(input("Rear left pressure: "))
if not (MIN_ALLOWABLE <= rear_left <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

```

```

rear_right = float(input("Rear right pressure: "))
if not (MIN_ALLOWABLE <= rear_right <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

front_diff = abs(front_left - front_right)
front_diff = abs(rear_left - rear_right)
if (front_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
    and rear_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
    and not is_out_of_range):
    print("Inflation is OK.")
else:
    print("Inflation is not OK!")

```

is_out_of_range ဘူလီယန် သုံးထားတာ နည်းနည်း ရှင်းပြန့် လိုပါမယ်။ စစချင်း False တန်ဖိုး ထည့်ထားတာ တွေ့ရမှာပါ။ if စတိတ်မန့်တွေက တာယာတစ်ခုချင်းကို out of range ဖြစ်နေလား စစ် ထားတာတွေ့ရတယ်။ တာယာတစ်ခု out of range ဖြစ်တာနဲ့ is_out_of_range က True ဖြစ်သွား မှာပါ။

အောက်ပါင်းမှ front_diff နဲ့ rear_diff ရှာတဲ့အခါ abs ဖန်ရှင်းနဲ့ ပကတိတန်ဖိုး ယူထားတာ သတိပြုပါ။ ပရ်ရှာ ခြားနားချက်ရှာတဲ့အခါ အနှံတ်တန်ဖိုး ထွက်နိုင်တဲ့အတွက်ကြောင့်ပါ။ ဘယ်ဘက် တာယာက ပရ်ရှာနည်းနေတဲ့အခါ (ဥပမာ 32 – 37 = -5) အနှံတ်တန်ဖိုး ဖြစ်နေမယ်။ ဒါကြောင့် ပကတိတန်ဖိုးယူမယဲ့ ကွာဟချက် 3 psi ကျော်မကျော်စစ်ပေးတဲ့အခါ အဖြော်ရပါမယ်။

```

front_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
    and rear_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
    and not is_out_of_range

```

and နှစ်ခုနဲ့ ဆက်ထားရင် သုံးခုလုံးမှန်မယဲ့ True ထွက်မှာပါ။ တစ်ခုမှားတာနဲ့ အပိုစ်ပရ်ရှင်တစ်ခုလုံး ရလေ၏ False ပဲ။ Out of range မဖြစ်ရဘူး ဆိုတာကို not is_out_of_range နဲ့ စစ်ထားတယ်။ is_out_of_range တန်ဖိုး False ဖြစ်မှ မှတ်ဆောင်ရွက်နေမယ်။

အခန်း ၈

ဖန်ရှင်များ

ဖန်ရှင်တွေဟာ ပရိုကရမဲ အဆောက်အညီတစ်ခုရဲ အခြေခံ အုပ်ချပ်တွေပါ။ ဒီအခြေခံ အုပ်ချပ်တွေကိုမူကောင်းကောင်းမွန်မွန် နားလည် အသုံးမချကတ်ရင် ကျမ်းကျင်တဲ့ ပရော်ဖက်ရှင်နယ် ပရိုကရမဲမာ တစ်ယောက် မဖြစ်နိုင်ဘူးဆိုတာ အထူးခြားစရာ လိုမယ် မထင်ပါဘူး။ ကားရဲ့မှာလည်း ဖန်ရှင် အခြေခံ သဘောတရားနဲ့ အသုံးချပုံကို လေ့လာခဲ့ကြရပါတယ်။ ဒီအခန်းမှာတော့ တန်ဖိုးပြန်ပေးခြင်း၊ ပါရာမီတာ တွေ အသုံးပြုပုံနဲ့ exception-handling တို့ကို ဆက်လက် လေ့လာကြမှာပါ။ လက်တွေနဲ့ နီးစပ်ပြီး အမှန် တကယ် အသုံးဝင်မဲ့ ပရိုကရမဲလေးတွေကို အစကနေ အဆုံးထိ တစ်ဆင့်ပြီးတစ်ဆင့် ဘယ်လိုအိမ်းပြုလုပ် တည်ဆောက်လဲ ကိုလည်း ဖော်ပြပေးသွားမှာပါ။

၈.၁ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်များ

အခန်း (၅) မှာ ဖော်ပြခဲ့တဲ့ နှစ်ထပ်ကိန်းရာတဲ့ `square` ဖန်ရှင်ကိုပဲ အသေးစိတ် တစ်ခါတပ်ကြည့်ရအောင်။ ဒီလောက် ရှင်းရှင်းလေးကို အကျယ်ချဲ့နေတယ်လို့ ထင်ကောင်း ထင်ပါလိမ့်မယ်။ နည်းနည်းတော့ စိတ်ရှည်သည်းခဲ့ ပေးပါမယ်။ အခြေခံကျတဲ့ သဘောတရားတွေ ကျေည်ကားမှ ရှေ့ဆက်တဲ့ အခါ လွှာယ်ကူးမှ မို့လိုပါ။

```
>>> def square(x):
...     return x ** 2
...
>>>
```

ပိုက်ကွင်းထဲက ဖော်ရောဲလ် `x` က ဖန်ရှင် ပါရာမီတာ (parameter) ဖြစ်ပြီး ဖန်ရှင်ခေါ်တဲ့အခါ ထည့်ပေးမဲ့ အားဂုံးမန်း (argument) တန်ဖိုးကို ကိုယ်စားပြုတယ်။ `return` စတိတ်မန်းက ဖန်ရှင်ခေါ်တဲ့နေရာ ကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန်းပါ။

ဖန်ရှင်အသုံးပြုတာကို *function call* လုပ်တယ်လို့ သိထားပြီးပါပြီ။ မြန်မာလိုတော့ ‘ဖန်ရှင်ခေါ်တယ်’ သို့မဟုတ် ‘ဖန်ရှင်ကောလ်တယ်’ လို့ အပြောများတယ်။ ဖန်ရှင်ခေါ်တဲ့ ပုံစံက ဒီလိုပါ

```
>>> square(2.5)
6.25
```

အခါ ဖန်ရှင်ကောလ် အတွက် ပါရာမီတာ `x` ရဲ့ တန်ဖိုးက 2.5 ဖြစ်မှာပါ။ (ဖန်ရှင်ခေါ်တဲ့အခါ ပါရာမီတာ ဖော်ရောဲလ် `x` ကို အားဂုံးမန်းနဲ့ အဆိုင်းမန်းလုပ်ပေးတယ်လို့ ယူဆနိုင်တယ်။ ဒီကိစ္စအတွက် အားဂုံးက

2.5 ဖြစ်တယ်)။ အားလုံးသိပြီး ဖြစ်တဲ့အတိုင်း ဖန်ရှင်းခေါ်ရင် ဖန်ရှင်းလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဖန်ရှင်းလောက်ထဲက `return` စတိတ်မနဲ့ လုပ်ဆောင်တဲ့အခါ အိပ်စ်ပရက်ရှင် $x \star\star 2$ ကို တန်ဖိုးအရင်ရှာတယ်။ 6.25 ရတယ်။ ဒီတန်ဖိုးကို ဖန်ရှင်းခေါ်ထားတဲ့ နေရာကို `return` က ပြန်ပိုးပေးလိုက်တာပါ။ အောက်ပါ ဖန်ရှင်းကောလုံးမှာလည်း ဒီဖြစ်စဉ် သဘောအတိုင်း တစ်ခါထပ်ဖြစ်မှာ ဖြစ်တယ်။

```
>>> a = 1024
>>> result = square(a)
>>> result
1048576
```

အခုတစ်ခါ ပါရာမီတာ x ဟာ အားဂုံး a ရဲ့ တန်ဖိုး ဖြစ်တယ် ($x = a$ အဆိုင်းမနဲ့ လုပ်တဲ့သဘောပဲ)။ $x \star\star 2$ က ရလာတဲ့ 1048576 ကို ဖန်ရှင်းခေါ်တဲ့ နေရာက ပြန်ရတယ်။ နောက်ဆုံးတော့ ဒီတန်ဖိုးကို `result` မှာ အဆိုင်းမနဲ့လုပ်တယ်။ ဖြစ်စဉ်အရ ရှိုးရှင်းပါတယ်။

```
>>> x = 10
>>> square(x)
```

ဒီလိုဆိုရင်ရော ဘယ်လို ဖြစ်မလဲ။ နည်းနည်းထူးခြားတာက အားဂုံးနဲ့ ပါရာမီတာ နံမည်တူနေတာ။ ပါရာမီတာရဲ့ စက်ပော ဖန်ရှင်းသတ်မှတ်ချက် အတွင်းမှာပဲ ရှိတယ်လို ယူဆရမှာပါ။ ဒါကြောင့် အားဂုံး $x \star\star$ ပါရာမီတာ x နဲ့က သီးခြား ပေါ်ရောက်တွေ့။

```
>>> u = 15
>>> t = 5
>>> square(u + 2*t)
```

အားဂုံးက အိပ်စ်ပရက်ရှင် ဖြစ်နေရင် တန်ဖိုးအရင်ရှာပြီး ရလဒ်ကို ပါရာမီတာနဲ့ အဆိုင်းမနဲ့ လုပ်ပါတယ် ($x = u + 2*t$)။

```
>>> z = square(2.0) + 5
>>> square(z)
81.0
>>> square(square(2.0) + 5)
81.0
```

ဒုတိယ ဖန်ရှင်းခေါ်တဲ့နေရာမှာ အိပ်စ်ပရက်ရှင်ကို z နဲ့ အဆိုင်းမနဲ့ မလုပ်တော့ဘဲ တစ်ခါတည်း အားဂုံး အနေနဲ့ ထည့်လိုက်တာပါ။ သဘောတရား တူးတူပါပဲ။

ဖန်ရှင် `return` လုပ်တဲ့ သဘောကို နားလည်ထားဖို့လည်း အရေးကြီးတယ်။ `return` စတိတ်မနဲ့ ဟာ ဖန်ရှင်ကနေ တန်ဖိုးတစ်ခုကို ဖန်ရှင်းခေါ်တဲ့ဆိုကို ပြန်ပေးတယ်လို သိထားပြီးပါပြီ။ ဖန်ရှင်ထဲကနေ `return` လုပ်လိုက်တာနဲ့ ခေါ်ထားတဲ့နေရာကို ချက်ချင်း ပြန်ရောက်သွားတာ။

```
>>> def get_sign(r):
...     if r > 0:
...         return 'positive'
...     elif r < 0:
...         return 'negative'
...     else:
...         return 'zero/nosign'
```

...
 >>>

>>> '10 is ' + get_sign(10)
 '10 is positive'

အခုအိပ်ပံ့ရက်ရှင်ရဲ့ တန်ဖိုးရှာဖို့ get_sign(10) ခေါ်လိုက်တဲ့အခါ လက်ရှိနေရာကနေ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ဆီ ပြောင်းရွှေ့ ရောက်ရှိသွားပါမယ်။ ဖန်ရှင်ထဲက စတိတ်မန်တွေ အစဉ်အတိုင်း စတင် လုပ်ဆောင်တယ်။ ဖန်ရှင်က return လုပ်တဲ့အခါ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ထဲကနေ ခေါ်ခဲ့တဲ့ နေရာကို တဖန်ပြန်၍ ပြောင်းရွှေ့သွားတယ်။ return ပြန်လိုက်တဲ့ တန်ဖိုးကို ဖန်ရှင်ခေါ်တဲ့နေရာမှာ ရရှိ ပြီး လုပ်လက်စ အိပ်ပံ့ရက်ရှင်ကို ဆက်လုပ်ပါတယ်။ ဒီလိုမြင်ကြည့်ပါ ...

```
def get_sign(r):  

    if r > 0:  

        return 'positive'  

    elif r < 0:  

        return 'negative'  

    else:  

        return 'zero/nosign'  

'10 is ' + get_sign(10)
```

မြှားအနက်က ဖန်ရှင်ခေါ်လိုက်တဲ့အခါ လုပ်ဆောင်မှု ပြောင်းရွှေ့သွားတာကို ပြတယ်။ မြှားအနီးက return ပြန်တဲ့အခါ ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားတာကို ပြတာပါ။

ဆက်လက်ပြီး ပါရာမီတာ တစ်ခုထက်ပိုတဲ့ ဖန်ရှင်တရာ့၍ကို ကြည့်ပါမယ်။ ပါရာမီတာဆိုတာ ဖန်ရှင် အတွက် လိုအပ်တဲ့ input ကို လက်ခံတဲ့ ပော်ရော်လုပ်ပါပဲ။ ထောင့်မှန်စတုဂံရဲ့ အလျားနဲ့ အနံကနေ ချို့ယာရှာပေးတဲ့ ဖန်ရှင်က ဒီလိုပါ

```
def rect_area(wid, len):  

    return wid * len
```

ဖန်ရှင်တစ်ခုကို အခြေခံ အုပ်ချပ်သဖွယ် အသုံးပြု၍ အခြားဖန်ရှင်တွေ တည်ဆောက်ယူနိုင်တယ်။ rect_area ကို box_vol မှာ သုံးထားတာပါ

```
def box_vol(w, l, h):  

    return rect_area(w, l) * h
```

ဒီဖန်ရှင်ကို ခေါ်ချင် ဘယ်လိုဖြစ်မလဲ ကြည့်တတ်သင့်တယ်။ အခုလို ခေါ်မယ ဆိုပါစို့

>>> box_vol(10, 5, 3)

w=10, l=5, h=3 ဖြစ်တယ်။ ဖန်ရှင် ဘလောက်ထဲကို ရောက်သွားမယ်။ return ပြန်ပေးဖို့ အိပ်ပံ့ရက်ရှင်ကို တန်ဖိုးရှာပါတယ်

rect_area(w, l) * h

rect_area ဖန်ရှင်ခေါ်တယ်။ wid=w, len=l ဖြစ်မယ်။ အခုကိစ္စအတွက် ပါရာမီတာနှစ်ခုရဲ့ တန်ဖိုးက 10 နဲ့ 5 အသီးသီး ဖြစ်မှုပါ။ 50 ရပါမယ်။ 50 * h ကို တန်ဖိုးဆက်ရှုပြီး ရလာတဲ့ 150 ကို

box_v01 ခေါ်ထားတဲ့နေရာကို `return` ပြန်ပေးမှာ ဖြစ်တယ်။ အခြေခံသဘောတရားတွေ သိပြီးတဲ့အခါ အတန်အသင့်ရှုပ်ထွေးတဲ့ ဖန်ရှင်တချို့ကို ကြည့်ပါမယ်။

ဖန်ရှင်များနှင့် အက်ဘ်စက်ရှင်းလုပ်ခြင်း

မွေးသက္ကရာဇ် (date of birth) ကနေ အသက် တွက်ပေးတဲ့ ဖန်ရှင်ကို လေ့လာကြည့်ပါ။ အသက်တွက် တဲ့ လေ့ဂျွဲ့ကို မရှင်းပြတော့ဘူး။ လေ့ကျင့်ခန်းအနေနဲ့ မိမိဘာသာ နားလည်အောင်ကြည့်ပါ။

```
# File: age_today.py
from datetime import *

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

print(age_today(date(1990, 4, 2)))
```

ဖန်ရှင်အတွင်းပိုင်း လေ့ဂျွဲ့တွေ ဘယ်လိုပဲ ရှုပ်ထွေးပါစေ၊ အသုံးပြုရတာကတော့ မခက်ပါဘူး။ ဖန်ရှင် ခေါ်တဲ့အခါ ဘယ်လို တည်ဆောက်ထားလဲ အတွင်းပိုင်း အယ်လိုဂုဏ်သမ်တွေ၊ လေ့ဂျွဲ့တွေ သိစရာမ လိုဘဲ သုံးရတာပါ။ ဖန်ရှင်က ငှါးရဲ့ အတွင်းပိုင်း ကုဒ်တွေကို အက်ဘ်စက်ရှင်း (abstraction) လုပ်ပေးလိုက်တာ ဖြစ်တယ်။ ဒါဟာ ဖန်ရှင်ရဲ့ အရေးပါဆုံး ဂုဏ်သတ္တိလို ဆိုရင်လည်း မမှားဘူး။

`age_today` ဖန်ရှင်ဟာ ပိုကြီးတဲ့ ပရိုကရမ်တစ်ခုရဲ့ တစ်စိတ်တစ်ပိုင်း ဖြစ်လာနိုင်ပါတယ်။ ပရိုကရမ် အသေးစားလေးတစ်ခုမှာ အသုံးပြုထားတာကို လေ့လာကြည့်ပါ။ နိုင်ငံအများစုမှာ (၁၈) နှစ် မပြည့်သေးတဲ့သူကို ဆေးလိပ်ရောင်းခွင့် မရှိဘူး။ ဥပဒေရှိပါတယ်။ စားသုံးသူရဲ့ မွေးသက္ကရာဇ် ထည့်ပေးလိုက်တာနဲ့ ရောင်းလို ရ/မရ ပြပေးတဲ့ ပရိုကရမ်လေးပါ။

```
# File: sell_cigarette.py
from datetime import *

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

def can_by_cig(dob):
    age = age_today(dob)
    return True if age >= 18 else False

def main():
```

```

"""
Given date of birth, this program tells whether the customer
is eligible to buy cigarette or not.

Enter 'exit' to quit the program.
"""

print("Please enter 'quit' to exit this program.")
while True:
    dobstr = input('Enter date of birth (yyyy-mm-dd): ')
    if dobstr == 'quit': break
    dob = date.fromisoformat(dobstr)
    print(dob)
    if can_by_cig(dob):
        print("Okay!")
    else:
        print('Too young to sell cigarette!')
print('Program exited...')

```

```

if __name__ == "__main__":
    main()

```

age_today ဖန်ရှင်ဟာ အခြားပရီဂရမ်တွေမှာလည်း အသုံးကျနိုင်ပါတယ်။ ဒီဖန်ရှင်ကို ခကေခက မရေးရဘဲ လိုအပ်တဲ့ ပရီဂရမ်တွေကနေ အလွယ်တကူ ပြန်သုံးလို့ရအောင် လိုက်ဘရဲ့ ဒါမှမဟုတ် မော်ဒုံးလိုခေါ်တဲ့ ကုဒ်အဖွဲ့အစည်း ယူနစ်တစ်ခုအနေနဲ့ ထုတ်ထားနိုင်တယ်။ လိုက်ဘရဲ့ (သို့) မော်ဒုံးတစ်ခု ဟာ ဆက်စပ်တဲ့ ဖန်ရှင်တွေ၊ ကလပ်စွေတွေနဲ့ ဖွဲ့စည်းထားတာ ဖြစ်တယ်။ နောက်ပိုင်းမှာ ဒါနဲ့ပါတ်သက်ပြီး သီးခြားလေ့လာရမှာပါ။

၈.၂ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်များ

ဖန်ရှင်အားလုံးတော့ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်တွေ မဟုတ်ကြပါဘူး။ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေလည်း ရှိတယ်။ ဥပမာ output ထုတ်တဲ့ print ဖန်ရှင်ဟာ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်မျိုးပါ။ အောက်ပါ print_sign ဖန်ရှင်ဟာ get_sign နဲ့ ဆင်တူပေမဲ့ တန်ဖိုး return ပြန်မပေးပါဘူး။

```

def print_sign(r):
    if r > 0:
        print('positive')
    elif r < 0:
        print('negative')
    else:
        print('zero/nosign')

```

ဒီဖန်ရှင်မှာ return မပါတာ တွေ့ရပါမယ်။ ကားရဲလ်ဖန်ရှင်တွေမှာလည်း return မသုံးခဲ့တာ ပြန် အမှတ်ရမှာပါ။ append_n_times ကို လေ့လာကြည့်ပါ

```

def append_n_times(lst, item, n):
    for i in range(n):
        lst.append(item)

lst = []
append_n_times(lst, 'hello', 10)
print(lst)

```

အိုက်တမ်တစ်ခုကို သတ်မှတ်ထားတဲ့ အရေအတွက်ပြည့်အောင် list တစ်ခုနောက်ကနေ ဆက်ပေးတယ်။ နှင့် list မှာ အိုက်တမ်တွေ တိုးသွားပြီး စတိတ်အပြောင်းအလဲ ဖြစ်စေတယ်။

Output ထုတ်တဲ့ ဖန်ရှင်တွေဟာ တန်ဖိုးပြန်ပေးလေ့မရှိဘူး။ စခရင်မှာ စာသား (သို့) ရုပ်ပုံ ပြပေး တော့ output ဖြစ်တယ်။ ဖိုင်တစ်ခုမှာ ရေးတာလည်း output ပဲ (ဥပမာ Python ကုဒ်ဖိုင်ကို ပြင် ပြီး save လုပ်တာ) ။ အော့ဘ်ဂျက် စတိတ်ကို ပြောင်းလဲစေတဲ့ ဖန်ရှင်တွေဟာလည်း တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေ ဖြစ်လေ့ရှိတယ် (ဥပမာ list ရဲ့ append နဲ့ insert ဖန်ရှင်)။ စတိတ်အပြောင်းအလဲ ဖြစ်စေတဲ့ ဖန်ရှင်အားလုံး တန်ဖိုးပြန်မပေးတာတော့ မဟုတ်ဘူး။ ဥပမာ pop ဟာ တန်ဖိုးပြန်ပေးပါတယ်။ စတိတ်အပြောင်းအလဲလည်း ဖြစ်စေတယ်။

တန်ဖိုးပြန်တဲ့ ဖန်ရှင်ပဲ return ပြန်လိုက်တာ မဟုတ်ပါဘူး။ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေမှာလည်း return ပါရှိနိုင်ပါတယ်။ print_sign ကို ဒီလိုရေးလို့လည်း ရပါတယ်

```

def print_sign2(r):
    if r > 0:
        print('positive')
        return
    elif r < 0:
        print('negative')
        return
    else:
        print('zero/nosign')
        return

```

တန်ဖိုးပြန်မပေးတဲ့အတွက် return ပဲဖြစ်ရပါမယ်။ တန်ဖိုး/အိပ်စံပရက်ရှင် တဲ့ပြီး ပါလိုမရပါဘူး။ ဖန်ရှင်ဘလောက် ပြီးတဲ့အခါ ခေါ်တဲ့နေရာကို ပြန်ရောက်သားရမှာ ဖြစ်တဲ့အတွက် return မပါတဲ့ ဖန်ရှင်တွေရဲ့ ဘလောက်အဆုံးမှာ return ရှိတယ်လို့ ယူဆနိုင်တယ်။ ဥပမာ return မပါတဲ့ print_sign ကို အခုခုလို ယူဆနိုင်တယ်

```

def print_sign(r):
    if r > 0:
        print('positive')
    elif r < 0:
        print('negative')
    else:
        print('zero/nosign')
    return

```

(ဖန်ရှင် ဘလောက်အဆုံးမှာ return လုပ်ထားတယ်လို့ ယူဆရမှာပါ)။

ပို့ပြီးတိတိကျကျ ပြောမယ်ဆိုရင် Python မှာ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေက None အော် ဂျက် ပြန်ပေးပါတယ်။ None အော်ဂျက်ကို 'တန်ဖိုးတစ်စုံတစ်ရာမရှိ' (သို့) 'မည်သည့် တန်ဖိုးကိုမှ ကိုယ်စားမပြု' ဆိုတဲ့ အဓိပါယ်အတွက် အသုံးပြုတာပါ။

၈.၃ Exceptions

ပုံမှန်အားဖြင့်တော့ ဖန်ရှင်တစ်ခုဟာ သူလုပ်ဆောင်ပေးရမဲ့ ကိစ္စကို ပြီးမြောက် အောင်မြင်အောင် ဆောင် ရွက်ပေးရမှာပါ။ ဒါပေမဲ့ ပုံမှန်မဟုတ်တဲ့ (သို့) မမျှော်လင့်ထားတဲ့ အခြေအနေ တစ်စုံတစ်ရာကြောင့် ဖန်ရှင် တစ်ခုဟာ သူလုပ်ဆောင်ပေးရမဲ့ ကိစ္စကို ပြီးအောင်ဆက်လုပ်ပေးလို့ မရရှိနိုင်တော့တာ ဖြစ်နိုင်ပါတယ်။ 'ပုံမှန် မဟုတ်တဲ့ (သို့) မမျှော်လင့်ထားတဲ့ အခြေအနေ' ဆိုတာ ဘယ်လိုပါမိပါလဲ။ အီးမေးလိုပို့စွဲ send_email ဖန်ရှင် ခေါ်တယ်ဆိုပါစို့။ အင်တာနက် ကွန်နက်ရှင် ရှိပါမယ်။ မရရှိရင် send_email က အီးမေးလိုပို့စွဲ မရရှိနိုင်ပါဘူး။ date အော်ဂျက်တစ်ခု ဖန်တီးမယ်ဆိုပါစို့။ လန့်ရက် မဖြစ်နိုင်တဲ့ ကဏ္နာတွေ ထည့်ပေးရင် အော်ဂျက် ဖန်တီးလိုမရရှိနိုင်ပါဘူး (သို့) ဖန်တီးမပေးသင့်ဘူး။

```
>>> date(2024, 13, 32)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: month must be in 1..12
>>> date(2024, 12, 32)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: day is out of range for month
```

ဖိုင်တစ်ခုကို ဖွင့်တဲ့အခါ ဖိုင်နဲ့အခါ ဖိုင်နဲ့မည် (သို့) path လမ်းကြောင်းမှားနေရှင် ဖွင့်လိုမရပါဘူး။ ဖိုင်သိမ်းတဲ့အခါ မှားလည်း ခွင့်မပြုထားတဲ့ နေရာမှာ သိမ်းလိုမရပါဘူး။ ဖျက်ပစ်မယ်ဆိုလည်း ခွင့်မပြုတဲ့ဖိုင်ကို ဖျက်လိုမရဘူး။ ဖိုင်စနစ်နဲ့ သက်ဆိုင်တဲ့ ဖန်ရှင်တွေကို အခန်း (၁၀) မှာ လေ့လာကြတဲ့အခါ ဒီလိုပိုးပြသုနာတွေ ကြော်တွေရမှာပါ။

```
>>> open('abc.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'
```

အခုဖော်ပြခဲ့တာတွေဟာ 'ပုံမှန်မဟုတ်တဲ့ (သို့) မမျှော်လင့်ထားတဲ့ အခြေအနေ' ဥပမာတချို့သာ ဖြစ်ပါတယ်။ နောက်ပိုင်းမှာ အခြားဟာတွေ ထပ်တွေရမှာပါ။ ပရိုဂရမ်တစ်ခု run နေတဲ့အချို့ ဒီလို အခြေအနေတွေ ဖြစ်လာခဲ့ရင် ပရိုဂရမ်က ရပ်သွားပြီး ဆက်လက် အလုပ် မလုပ်ပေးနိုင်တော့တာမျိုး မဖြစ်သင့်ပါဘူး။ ဒီလိုမဖြစ်အောင် ကိုင်တွယ်ထိန်းကြောင်း ပေးဖို့အတွက် ခေတ်မီ programming language တွေ အားလုံးလိုလိုမှာ ရှိုးရှင်းတဲ့ နည်းစနစ်တစ်ခု ထည့်သွင်းပေးထားပါတယ်။ အဲဒီ နည်းစနစ် ကတော့ exception-handling ပဲ ဖြစ်ပါတယ်။

raise -ing Exceptions

Exception-handling မှာ အပိုင်း နှစ်ပိုင်းပါဝင်တယ်။ ပထမတစ်ခုက တစ်ခုခဲ့ ပြသုနာဖြစ်နေပြီ ဆိုတာ အသိပေးတဲ့ အပိုင်း။ ဖန်ရှင်တစ်ခုဟာ ပုံမှန်မဟုတ်တဲ့ ပြသုနာကြောင့် သူတာဝန်ကို ပြီးမြောက် မှန်ကန်အောင် မလုပ်ပေးနိုင်တော့တဲ့အခါ ဖန်ရှင်ခေါ်တဲ့သူ (သို့) ခေါ်ထားတဲ့နေရာ ကို အသိပေးနိုင်ဖို့ လို

ပါတယ်။ ဒါမှာလည်း ဖြစ်တဲ့ ပြဿနာပေါ် မူတည်ပြီး ဘာလုပ်ရမလဲ ဆုံးဖြတ်လို့ ရမယ်။ (အီးမေးလိုပိုတာ အင်တာနက်မရရှင် ခဏနောင့်ပါလို့ အသိပေးတာမျိုး၊ ရက်စွဲထည့်တာ မှားရင် ပြန်ထည့်ခိုင်းတာမျိုး၊ ဖိုင်မရှိလို့ ဖွင့်မရရင် နောက်တစ်နာရီကြာမှ ပြန်စမ်းဖို့ တိုင်မာပေးတာမျိုး ... စသည်ဖြင့်ပေါ့)။

```
# File: print_n_times.py
def print_n_times(txt, n):
    if not (isinstance(n, int) and n > 0):
        raise ValueError('Positive integer expected')
    for i in range(n):
        print(txt)

raise ValueError('Positive integer expected')
```

ဒီဖန်ရှင်က စာသားတစ်ခုကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ပရင့်ထွက်ပေးမှာပါ။ အကြိမ်အရေအတွက်က အပေါင်း ကိုန်းပြည့်ကဏ္ဍး ဖြစ်သင့်ပါတယ်။ မဟုတ်ဘူးဆိုရင် ဖန်ရှင်ခေါ်တဲ့အခါ ထည့်ပေးတဲ့ အကြိမ်အရေအတွက် အခြောင်းတစ်ခုခုကြောင့် မှားနေတာပဲ ဖြစ်ရမယ်။ ဒီလိုဖြစ်လာခဲ့ရင် တစ်ခုခု မှားနေပြီဆိုတာ အသိပေးဖို့အတွက် raise စတိတ်မန်ကို အသုံးပြုခိုင်တယ်။ isinstance ဖန်ရှင်က ဖော်ရောဘဲလဲရဲ့ တိုက်ပိုက် စစ်ပေးတာပါ။ n ဟာ int ဖြစ်ရင် isinstance(n, int) ၏ True ရမှာပါ။ အပေါင်းကိုန်းပြည့် မဟုတ်ခဲ့ရင်

နဲ့ ပြဿနာကို အသိပေးပါတယ်။ ဒါကို exception ကို raise လုပ်တယ်လို့ ခေါ်တယ်။ ValueError ကတော့ exception (ပုံမှန်မဟုတ်တဲ့/မမျှော်လင့်ထားတဲ့ အခြေအနေ/ပြဿနာကို ဆိုလို) ကို ဖော်ပြတဲ့ အော့ဘိုက်ပါ။ ValueError အပြင် ArithmeticError, FileNotFoundError စသည်ဖြင့် ဖြစ်တဲ့ exception ပေါ်မှာလည်ပြီး သင့်တော်တဲ့ အော့ဘိုက်ကို raise လုပ်ခိုင်ပါတယ်။

print_n_times.py ကို run ကြည့်ပါ။ ဖိုင်အောက်ပိုင်းက ဒုတိယ ဖန်ရှင်ကောလ်မှာ exception တက်မှာပါ။

```
print_n_times('Hello', 10)
print_n_times('Hi', 0)
print_n_times('Hola', 3)
```

အခုလို မက်ဆွဲချုံ တွေပြုပြီး ပရိုကရမ် ဆက်အလုပ် မလုပ်တော့ဘဲ ရပ်သွားမှာပါ။

```
...
Hello
Hello
Traceback (most recent call last):
  File ".../ch08/print_n_times.py", line 10, in <module>
    print_n_times('Hi', 0)
  File ".../ch08/print_n_times.py", line 4, in print_n_times
    raise ValueError('Positive integer expected')
ValueError: Positive integer expected
```

တတိယဖန်ရှင် ဆက်မလုပ်တဲ့အတွက် Hola သုံးခါ မပါတာ သတိပြုပါ။

Handling Exceptions

Exception-handling ရဲ့ ဒုတိယပိုင်းကတော့ handle (ကိုင်တွယ် ထိန်းကျောင်းတာကို ဆိုလို) လုပ်တဲ့ ကိုဖြစ်ပါတယ်။ `raise` လုပ်လိုက်တဲ့ exception ကို handle မလုပ်ရင် ပရီဂရမ်ဟာ ဆက်အလုပ်မလုပ်နိုင်တော့ဘဲ ရပ်သွားမှာပါ။ Handle လုပ်ဖို့ `try` နဲ့ `except` ကို သုံးပါပါတယ်။

ဖန်ရှင်တစ်ခုက `raise` လုပ်လိုက်တဲ့ exception ကို handle လုပ်ဖို့ရည်ရွယ်ချက်ရှိရင် အဲဒီဖန်ရှင်ကို `try` ဘလောက်ထဲမှာ ခေါ်ပါမယ်။ `Exception` ဖြစ်ခဲ့ရင် ဘယ်လို handle လုပ်ချင်လဲ။ ဒါ အပိုင်းကိုတော့ `except` ဘလောက်ထဲမှာ ရေးရပါမယ်။

```
try:
    print_n_times('Hi', 0)
except ValueError as err:
    print(f'Error: {err}')
```

`ValueError` ကတော့ handle လုပ်မဲ့ exception ရဲ့ တိုက်ပဲ။ `ValueError` exception ကိုပဲ handle လုပ်မယ်လို့ ဆိုလိုတာ။ အေားဟာတွေဆိုရင် handle မလုပ်ဘူးပေါ့။ `FileNotFoundException` အတွက်ဆိုရင် `except FileNotFoundException as err:` ဖြစ်ပါမယ်။ `err` က exception ကို ကိုယ်စားပြုတဲ့ ပေရီရောဘဲလဲ (အေား နံမည်ဖြစ်လိုရတယ်)။ `Exception` ဖြစ်ခဲ့ရင် (သို့) `raise` လုပ်ခဲ့ရင် အဲဒီ exception နဲ့ သက်ဆိုင်တဲ့ အချက်အလက်တွေကို `err` ကနေတစ်ဆင့် ရယူနိုင်ပါတယ်။ သုံးဖို့မလိုရင်တော့ `as err` မပါဘဲ `except ValueError:` နဲ့ ရတယ်။

`try` ဘလောက်ထဲမှာ exception ဖြစ်ခဲ့ရင် ဖြစ်တဲ့နေရာကန် သက်ဆိုင်တဲ့ `except` ဘလောက်ဆိုကို 'ချက်ချင်း' ရောက်သွားမှာပါ။ မဖြစ်ခဲ့ရင်တော့ `try` ဘလောက် ပြီးတဲ့အထိ လုပ်ဆောင်ပြီး `except` ဘလောက်ကို လစ်လျှော့၍ သွားမှာပါ။ အခုလို စမ်းသပ်ကြည့်ပါ

```
try:
    print_n_times('Hi', 0)
    print('Done printing') # exception ဖြစ်ရင် ဒီလိုင်းကို ရောက်မလာဘူး
except ValueError as err:
    print(f'Error: {err}')

print('Program exits')
```

Output:

```
Error: Positive integer expected
Finish program
```

Exception `raise` လုပ်ရင် ကွန်းမန်ရေးထားတဲ့ လိုင်းကို မလုပ်ပါဘူး။ `except` ဘလောက်နဲ့ အောက်ဆုံး `print` လုပ်တဲ့အထိ ဆက်အလုပ်လုပ်သွားတယ်။ 2 ထည့်ပြီး စမ်းကြည့်ရင် exception မဖြစ်ဘူး။ `try` ဘလောက် ပြီးတဲ့ထို့မှုန်အတိုင်း လုပ်ဆောင်တယ်။ `Exception` မဖြစ်တော့ `except` ဘလောက်အလုပ် မလုပ်ဘဲ ကျော်သွားပါတယ်။

Exception-handling ကို အသုံးပြုပြီး အင်တိဂျာ တစ်ခု `input` ထည့်ခိုင်းတဲ့ ဖန်ရှင်ကို အခုလိုရေးနိုင်ပါတယ်။

```
# File: read_int.py
def read_int(prompt):
    while True:
```

```

try:
    return int(input(prompt))
except ValueError as err:
    print('Error: Non-integer data!')

# test
num = read_int('Enter number: ')
print(num)

```

အင်တိဂျာမဟုတ်တဲ့ တန်ဖိုး ထည့်ပေးရင် `int(input(prompt))` မှာ `ValueError` exception ဖြစ်ပြီး handle လုပ်တဲ့ ဘလောက်ကို ရောက်သွားမှုပါ။ ဒီတော့ ဖန်ရှင်က `return` မဖြစ်ဘူး။ `while` loop နောက်တစ်ကြိမ် ထပ်ကျော်ပါတယ်။ အင်တိဂျာ ပြောင်းလိုက်ပဲ တန်ဖိုး ထည့်ပေးမယ့် exception မဖြစ်ဘဲ `return` လုပ်ရင် ဖန်ရှင်ခေါ်တဲ့ဆိုကို ချက်ချင်း ပြန်ရောက်သွားတဲ့ အတွက် loop ကနေလည်း ထွက်သွားစေတယ်။

လိုက်ဘရီ ဖန်ရှင်တွေ အသုံးပြုတာပဲဖြစ်ဖြစ် ကိုယ်ပိုင် ဖန်ရှင် သတ်မှတ်တာပဲ ဖြစ်ဖြစ် exception တွေနဲ့ exception-handling အခြေခံအဆင့် နားလည်ထားဖို့ လိုအပ်တယ်။ Exception-handling နဲ့ ပါတ်သက်ပြီး အခန်း (၁၀) မှာ ဒီထက် ကျယ်ကျယ်ပြန့်ပြန့် ဆက်လက် လေ့လာရအုံမှုပါ။ အခုတော့ ဒီလောက်နဲ့ ခဏရုပ်ထားပြီး လက်တွေနဲ့ ပိုနဲ့စပ်တဲ့ အသုံးချေပမာတရချို့ကို ဆက်ကြည့်ရအောင်။

၈.၄ ဖန်ရှင်နှင့် ပရီဂရမ်ဒီဇိုင်း ဥပမာ (၁) “Insurance Premium”^{၁၁}

နှစ် နှစ်ဆယ် သက်တမ်းကာလ \$500,000 အသက်အာမခံထားရင် ကျန်းမာတဲ့ အသက် (၃၀) အရွယ် အမျိုးသမီး တစ်ယောက်အတွက် နှစ်စဉ်ပျေးမျှ အာမခံကြေး (premium) \$229 ဒေါ်လာ ကုန်ကျတယ်။ ရွယ်တူ အမျိုးသမီးတစ်ယောက် ဆိုရင်တော့ \$373 ဒေါ်လာပါ။ ဒါကယောကုပြောတာ။ အပိုင်မှာ အသက်အာမခံထားရင် သက်တမ်းကာလ၊ အကျိုးခံစားနှင့်မည့် ငွေပမာဏ (coverage)၊ ကျန်းမာရေး၊ အသက်အရွယ် စတဲ့ အချက်တွေပေါ် မူတည်ပြီး တစ်ညီးချင်းအတွက် အာမခံကြေး သီးသန့်တွက်ချက်တာပါ။

အာမခံသက်တမ်း နှစ်ဆယ်နှစ်အတွက်ပဲ စဉ်းစားပါမယ်။ အသက် (၃၀) အတွက် premium လိုပြောပေမဲ့ တကယ်တမ်းက အသက် (၁၈) နှစ် ကနေ (၃၀) (အပါအဝင်) အတွက် premium ကို ဆိုလိုတဲ့ တာပါ။ ထိုနည်းတူစွာ အသက် (၄၀)၊ (၅၀)၊ (၆၀) premium ဟာ (၃၁) နှစ် ကနေ (၄၀)၊ (၅၁) နှစ် ကနေ (၅၀)၊ (၅၁) နှစ် ကနေ (၆၀) ကြေး (အပါအဝင်) ကို ဆိုလိုတဲ့ဖြစ်တယ်။ (၁၈) နှစ်အောက်နဲ့ (၆၀) အထက်ဆိုရင်တော့ အကျိုးမဝင်ပါဘူး (အာမခံ ထားလို့ မရဘူး ယူဆပါ)။ ပေါ်မှု (၈.၄) နှင့် (၈.၄) တွင် ကြည့်ပါ။

Coverage Amount	Age 30	Age 40	Age 50	Age 60
\$250,000	\$142	\$193	\$392	\$989
\$500,000	\$205	\$307	\$685	\$1,781
\$1 million	\$325	\$526	\$1,227	\$3,375
\$2 million	\$593	\$984	\$2,388	\$6,758

တေဘဲလ် ၈.၁ နှစ်နှစ်ဆယ် အသက်အာမခံကြေး (၁)

Coverage Amount	Age 30	Age 40	Age 50	Age 60
\$250,000	\$162	\$224	\$499	\$1,375
\$500,000	\$251	\$360	\$891	\$2,567
\$1 million	\$408	\$628	\$1,681	\$4,952
\$2 million	\$749	\$1,190	\$3,267	\$9,660

တေဘာ် ၈.၂ နှစ်နှစ်ဆယ် အသက်အာမခံပြေား (ကျား)

မွေးသက္ကရာဇ်၊ အကျိုးခံစားလိုသည့် ပမာဏ (coverage amount)၊ ကျား/မ အလိုက် တစ်နှစ် ပျမ်းမျှ ပရီမီယံကြေး တွက်ပေးတဲ့ ပရီဂျမ်တစ်ခု အာမခံ အေးကျင့်တစ်ယောက် အတွက် ရေးပေးရမယ် ဆိုပါစို့။ အေးကျင့်က သူလိုချင်တဲ့ ပုံစံအတွက် ဥပမာတချို့ကို အခုလို ပြပါတယ်။

dob? 19-12-1980

Enter gender F (Female)/M (Male): M

Choose 1/2/3/4 for one of the coverage options:

1. \$250,000
2. \$500,000
3. \$1,000,000
4. \$2,000,000

3

Premium per year: 1681.00

dob? Feb-15-1990

Enter gender F (Female)/M (Male): F

Choose 1/2/3/4 for one of the coverage options:

1. \$250,000
2. \$500,000
3. \$1,000,000
4. \$2,000,000

4

Premium per year: 984.00

လုပ်ငန်းသဘောနဲ့ သက်ဆိုင်တဲ့ ဖန်ရှင်များ

ပရီဂျမ် တစ်ခု ဒီဇိုင်းလုပ် ရေးသားပုံအဆင့်ဆင့်ကို ဆက်လက်ဖော်ပြပါမယ်။ လုပ်ငန်းသဘောနဲ့ သက်ဆိုင် တဲ့ အပိုင်းနဲ့ input/output အပိုင်းခဲ့ ပြီးရေးလေ့ရှိတယ်။ (ကိုဘုဒ်ကန် ထည့်ပေးတာကို ဖတ်တာ၊ စခရင်မှာ ရလဒ်ကို ထုတ်ပြတာ စတာတွေကို input/output အပိုင်းမှာ ပါဝင်တယ်)။

လုပ်ငန်းသဘောနဲ့ ဆိုင်တဲ့ အပိုင်းကို အရင်ကြည့်ရအောင်။ Premium ကြေး ယေားနှစ်ခုပါ ဒေတာ တွေကို သိမ်းထားဖို့ လိုတယ်။ Row, column တွေနဲ့ တေဘာ်လုပ်မျိုး စီစဉ်ထားချင်ရင် nested list သုံးလို့ရပါတယ်။

```
# File: insurance_prem.py
from decimal import *

# Premium for male
MALE_PREM = [[Decimal('162.00'), Decimal('224.00'), # 1st row, $250,000
               
```

```

        Decimal('499.00'), Decimal('1375.00')],
        [Decimal('251.00'), Decimal('360.00'), # 2nd row, $500,000
         Decimal('891.00'), Decimal('2567.00')],
        [Decimal('408.00'), Decimal('628.00'),
         Decimal('1681.00'), Decimal('4952.00')],
        [Decimal('749.00'), Decimal('1190.00'),
         Decimal('3267.00'), Decimal('9660.00')]

# Premium for female
FEMALE_PREM = [[Decimal('142.00'), Decimal('193.00'),
                  Decimal('392.00'), Decimal('989.00')],
                  [Decimal('205.00'), Decimal('307.00'),
                   Decimal('685.00'), Decimal('1781.00')],
                   [Decimal('325.00'), Decimal('526.00'),
                    Decimal('1227.00'), Decimal('3375.00')],
                    [Decimal('593.00'), Decimal('984.00'),
                     Decimal('2388.00'), Decimal('6758.00')]]
```

ဒီထဲကနေ လိုအပ်တဲ့ premium ကြေးကို ကျေးမှု/မ၊ အသက်နဲ့ coverage ပေါ်မှုတည်ပြီး ထုတ်ယူရမှာပါ။ ကျေးမှု (၄၅) နှစ်၊ coverage (၅၀၀,၀၀၀) အတွက် premium ကြေးကို MALE_PREM[1][2] နဲ့ ယူရမှာပါ။

Premium ကြေးကို ဖန်ရှင်တစ်ခုနဲ့ အခုလိုမျိုး ယူလိုရသင့်တယ်။ အသက် အရွယ်၊ ကျေးမှု/မ၊ coverage ပမာဏ ထည့်ပေးရမယ်။

```

retrieve_prem(45, 'M', Decimal('500_000.00'))      # should get 891.00
retrieve_prem(35, 'F', Decimal('1_000_000.00'))    # should get 526.00
```

ဒီကိစ္စအတွက် ဖန်ရှင် ဘယ်လိုရေးထားလဲ ကြည့်ရအောင်

```

# File: insurance_prem.py
COVERAGES = [Decimal("250_000.00"), Decimal("500_000.00"),
              Decimal("1_000_000.00"), Decimal("2_000_000.00")]

FEMALE = 'F'
MALE = 'M'

def retrieve_prem(age, gender, coverage):
    if not (18 <= age <= 60):
        raise ValueError(f"Age of {age} yrs not applicable!")

    age_band = None
    if age <= 30:
        age_band = 0
    elif age <= 40:
        age_band = 1
    elif age <= 50:
        age_band = 2
```

```

elif age <= 60:
    age_band = 3

try:
    coverage_idx = COVERAGES.index(coverage)
except ValueError:
    raise ValueError(f'No such coverage amount, {str(coverage)}!')

if gender == FEMALE:
    return FEMALE_PREM[coverage_idx][age_band]
elif gender == MALE:
    return MALE_PREM[coverage_idx][age_band]

```

လိုအပ်တဲ့ constant တချို့ ကြော်ထားတယ်။ COVERAGES က coverage ပမာဏတွေ ပါတဲ့ list ဖြစ်တယ်။ ဘာအတွက်လဲ ခဏနေ တွေ့ရပါမယ်။ age, gender နဲ့ coverage က လိုအပ်တဲ့ ဖန်ရှင်ပါ ရာမိတာတွေပါ။ ဖန်ရှင် တစ်ခုဟာ လက်မခဲ့နိုင်တဲ့ သို့မဟုတ် မဖြစ်သင့်တဲ့ ပါရာမိတာ တန်ဖိုးတွေအတွက် အဖြစ်ထုတ်ပေးဖို့ မဖြစ်နိုင်ပါဘူး။ ဖန်ရှင်က သူလုပ်ဆောင်ရမဲ့ ကိစ္စကို အကြောင်းတစ်ခုခဲ့ကြောင့် မလုပ် ပေးနိုင်တဲ့အခါ exception raise လုပ်လေ့ရှိတယ်ဆိုတာ ဒီမိတိုင်ခင် စက်ရှုင်မှာ တွေ့ခဲ့ရတယ်။ အခါ ဖန်ရှင်ကလည်း အသက်အရွယ် လိုအပ်ချက် မကိုက်ညံရင် exception raise လုပ်တယ်။ Coverage ပမာဏ လေးမျိုးထဲမှာ မပါဝင်ရင်လည်း raise လုပ်ထားပါတယ်။ COVERAGES.index(coverage) က ValueError ဖြစ်ရင် coverage က list ထဲမှာ မရှိလို့ မှားနေလိုပဲ။ ဒီလိုဖြစ်ခဲ့ရင် except ဘလောက် ကလည်း ValueError ကိုပဲ raise လုပ်တယ်။ ဖြစ်တဲ့ပြဿနာကို တိတိကျကျ ဖော်ပြတဲ့မက်ဆော်ချုပ်ဖြစ်ဖို့ အတွက် အရေးကြီးပါတယ်။

```
ValueError('Something went wrong!')
```

ဆိုတာထက်

```
ValueError(f'No such coverage amount, {str(coverage)}!')
```

က ပြဿနာ၏ အရင်းအမြစ်ကို နားလည်ဖို့ အများကြီး အထောက်အကူ ပိုဖြစ်ပါတယ်။

age_band အတွက် သီးသန့်ဖန်ရှင်တစ်ခု ခဲ့ထုတ်လိုက်ရင် ပိုကောင်းနိုင်တယ်။ age ကနေ မှန်ကန် တဲ့ ကော်လဲ index ထုတ်ပေးကိစ္စ တစ်ခုကိုပဲ သီးသန့်လုပ်ဆောင်ပေးတာပေါ့။

```

def age_to_age_band(age):
    if not (18 <= age <= 60):
        raise ValueError(f"Age of {age} yrs not applicable!")
    if age <= 30:
        return 0
    elif age <= 40:
        return 1
    elif age <= 50:
        return 2
    elif age < 60:
        return 3

```

ဒီဖန်ရှင်က `age` နဲ့ သက်ခိုင်တာကို အမိကလုပ်တာဆိုတော့ `age` က ဘောင်ဝင်မဝင်လည်း ဒီမှာပဲ ဆုံးဖြတ်တာ ပိုသင့်တော်မယ်လို့ ယူဆနိုင်တယ်။ ဒါကြောင့် ဒီဖန်ရှင်မှာပဲ စစ်ပြီး exception raise ထားတယ်။ ခုနက `retrieve_prem` ဖန်ရှင်က ဒီလိုဖြစ်လာမယ်

```
def retrieve_prem(age, gender, coverage):
    try:
        age_band = age_to_age_band(age)
    except ValueError as age_err:
        raise age_err
    try:
        coverage_idx = COVERAGES.index(coverage)
    except ValueError:
        raise ValueError(f'No such coverage amount, {str(coverage)}!')
    if gender == FEMALE:
        return FEMALE_PREM[coverage_idx][age_band]
    elif gender == MALE:
        return MALE_PREM[coverage_idx][age_band]
```

`age` နဲ့ဆိုင်တဲ့ exception ဖြစ်ခဲ့ရင် အဲဒီ exception ကိုပဲ ပြန် `raise` လုပ်လိုက်တယ် (`raise age_err`)။ `Exception` ဖြစ်မဲ့ `input` တွေရော့၊ မဖြစ်ဘဲ အဆင်ပြေမဲ့ `input` တွေနဲ့ပါ မှန်/မမှန် စိစစ်ကြည့်ပါ။ အောက်ပါအတိုင်း တစ်ကြောင်းချင်း စမ်းကြည့်ပါ။

```
print(retrieve_prem(80, 'F', Decimal('1_000_000.00'))) # age error
print(retrieve_prem(50, 'F', Decimal('3_000_000.00'))) # coverage error
print(retrieve_prem(50, 'F', Decimal('1_000_000.00'))) # 1227.00
print(retrieve_prem(50, 'K', Decimal('1_000_000.00'))) # None (Why?)
```

နောက်ဆုံးတစ်ကြောင်းက `None` ဖြစ်နေတာ တွေ့ရတယ်။ ဘုံးကြောင့်လဲ အကြောင်းအရင်းကို နောက်ပိုင်းမှာ သိရှုမှုပါ။

Input/Output အပိုင်း

Input ဆုံးထားပေးရမယ်။ မွေးသက္ကရာဇ် (date of birth) , coverage amount နဲ့ ကျားမာ (gender) တို့ဖြစ်တယ်။ အသုံးပြုသူအနေနဲ့ အဆင်ပြေဆုံး၊ အလွယ်ကူဆုံးဖြစ်အောင်၊ အမှားအယွင်းနည်းနိုင်သမျှန်သုံးအောင် စဉ်းစားသင့်တယ်။

မွေးသက္ကရာဇ်ကို ကြည့်ရအောင်။ ရက်စွဲကို နိုင်ငံနဲ့ နေရာအော် ပေါ်မှုတည်ပြီး ဖော့မတ်အမျိုးမျိုးနဲ့ ရေးကြတယ်။ 04/05/2020, 04-05-2020, Apr-4-2020 စသည်ဖြင့်။ ခုနစ်ကို ကောန်းနှစ်လုံးပဲ ရေးတာလည်း ရှိတယ်။ ရက်နွဲလကို ရေးမှာ သုညမပါဘဲလည်း ရေးတယ်။ ဥပမာ 4/5/2020, 4/5/20, Apr-4-2020 ။ ဖော့မတ်တစ်မျိုးကိုပဲ သတ်သတ်မှတ်မှတ် သုံးသင့်တာ ဖြစ်ပေမဲ့ ဆော်ဖဲ့အပ်သူက ဖော့မတ် သုံးမျိုးနဲ့ ထည့်လို့ရအောင် လုပ်ပေးဖို့ တောင်းဆိုထားတယ်။

01-01-2024

01/01/2024

Jan-1-2024

မွေးသက္ကရာဇ်ကို ဒီဖော့မတ်သုံးမျိုးနဲ့ ပရိုကရမ်က လက်ခံရပါမယ်။ `input` ဖန်ရှင်က ကိုးဘုံးကထည့်ပေးတာကို `string` အနေနဲ့ ပြန်ပေးပါတယ်။

မွေးသက္ကရာဇ်ကနေ အသက်ကို တွက်ယူရမှုပါ။ ဒီအတွက် ဖန်ရှင် (ဥပမာ calc_age) သတ်မှတ်နိုင်တယ်။ နေ့ရက်၊ အချိန်နဲ့ သက်ဆိုင်တဲ့ အတွက်အချက်တွေ အတွက် စာသားကို အသုံးမပြုသင့်ဘူး။ date, datetime စတာတွေ သုံးသင့်တယ်။ ဒါကြောင့် စာသားကနေ မွေးသက္ကရာဇ်ကို ဖော်ပြတဲ့ date (သို့) datetime အော့သုတေသန ပြောင်းဖို့ လိပါမယ်။ '01-01-2024' ကနေ 1, 1, 2024 ကောင်းတွေ ရအောင် စာသားကို တစ်ဖြတ်ချင်း ဖြတ်ပြီး ပြောင်းမယ်။

```
>>> egstr = '123abc456abc789'
>>> egstr.split('abc')
['123', '456', '789']
>>> dt1 = '01-01-2024'
>>> parts1 = dt1.split('-')
>>> parts1
['01', '01', '2024']
>>> dt2 = '01/01/2024'
>>> parts2 = dt2.split('/')
>>> parts2
['01', '01', '2024']
>>> dt3 = 'Feb-02-2024'
>>> parts3 = dt3.split('-')
>>> parts3
['Feb', '02', '2024']
```

split ဖန်ရှင်က string တစ်ခုကို အပိုင်းတွေ ပိုင်းပေးတာပါ။ အပိုင်းတွေ အားလုံးကို list အနေနဲ့ ပြန်ပေးတယ်။ ဘယ် ကာရက်တာ (သို့) string နဲ့ ဗြားထားရင် ပိုင်းဖြတ်ချင်လဲ ထည့်ပေးလိုရတယ်။ အပေါ်မှာ 'abc', '-', '/' စတာတွေနဲ့ ပိုင်းဖြတ်ထားတယ်။

```
>>> dt1 = '01-01-2024'
>>> parts1 = dt1.split('-')
>>> date(int(parts1[2]), int(parts1[1]), int(parts1[0]))
datetime.date(2024, 1, 1)
```

'Feb-02-2024' ဖော်မတ်က နည်းနည်း ပို့ချုပ်မယ်။ လက 'Jan', 'Feb' စသည်ဖြင့် စာသားဖြတ်နောက်တယ်။ အခုလို dictionary တစ်ခု ရှိရင် လန်းမည်ကနေ သက်ဆိုင်တဲ့ ကောင်းကို အလွယ်တကူ ရနိုင်ပါတယ်

```
>>> mths = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
...           'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
...           'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
>>> mths['Jan']
1
>>> mths['Dec']
12
```

ဒီလိုဆိုရင် date အော့သုတေသန ရဖို့အတွက်လည်း သိပ်မခက်တော့ဘူး။ ဥပမာ

```
>>> dt4 = 'Dec-26-2024'
>>> parts4 = dt4.split('-')
```

```
>>> date(int(parts4[2]), mths[parts4[0]], int(parts4[1]))
datetime.date(2024, 12, 26)
```

နေ့ကြော် ဖော်မတ် သုံးမျိုးနဲ့ လက်ခံနိုင်တဲ့ ဖန်ရှင်ကို ရိုးရိုးရှင်းရှင်းလေးပဲ အခုလို

```
# File: insurance_prem.py
from datetime import *

MONTHS = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
          'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
          'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

def parse_date_str(dtstr):
    parts = dtstr.replace('/', '-').split('-')
    if parts[0].isdigit():
        return date(int(parts[2]), int(parts[1]), int(parts[0]))
    else:
        return date(int(parts[2]), months[int(parts[0])], int(parts[1]))
```

သတ်မှတ်ပါမယ်။ replace ဖန်ရှင်သုံးပြီး / ကို - နဲ့ အစားထိုးလိုက်တယ်။ ဒါဆိုရင် ဖော့မတ် နှစ်ခုပဲ စဉ်းစားဖို့လိုတော့မယ် (10/01/2024 ကနေ 10-01-2024 ဖြစ်သားမှာပါ။)။ ပြီးတော့မှာ split လုပ်တယ်။ ရှေ့လုံးအပိုင်း parts[0] ကတန်း ဟုတ်လား isdigit နဲ့ စစ်တားတယ်။ ဖော့မတ် အမှုနဲ့ စမ်းကြည့်ရင် date အော့သုတေသနပါတယ်

```
print(parse_date_str('12-03-1995'))  
print(parse_date_str('12/03/1995'))  
print(parse_date_str('Mar-12-1995'))
```

```
# Spelling error: Fab instead of Feb
print(parse_date_str('Fab-12-1995'))
```

Error Output:

Dictionary ထဲမှာ Fab ကို မရှိတဲ့အတွက် ဒီပြဿနာဖြစ်တယ်။ မဖြစ်နိုင်တဲ့ ခုနှစ်၊ လ၊ ရက် တန်ဖိုးတော်းရင် `ValueError` ဖြစ်တယ် (`ညာမှာ parse_date_str('30-02-1995'))။`

အချေဖြေပြခဲ့တဲ့ နည်းအပြင် အခြားနည်းလမ်းတွေလည်း ရှိရမှာပါ။ `datetime` ကလပ်စူးမှာ နေရက် ကို စာသားကနေ `datetime` ပြောင်းပေးတဲ့ `strptime` ဖန်ရှင်းရှိတယ်။ သူကို အသံးပြုပြီး `parse_date_str` ကို ဘယ်လိုရေးလို့ ရမလဲ။ စဉ်းစားကြည့်ရအောင် ...

```
>>> dtstr1 = 'Jan-01-2024'
>>> datetime.strptime(dtstr1, "%b-%d-%Y")
datetime.datetime(2024, 1, 1, 0, 0)
>>> dtstr2 = '30-12-2024'
>>> datetime.strptime(dtstr2, "%d-%m-%Y")
datetime.datetime(2024, 12, 30, 0, 0)
>>> dtstr3 = '30/Dec/2024'
>>> datetime.strptime(dtstr3, "%d/%b/%Y")
datetime.datetime(2024, 12, 30, 0, 0)
```

ဒီဖန်ရှင်က စာသားနဲ့ ဖော်ပြထားတဲ့ အချို့နှင့်ရက်ကို `datetime` အော့ဘ်ဂျက် ပြောင်းပေးဖို့ ဖော့မတ် ကုဒ် (format code) လို့ခေါ်တဲ့ % နဲ့ စတဲ့ ကာရက်တာတွေကို လက်ခံတယ်။ `%d` က ရက်၊ `%m` က လ ကို ကဏ်း တ်လုံး (သို့) နှစ်လုံးနဲ့ ဆိုတဲ့ အဓိပါယ် (ဥပမာ ကိုရက်နေ့ကို 09 သို့မဟုတ် ၁၊ ငါးလပိုင်း ကို 05 သို့မဟုတ် 5)။ `%Y` က ခုနှစ်ကို ကဏ်းလေးလုံးနဲ့ ရေးတယ်လို့ ဆိုလိုတာပါ။ `%b` ကတော့ လရဲ့ နံမည်ကို Jan, Feb, Mar စသည်ဖြင့် အတိုကောက်ရေးတယ်လို့ ဆိုလိုတာပါ။ 'Jan-01-2024' ကို ပြောင်းမယ်ဆိုရင် သူရဲ့ဖော့မတ်နဲ့ ကိုက်ညီတဲ့ '`%b-%d-%Y`' ကို ဖန်ရှင်ခေါ်တဲ့အခါ ထည့်ပေးရပါမယ်။ '30/Dec/2024' ဆိုရင် '`%d/%b/%Y`'၊ '30-12-2024' အတွက် '`%d-%m-%Y`' ဖြစ်မှာပါ။ ဖော့မတ် ကုဒ် သုံးထားတဲ့ `parse_date_str2` ကို အခုလို သတ်မှတ်လိုပါမယ်

```
formats = ['%d-%m-%Y', '%d/%m/%Y', '%b-%d-%Y']
def parse_date_str2(dtstr):
    for fmt in formats:
        try:
            return datetime.strptime(dtstr, fmt).date()
        except ValueError:
            pass
    # ဖော့မတ် သုံးမျိုးလုံးနဲ့ မကိုက်ညီလို့ ပြောင်းလိုမရရင် ဒီကိုရောက်လာမယ်
    raise ValueError(f"{dtstr} doesn't match any of acceptable formats")
```

`dtstr` ကို ဖော့မတ်တစ်ခုချင်း ပြောင်းကြည့်တယ်။ ပြောင်းလိုရရင် `return` ဖြစ်သွားမယ်။ ပြောင်းလိုမရရင် `strptime` က `ValueError` raise လုပ်တယ်။ ဖော့မတ်တစ်ခုနဲ့ ပြောင်းလိုမရရင် နောက်တစ်ခု နဲ့ရှိနိုင်ပါတယ်။ ဒါကြောင့် `ValueError` ဖြစ်ခဲ့ရင် `pass` ပဲ လုပ်ပေးလိုက်တယ် (ဘာမှုလုပ်စရာမလိုရင် `pass` စတိတ်မနဲ့ သုံးတာပါ)။ ဖော့မတ် သုံးမျိုးလုံးနဲ့ အဆင်မပြောင်တော့ တစ်ခုခု မှားနေလိုပဲ။ ဒါကြောင့် `return` မဖြစ်ဘဲ `for` loop ပြီးတဲ့ထိ ရောက်လာခဲ့ရင် `ValueError` exception ဖြစ်အောင် `raise` လပ်ထားတာ။ `strptime` ပြန်ပေးတာက `datetime` ပါ။ `date` လိုချင်ရင် `date()` ဖန်ရှင် ဆက်ခေါ်လို့ ရတယ်။

```
datetime.strptime(dtstr, fmt).date()
```

ပရိုဂရမ်တည်ဆောက်တဲ့အခါ ကိုစွဲတစ်ခုကို ဖြော်ရှင်းလိုရတဲ့ နည်းလမ်းက တစ်ခုတည်းပဲ ဖြစ်လေ့မရှိတာကတော့ ထုံးစံပါပဲ။ ဖြစ်နိုင်တဲ့ ဖြော်ရှင်းနည်းတွေထဲက အသင့်တော်ဆုံးတစ်ခုကို ရွှေးချယ် အသံးပြုရတယ်ပါ။ နည်းလမ်းတစ်ခုက လက်ရှိမှာ အသင့်တော်ဆုံး ဖြစ်ပေမဲ့ နောင်တစ်ချိန်မှာ သူထက် ပိုကောင်းတာ၊ ပိုသင့်တော်တာလည်း ရှိလာနိုင်ပါတယ်။

မွေးသက္ကရာဇ် ဖတ်တဲ့ ဖန်ရှင်နဲ့ အသက်တွက်တဲ့ ဖန်ရှင်ကို အခုလို သတ်မှတ်ပါမယ်

```
def read_date():
    while True:
        try:
            return parse_date_str(input('dob? '))
        except Exception as err:
            print("Incorrect date. Please enter the date again.")
            print('Error is probably: ' + str(err))

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1
```

read_date ၏ ဖော်မတ် မမှန်မချင်း ပြန်ထည့်ခွင့်းမှုပါ။

ဒါကတော့ coverage option အတွက် ဖန်ရှင် နဲ့ gender လက်ခံတဲ့ ဖန်ရှင်တွေပါ

```
def read_coverage():
    print("Enter 1/2/3/4 for one of the coverage options: ")
    print("1. $250,000")
    print("2. $500,000")
    print("3. $1,000,000")
    print("4. $2,000,000")
    while True:
        try:
            choice = int(input())
            if 1 <= choice <= 4:
                return COVERAGES[choice - 1]
            else:
                print('Incorrect choice! Accept only 1 to 4!')
        except ValueError as err:
            print('Incorrect choice! Accept only 1 to 4!')
```

တစ်ကနေ လေးထိုး လိုချင်တာ။ ကိုနဲ့ပြည့်မဟုတ်တာ (ဥပမာ 12yz, 12.3) ထည့်ရင် int() ပြောင်းတဲ့နေရာ exception ဖြစ်မယ်။ Handle အပိုင်းမှာ ဒီအတွက် အယ်ရာပြပေးတယ်။ အင်တီဂျာတော့ ထည့်တာမှန်တယ်၊ ဒါပေမဲ့ တစ်နဲ့ လေးကြား မဟုတ်ရင်လည်း လက်ခံလို့ မဖြစ်ဘူး။ ဒီအတွက် if နဲ့ စစ်ပြီး အယ်ရာပြပေးတယ်။

```
def read_gender():
    while True:
        gender = input('Enter gender F (Female)/M (Male): ')
        if gender in ['F', 'M']:
            return gender
```

```

else:
    print('Invalid gender! Accept only F or M! ')

```

ကိုဘုဒ် input ဖတ်ဖို့ လိုအပ်တဲ့ ဖန်ရှင်အားလုံးလည်း ရှိပြီ။ နောက်ဆုံးတော့ ပရီဂရမ်တစ်ခုလုံးရဲ့ top level ဖန်ရှင်က အခုလိုပါ။ ရှိခိုး ရှင်းရှင်းလေးပါပဲ။

```

def run_prem_program():
    while True:
        dob = read_date()
        gender = read_gender()
        coverage = read_coverage()

    try:
        print(retrieve_prem(age_today(dob),
                            gender,
                            coverage))
    except ValueError as dm_err:
        print(f'Domain error: {dm_err}')

    quit_or_cont = input(('Press enter/return to continue.'
                         ' To quit the program, enter quit.'))
    if quit_or_cont == 'quit':
        break

```

retrieve_prem မှာ exception ဖြစ်ရင် handle လုပ်တဲ့အပိုင်းက ဘာကြောင့် error ဖြစ်လဲ ပြေားပါတယ်။ ထွက်ချုပ်ရင် quit ထည့်ပေးရမယ်။ Enter/return နှိပ်ရင် နောက်တစ်ခါ input ပြန်တောင်းပြီး ဆက်အလုပ်လုပ်နေမှာပါ။ ကိုယ်ပိုင် သတ်မှတ်တဲ့ ဖန်ရှင်အားလုံးကနေ ဖြစ်လိုနိုင်တဲ့ exception အားလုံးကို သင့်တော်တဲ့နေရာမှာ handle လုပ်ထားတဲ့အတွက် ဘာပြဿနာမှ မဖြစ်နိုင်တော့ဘူးလို ယူဆ စရာ အကြောင်းရှိပါတယ်။ ဒါပေမဲ့ အားကိုယ် မမျှော်မှန်းနိုင်တဲ့ exception တော်လည်း ရှိနိုင်ပါသေးတယ်။ (ဥပမာ ဂွန်ပျူးတာ operating system နဲ့ ဆက်ဆိုင်တဲ့ OSError မဖို့မရ မလုပ်လောက်ရင် MemoryError)။ ဒီလို exception တွေဖြစ်ရင် ပရီဂရမ် ပုံမှန်ဆက်အလုပ် လုပ်နိုင်ဖို့လည်း မဖြစ်နိုင်တာ များပါတယ်။ ပရီဂရမ်ကို ဆက် run လို မရတော့လောက်အောင် မဆိုးပေမဲ့ ကိုယ်မမျှော်လင့်ထားမိတဲ့ exception ဖျိုးလည်း ဖြစ်နိုင်တာပဲ။ ဒီလိုအခါဖျိုးမှာ တစ်ခုခဲတော့ ပြဿနာဖြစ်သွားတယ် ပရီဂရမ်ရေးတဲ့သူကို ဆက်သွယ်ပါ ဆိုတာမျိုး user ကို မက်ဆော်ချုံ ကောင်းကောင်းမွန်မွန် ပြေားချင်တယ်ဆိုရင် အခုလို လုပ်လိုရပါတယ်။

```

def run_prem_program():
    while True:
        try:
            dob = read_date()
            gender = read_gender()
            coverage = read_coverage()

        try:
            print(retrieve_prem(age_today(dob),
                                gender,

```

```

                coverage))
except ValueError as dm_err:
    print(f'Domain error: {dm_err}')

quit_or_cont = input('Press enter/return to continue.
                     ' To quit the program, enter quit.')
if quit_or_cont == 'quit':
    break
except Exception as sys_err:
    print(f'Application/System Error: {sys_err}')
    print('Please kindly report to system owner.')

```

အပြင် try: ... except: က အခြား မမျှော်လင့်ထားတဲ့ exception အားလုံးကို handle လုပ်ပေးပါတယ်။ except Exception ကတော့ သီးသန့် exception တစ်မျိုးအတွက်မဟုတ်ဘဲ ဘယ်အမျိုးအထားဖြစ်ဖြစ် handle လုပ်ချင်တဲ့အခါ သုံးရတာပါ။ ဒါနဲ့ ပါတ်သက်ပြီး Exception-handling အခန်းမှာ ဆက်လက်လေ့လာကြပါမယ်။

နောက်ပိုင်းမှာတော့ retrieve_prem မှာ ပထောက်မှုများ ပါတယ် (သို့ ဖြစ်သင့်တယ်) ဆိုတာ နားလည်လာပါမယ်။

```

def retrieve_prem(age, gender, coverage):
    age_band = age_to_age_band(age)
    try:
        coverage_idx = COVERAGES.index(coverage)
    except ValueError:
        raise ValueError(f'No such coverage amount, {str(coverage)}!')
    if gender == FEMALE:
        return FEMALE_PREM[coverage_idx][age_band]
    elif gender == MALE:
        return MALE_PREM[coverage_idx][age_band]

```

age_to_age_band မှာ ဖြစ်တဲ့ exception ကို handle မလုပ်ထားတော့ဘူး။ ဒါပေမဲ့ ပရိုဂရမ်တစ်ခုလုံးရဲ့ လုပ်ဆောင်တဲ့ ဘီးယောကတော့ မပြောင်းလဲသွားပါဘူး။ နို့အတိုင်းပါပဲ။ ဘူးကြောင့်လဲ အပိုင်း (၂) မှာ ဆက်ကြည့်ပါမယ် ...

ပရိုဂရမ် အစအဆုံး လေ့လာကြည့်ပါ စမ်းကြည့်ပါ ...

```

from decimal import *
from datetime import *

MALE_PREM = [[Decimal('162.00'), Decimal('224.00'), # 1st row, $250,000
              Decimal('499.00'), Decimal('1375.00')],
              [Decimal('251.00'), Decimal('360.00'), # 2nd row, $500,000
              Decimal('891.00'), Decimal('2567.00')],
              [Decimal('408.00'), Decimal('628.00'),
              Decimal('1681.00'), Decimal('4952.00')],
              [Decimal('749.00'), Decimal('1190.00'),
              Decimal('3267.00'), Decimal('9660.00')]]

```

```

# Premium for female
FEMALE_PREM = [[Decimal('142.00'), Decimal('193.00'),
                 Decimal('392.00'), Decimal('989.00')],
                [Decimal('205.00'), Decimal('307.00'),
                 Decimal('685.00'), Decimal('1781.00')],
                [Decimal('325.00'), Decimal('526.00'),
                 Decimal('1227.00'), Decimal('3375.00')],
                [Decimal('593.00'), Decimal('984.00'),
                 Decimal('2388.00'), Decimal('6758.00')]]


COVERAGES = [Decimal("250_000.00"), Decimal("500_000.00"),
             Decimal("1_000_000.00"), Decimal("2_000_000.00")]


FEMALE = 'F'
MALE = 'M'

MONTHS = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
          'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
          'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

def retrieve_prem(age, gender, coverage):
    try:
        age_band = age_to_age_band(age)
    except ValueError as age_err:
        raise age_err
    try:
        coverage_idx = COVERAGES.index(coverage)
    except ValueError:
        raise ValueError(f'No such coverage amount, {str(coverage)}!')
    if gender == FEMALE:
        return FEMALE_PREM[coverage_idx][age_band]
    elif gender == MALE:
        return MALE_PREM[coverage_idx][age_band]

def age_to_age_band(age):
    if not (18 <= age <= 60):
        raise ValueError(f"Sorry. Age of {age} yrs not applicable!")
    if age <= 30:
        return 0
    elif age <= 40:
        return 1
    elif age <= 50:
        return 2
    elif age <= 60:
        return 3

```

```
    return 3

def parse_date_str(dtstr):
    parts = dtstr.replace('/', '-').split('-')
    if parts[0].isdigit():
        return date(int(parts[2]), int(parts[1]), int(parts[0]))
    else:
        return date(int(parts[2]), MONTHS[parts[0]], int(parts[1]))

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

def read_date():
    while True:
        try:
            return parse_date_str(input('dob? '))
        except Exception as err:
            print("Incorrect date. Please enter the date again.")
            print('Error is probably: ' + str(err))

def read_coverage():
    print("Enter 1/2/3/4 for one of the coverage options: ")
    print("1. $250,000")
    print("2. $500,000")
    print("3. $1,000,000")
    print("4. $2,000,000")
    while True:
        try:
            choice = int(input())
            if 1 <= choice <= 4:
                return COVERAGES[choice - 1]
            else:
                print('Incorrect choice! Accept only 1 to 4!')
        except ValueError as err:
            print('Incorrect choice! Accept only 1 to 4!')
```

```

def read_gender():
    while True:
        gender = input('Enter gender F (Female)/M (Male): ')
        if gender in ['F', 'M']:
            return gender
        else:
            print('Invalid gender! Accept only F or M! ')

def run_prem_program():
    while True:
        dob = read_date()
        gender = read_gender()
        coverage = read_coverage()

        try:
            print(retrieve_prem(age_today(dob),
                                gender,
                                coverage))
        except ValueError as dm_err:
            print(f'Domain error: {dm_err}')

        quit_or_cont = input(('Press enter/return to continue.'
                            ' To quit the program, enter quit.'))
        if quit_or_cont == 'quit':
            break

run_prem_program()

```

၈.၅ ဖန်ရှင်နှင့် ပရီဂရမ်ခိုဝင်း ဥပမာ (၂) “Colorful House”^{၁၁}

အခုတစ်ခါ ပုံ (၈.၁) မှ ရောင်စုအိမ်ပုံလေး ရေးဆွဲကြပါမယ်။ အိမ်အမြင့်ဟာ အိမ်အောက်ခြေကန် အိမ်ခေါင်မှိုး ထိပ်ထိ ဖြစ်တယ်။ အမြင့်နဲ့ အကျယ်ဟာ အိမ်အရွယ်အစားကို ဖော်ပြတယ်။ ပုံဆွဲတဲ့အခါ အိမ် အရွယ်အစားနဲ့ အရောင် စိတ်ကြိုက် အလွယ်တကူ ပြောင်းလို့ရသုတေသနတယ်။ အိမ်တည်နေရာကို ဘယ်ဘက် အပေါ်ထောင့်အမှတ် နဲ့ဖော်ပြုမယ် (ပုံတွင်ကြည့်ပါ)။ အိမ်တစ်အိမ်လုံး ကွက်တိဝင်တဲ့ ထောင့်မှုန်စတုဂံ ရဲ့ (x, y) လို့လည်း ယူဆနိုင်တယ်။

ခေါင်မှိုးအမြင့် (အပြာရောင် ကြိုက်ပုံ) က အိမ်တစ်ခုလုံး အမြင့် လေးပုံတစ်ပုံ။ ကျွန်ုတ်လေးပုံ သုံးပုံက နံရံအမြင့် (အစိမ်းနေရာင် ထောင့်မှုန်စတုဂံ) ဖြစ်တယ်။ နံရံကို အညီအမျှ လေးပိုင်းခွဲပြီး တစ်ပိုင်းစီရဲ့ ဗဟို နဲ့ ပြတ်ငါးပေါက်ဗဟို တစ်ထပ်တည်းကျအောင် နေရာချထားရပါမယ်။

draw_window

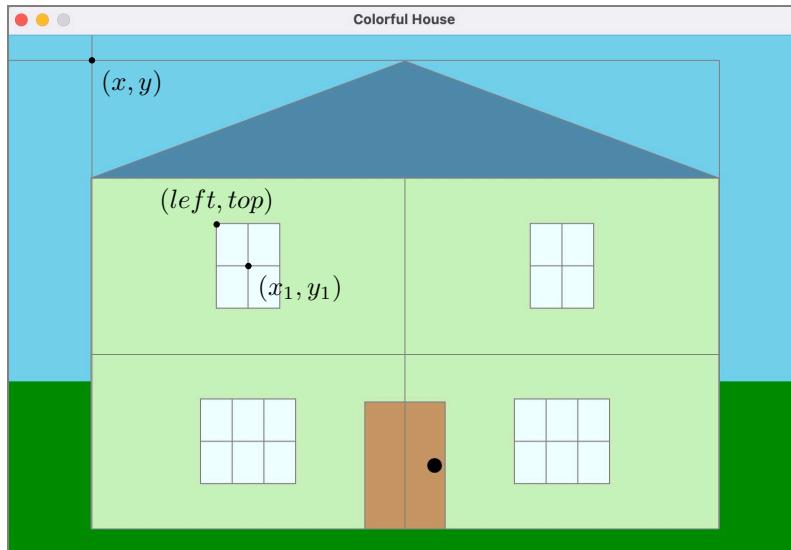
ပြတင်းပေါက်တွေဆဲဖို့ draw_window ဖန်ရှင် သတ်မှတ်သင့်တယ်။ ပြတင်းပေါက် အရွယ်အစား (မျှန်ချပ် အရေအတွက်) ကို $rows \times columns$ နဲ့ ဖော်ပြနိုင်တယ်။ အပေါ်ပြတင်းပေါက် နှစ်ခုက 2 \times 2 ဖြစ်ပြီး အောက်နှစ်ခုက 2 \times 3 ။ အောက်ဖော်ပြပါ draw_window မှာ x , y , row , col တို့ဟာ ပြတင်းပေါက် ဗဟိုမှတ်, အရွယ်အစားနဲ့ အရောင် တို့အတွက် ပါရောမိတာတွေဖြစ်တယ်။ နံရုံး ပထမတစ်ပိုင်းရဲ့ ဗဟိုမှတ် ဟာ (x_1, y_1) ဆိုပါမိ (ပုံတွင်ကြည့်ပါ)။ ပြတင်းပေါက်ကို အခုလို

```
| draw_window(x1, y1, 2, 2);
```

ဆွဲနိုင်ပါတယ်။ နံရုံးတစ်ပိုင်းစီအတွက် ဗဟိုမှတ်ရှာတာ သိပ်မခက်ဘူး။ ဒါကြောင့် ဗဟိုမှတ်ကို ပါရောမိတာ အနေနဲ့ ထားတာ အဆင့်ပြေတယ်။ (ပြတင်းတစ်ပေါက်ချင်းအတွက် သူရဲ့ ဘယ်ဘက်အပေါ် ထောင့်စွန်း မှတ် (x, y) တန်ဖိုးကို တွက်ကြည့်ပါ။ ဗဟိုမှတ်တွက်တာလောက် မလွယ်ကူတာ တွေ့ရလိမ့်မယ်)။

```
# File: colorful_house.py
import arcade
from arcade.color import *

# x, y က ပြတင်းပေါက် အလယ်မှတ်ပါ ပုံထဲမှာ  $(x_1, y_1)$  နဲ့ပြထားတယ်
def draw_window(x, y, row, col):
    pane_width = 30
    pane_height = 40
    window_width = pane_width * col
    window_height = pane_height * row
    top = y - window_height / 2
    left = x - window_width / 2
    for i in range(row):
        for j in range(col):
```



ပုံ ၈.၁ ရောင်ခံအောမ်လေး

```

arcade.draw_lrtb_rectangle_filled(left + pane_width * j,
                                  left + pane_width * (j+1),
                                  top + pane_height * (i+1),
                                  top + pane_height * i,
                                  AZURE_MIST)
arcade.draw_lrtb_rectangle_outline(left + pane_width * j,
                                   left + pane_width * (j+1),
                                   top + pane_height * (i+1),
                                   top + pane_height * i,
                                   GRAY)

```

ပြတင်းပေါက် မှန်ချပ်တွေ ဆွဲတာက အနေး (၇) တဗျာက်နာ (၁၀၀) checkerboard လိုပါပဲ။ တက္က် ချင်းရဲ့ left, right, top, bottom တွေ loop တစ်ကျော်တိုင်းမှာ မှန်အောင်တွက်နှင့်ရင် ပြတင်းပေါက် ဆွဲလို့ ရုံးပါ။ မှန်တစ်ချပ် အကျယ်၊ အမြင့် 30, 40 သတ်မှတ်တယ်။ ပြတင်းပေါက်တစ်ခုလုံး အကျယ်၊ အမြင့်ကို

```

window_width = pane_width * col
window_height = pane_height * row

```

ဖော်မျှလာနဲ့ တွက်ရပါမယ်။ ပြတင်းပေါက် အလယ်မှတ် x, y ကနေ ဘယ်ဘက် အပေါ်ထောင့်စွန်းမှတ် ကို ရှာမယ်ဆိုရင်

```

top = y - window_height / 2
left = x - window_width / 2

```

(ပုံကြိုးပြီး x_1, y_1 အမှတ်ကနေ $left, top$ အမှတ်ကို ဘယ်လိုရှာမလဲ စဉ်းစားကြည့်ပါ)။ မှန်ချပ်တစ် ချပ်ချင်း ကိုထြုဒိန်တွေကို $i \neq j$ (မှန်ချပ်ရဲ့ row နဲ့ column နံပါတ်) ပေါ်မှတည်ပြီး အခုလို့ တွက် လို့ရမယ်

```

left + pane_width * j,           # မှန်ချပ်ရဲ့ left x
left + pane_width * (j+1),      # မှန်ချပ်ရဲ့ right x
top + pane_height * (i+1),      # မှန်ချပ်ရဲ့ bottom y
top + pane_height * i,          # မှန်ချပ်ရဲ့ top y

```

ဥပမာ $(left, top) = (100, 120)$ ဆိုပါစို့။ ညာဘက်အောက် မှန်ချပ်အတွက် $i = 1, j = 1$ ဖြစ်တယ် ($i \neq j$ တန်ဖိုး သုညာက စတယ်ဆိုတာ အမှတ်ရပါ)။ အဲဒီမှန်ချပ်ရဲ့ ကိုထြုဒိန်တွေက

```

100 + 30 * 1           = 130
100 + 30 * (1 + 1)     = 160
120 + 40 * (1 + 1)     = 200
120 + 40 * 1           = 160

```

ဖြစ်ပါတယ်။

draw_house

draw_house ကို အောက်ပါအတိုင်း သတ်မှတ်ထားတယ်။ $left \neq top$ က အိမ်တည်နေရာ (x, y) လက်ခံတဲ့ ပါရာမိတာတွေပါ။ $width \neq height$ က အိမ်တစ်ခုလုံးရဲ့ အကျယ်နဲ့ အမြင့်အတွက်။

```

# left = x top = y (ပုံစံကြည်ပါ)
def draw_house(left, top, width, height):
    middle = left + width / 2
    wall_top = top + height / 4
    right = left + width
    wall_bottom = top + height

    arcade.draw_lrtb_rectangle_filled(left, right,
                                      wall_bottom, wall_top,
                                      TEA_GREEN)
    arcade.draw_lrtb_rectangle_outline(left, right,
                                      wall_bottom, wall_top,
                                      GRAY)

    arcade.draw_triangle_filled(left, wall_top,
                               right, wall_top,
                               middle, top,
                               AIR_FORCE_BLUE)
    arcade.draw_triangle_outline(left, wall_top,
                               right, wall_top,
                               middle, top,
                               GRAY)

    left_windows_x = left + width/4
    right_windows_x = left + width * 3/4

    wall_height = height * 3/4
    upper_windows_y = wall_top + wall_height/4
    lower_windows_y = wall_top + wall_height/4 * 3

    draw_window(left_windows_x, upper_windows_y, 2, 2)
    draw_window(left_windows_x, lower_windows_y, 2, 3)
    draw_window(right_windows_x, upper_windows_y, 2, 2)
    draw_window(right_windows_x, lower_windows_y, 2, 3)

    door_width = 76
    door_height = 120
    door_top = wall_bottom - door_height
    door_left = left + width/2 - door_width/2
    draw_main_door(door_left, door_top,
                   door_width, door_height,
                   WOOD_BROWN)

```

အိမ်ခေါင်မှူး ထိပ်စန်းမှတ်က အိမ်ကို အလယ်ကနေ ပိုင်းဖြတ်တဲ့ ဒေါင်လိုက်မျှင်းပေါ်မှာပါ (ခေါက်ချိုးညီမျှင်း)။ ဒီမျှင်းပေါ်က x အမှတ်ကို middle လို့ ထားမယ်။ သူတေနဖိုးက

```

middle = left + width / 2

ဖြစ်တယ်။ နံရုံ အပေါ်အနာဂတ်၏ y အမှတ်ကို wall_top လို့ ထားတယ်။ သူတန်ဖိုးက
wall_top = top + height / 4

ဖြစ်မယ်။ top ကို အိမ်အမြင့်ရဲ့ လေးပုံတစ်ပုံ ပေါင်းပေးရမှာပါ။ နံရုံ အောက်ဘက်အနာဂတ်၏ y အမှတ်ကို
wall_bottom လို့ ထားတယ်။ သူတန်ဖိုးက
wall_bottom = top + height

နံရုံ ညာဘက်အနာဂတ်၏ x အမှတ်ကို right လို့ ထားတယ်။ သူတန်ဖိုးက
right = left + width

(ရေပြင်ညီမျဉ်းပေါ်က အမှတ်အားလုံး y တန်ဖိုး တူညီတယ်၊ ဒေါင်လိုက်မျဉ်းပေါ်က အမှတ်အားလုံး x
တန်ဖိုး တူညီတယ်လို့ သိတယ်ပါ။)

ခေါင်မိုးကို draw_triangle_filled နဲ့ အခုလိုခွဲခိုင်ပါတယ်။ ပြိုက်ရဲ့ ထောင့်စွာနှင့်မှတ်တွေ ထည့်
ပေးရမှာပါ။ (ပြိုက်အနာဂတ်ကို draw_triangle_outline နဲ့ ဆွဲတယ်။ ရွှေ့မျက်နှာ draw_house
မှာ ကြည့်ပါ။)

# draw roof
arcade.draw_triangle_filled(left, wall_top,      # ဘယ်ထောင့်စွာနှင့်
                            right, wall_top,    # ညာထောင့်စွာနှင့်
                            middle, top,        # အပေါ်ထိပ်စွာနှင့်
                            AIR_FORCE_BLUE)

```

ပြတင်းပေါက်တွေရဲ့ အလယ်မှတ်တွေကို အောက်ပါအတိုင်း တွေက်ထားတယ်။ ဘယ်ဘက် ပြတင်းနှစ်
ပေါက် အလယ်မှတ် x တန်ဖိုး တူတယ်။ ညာဘက်နှစ်ခု အလယ်မှတ်လည်း x တန်ဖိုး တူတယ်။ အပေါ်
ပြတင်းနှစ်ပေါက် အလယ်မှတ် y တန်ဖိုး တူတယ်။ အောက်နှစ်ခု အလယ်မှတ်လည်း y တန်ဖိုး တူတယ်။
နံရုံအမြင့်က အိမ်အမြင့်ရဲ့ လေးပုံသုံးပုံ (ခေါင်မိုးက လေးပုံတစ်ပုံ ဖြစ်တဲ့အတွက်)။

```

# center points of the windows
left_windows_x = left + width/4
right_windows_x = left + width * 3/4

wall_height = height * 3/4
upper_windows_y = wall_top + wall_height/4
lower_windows_y = wall_top + wall_height/4 * 3

ဒီအမှတ်တွေသိရင် ပြတင်းပေါက်တွေကို ဒီလိုခွဲခိုင်ပဲ
draw_window(left_windows_x, upper_windows_y, 2, 2)
draw_window(left_windows_x, lower_windows_y, 2, 3)
draw_window(right_windows_x, upper_windows_y, 2, 2)
draw_window(right_windows_x, lower_windows_y, 2, 3)

```

အိမ်တံခါးမ ကို အခုလို ဆွဲပါတယ်။ အမြင့်နဲ့ အကျယ်ကို ကြည့်ကောင်းအောင် သင့်တော်သလိုထားနိုင်
တယ်။ နေရာအတွက် door_top, door_left တွေက်တာလည်း မခက်ပါဘူး။ နားလည်မှာပါ။

```
# draw main door
door_width = 76
door_height = 120
door_top = wall_bottom - door_height
door_left = left + width/2 - door_width/2
draw_main_door(door_left, door_top,
                door_width, door_height,
                WOOD_BROWN)
```

အတွက်အချက်တွေ အားလုံး ဒီလောက်ပါပဲ။ အရှင်းမခက်ဘူး ထင်ပါတယ်။ draw_house ဖန်ရှင်တစ်ခုလုံး ရှုံးမျက်နှာမှာ အစအဆုံး ပြန်လည့်လာကြည့်ပါ။

draw_main_door

draw_main_door ဖန်ရှင်က ဒီလို ... left နဲ့ top က တံခါးမ အပေါ်ဘယ်ဘက်ထောင့် အမှတ်ပါ။ တံခါးသော့ကို draw_circle_filled နဲ့ ဆွဲတယ်။ စက်ဝိုင်း ဗဟိုမှတ်ကို ထည့်ပေးရတယ်။ ဘယ်၊ ညာ၊ အထက်၊ အောက် ဘယ်လိုတွက်ထားလဲ အခုလောက်ဆို နားလည်လောက်ပြီလို့ ယူဆတယ်။

```
def draw_main_door(left, top, width, height, color):
    arcade.draw_lrtb_rectangle_filled(left, left + width,
                                      top + height, top,
                                      color)
    arcade.draw_lrtb_rectangle_outline(left, left + width,
                                      top + height, top,
                                      GRAY)

    # draw door lock
    arcade.draw_circle_filled(left + width - 10,  # circle center x
                             top + height/2,    # circle center y
                             7,                  # radius
                             BLACK)
```

အိမ်ပုံဆွဲတဲ့အခါ ခေါက်ချိုးညီ ပေါ်အောင်နဲ့ သင့်တော်တဲ့ အရွယ်အစားဖြစ်အောင် အခုလိုတွက်ထားပါတယ်

```
WIN_WIDTH = 754
WIN_HEIGHT = 492
SIDE_MARGIN = 80
TOP_BTM_MARGIN = 25
HOUSE_WIDTH = WIN_WIDTH - 2 * SIDE_MARGIN
HOUSE_HEIGHT = WIN_HEIGHT - 2 * TOP_BTM_MARGIN
```

ဝင်းဒီး အကျယ်နဲ့ အမြင် 754, 492 pixels ထားတယ်။ ဝင်းဒီးရဲ့ ဘယ်၊ ညာနဲ့ အထက်၊ အောက် ဘောင်တွေကနေ အိမ်ကို 80, 25 pixels ခွာထားတယ်။

နောက်ခံးတော့ Arcade လိုက်ဘရဲ့ လိုအပ်ချက်အတိုင်း အခုလို တစ်ဆင့်ချင်း ပြင်ဆင်ပြီး ရောင်စုံအိမ်ပုံလေး ဆွဲလို့ရပါပြီ

```
arcade.open_window(WIN_WIDTH, WIN_HEIGHT, "Colorful House")
arcade.set_viewport(0, WIN_WIDTH, WIN_HEIGHT, 0)

# Set the background color
arcade.set_background_color(arcade.color.SKY_BLUE)

# Get ready to draw
arcade.start_render()

# იძინეთ დანართის მიზანის დანართის გვერდზე
(მინიჭით დანართის გვერდზე)
arcade.draw_lrtb_rectangle_filled(0, WIN_WIDTH,
                                  WIN_HEIGHT, WIN_HEIGHT * 2/3,
                                  FOREST_GREEN)
draw_house(SIDE_MARGIN, TOP_BTM_MARGIN, HOUSE_WIDTH, HOUSE_HEIGHT)

# Finish drawing
arcade.finish_render()

# Keep the window up until someone closes it.
arcade.run()
```

မှတ်ချက်။ ။ အခန်း (၇) စာမျက်နှာ (၁၀၂) မှာ Arcade လိုက်ဘရီအကြောင်း မိက်ဆက်ပေးထားပါတယ်။ အဲဒီအပိုင်းဖတ်ထားမှ အခုရှင်းပြတာတွေကို သေချာနားလည်နိုင်မှာပါ။

ရောင်စုံအိမ်လေးဆွဲတဲ့ ပရိုဂရမ်ကို တစ်ဆက်တည်း ဆက်စပ် ကြည့်လိုအဆင်ပြအောင် အစအဆုံးထည့်ပေးထားတယ်။ လေ့လာကြည့်ပါ။

```
                top + pane_height * (i+1),
                top + pane_height * i,
                GRAY)

def draw_main_door(left, top, width, height, color):
    arcade.draw_lrtb_rectangle_filled(left, left + width,
                                      top + height, top,
                                      color)
    arcade.draw_lrtb_rectangle_outline(left, left + width,
                                      top + height, top,
                                      GRAY)

    # draw door lock
    arcade.draw_circle_filled(left + width - 10,
                             top + height/2,
                             7,                      # radius
                             BLACK)

def draw_house(left, top, width, height):
    middle = left + width / 2
    wall_top = top + height / 4
    right = left + width
    wall_bottom = top + height

    arcade.draw_lrtb_rectangle_filled(left, right,
                                      wall_bottom, wall_top,
                                      TEA_GREEN)
    arcade.draw_lrtb_rectangle_outline(left, right,
                                      wall_bottom, wall_top,
                                      GRAY)

    arcade.draw_triangle_filled(left, wall_top, right,
                               wall_top, middle, top,
                               AIR_FORCE_BLUE)
    arcade.draw_triangle_outline(left, wall_top,
                               right, wall_top,
                               middle, top,
                               GRAY)

    # center points of the windows
    left_windows_x = left + width/4
    right_windows_x = left + width * 3/4

    wall_height = height * 3/4
    upper_windows_y = wall_top + wall_height/4
```

292

```
lower_windows_y = wall_top + wall_height/4 * 3

draw_window(left_windows_x, upper_windows_y, 2, 2)
draw_window(left_windows_x, lower_windows_y, 2, 3)
draw_window(right_windows_x, upper_windows_y, 2, 2)
draw_window(right_windows_x, lower_windows_y, 2, 3)

# draw main door
door_width = 76
door_height = 120
door_top = wall_bottom - door_height
door_left = left + width/2 - door_width/2
draw_main_door(door_left, door_top,
                door_width, door_height,
                WOOD_BROWN)

WIN_WIDTH = 754
WIN_HEIGHT = 492
SIDE_MARGIN = 80
TOP_BTM_MARGIN = 25
HOUSE_WIDTH = WIN_WIDTH - 2 * SIDE_MARGIN
HOUSE_HEIGHT = WIN_HEIGHT - 2 * TOP_BTM_MARGIN

# Open up a window.
# From the "arcade" library, use a function called "open_window"
# Set the window title to "Colorful House"
# Set the dimensions (width and height)
arcade.open_window(WIN_WIDTH, WIN_HEIGHT, "Colorful House")
arcade.set_viewport(0, WIN_WIDTH, WIN_HEIGHT, 0)

# Set the background color
arcade.set_background_color(arcade.color.SKY_BLUE)

# Get ready to draw
arcade.start_render()

arcade.draw_lrtb_rectangle_filled(0, WIN_WIDTH,
                                 WIN_HEIGHT, WIN_HEIGHT * 2 / 3,
                                 FOREST_GREEN)
draw_house(SIDE_MARGIN, TOP_BTM_MARGIN, HOUSE_WIDTH, HOUSE_HEIGHT)

# Finish drawing
arcade.finish_render()

# Keep the window up until someone closes it.
```

arcade.run()

ပိုကောင်းအောင် ပြုပြင်သင့်တဲ့ နေရာလေးတွေ

ရောင်းစံအိမ်လေး ပရိုဂရမ်က အတန်အသင့် အဆင်ပြုနေပါပြီ။ တရာ့မျိုးနေရာလေးတွေ ပိုကောင်းအောင် improve လုပ်လို့ရပါတယ်။ ပြင်ဖို့က သိပ်လည်းမခက်ဘူး။ ထောင့်မှန်စတုဂံတွေကို နှစ်ခါဆွဲနေရတယ်။ အရောင်ဖြည့်ဖို့ draw_lrtb_rectangle_filled နဲ့တစ်ခါ အနားသတ်အတွက် draw_lrtb_rect angle_outline နဲ့တစ်ခါ။ ဖန်ရှင်မှာ ထည့်ပေးတဲ့ တန်ဖိုးတွေကလည်း အားလုံး တူတူပဲ (အရောင်က လွှဲလို့)။ အခုလို ဖန်ရှင် သတ်မှတ်ထားလိုက်ရင် ပိုအဆင်ပြုသွားမှာပါ

```
def draw_rect_filled_with_outline(left, right, top, bottom, color):
    arcade.draw_lrtb_rectangle_filled(left, right, top, bottom, color)
    arcade.draw_lrtb_rectangle_outline(left, right, top, bottom, GRAY)
```

မီးခိုးရောင် အနားသတ်နဲ့ rectangle ကို အရောင်အမျိုးမျိုး ဖြည့်ဆွဲလို့ရမှာပါ။ ဥပမာ draw_window မှာ အခုလို သုံးလို့ရမယ်

```
draw_rect_filled_with_outline(left + pane_width * j,
                               left + pane_width * (j+1),
                               top + pane_height * (i+1),
                               top + pane_height * i,
                               AZURE_MIST)
```

နံရံနဲ့ တံခါးမဆွဲရင်လည်း အလားတူ သုံးရုံပဲ။

၈.၆ ပါရာမီတာ ခြေးချယ်သတ်မှတ်ခြင်း

ဖန်ရှင်သတ်မှတ်တဲ့အခါ ‘ဘယ်ဟာတွေက ပါရာမီတာ ဖြစ်သင့်လဲ’၊ ‘ဘယ်ဟာတွေကတော့ရော မဖြစ်သင့်ဘူးလား’ ဝေခဲ့ဆုံးဖြတ်မရ ဖြစ်လေ့ရှိတာဟာ စလေ့လာတဲ့သူတွေရော အတွေးအကြုံရှိပြီး သူတွေအတွက်ပါ ပုံမှန် ဖြစ်ရှိဖြစ်စဉ် တစ်ခုပါပဲ။ ဖော်မြှုပါလာ ရှိပြီးသား သချာဖန်ရှင်တော့အတွက်တော့ သိပ်ခေါင်းစားစာရာ မလိုပါဘူး။ စက်ဝိုင်းခေါ်ယာရှာရင် အချင်းဝက်၊ ကြိုက်ဆိုရင် အခြေနဲ့ အမြင့်တိုက် ပါရာမီတာ အနေနဲ့ ထားရမယ်ဆိုတာ သိသာတယ်။ ဒါပေမဲ့ တရာ့မျိုးနေရာတွေမှာ ထင်သလောက် မရှိရှင်းပါဘူး။

ယော့ယျာအားဖြင့် ဖန်ရှင် ခေါ်ပဲ့အခါကျတော့မှ တန်ဖိုးထည့်ပေးချင်တဲ့ အရာတွေကို ပါရာမီတာ အနေနဲ့ ထားလေ့ရှိတယ်။ ရောင်းစံအိမ်လေးမှာ draw_window ဖန်ရှင်ခေါ်တော့မှ ဆွဲမဲ့ ပြတင်းပေါက် တည်နေရာနဲ့ row နဲ့ column အရေအတွက် ထည့်ပေးရမှာပါ။ ဒီတော့မှ လိုချင်တဲ့နေရာမှာ လိုချင်တဲ့ row, column အရေအတွက်နဲ့ ဆွဲလို့ရမယ်။

အခုလက်ရှိ draw_window ဟာ ပြတင်းပေါက်အရောင် တစ်မျိုးနဲ့ပဲ ဆွဲလို့ရနိုင်မယ်။ အကယ်၍ အပေါ်ပြတင်းပေါက် နှစ်ခုနဲ့ အောက်ပြတင်းပေါက် နှစ်ခု အရောင်ကဲ့နေတယ်ဆိုပါစို့။ ဒါဆိုရင်တော့ draw_window ကို အခုလို ပြင်ပေးရပါမယ်

```
def draw_window(x, y, row, col, color):
    # ... ဒီနေရာက လိုင်းတွေ တရာ့မျိုးကို ချုံထားတယ်
    for i in range(row):
        for j in range(col):
            draw_rect_filled_with_outline(left + pane_width * j,
```

```
left + pane_width * (j+1),
top + pane_height * (i+1),
top + pane_height * i,
color)
```

အပေါက် အဖြူရောင် အောက်ကို မီးခါး ဆွဲမယ်ဆိုရင်

```
draw_window(left_windows_x, upper_windows_y, 2, 2, WHITE)
draw_window(right_windows_x, upper_windows_y, 2, 2, WHITE)
draw_window(left_windows_x, lower_windows_y, 2, 3, GRAY)
draw_window(right_windows_x, lower_windows_y, 2, 3, GRAY)
```

ဒီလိုဆွဲလို ရပါတယ်။

တကယ်လို မှန်ချပ်အရွယ်ပါ ဖန်ရှင်ခေါ်တဲ့အခါကျတော့မှ သတ်မှတ်လိုရချင်ရင် မှန်ချပ် အကျယ်၊ အမြင့် အတွက် ပါရာမီတာနှစ်ခု ထပ်ထည့်နိုင်ပါတယ်

```
def draw_window(x, y, row, col, color, pane_width, pane_height):
    # pane_width = 30 ဒီနံပါတ်ကြာင်း မလိုတော့ဘူး
    # pane_height = 40
    window_width = pane_width * col
    window_height = pane_height * row
    top = y - window_height / 2
    left = x - window_width / 2
    for i in range(row):
        for j in range(col):
            draw_rect_filled_with_outline(left + pane_width * j,
                                           left + pane_width * (j+1),
                                           top + pane_height * (i+1),
                                           top + pane_height * i,
                                           color)
```

ပါရာမီတာတွေ ထပ်ဖြည့်ခြင်းအားဖြင့် ပြတင်းပေါက်ကို အရောင် အမျိုးမျိုးနဲ့ ဆွဲလိုရလာတယ်။ မှန်ချပ် အရွယ် အမျိုးမျိုးနဲ့ ဆွဲလိုရလာတယ်။ ဖန်ရှင်ဟာ ပိုပြီး ဂျွန်နှုန်းကျလာတယ်။ flexible ပိုဖြစ်လာတယ်။ ဒီတော့ တတ်နိုင်သယူ ပါရာမီတာတွေ များများထားရင် ပိုကောင်းတယ် ယူဆရမှာလား။ ပါရာမီတာတွေ များလာတာနဲ့အမျှ သူတို့ကို ဘာအတွက်ထည့်ရော်လဲ၊ ဘာတန်ဖိုးထည့်သင့်လဲ စဉ်းစားဖို့လိုလာတယ်။ ဖန်ရှင်ကို အသုံးပြုရတဲ့ ပို့ခေက်လာတယ်။ ဒါကြောင့် flexibility နဲ့ အသုံးပြုရ လွှာယ်ကူမှု ပေါ်ပူတည်ပြီး ချင့်ချိန် သတ်မှတ်ရမှာပါ။

ဂျွန်နှုန်းကျတဲ့၊ flexible ဖြစ်တဲ့ ဖန်ရှင်တစ်ခုကို အခြေခံပြီး specific ဖြစ်တဲ့ ဖန်ရှင်တွေကို သတ်မှတ်ယူလိုရတယ်။ ဒီဖန်ရှင်နှစ်ခုမှာ အပေါက် draw_window ကို ပြန်သုံးထားပါတယ်။

```
def draw_2x2_pearl_window(x, y):
    draw_window(x, y, 2, 2, PEARL, 30, 40)

def draw_2x3_gray_window(x, y):
    draw_window(x, y, 2, 3, LIGHT_GRAY, 30, 40)
```

2×2 ပုလဲရောင်နဲ့ 2×3 မီးခိုးရောင် ပြတင်းပေါက်ဆွဲဖို့ပါ။ နေရာအတွက် ပါရာမိတာ နှစ်ခုပဲ ပါတော့တယ်။ သုံးရတာ ပိုလွယ်သွားတာ သိသိသာသာတွေရမှုပါ။ ဥပမာ ရောင်စုအမ်လေးမှာ အပေါ်ပြတင်းနှစ်ပေါက်ကို ပုလဲရောင်၊ အောက်နှစ်ခုကို မီးခိုးရောင် လိုချင်ရင် အခုလို ဆွဲရုပ်ပါပဲ

```
draw_2x2_pearl(left_windows_x, upper_windows_y)
draw_2x3_gray(left_windows_x, lower_windows_y)
draw_2x2_pearl(right_windows_x, upper_windows_y)
draw_2x3_gray(right_windows_x, lower_windows_y)
```

အရောင်နဲ့ row, column တော့ ပြောင်းချင်လို့ မရဘူး။ ပုံသေ ဒီနှစ်မျိုးပဲ ရပါမယ်။ ဒါကြောင့် flexibility အရတော့ အားနည်းတယ်ပေါ့။ ဒါပေမဲ့ သုံးရတာ အတော်လေးလွှယ်တယ်။ ဂျှိနရယ်ကျတဲ့ flexible ဖြစ်တဲ့ ဖန်ရှင်ကို အခြေခံပြီး specific ဖြစ်တဲ့ ဖန်ရှင် သတ်မှတ်တဲ့ နည်းလမ်းကို ပရိုကရမ်ဘတွေ မကြာခဏ အသုံးပြုလေ့ရှိတယ်။ ကိုယ်တိုင် အသုံးချုတ်အောင် လေ့လာထားသင့်ပါတယ်။ အသုံးတည်းမှုပါ။ နားလည်ဖို့ သိပ်ခက်ခဲတဲ့ ကိစ္စလည်း မဟုတ်ပါဘူး။

၈.၃ တန်ဖိုးပြန်ပေးခြင်း၊ မပေးခြင်း

တရာ့၏ ဖန်ရှင်တွေမှာ တန်ဖိုးပြန်ပေးသင့် (သို့) မပေးသင့် ဆုံးဖြတ်ရ ခက်တတ်ပါတယ်။ ဒါနဲ့ပါတ်သက်ပြီး ပထမဆုံး သိထားသင့်တာကတော့ တန်ဖိုးပြန်ပေးခြင်း (သို့) မပေးခြင်းရဲ့ အကျိုးဆက်က ဘယ်လို ရှိမလဲ ဆိုတာပါ။

```
def greet_guest_nort(guest_name):
    print(f'Hello, {guest_name}.')

def greet_guest_rt(guest_name):
    return f'Hello, {guest_name}.'
```

ဒီဖန်ရှင်နှစ်ခုက သဘောတရား ဆင်တူပါတယ်။ ကွာခြားတာဆိုလို့ ပထမတစ်ခုက တစ်ခါတည်း print နဲ့ စာသားကို output ထုတ်ပေးပြီး နောက်တစ်ခုကတော့ စာသားကိုပဲ return ပြန်ပေးတယ်။ ဒုတိယတစ်ခုနဲ့ စခရင်မှာထုတ်ချက်ရင်

```
print(greet_guest_rt("Joe"))
```

ဖန်ရှင်ခေါ်တဲ့သူက print လုပ်ပေးရပါမယ်။ သုံးတဲ့သူအနေနဲ့တော့ အလုပ်ပိုတာပေါ့။ ဒါပေမဲ့ ဖန်ရှင်က ပြန်ရတဲ့ တန်ဖိုးကို လိုအပ်သလို ဆက်လက် အသုံးပြုနိုင်တယ်။ ဥပမာ greet လုပ်ပြီးရင် နေကောင်းလား ဆက်မေးချင်တယ်။ ဒါမှုမဟုတ် စာသားထဲက အကွဲရာ/စကားလုံး အစားထိုးမယ် ဆိုပါစွာ။

```
print(greet_guest_rt("Joe") + '\nHow are you doing?')
print(greet_guest_rt('Kathy').replace('. ', '!!!!'))
```

ဖန်ရှင်ခေါ်တဲ့သူက ပြန်ရတဲ့ တန်ဖိုးကို လိုအပ်သလို ဆက်လက် အသုံးပြုလို့ ရပါမယ်။ တန်ဖိုးပြန်မရရင် တော့ ဒီလိုလုပ်လို့ ရမှာ မဟုတ်ပါဘူး။

နောက်တစ်ခုက တန်ဖိုးပြန်ပေးရင် မှန်/မမှန်စစ် (test) လုပ်ရတာ ပိုလွယ်တယ်။ ဒါက စာမျက်နှာ (၁၀၄) က ခရစ်မတ် သစ်ပင်လေးကို ဖန်ရှင်ရေးထားတာပါ

```
def christmas_tree(leaf_rows, trunk_rows):
    tree = []
```

```

for r in range(leaf_rows):
    tree_row = ' ' * (leaf_rows - 1 - r)
    tree_row += '*' * (r * 2 + 1)
    tree.append(tree_row)
trunk_width = 3
for r in range(trunk_rows):
    tree_row = ' ' * (leaf_rows - 2)
    tree_row += '*' * trunk_width
    tree.append(tree_row)

return tree

```

Row တစ်ခုချင်းကို string အနေနဲ့ list ထဲမှာ ထည့်ပြီး return ပြန်ပေးထားတယ်။ Output ထုတ်မယ်ဆိုရင်

```

my_tree = christmas_tree(8, 4)
for row in my_tree:
    print(row)

```

ဒီဖန်ရှင် မှန်/မမှန် test လုပ်ဖို့ အခုလို ဖန်ရှင်တစ်ခု ရေးထားနိုင်ပါတယ်။

```

def test_christmas_tree():
    test_tree = [
        '      *',
        '     ***',
        '    *****',
        '   *****',
        '*****',
        '  ***',
        '  ***',
        '  ***',
    ]
    result_tree = christmas_tree(5, 3)
    if result_tree != test_tree:
        print("Error: expected and actual result don't match!")
    else:
        print("Expected and actual result do match!")

```

အကြောင်းတစ်ခုခဲ့ကြောင့် `christmas_tree` ရဲ့ အတွင်းပိုင်း တည်ဆောက်ထားပုံကို (ပိုမြင်အောင် ဒါမှမဟုတ် ပိုနားလည်ရ လွယ်ကူးအောင်) ပြင်ရေးမယ်ဆိုရင် ရလဒ် နိုင်အတိုင်း ဖြစ်မဖြစ် ပြန်စစ်ဖို့ လိုပါတယ်။ တကယ်လို့ `christmas_tree` က တန်ဖိုးပြန်မပေးဘဲ တစ်ခါတည်း `print` ထုတ်လိုက်ရင် စခရင်ဗာ ပေါ်လာတဲ့ output ကိုကြည့်ပြီး လူကိုယ်တိုင် စစ်ဆေးမှုပဲ မှန်/မမှန် သိနိုင်မှုပါ။ ပရိုဂရမ်ရေးပြီး အခုလို automated testing (လူက test မလုပ်ဘဲ ပရိုဂရမ်နဲ့ test လုပ်တာကို ဆိုလို) လုပ်ဖို့ မဖြစ်နိုင်ဘူး။

အခုဖော်ပြခဲ့သလို testing လုပ်တာဟာ `unit testing` ရဲ့ အခြေခံသောတရားပါပဲ။ ဖန်ရှင်တစ်ခု (သို့) မောဒ္ဒါးတစ်ခုကို သူချည်းသီးခြား မှန်/မမှန် test လုပ်တာကို `unit testing` လို့ ခေါ်တာပါ။

တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်တွေဟာ unit test လုပ်ရ လွယ်ကူတာ တွေနဲ့ပါတယ်။ ဆွဲဖို့တည်ဆောက်ရမှာ unit testing ဟာ အရေးကြီးတဲ့ အခန်းကဏ္ဍတစ်ခု အနေနဲ့ ပါဝင်တယ်။ ကျယ်ပြန်တဲ့ နယ်ပယ်ဖြစ်ပြီး သီးခြားလေ့လာဖို့ လိုအပ်ပါတယ်။ ဒီစာအုပ်မှာတော့ ဖန်ရှင်တန်ဖိုး ပြန်ပေးခြင်းနဲ့ ဆက်စပ်နေတဲ့ အတွက် အခြေခံလောက်လေးကိုပဲ ဗဟိုသုတအနေနဲ့ ထည့်သွင်းဖော်ပြတာပါ။

၈.၈ return

return ဟာ ဖန်ရှင်ကနေ သူကိုခေါ်တဲ့သူ (သီး) ခေါ်ထားတဲ့နေရာဆီ ပြန်ရောက်သွားစေတယ်လို့ သိထားပြီ။ သတိပြုမိသင့်တဲ့ အသေးစိတ်အချက် တချို့ကို ဆက်ကြည့်ရအောင်။ `get_sign` နဲ့ `print_sign` ကို cascading if မသုံးဘဲ အခုလိုရေးလိုရပါတယ်

```
def get_sign(r):
    if r > 0:
        return 'positive'
    if r < 0:
        return 'negative'

    return 'zero/nosign'

def print_sign(r):
    if r > 0:
        print('positive')
        return
    if r < 0:
        print('negative')
        return

    print('zero/nosign')
    return # ဒါမပါလည်းရတယ်
```

ရုံးရိုး if သုံးထားပေမဲ့ ကွန်ဒါရှင်မှန်ရင် `return` ဖြစ်သွားမှုမိုလို့ အောက်ကို မရောက်တော့ဘူး။ ဒီ အတွက် cascading if အလုပ်လုပ်ပုံလိုပဲ ဖြစ်သွားတယ်။ အပေါ်မှာ မှန်ရင်အောက်ကို မရောက်လာနိုင်ဘူး။ အောက်က `if` တွေကို ရောက်လာတော့ အပေါ်က ကွန်ဒါရှင်တွေ မှားလိုပဲ။ (အခုလိုမျိုးရေးသင့်တယ်လို့ မဆိုလိုပါ။ `return` ခဲ့သောတရားကို သတိပြုမိအောင် ဥပမာပြတာပါ။)

ဖန်ရှင်မှာ `loop` ထဲကနေ `return` လုပ်လိုက်ရင် `loop` ကို `break` လိုက်သလို ဖန်ရှင်ခေါ်တဲ့နေရာ ဆီကိုလည်း ပြန်ရောက်သွားမှာပါ။

```
def read_int(prompt):
    while True:
        valin = input(prompt)
        try:
            return int(valin)
        except ValueError as err:
            if valin == 'quit':
                return
            print('Non-integer data!')
```

```
num = read_int('Enter number: ')
print(num)
```

မတော်တဆာ **None** **return** ဖြစ်ခြင်း

အောက်ပါ ပကဗ္ဗတန်ဖိုး ဖန်ရှင်မှာ ၂ တန်ဖိုး သူညာအတွက် မစစ်မိတားဘူး။

```
def my_abs(n):
    if n > 0:
        return n
    elif n < 0:
        return -n
```

အကယ်၍ `print(my_abs(0))` နဲ့ ထုတ်ကြည့်ရင် `None` လို့ ပြပါလိမ့်မယ်။ ၂ သူညာဖြစ်တဲ့အတွက် `if...elif` ထဲက `return` နှစ်ခုလုံး မလုပ်ဘူး။ ဒီတော့ `if...elif` အပြီး နောက်တစ်ကြောင်း ကို ရောက်လာတဲ့ သဘောပဲ။ ဖန်ရှင်ဘလောက်အဆုံးထိ ရောက်လာပြီးရင် ခေါ်ခဲ့တဲ့နေရာကို ပြန်သွား ဖို့ `return` (နောက်မှာ တန်ဖိုးမပါ) လုပ်တယ်လို့ ယူဆရမှာပါ။ တန်ဖိုးမပါတဲ့ `return` က `None` ကို ပြန်ပေးပါတယ်။ ဒီလောက်ဆိုရင် ဘာလို့ `print(my_abs(0))` က `None` ရတာလဲ နားလည်းမှာပါ။ ဒီ အမှားကို ပြင်ထားတဲ့ ဥပမာနစ်ခုကို ပြထားပါတယ်။ ဒုတိယတစ်ခုက အတွေ့အကြုံတဲ့ Python ပရီ ဂရမ်မာတစ်ယောက် ရေးလေ့ရှိတဲ့နည်း။ Pythonic ပိုဖြစ်တယ် (Python ပိုဆန်တယ်) ပေါ့။

```
def my_abs(n):
    if n > 0:
        return n
    elif n < 0:
        return -n
    else:
        return 0

def my_abs(n):
    if n < 0:
        return -n
    return n
```


အခန်း ၉

From Classes to Objects (ကလပ်စ်များမှ အော်ဂျက်များသို့)

အခန်း (၆) မှာ အော်ဂျက်တရီးနဲ့ မိတ်ဆက်ပေးခဲ့တယ်။ `date`, `list`, `dict` စသည်ဖြင့်။ အော်ဂျက်တွေကို ကလပ်စ်တစ်ခုကနေ တည်ဆောက်ယူရတာပါ။ `date` အော်ဂျက်တွေကို `date` ကလပ်စ် `Fraction` အော်ဂျက်တွေကို `Fraction` ကလပ်စ်ကနေ ဖန်တီးယူတယ်။ အော်ဂျက်တွေကို `class` တစ်ခုရဲ့ `instance` လိုလည်းခေါ်တယ်။ `Fraction` အော်ဂျက်တွေဟာ `Fraction` ကလပ်စ် `instance` လိုလည်း ပြောလေ့ရှိတယ်။ အော်ဂျက်တွေဟာ `class` `instance` တွေဖြစ်တယ်။ (အကြမ်းဖျဉ်းအားဖြင့် အော်ဂျက်နဲ့ `instance` တူတယ်ပြောလိုရပေမဲ့ အနက်အဓိပါယ်အရ အနိတ် ကွာဟနျက်တွေ ရှိတာကိုလည်း inheritance အကြောင်း လေ့လာတဲ့အခါ တွေ့ရပါယ်)။

ဒီအခန်းမှာ ကလပ်စ်အကြောင်းကို လေ့လာကြမှာပါ။ ကိုယ်ပိုင် ကလပ်စ်သတ်မှတ်ပြီး အဲဒီကလပ်စ်ကနေ အော်ဂျက်တွေ တည်ဆောက် ယူနိုင်တော့မှာဖြစ်တယ်။ အခြားသူတွေရေးထားပေးတဲ့ လိုက်ဘရိဖန်ရှင်တွေကို အသုံးပြုရာကနေ နောက်ပိုင်းမှာ ကိုယ်ပိုင် ဖန်ရှင်တော့ သတ်မှတ်လာနိုင်တာဟာ အဆင့်တစ်ဆင့် ပုံမြေင့်လာသလိုပါပဲ။ ကိုယ်ပိုင်ကလပ်စ် ဒီဇိုင်းပြုလုပ် ဖန်တီးနိုင်လာတာဟာလည်း နောက်တစ်ဆင့်ထပ်မြုင်လာတယ်လို့ ဆုံးရမှာပါ။ ပရိုဂရမ်တွေကို အမှားနည်းအောင် ပြင်ဆင်ရလွယ်ကူအောင် စနစ်တကျ ဒီဇိုင်းလုပ် တည်ဆောက်လို့ရလာမှာ ဖြစ်ပါတယ်။

ကလပ်စ်ကို abstraction mechanism တစ်ဖုံးလိုလည်း ရှိမြင်နိုင်တယ်။ Abstraction လုပ်တယ်ဆိုတာ ဘယ်လိုတည်ဆောက်ထားလဲ သိစရာမလိုဘဲ အသုံးပြုလိုရစေတာကို ဆုံးလိုတာ။ ဖန်ရှင်တွေနဲ့ abstraction လုပ်တာကို တွေ့ခဲ့ကြပြီးပါပြီ။ ကလပ်စ်တွေဟာလည်း abstraction အတွက် အထောက်အကူဗြို့ နည်းလမ်းတစ်မျိုး (mechanism) ပါပဲ။ အများကြီး ပုံးဖွဲ့များထက်တဲ့ နည်းလမ်းလို့ ဆုံးရမှာပါ။ အစွမ်းထက်လာတာနဲ့အမျှ သဘောတရားအရရော အသုံးချတဲ့အပိုင်းမှာပါ ပိုပြီးရှုပ်ထွေးခက်ခဲနိုင်ပါတယ်။ ဒီဇိုင်းပိုင်းအတွက် ကျယ်ကျယ်ပြန်ပြန့် ဆက်လက်လေ့လာဖို့ လိုအပ်မှာဖြစ်ပြီး အတွေ့အကြုံပါ ပေါင်းစပ်ယူရမှာပါ။ စာချည်းဖတ်ရှု့နဲ့ မရှိနိုင်ဘူး (စာရေးသူ ကိုယ်တွေ့ အတွေ့အကြုံအရ ကိုယ်ပိုင်အမြင်ကို ပြောခြင်းသာ)။ ဒါတွေက နောက်ပိုင်း ဆက်လက်လေ့လာဖို့အတွက်ပေါ့။ အခုတော့ အများကြီးမပြောတော့ဘဲ ကလပ်စ်တွေအကြောင်း စလိုက်ကြရအောင် ...

၉.၁ Account Class (အကောင့် ကလပ်စ်)

ဘဏ်အကောင့်တွေကို `Account` အော်ဂျက်နဲ့ ဖော်ပြုမယ်ဆိုပါမြို့။ အကောင့်နံပါတ်၊ ပိုင်ရှင်၊ လက်ကျွန် (balance) စတဲ့ အချက်အလက်တွေဟာ အပြင်က တကယ့် ဘဏ်အကောင့်တစ်ခုအတွက် အရေးပါပါတယ်။ မှတ်သား သိမ်းဆည်းထားရတယ်။ ဒါကြောင့် အပြင်က တကယ့် ဘဏ်အကောင့်ကို ထင်ဟပ်

ဖော်ပြတဲ့ ဆော်ဖို့ အော့ဘ်ဂျက်တွေမှာလည်း ဖော်ပြပါ အချက်အလက်တွေ ပါဝင်သင့်တယ်ဆိုရင် ကျိုး ကြောင်းဆီလျဉ်တယ်ပဲ ယူဆရမှာပါ။ အကောင့်နဲ့ ပါတ်သက်ပြီး အခြား အရေးပါတဲ့ အချက်အလက် တွေ ဒီးထက်မက ရှိပါတယ်။ ဥပမာ အကောင့် အမျိုးအစား (saving, current) ၊ ဖွင့်ရက် (open date)၊ ဘဏ်ခွဲအမည်၊ ငွေသွင်း/ငွေထုတ်/ငွေလွှဲမှတ်တမ်း စတဲ့အချက်တွေ ရှိပါတယ်။ တကယ့်လက်တွေ မှာ လိုအပ်မှာဖြစ်ပေမဲ့ အခုံပမာမှာ ရိုးရှင်းရှင်းဖြစ်ဖို့ အဲဒါတွေ ထည့်မစွာစားဘဲ ချုန်ခဲ့ရအောင်။ ကလပ်စဲရဲ့ သဘောတရားကို ရှင်းပြန့်အတွက် ပထမသုံးချက်နဲ့ လုံလောက်ပါတယ်။

ကေသိမှာ အကောင့်နဲ့ပါတ် 0086-6002-1111 နဲ့ လက်ကျွန်းငွေ တစ်သိန်းရှိတဲ့ အကောင့်တစ်ခု ရှိတယ် ဆုံးပါမို့။ Account ကလပ်စဲသာ ရှိမယ်ဆုံးရင် အဲဒီ ကေသိအကောင့်ကို ကိုယ်စားပြုတဲ့ အော့ဘ်ဂျက် ကို အခုံလို့

```
acc1 = Account('Kathy',
'0086-6002-1111',
Decimal("100000.00"))
```

ဖန်တီးယူလို့ ရမှာပါ။ ထိုနည်းတူစွာ ငွေလက်ကျွန်းငွေ လေးသိန်းခဲ့၊ နဲ့ပါတ် 0086-6002-2211 နဲ့ စုနှုန်းအကောင့် အော့ဘ်ဂျက်ကို ဒီလို

```
acc2 = Account('Sandy',
'0086-6002-2211',
Decimal("450000.00"))
```

ဖန်တီးယူနိုင်မယ်။ ဒါက Account ကလပ်စဲသာ ရှိခဲ့ရင် အော့ဘ်ဂျက် ဘယ်လို့ ဖန်တီးယူရမလဲ စဉ်းစားကြည့်တာပေါ့။ ကလပ်စဲမရှိတဲ့အတွက် အမှန်စာကယ်တော့ မရသေး။

date, list, စတဲ့ အော့ဘ်ဂျက်တွေ အပေါ်မှာ အော်ပရေးရှင်းတွေ လုပ်ဆောင်လို့ရတာ တွေ့ခဲ့ရတယ်။ Account အော့ဘ်ဂျက်တွေမှာရော ဘယ်လို့ အော်ပရေးရှင်းတွေ လုပ်ဆောင်လို့ရသင့်လဲ။ အပြင် မှာ အကောင့်တစ်ခုကနေ ငွေထုတ်လို့ရတယ်။ အကောင့်ထဲကို ငွေသွင်းလို့ရှိပါတယ်။ Account အော့ဘ်ဂျက်တွေမှာလည်း ဒါတွေလုပ်လို့ရသင့်တာပေါ့

```
acc1.deposit(Decimal("50000.00"));
acc2.withdraw(Decimal("70000.00"));
```

ဒီအော်ပရေးရှင်းတွေက အော့ဘ်ဂျက်တွေအပေါ် ဘယ်လို့သက်ရောက်မှုရှိမယလဲ။ ကေသိအကောင့် လက်ကျွန်းငွေ တစ်သိန်းခဲ့ ဖြစ်သွားသင့်တယ်။ စုနှုန်းအကောင့်က သုံးသိန်းရှုစ်သောင်း ဖြစ်သင့်တယ်။ ဒီလို့ အော်ပရေးရှင်းတွေဟာလည်း ကလပ်စဲပေါ်မှာ မူတည့်တယ်။ ကလပ်စဲက ထောက်ပုံပေးထားမှုပဲ ရမယ်။ ကလပ်စဲသည်သာ အခရာလို့ ပြောရမှာပါ။

ကလပ်စဲတစ်ခုဟာ အော့ဘ်ဂျက်တွေမှာ ပါရှိရမဲ့ အချက်အလက်တွေနဲ့ ငါးငါးအော့ဘ်ဂျက်တွေအပေါ် မှာ လုပ်ဆောင်နိုင်တဲ့ အော်ပရေးရှင်းတွေကို သတ်မှတ်ပေးပါတယ်။ ဥပမာ အနေနဲ့ Account ကလပ်စဲ ဘယ်လို့သတ်မှတ်ရမလဲ ကြည့်ရအောင် ...

```
# File: account.py
from decimal import *

class Account:
    def __init__(self, holder, acc_number, balance):
        self.holder = holder
```

```

    self.acc_number = acc_number
    self.balance = balance

    def deposit(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount for deposit!')
        self.balance += amt

    def withdraw(self, amt):
        if amt > self.balance:
            raise ValueError('Not enough balance!')
        self.balance -= amt

```

‘Account ကလပ်စ် သတ်မှတ်ပါမယ်’ လို့ ပြောဖို့အတွက် class Account: နဲ့ စရပါတယ်။ သူ အောက်မှာရှိတာက Account ကလပ်စ်ရဲ့ ဘေးဒီ (body) ပါ။ Body ဆိုတာ block ကို ပြောတာပါပဲ။ ကလပ်စ်ဘေးဒီထဲမှာ ဖန်ရှင် သုံးခု သတ်မှတ်ထားတာ တွေ့ရမယ်။ သုံးခုလုံးမှာ ပထမ ပါရာမီတာက self ဖြစ်နေတာကို သတိပြုမှုပါ။ သူ (self) က လက်ရှိအော့ဘူက် (current object) လို့ ဆိုလို တာဖြစ်ပြီး သိပ်မကြာခင် သူအိမိပါယ်ကို ရှင်းပြပါမယ်။

__init__ ဖန်ရှင်က အော့ဘူက်မှာ ပါဝင်တဲ့ ပေးပို့ရော့လဲတွေ သတ်မှတ်ပေးတယ်။ ငွေးတို့ ရဲ့ ကန်းတို့ ကန်းကိုလည်း ဒီဖန်ရှင်က initialize (စထည်) လုပ်ပေးရတာပါ။ အော့ဘူက်ရဲ့ ကန်းတို့ အခြေအနေ (initial state) ကို စတင်ပေးတဲ့ ဖန်ရှင်ဖြစ်တာကြောင့် initializer လို့ ခေါ်လေ့ရှိတယ်။ Account ကလပ်စ်ကို ကြည့်ရင် ဒီဖန်ရှင်မှာ self.holder, self.acc_number, self.balance စတဲ့ အော့ဘူက်ရဲ့ အချက်အလက်တွေနဲ့ သက်ဆိုတဲ့ ပေးပို့ရော့လဲတွေ ကြော်ကြားတာ တွေ့ရမှာ ပါ။ ဒီနေရာမှာ dot အမှတ်အသား အမိပါယ်ကို ‘၏/ရဲ့’ လို့ ယူဆရင် ဖတ်ရတာ အဆင်ပြေတယ်။ self.holder ကို ‘လက်ရှိအော့ဘူက်ရဲ့ holder’၊ self.balance ကို ‘လက်ရှိအော့ဘူက်ရဲ့ balance’ လို့ ဖတ်ရှိတယ်။ __init__ ရဲ့ ထူးခြားချက်က သူကို စိုက်ရှိက် ခေါ်သုံးရလေ့မရှိဘူး။ အော့ဘူက် တည်ဆောက်တဲ့ ဖြစ်စဉ်ရဲ့ နောက်ဆုံးအဆင့်မှာ Python က အလိုအလျောက် ခေါ်ပေးတဲ့ ဖန်ရှင် ဖြစ်တယ်။

Account instance တစ်ခုကို ဖန်တီးယူမယ်ဆိုရင် အခုလို ရေးရမှာပါ

```
Account('Amy', '0086-6002-2233', Decimal('350_000.00'))
```

ဒီအခါမှာ အော့ဘူက် တည်ဆောက်တဲ့ ဖြစ်စဉ်ကို စတင်လုပ်ဆောင်ပေးမှာ ဖြစ်တယ်။ ထည့်ပေးထားတဲ့ တန်ဖိုးတွေက __init__ ဖန်ရှင်အတွက် ဖြစ်တယ်။ self နောက်က ပါရာမီတာတွေအတွက်ပဲ ထည့်ပေးရပြီး self အတွက် တန်ဖိုးထည်မပေးရပါဘူး (self ဟာ ကလပ်စ်ရေးတဲ့ သူအတွက် သီးသန်ဖြေပြီး ကလပ်စ်ဘေးဒီထဲမှာပဲ အသုံးပြုရတာပါ)။ ဒီတန်ဖိုးတွေက လက်ရှိအော့ဘူက် ပေးပို့ရော့လဲတွေရဲ့ ကန်းတို့ တန်ဖိုးတွေဖြစ်သွားမှုပါ

```

self.holder = holder
self.acc_number = acc_number
self.balance = balance

```

ဒီသုံးကြောင်းအရ လက်ရှိအော့ဘူက်ရဲ့ holder က ‘Amy’ ဖြစ်ပါမယ်။ လက်ရှိအော့ဘူက် balance နဲ့ acc_number က Decimal('350_000.00') နဲ့ ‘0086-6002-2233’ အသီးသီး ဖြစ်ပါမယ်။ ပုံ (၉.၁) မှာလို ဖြင့်ကြည့်နိုင်ပါတယ်

holder	Amy
accountNumber	0086-6002-2233
balance	350,000.00

(Account)

ပုံ ၉.၁ အေမ့် Account အော်ဂျက် (acc3)

ဒီအော်ဂျက်ကို ရည်ညွှန်းအသုံးပြန်စိစ္စရင် ထုံးစံအတိုင်း ပေရီရေဘဲလ်နဲ့ အဆိုင်းမန် လုပ်ထား ဖို့ လိုပါတယ်။

```
acc3 = Account('Amy', '0086-6002-2233', Decimal('350_000.00'))
```

```
print(acc3.holder)      # Amy
print(acc3.acc_number)  # 0086-6002-2233
print(acc3.balance)    # 350000.00
```

ဒီနေရာမှာ self အကြောင်းကို ရှင်းပြွို လိုလာပြီ။ လက်ရှိအသုံးပြုနေတဲ့ အေမ့် Account အော်ဂျက် ဟာ 'လက်ရှိအော်ဂျက်' self ပဲ ဖြစ်တယ်။ ဒါကြောင့် အခုက္ခာမှာ acc3 နဲ့ self နှစ်ခုလုံးဟာ 'လက်ရှိအော်ဂျက်' (တစ်ခုတည်း) ပဲ။ acc3.holder နဲ့ self.holder နှစ်ခုလုံးက လက်ရှိအော်ဂျက်ရဲ့ holder ကို ဆုံးလိုတာ ဖြစ်တယ်။ holder က 'Amy' ပါ။ ဒီသဘောအတိုင်း acc3.balance ရော့ self.balance ပါ လက်ရှိအော်ဂျက်ရဲ့ balance ကို ဆုံးလိုတာ။ Decimal("350_000.00") ဖြစ်ပါမယ်။ လက်ရှိအော်ဂျက်က ပြောင်းသွားရင်ရော့ ဘယ်လိုဖြစ်မလဲ။

```
acc1 = Account('Kathy', '0086-6002-1111', Decimal("100000.00"))
```

အခုလက်ရှိ အော်ဂျက်က ကေသိ Account အော်ဂျက် ဖြစ်သွားပြီ။ ဒီအချိန်မှာ self.holder က 'Kathy' ပဲ acc1.holder လည်း ဒါပဲဖြစ်မယ်။ acc_number နဲ့ balance တို့ကိုလည်း အလားတူစဉ်းစားရမှာ ဖြစ်တယ်။ '0086-6002-1111' နဲ့ Decimal("100000.00") ဖြစ်ပါမယ်။ အခုလက်ရှိ အော်ဂျက်ဟာ သီးမြှေးတည်ရှုနေတဲ့ အော်ဂျက်တစ်ခုဖြစ်ပြီး အခုလို မြင်ကြည့်ရမှာပါ

holder	Kathy
accountNumber	0086-6002-1111
balance	100,000.00

(Account)

ပုံ ၉.၂ ကေသိ Account အော်ဂျက် (acc1)

ဆုံးလိုတာက အေမ့် Account အော်ဂျက်မှာ အကောင့်ပိုင်ရှင်၊ နံပါတ်နဲ့ လက်ကျန်ငွေအတွက် သူ

ကိုယ်ပိုင် ဖေရီရောဘဲလ်သုံးခု ရှိနေမှုဖြစ်ပြီး ကေသိ Account အော်ဂျက်ကလည်း သူ့ဟာနဲ့သူ သိုးခြား ရှိနေမှုပါ။

အခုလောက်ဆိုရင် self ရဲ့ သဘောကို နားလည်လောက်ပါပြီ။ ကလပ်စ်သတ်မှတ်တဲ့အခါ အော် ဂျက်တစ်ခုစီမှာ သီးခြားကိုယ်ပိုင် ပါရှိမဲ့ ဖေရီရောဘဲလ်တွေကို dot အမှတ်အသားအသုံးပြုပြီး self နဲ့ ရည်ညွှန်းပါတယ်။ deposit နဲ့ withdraw ကို ဆက်ကြည့်ရအောင်။

```
# File: account.py
class Account:
    ... # __init__ ကို ထပ်မပြား ချွ်ထားခဲ့တယ်
    def deposit(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount for deposit!')
        self.balance += amt

    def withdraw(self, amt):
        if amt > self.balance:
            raise ValueError('Not enough balance!')
        self.balance -= amt
```

ဒီဖန်ရှင်တွေမှာလည်း ပထမ ပါရာမီတာက self ဖြစ်နေတာ တွေ့ရမှုပါ။ လက်ရှိအော်ဂျက်ရဲ့ အခြေ အနေအောက်မှာ အလုပ်လုပ်ပေးမဲ့ ဖန်ရှင်ရဲ့ ပထမ ပါရာမီတာက self ဖြစ်ရပါမယ်။ ဒီဖန်ရှင်တွေက အော်ဂျက်အပေါ်မှာ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေပါပဲ။ Account instance တွေအပေါ်မှာ deposit နဲ့ withdraw လုပ်ဆောင်လို့ ရမှုဖြစ်တယ်

```
acc3.withdraw(Decimal('50_000.00'))
acc1.deposit(Decimal('25_000.00'))
```

ပထမတစ်ခုက အော့မှု အကောင့်ကနေ ငွေထွက် (withdraw) လုပ်တာပါ။ သိပြီးဖြစ်တဲ့အတိုင်း acc3 နဲ့ self ဟာ လက်ရှိအော်ဂျက် ဖြစ်တယ်။ withdraw ကို ကြည့်ရင် amt က လက်ရှိအော်ဂျက်ရဲ့ balance ထက် များနေရင် exception raise လုပ်ထားတယ်။ ရှိတဲ့ လက်ကျွန်ငွေထက် ပို့ထုတ်လို့ မရ သင့်ဘူး။

```
    self.balance -= amt
```

ကတော့ လက်ရှိအော်ဂျက်ရဲ့ balance ကနေ amt နှုတ်လိုက်တာပါ။ အော့မှု အကောင့်မှာ လက်ကျွန်ငွေ 300,000.00 ဖြစ်သွားမယ်။ ဒုတိယတစ်ကြောင်းက ကေသိ အကောင့်ကို ငွေသွင်း (deposit) လုပ်တာ။ ဒီတစ်ခါကျတော့ လက်ရှိအော်ဂျက်က ကေသိအကောင့်ပေါ့။ self.balance += amt က ကေသိအကောင့် သက်ဆိုင်တဲ့ အော်ဂျက်ရဲ့ balance ကို amt ပမာဏ ပေါင်းပေးတာ။ ဒါကြောင့် ကေသိ အကောင့်လက်ကျွန်ငွေ 125,000.00 ဖြစ်သွားပါမယ်။

```
print(acc3.balance) # 300000.00
print(acc1.balance) # 125000.00
```

holder	Amy	holder	Kathy
accountNumber	0086-6002-2233	accountNumber	0086-6002-1111
balance	300,000.00	balance	125,000.00

(Account)

(Account)

(က) Account အော်ဂျက် (acc3)

(ခ) Account အော်ဂျက် (acc1)

ပုံ ၉.၃ After withdraw and deposit on acc3 and acc1 respectively

Attributes, Methods and Members

အော်ဂျက်မှာပါဝင်တဲ့ အချက်အလက်ကို attribute လိုခေါ်ပြီး အော်ဂျက်ပေါ်မှာ လုပ်ဆောင်ရှင် တဲ့ အော်ပရေးရှင်းကို method (မက်သဒ်) လိုခေါ်တယ်။ အော်ဂျက်တစ်ခုမှာ ပါဝင်တဲ့ attribute တွေနဲ့ method တွေ နှစ်မျိုးလုံးကိုခြင်ပြီး members တွေလို့ ခေါ်ပါတယ်။

၉.၂ Access Control

အိမ်ရှင်တစ်ယောက်ဟာ သူအိမ်မှာရှိသမျှ ပစ္စည်းတွေ (သို့) အခန်းတွေ အားလုံးကိုတော့ အိမ်လာတဲ့ ပြည့်သည်ကို အသုံးပြုခဲ့ မပေးချင်ပါဘူး။ အိမ်ရှင်ကိုယ်တိုင်က အားလုံးကို အသုံးပြုခိုင်ပေမဲ့ ပြည့်သည်ဖြစ် သူကတော့ အိမ်ရှင်ခွဲပြုတာကိုပဲ အသုံးပြုသင့်ပါတယ်။ ထိုနည်းတူစွာ ကလပ်စိုင်ရှင် (ကလပ်စ်သတ်မှတ်သူ) ဟာ သူကလပ်စ် instance တွေမှာ ပါဝင်တဲ့ members တွေကို အသုံးပြုသူတွေ ‘အကန့်အသတ်မရှိ ဆန္ဒရှိသလိုသာသုံး’ ခွင့်ပြုလိုမပါဘူး။ ကလပ်စိုင်ရှင်ဟာ ကလပ်စ် instance တွေ မှန်ကန်တဲ့ state မှာ အမြန်နေအောင် ကန့်သတ် ထိန်းချုပ်ထားရပါတယ်။ အသုံးပြုသူတွေ မသိသင့်တဲ့ အသုံးမပြုသင့်တဲ့ အတွင်းပိုင်းတည်ဆောက်ပဲ အသေးစိတ်နှင့်သက်ဆိုင်တဲ့ implementation details တော်းလည်း ဖုံးကွယ်ထားနိုင်ဖို့ နည်းလမ်းတွေ လိုအပ်ပါတယ်။ ဒီအခန်းကဏ္ဍကို access control လို့ ခေါ်ပါတယ်။ ပရိုဂရမ်းမင်း စလေ့လာသူတွေအတွက် အခုပြောတာတွေကို မြင်နိုင်ဖို့ အခက်အခဲ ရှိနေမှာပါ။ ဥပမာတချို့နဲ့ ရှင်းပြုမယ်။

Account instance တွေမှာ ပြသုနာတာချို့ ရှိနေပါတယ်။ withdraw နဲ့ deposit မက်သဒ်တွေ က ငွေရှိတာထက် ပိုထုတ်လို့မရအောင် ငွေသွင်းရင်လည်း ဖြစ်သင့်တဲ့ပမာဏပဲ သွင်းအောင် ကာကွယ်ထားတာ တော်းရမှာပါ။ ဒါပေမဲ့ ခက်တာက အသုံးပြုသူတွေက ဒီအကာအကွယ်တွေကို ကျော်ပြီး အခုလို လုပ်လိုရနေပါလိမ့်မယ်

```
acc1.balance = Decimal(' -50_000.00 ')      #
acc1.acc_number = None                      #
```

လက်ကျွန်ငွေဟာ အနှုတ်တန်ဖိုး ဖြစ်လို့မရသင့်ဘူး။ ဘက်အကောင့်တစ်ခုမှာ အကောင့်နံပါတ်ကလည်း ရှိကိုရှိပါမယ်။ ဒီလိုသာ ထင်သလို တန်ဖိုး ထည့်လို့ရနေရင် ပြသုနာပဲ။ အခုလိုဆိုရင် acc1 အော်ဂျက်ဟာ invalid state (မှန်ကန်တဲ့ အနေအထား) ဖြစ်သွားမှာပါ။ အသုံးပြုသူက သေချာစဉ်းစားပြီး မဖြစ်သင့်တာ မလုပ်နဲ့ပေါ့လို့ စောဒကတ်စရာတော့ ရှိပါတယ်။ လျှပ်စစ်မီး ပလပ်ပေါက်ထဲ သူတော်းလည်းထိုးထည့်လို့ ပါတ်လိုက်ရင် user fault လိုပြောတာက မှားတော့မှားပါဘူး။ ဒါပေမဲ့လည်း မီးပလပ်ပေါက်ထဲတို့လည်းအနေနဲ့ အတတ်နိုင်ခဲ့း အဲဒီလိုလုပ်လို့မရအောင်၊ အန္တရာယ်နည်းနိုင်သမျှ နည်းအောင် ဒီမိုင်း

ပြုလုပ်ထားသင့်ပါတယ်။

အော့သ်ဂျက် attribute တွေကို တိုက်ရှိက် အသုံးမပြုစေချင်တဲ့အခါ Python စေလေထုံးစံက attribute နံမည်ရေးမှာ _ (underscore) ထည့်ပေးပါတယ်။

```
class Account:
    def __init__(self, holder, acc_number, balance):
        self._holder = holder
        self._acc_number = acc_number
        self._balance = balance
```

Attribute နံမည်တွေကို _holder, _acc_number စသည်ဖြင့် _ နဲ့စထားတာ သတိပြုပါ။ ဒီလိုနံမည် တွေဆိုရင် ကလပ်စံအသုံးပြုသူ (အားကျေပြာရင် ကလပ်စံကနေ ဖန်တီးယူတဲ့ instance အသုံးပြုသူ) အနေနဲ့ ‘တိုက်ရှိက် အသုံးမပြုရ’ လို့ အမိပါယ်ရတယ်။ ဒါက Python ပရိုက်မှု အများစု လက်ခံထားတဲ့ စေလေထုံးစံတစ်ခုသာ ဖြစ်တယ်။ အေးလုံးက _ နဲ့စုတဲ့ attribute (မက်သံတွေလည်းပါတယ်) တွေဆိုရင် တိုက်ရှိက်ယူမသုံးသင့်ဘူးလို့ နားလည် လက်ခံထားတယ်။ ဒါပေမဲ့ အခြား programming language တွေလို့ လုံးဝသုံးလို့မရအောင် ကန်သတ်လိုတော့ Python မှာ မရနိုင်ဘူး။ သုံးချင်သပဆိုရင်လည်း သုံးလိုတော့ရတယ်။ ဒါပေမဲ့ ဖြစ်လာမဲ့အကျိုးဆက်ဟာ အသုံးပြုသူရဲ့ တာဝန်သာဖြစ်တယ်။ ဒါကြောင့် ဒီလိုလုပ်လို့ ရတယ်

```
acc1._balance = Decimal('-50_000.00')      # မလုပ်သင့်ပါ
```

ဒါပေမဲ့ ထုံးစံကိုဖောက်ဖျက်တာ ဖြစ်တယ်။ မလုပ်သင့်တဲ့ အရာပေါ့။

ဒီလို ဘာကြောင့် မသုံးသင့်လဲ အခြားအကြောင်းအရင်း တစ်ခုလည်း ရှိပါသေးတယ်။ ကလပ်စံပိုင်ရှင် က သူကလပ်စံ implementation details တွေကို အကြောင်းအမျိုးမျိုးကြောင့် ပြင်ဆင် ပြောင်းလဲ ရလေ့ရှိတယ်။ ဥပမာ ပိုမြန်တဲ့ ဒါမှမဟုတ် မမိမိရှိ အစားသက်သာစေမဲ့ နည်းလမ်းတွေကို ပြောင်းလဲ အသုံးပြုရနိုင်တယ်။ ဘက်အကောင်းတွေမှာ ငွေသွင်းငွေထုတ် စာရင်းကို အစဉ်အတိုင်း သိမ်းထားဖို့ လို တယ်ဆုံးပါစိုး။ List တစ်ခုထဲမှာ သွင်းငွေပမာဏကို အပေါင်းတန်ဖိုး၊ ထုတ်ငွေပမာဏကို အနှုတ်တန်ဖိုးနဲ့ သိမ်းထားနိုင်တယ်။ တစ်သိန်းသွင်းပြီး နှစ်သောင်းခဲ့ ထပ်သွင်းတယ်။ နောက်တော့ သုံးသောင်းခဲ့ ပြန်ထုတ်တယ် ဆုံးဖို့။ List ထဲမှာ အခုလု့ ရှိနေရပါမယ်

```
[100000.00, 25000.00, -35000.00]
```

ဒီတန်ဖိုးတွေကို ပေါင်းလိုက်ရင် လက်ရှိလက်ကျွန်ုင် balance ကိုရမှုပါ။ ဒီလိုဆိုရင် သီးသန့် attribute တစ်ခုနဲ့ balance ကို သိမ်းထားဖို့ မလိုအပ်တော့ဘူး။ ငွေသွင်းငွေထုတ် စာရင်းကနေ တွေက် ယူနိုင်တယ်။ အဖက်ဖက်ကနေ စဉ်းစားသုံးသပ်ပြီး ပိုင်ရှင်က သူကလပ်စံကို အခုလု့ update လုပ်ဖို့ ဆုံးဖြတ်ချက် ချုနိုင်ပါတယ်

```
from decimal import *

class Account:
    def __init__(self, holder, acc_number, balance):
        self._holder = holder
        self._acc_number = acc_number
        self._transactions = [balance]

    def deposit(self, amt):
```

```

if amt <= Decimal(0.00):
    raise ValueError('Invalid amount for deposit!')
self._transactions.append(amt)

def withdraw(self, amt):
    if amt > self.balance:
        raise ValueError('Not enough balance!')
    self._transactions.append(-amt)

@property
def balance(self):
    return sum(self._transactions)

```

လောလေဆယ် အသေးစိတ်တွေက သိပ်အရေးမကြီးဘူး။ `_balance` ဖြူတ်လိုက်တာကိုသာ အဓိက အရှုံးပါ။ ဒီအခါ သူကို တိုက်ရိုက် သုံးထားတဲ့ ကုဒ်အားလုံး (ဥပမာ `acc1._balance`) ကလပ်စ် ဗုံးရှင်း အသစ်ကို `update` ယူပြီးသုံးရင် ပြဿနာတက်သွားမှာပါ။ ပိုကောင်းတဲ့ ဗုံးရှင်းအသစ်ကို သုံးချင်တယ် ဆိုရင် တိုက်ရိုက်သုံးထားတဲ့နေရာတွေ အားလုံးကို လိုက်ပြင်ပါလိမ့်မယ်။ ကလပ်စ်ပိုင်ရှင်က တမင်တကာ ဖုံးကွယ်ထားတဲ့ implementation details တွေကို တိုက်ရိုက်မသုံးရင် ဒီလိုပြဿနာမျိုးတွေ မဖြစ်နိုင်ဘူးပေါ့။

ဒီလောက်ဆိုရင် access control ကာကြောင့် အရေးကြီးသလဲ နားလည် သဘောပါက်လောက် ပါပြီ။ ကလပ်စ် ပိုင်ရှင်ရော အသုံးပြုသူတွေ အတွက်ပါ ကရိစိုက်စဉ်းစားရမဲ့ ကိစ္စဖြစ်ပါတယ်။ ဘယ်သူမ ဆို တရားဝင် အသုံးပြုခွင့် ပေးထားတဲ့ `members` ကို ကလပ်စ်ရဲ့ *interface* သို့ API (*Application Programming Interface*) လိုအပ်ပါတယ်။ ကလပ်စ်ပိုင်ရှင်က စဉ်းစားဆင်ပြင် သုံးသပ်ပြီး ကလပ်စ် API က ဘယ်လိုဖြစ်သင့်လဲ တိတိကျကျ သတ်မှတ်ပေးရတာပါ။ ဖုံးကွယ်ထားသင့်တဲ့ implementation details တွေကို ပေးသိ/သုံး မိတာဟာ နောင်တစ်ချိန် ကလပ်စ်ကို ပြင်ဆင်မွမ်းမဖို့ အဟန်အတား ဖြစ်စေပါတယ်။ အသုံးပြုသူတွေကလည်း မသိ/မသုံးစေချင်တဲ့ internal တွေကို အသုံးပြုခြင်းဖြင့် စဉ်းမဖောက်ဖို့ လိုပါတယ်။

၉.၃ Composition or Has-a Relationship

အခုတွေခဲ့တဲ့ `Account` instance တွေက ပိုင်ရှင် နံမည်ကိုပဲ attribute အနေနဲ့ သိမ်းတယ်။ နံမည် အပြင် အကောင့် ပိုင်ရှင် မှတ်ပုံတင်နံပါတ် (NRIC)၊ မွေးသက္ကရာဇ်၊ နေရပ်လိပ်စာတိုကိုလည်း ဘက် တွေ့က မှတ်သား သိမ်းဆည်းရှုပါတယ်။ ပရိုကရမ်လဲမှုဗုလည်း ဒီအချက်အလက်တွေကို ထင်ဟပ်ဖော်ပြန့် လိုပါမယ်။ `Account` ကလပ်စ်မှာ ဒီအချက်အလက်တွေအတွက် attribute အသစ်တွေ ထပ်ထည့်ရမလား၊ ဒါမှုမဟုတ် ပိုင်ရှင်တစ်ယောက်ချင်းကို အော့ဘ်ဂျက်အနေနဲ့ ဖော်ပြရမလား စဉ်းစားစရာ ဖြစ်လာပါတယ်။ အပြင်မှုရှိတဲ့ ဘက်အကောင့်ကို `Account` instance နဲ့ ကိုယ်စားပြုတယ်ဆိုရင် အပြင်က အကောင့်ပိုင်ရှင်ကို `Holder` instance နဲ့ ဖော်ပြတာဟာလည်း ကျိုးကြောင်းဆီလျော်တယ်လို့ပဲ ယူဆရမှာပါ။ အော့ဘ်ဂျက် ဖြစ်သင့်/မသင့် စဉ်းစားဆုံးဖြတ်တဲ့အခါ အခြေခံအကျဆုံး မေးခွန်းတစ်ခုကတော့ မိမိ ဖော်ပြလိုတဲ့ အရာဟာ attributes တွေအပြင် ဘယ်လို methods တွေ ပါဝင်သင့်လဲ ဆိုတာပါ။ Attribute တွေပဲ ရှိပြီး method တစ်ခုမှ မရှိရင် အော့ဘ်ဂျက်အနေနဲ့ မရှုပြင်သင့်ဘူး။ `Holder` instance တွေ အပေါ်မှာ လုပ်ဆောင်လို့ ရသင့်တဲ့ method တချို့ ခေါင်းထဲပေါ်လာဖို့ သိပ်မခက်သင့်ဘူး။ အခြားလည်း ရှိအုံမှာပါ။

```
holder1.change_address(new_address)
```

အပြင်မှုလည်း လိပ်စာပြောင်းတာ မကြေခဏဖြစ်လေးရှိပါတယ်။ Attributes ရော methods တွေပါရှိမယ်ဆုံးရင် အော်ဂျက်အနေနဲ့ စဉ်းစားနိုင်ပါတယ်။ ဒါဆုံးရင် ကလပ်စ်သတ်မှတ် ရပါမယ်

```
class Holder:  
    def __init__(self, name, dob, nric, gender):  
        self._name = name  
        self._dob = dob  
        self._nric = nric  
        self._gender = gender  
  
    # ...
```

ဒီလောက်နဲ့ အော့ဘ်ပျက် ဖန်တီးယူလို ရပါပြ (ကျွန်တဲ့ ပါသင့်တာတွေ ခဏနောက် ထပ်ဖြည့်ပါမယ်)။ ဒီ စက်ရှင်ငဲ့ အဓိက အင်ကြာင်းအရာကို အပ်ကြည့်ရအောင်။ ဘဏ်အကောင့်နဲ့ အကောင့်ပိုင်ရှင် ဆက်စပ် နေတာကို ဖော်ပြချင်ရန် ဒီလိုပါ

```
holder1 = Holder('Kathy',
                  date(1990, 3, 10),
                  'MaRaNa (N) 1343232',
                  'F')

acc1 = Account(holder1, '0086-6002-1111', Decimal('100_000.00'))
```

အကောင့်ပိုင်ရှင် ကေသိကို ကိုယ်စားပြုတဲ့ အော့ဘ်ဂျက် ဖော်တိုးယူမယ်။ ပြီးတော့ ကေသိအကောင့်မှာ အဲ ဒီအော့ဘ်ဂျက်ကို ထည့်ပေးလိုက်ရမယဲ့။ Account မှာ Holder က attribute တစ်ခုအနေနဲ့ ရှိနေမှာပါ။ အော့ဘ်ဂျက်နှစ်ခု အခုလို ဖွဲ့စည်းထားတာကို composition လို့ ခေါ်ပါတယ်။ Composition ဟာ ပိုင်ဆိုင်ခြင်း (သို့) ရှိခြင်း ကိုလည်း ဖော်ပြတယ်။ ‘ဘက်အကောင့်တစ်ခုမှာ အကောင့်ပိုင်ရှင်တစ်ယောက် ရှိတယ်’၊ ‘ကားတစ်စီးမှာ အင်ဂျင်တစ်လုံး ပါရှိတယ်’။ ဒီလိုဆက်စပ်မှုမျိုးကို ဖော်ပြတဲ့ အော့ဘ်ဂျက်နှစ်ခု ဟာ has-a relationship ရှိတယ်လို့ ပြောလေ့ရှိတယ်။

အော်ဂျက်တစ်ခုဟာ တစ်ခုထက်ပိုတဲ့ အမြား အော်ဂျက်တွေနဲ့ has-a relation ရှိနိုင်တာပေါ့။ ဥပမာ အကောင့်ဖွင့်တဲ့ ဘက်ခဲ့ကို အော်ဂျက်အနေနဲ့ ယူဆမယ်ဆိုရင်

```
class Branch:
    def __init__(self, name, address):
        self._name = name
        self._address = address
    # ...

class Account:
    def __init__(self, holder, acc_number, balance, branch):
        # ... അഥവാ attributes ടോറ്റിമാനി അല്ലെങ്കിൽ
        self.branch = branch
```

Account နဲ့ သူရဲ့ Branch ကို ခိုတ်ဆက်မယ်ဆိုရင်

```
# ... holder1 ရှိမယ် မပြထားတော့တော့
branch1 = Branch('Mandalay', '232A, 62 St, Chanayetharsi')
acc1 = Account(holder1,
                '0086-6002-1111',
                Decimal('100_000.00'),
                branch1)
```

Types of Composition and Sharing

ဘဏ်ခဲ့တစ်ခုတည်းမှာ ဖွင့်ထားတဲ့ ဘဏ် အကောင့်တွေဟာ Branch အော့သိဂျက်တစ်ခုတည်းကိုပဲ sharing လုပ်ထားပြီး အသုံးပြုနိုင်ပါတယ်။

```
branch1 = Branch('Mandalay', '232A, 62 St, Chanayetharsi')
acc1 = Account(holder1,
                '0086-6002-1111',
                Decimal('100_000.00'),
                branch1)
acc2 = Account(holder2,
                '0086-6002-1111',
                Decimal('100_000.00'),
                branch1)
```

acc1 နဲ့ acc2 နှစ်ခုလုံးက branch1 အော့သိဂျက်တစ်ခုတည်းကိုပဲ မျှသုံးထားတာပါ။ လူတစ်ယောက် တည်းက အကောင့် နှစ်ခု ဖွင့်ထားရင် Holder အော့သိဂျက်တစ်ခုတည်းကို မျှသုံးပါမယ်။

```
holder1 = Holder('Kathy',
                  date(1990, 3, 10),
                  'MaRaNa (N) 1343232',
                  'F')
acc1 = Account(holder1, '0086-6002-1111', Decimal('100_000.00'))
acc5 = Account(holder1, '0086-6002-5522', Decimal('500_000.00'))
```

Has-a relationship ကို အမျိုးအစားတွေခဲ့ကြည့်ရင် one-to-one, one-to-many, many-to-many စသည်ဖြင့်တွေရပါတယ်။ Car နဲ့ Engine ဟာ one-to-one ဖြစ်ပါမယ်။ Student နဲ့ Course (သို့) Actor နဲ့ Film ဆိုရင်တော့ many-to-many relationship ပေါ့။ One-to-many ကို Bank နဲ့ Branch, House နဲ့ Resident တိုအကြား တွေရမှာပါ။

Bank နဲ့ Account တို့ အကြား one-to-many relationship ကို ဒီအခန်းနောက်ဆုံးမှာ တွေ့ရပါမယ်။ One-to-one အတွက် Car နဲ့ Engine, many-to-many အတွက် Student နဲ့ Course ကို ဖော်ပြထားတာ လေ့လာကြည့်ပါ။

```
class Engine:
    def __init__(self, horsepower):
        self._horsepower = horsepower

class Car:
```

၁၆၁

```
def __init__(self, model, engine):
    self._model = model
    self._engine = engine

# Creating an engine object
my_engine = Engine(horsepower=200)

# Creating a car object with the engine
my_car = Car(model="Toyota Camry", engine=my_engine)

class Student:
    def __init__(self, name):
        self.name = name
        self.courses = []

    def enroll(self, course):
        self.courses.append(course)

class Course:
    def __init__(self, name):
        self.name = name
        self.students = []

    def add_student(self, student):
        self.students.append(student)

# Creating student objects
sandar = Student("Sandar")
waiyan = Student("Waiyan")

# Creating course objects
math_course = Course("Mathematics")
science_course = Course("Science")

# Enrolling students in courses
sandar.enroll(math_course)
sandar.enroll(science_course)
waiyan.enroll(math_course)

# Adding students to courses
math_course.add_student(sandar)
math_course.add_student(waiyan)
```

```
science_course.add_student(sandar)
```

ကျောင်းသူ/သား တစ်ယောက် တက်ရောက်ဖို့ စာရင်းပေးထားတဲ့ course တွေအတွက် course တစ်ခုကို တက်ရောက်မဲ့ ကျောင်းသူ/သား တေအတွက် list ကို သုံးထားတာ ရုပ်ပြုကြည့်ပါ။ Has-a relationship မှာ many ဘက်ခြမ်းကို ဖော်ပြန့်အတွက် အော်ဂျက်တွေ တစ်စုတစ်စည်းတည်း ထားနိုင်တဲ့ list, set, dictionary စတေတွေ သုံးရပါတယ်။

၉.၄ Properties

Access control စည်းကမ်းအတိုင်း လိုက်နာရမယ်ဆိုရင် Holder ရဲ့ attributes တွေကို တိုက်ရိုက် မသုံးသင့်ဘူး။ ဒါဆို attributes တွေကို သုံးဖို့လိုအပ်ရင် ဘယ်လိုလုပ်ရမလဲ မေးစရာ ရှိပါတယ်။ ကလပ်စိုင်ရှင် အနေနဲ့ attribute ကို မက်သခ်ကနေတစ်ခုင့် အသုံးပြုလိုရအောင် လမ်းဖွင့်ပေးထားနိုင်ပါတယ်။ မဖြစ်သင့်တာ မဖြစ်အောင် မက်သခ်မှာ စိစစ်လိုရတယ်။

```
class Holder:
    def __init__(self, name, dob, nric, gender):
        self._name = name
        self._dob = dob
        self._nric = nric
        self._gender = gender

    def get_name(self):
        return self._name

    def set_name(self, name):
        if name.strip() == '':
            raise ValueError('Name cannot be empty!')
        self._name = name.strip()
```

set_name မှာ နံမည်ကို empty string မဖြစ်အောင် စစ်ထားတယ်။ အသုံးပြုသူအနေနဲ့ နံမည်ကို ယူကြည့်တာ၊ ပြောင်းတာ လုပ်မယ်ဆိုရင် မက်သခ်ကိုပဲ သုံးရမှာပါ။ ဥပမာ

```
print(holder1.get_name())
holder1.set_name('Amy Lynn')
holder1.set_name('')           # ValueError ဖြစ်မယ်
```

(- နဲ့ မစရင် API, သုံးလိုရတယ်)။

အခြား attribute တွေလည်း လိုအပ်ရင် အလားတူ မက်သခ်တွေနဲ့ API ထုတ်ပေးထားလိုရပါတယ်။ ဒီနည်းလမ်းအပြင် Python မှာ *managed attributes* ဆိုတာလည်း ရှိပါသေးတယ်။ Managed attributes တွေကို *properties* လိုလည်း ခေါ်တယ်။ Managed attribute သုံးထားတဲ့ ဥပမာကို လေ့လာကြည့်ပါ

```
class Holder:
    def __init__(self, name, dob, nric, gender):
        self._name = name
        self._dob = dob
```

```

        self._nric = nric
        self._gender = gender

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, name):
        if name.strip() == '':
            raise ValueError('Name cannot be empty!')
        self._name = name

    @property
    def dob(self):
        return self._dob

    @dob.setter
    def dob(self, value):
        self._dob = value

    # ...

```

@ သက်တန် စထားတဲ့ @property, @name.setter စသည်ဖြင့် တွေ့ရပါတယ်။ Property decorator လိုခေါ်ပါတယ်။ ဒါလေးတွေက နောက်ကွယ်မှုတော့ ဖန်ရှင်တစ်ခုပါပဲ။ property() ဖော်ရှင် က သူတို့ရဲ့ နောက်ကွယ်ကနဲ့ အလုပ်လုပ်ပေးမှာပါ။ လောလောဆယ် အသေးစိတ်တွေ ခဏထားလိုက်ပြီး အသုံးပြုနည်းကို အမိကထား ကြည့်ရအောင်။ Decorator ရဲ့ အောက်မှာ ရှိနေတာက ပုံမှန်မက်သခံတွေ ပါပဲ။ Property decorator က အမိကဘာလုပ်ပေးလဲ။ မက်သခံတစ်ခုကို ပုံမှန် attribute ပေါ်ရော ဘဲလို သုံးလိုရအောင် လုပ်ပေးတာပါ။ ဥပမာ name မက်သခံကို ခေါ်ရင် holder1.name() လို ရေး နေစရာ မလိုဘူး။ holder1.name လို ရေးရုံပဲ။ တာကယ်တမ်း နောက်ကွယ်မှု အလုပ် လုပ်တာကတော့ မက်သခံကပဲ လုပ်သွားမှာပါ။ name အတွက် @name.setter, dob အတွက်ဆုံးရင် @dob.setter စသည်ဖြင့် ဖြစ်ရပါမယ်။

အဆိုင်းမန် လုပ်ရှင်လဲ ဒီသဘောပဲ။ holder1.name = 'Amy Lynn' လို ရေးရင် @name.setter decorator နဲ့ မက်သခံက အလုပ်လုပ်သွားမှာ ဖြစ်တယ်။ holder1.name('Amy Lynn') မက်သခံ ခေါ်တာနဲ့ သဘောတရား တူမယ်။ ဒါကြောင့် အခုလို စမ်းကြည့်ရင်

```
holder1.name = ''
```

ValueError exception ဖြစ်ပါလိမ့်မယ်။ Property decorator သုံးထားတဲ့အတွက် dob ကိုလည်း attribute အနေနဲ့ သုံးလိုရပါတယ်။

```
println(holder1.dob)
holder1.dob = date(1990, 3, 11)
```

Python ပရိုဂရမ်မာတွေက managed attribute အသုံးပြုတာကို Pythonic ပိုဖြစ်တဲ့ နည်းလမ်းလို ယူဆကြပါတယ်။ ဒါကြောင့် ရိုးရိုး get_name, set_name မက်သခံတွေနဲ့ attribute သုံး

လေ့မရှိဘူး။ ဘိုင်နာအနေနဲ့ မက်သဒ်ဖြစ်သင့်တာတွေကို attribute အနေနဲ့ မရှုမြင်ကြည့်မိဖို့ သတိပြုသင့်ပါတယ်။ withdraw နဲ့ deposit က attribute မဖြစ်သင့်ဘူး။ ဒါတွေက အော်ဂျက်အပေါ်မှာ လုပ်ဆောင်လိုရတဲ့ အော်ပရေးရှင်းတွေ အနေနဲ့ပဲ ရှုမြင်ရမှာပါ။ Attribute ဆိုတာက အော်ဂျက်ပိုင်ဆိုင်ထားတဲ့ အချက်အလက်သာ ဖြစ်တယ်။ ဒီအချက်အလက်တွေကိုမှ လိုအပ်ရင် managed attribute အနေနဲ့ အသုံးပြုခွင့် ပေးရတာဖြစ်တယ်။

Derived Attributes/Properties

Holder ရဲ့ အသက်ကို ပေါ်ရောဲလ်အနေနဲ့ ထားစရာမလိုဘဲ မွေးသက္ကရာဇ်ကနေ တွက်ယူလိုရတယ်။ ဒီလိုမျိုး attribute တွေကို derived attributes လို ခေါ်လှုရှိတယ်။ ငြင်းတိုးအတွက် ပေါ်ရောဲလ်ရှိဖို့ မလိုအပ်တာကြောင့် နေရာကုန် သက်သာမယ်။ အသုံးပြုတဲ့အခါမှ လိုတဲ့တန်ဖိုးကို တွက်ယူတာမို့လို အချိန် အကုန်အကျိုးမယ်။ တာချို့ တကယ့်လာက်တွေ့ အခြေအနေတွေမှာ နေရာနဲ့ အချိန် အကုန်အကျ ကို ဆုံးစစ်သုံးသပ်ပြီး တန်ဖိုးအနေနဲ့ သိမ်းထားမလား၊ ဒါမှုမဟုတ် တွက်ချက်ယူမလား ဆုံးဖြတ်ရတတ်ပါတယ်။ Derived attributes တွေက နေရာသက်သာပေမဲ့ အချိန်အကုန်အကျ ရှိတယ်လို့ အခြေခံအဆင့်မှာ သိထားသင့်ပါတယ်။

```
class Holder:
    def __init__(self, name, dob, nric, gender: str):
        # ...
        self._dob = dob
        # ...

    @property
    def age(self):
        return age_today(self._dob)
```

စာမျက်နှာ (၁၆၅) က Account ကလပ်စုံ balance ကို derived attribute အနေနဲ့ တွေ့ရမှာပါ။ Transaction (ငွေအဝင်အထွက်ကို ဆိုလို) အကြိမ်အရေအတွက် များရင်များသလို balance တွက်ထုတ်တဲ့အခါ အချိန်အကုန်အကျလည်း များမှာဖြစ်တယ်။

```
class Account:
    def __init__(self, holder, acc_number, balance):
        # ...
        self._transactions = [balance]
        # ...

    @property
    def balance(self):
        return sum(self._transactions)
```

self အကြောင်း သိကောင်းစရာ

self နဲ့ ပါတ်သက်ပြီး ထိုက်သင့်သလောက် ဖော်ပြခဲ့ပြီးပါပြီ။ ဒီလောက်သိထားရင်ကို ဘိုင်နာအဆင့်အတွက် လုံလောက်ပါတယ်။ ဒါပေမဲ့ စိတ်ဝင်စားလို ပိုသိချင်တဲ့သူတွေအတွက်ရော သိထားသင့်တဲ့အတွက်ပါ တာချို့ဟာလေးတွေ ထပ်ကြည့်ကြပါမယ်။

အော်ဂျက်တစ်ခု စဖန်တီးတဲ့အခါ heap လိုခေါ်တဲ့ မမ်းမှုရှိ ရော်ယာထဲမှာ သူအတွက် လိုအပ်တဲ့ နေရာ အရင်ပေးရတယ် (`__init__` အလုပ်မလုပ်ခင် ဒီကိစ္စကို အရင်လုပ်ရတာ)။ နေရာပေးပြီးသားရင်

အဲဒီအော်ဂျက်ကို ရည်ညွှန်းလိုပဲ reference ကို ရရှိပါတယ် (reference ဆိုတာ အော်ဂျက်ကို ရည်ညွှန်းတဲ့ ဖေရီရောဘဲလ်ကို ပြောတာပါပဲ)။ ဒီအဆင့်ပြီးရင် အကြမ်းထည်အဆင့် အော်ဂျက်တစ်ခု ရ နေပါပြီ။ ဒီအော်ဂျက်ကို initialization ဆက်လုပ်ရပါမယ်။ `__init__` ဖန်ရှင်က initialize လုပ် ပေးရမဲ့ အော်ဂျက်ကို သိရပါမယ်။ ဒါကြောင့် `__init__` ခေါ်တဲ့အခါ ခုန် referene ကို `self` နေရာမှာ Python က အလိုအလျောက် ထည့်ပေးသွားမှာပါ။

အော်ဂျက် အချေထည် (properly initialized object) တည်ဆောက်ပြီးသွားတဲ့အခါ သူ အတွက် reference ကို ပြန်လည်ရရှိမှာပါ။ ဒီ reference နဲ့ အော်ဂျက်ကို ဆက်လက် အသုံးပြုလို ရ တာဖြစ်တယ်။ ဥပမာ

```
| acc4 = Account('Waiyan', ...)
```

ဒါဟာ ပြန်ရတဲ့ reference ကို acc4 နဲ့ ဖမ်းယူထားလိုက်တဲ့ သဘောပဲ။

`withdraw`, `deposit` မက်သာဒွေ ခေါ်တဲ့အခါမှာလည်း `self` နေရာမှာ လက်ရှိအော်ဂျက်ကို Python က ထည့်ပေးတာ ဖြစ်တယ်။

```
| acc4.deposit(Decimal('50_000.00'))
```

အခုလို ခေါ်တဲ့အခါ acc4 ကို `deposit` မက်သာ `self` နေရာမှာ ထည့်ပေးမှာပါ။ ဒီလို transformation လုပ်ပေးတယ်လို ယူဆနိုင်တယ်

```
| acc4.deposit(acc4, Decimal('50_000.00'))
```

မှတ်ချက်။ ။ ။ အတိအကျ ဒီလိုဖြစ်တယ်လို မဆိုလို။ တကယ့်တကယ်က `Account.deposit(ac4, Decimal('50_000.00'))` အဖြစ် ပြောင်းပေးတာပါ။ ဒါပေမဲ့ ဘိုင်နာအတွက် လက်ရှိ knowledge နဲ့ ဒါကိုနားလည်နိုင်ဖို့က စေလွန်းသေးတယ်။

Instance Variables and Instance Methods

အော်ဂျက်တစ်ခုစီက သီး၏ပိုင်ဆိုင်တဲ့ ဖေရီရောဘဲလ်တွေကို *instance variable* တွေလိုလည်း ခေါ်ပါတယ်။ *Instance attribute* ပြောရင်လည်း အဓိပ္ပာယ်တူတူပါပဲ။ အော်ဂျက်တစ်ခုစီက သီး၏ပိုင်ဆိုင်တဲ့ attribute တွေပေါ့။

Instance method တွေကတော့ လက်ရှိအော်ဂျက်ရဲ့ အခြေအနေအောက်မှာ အလုပ်လုပ်တဲ့ မက သာဒွေဖြစ်ပြီး *instance variable* တွေကို အသုံးပြုလေ့ရှိတယ် (`acc1.withdraw(amt)` မှာ `withdraw` ဟာ လက်ရှိအော်ဂျက် `acc1` အခြေအနေအောက်မှာ အလုပ်လုပ်တယ်)။ *Instance method* ဟာ လက်ရှိအော်ဂျက်နဲ့ တွဲဖက်လုပ်ဆောင်ရတာ ဖြစ်တာကြောင့် `self` ပါရမိတာ မပါလို မရဘူး။

၉.၅ Class Attributes and Methods

Class attribute ဆိုတာ ကလပ်စာတစ်ခုရဲ့ *instance* တွေအားလုံးနဲ့ ဆိုင်တဲ့ ဘုံသုံး attribute တွေ ကို ဆိုလိုတာပါ။ *Class attribute* တွေကို ကလပ်စာတစ်ခုရဲ့ top level မှာ ကြော်ရပါမယ် (မက သဒေတွေရဲ့ အပြင်ဘက်၊ ကလပ်စာတစ်ခုရဲ့ တိုက်ရှိရှိရမှာ၊ မက်သဒေတဲ့မှာ မဖြစ်ရဘူး)။ ဘုံသုံးဆိုတဲ့ အတိုင်း လက်ရှိအော်ဂျက်နဲ့ ဆိုင်တာမရှုတ်တော့ဘူး။ ဒါကြောင့်မှုလို attribute ကြော်တဲ့အခါ `self.attribute_name` ပုံစံ မကြော်ရဘူး။

```

# File: class_attr_eg.py
class SomeClass:
    # class attributes
    MAX_ID = 9999
    cur_id = 0

    def __init__(self):
        self._id = SomeClass.cur_id + 1
        SomeClass.cur_id += 1

    @property
    def id(self):
        return self._id

# create 3 instance for testing
smc1 = SomeClass()
smc2 = SomeClass()
smc3 = SomeClass()

# အားလုံး 3 ထွက်တယ်
print(SomeClass.cur_id)          # ကလပ်စံမည်နဲ့ သုံးတာ
print(smc1.cur_id)              # ကလပ်စံ instance သုံးတာ
print(smc2.cur_id)
print(smc3.cur_id)

print(smc3.MAX_ID)
print(SomeClass.MAX_ID)

```

MAX_ID နဲ့ cur_id ကဲ class attribute တွေပါ။ အသုံးပြုတဲ့အခါ *ClassName.attribute_name* ပုံစံနဲ့ သုံးရမှာပါ။ ဒါပေမဲ့ ကလပ်စံ instance ကနေ သုံးလိုလည်း ရပါတယ်။ Instance တွေကို ကိုယ်စားပြုတဲ့ smc1, smc2 စတာတွေနဲ့ အသုံးပြုလိုရတာ တွေရပါမယ်။

`__init__` ထဲမှာ လက်ရှိအော်ဘုက်ရဲ့ `_id` ကို `cur_id` အပေါင်း တစ် ထည့်ထားတာ တွေရပါမယ်။ ပြီးတော့ `cur_id` တန်ဖိုးကိုလည်း တစ်တိုးလိုက်တယ်။ ပထမဆုံး SomeClass instance ရဲ့ `id` က တစ် ဖြစ်ပါမယ်။ အော်ဘုက်တစ်ခု ဆောက်လိုက်တိုင်း `cur_id` တန်ဖိုးက တစ်တိုးသွားမှာပါ။ အခုံပေမှာ အော်ဘုက် သုံးခုယူထားတဲ့အတွက် သုံး ဖြစ်သွားပါမယ်။

Attribute တစ်ခုတည်းကိုပဲ အော်ဘုက်အားလုံးက မျှသုံးထားတာ ဖြစ်တဲ့အတွက် `print` ထဲတွေ့ကြည့်တဲ့အခါ ဘယ်အော်ဘုက်နဲ့ပဲဖြစ်ဖြစ် 3 ပဲထွက်တာ တွေရမှာပါ။ ဒါဟာ အော်ဘုက်တစ်ခုစီက သီးသန်ပိုင်ဆိုင်တဲ့ `attribute` နဲ့ အခိုက်ကွားချက်ပါပဲ။ ဒီအချက်ကို ကွဲကွဲပြားပြား နားလည်ဖို့ အရေးကြီးပါတယ်။ `id` ကို ထဲတွေ့ကြည့်ပါ

```

print(smc1.id)      # 1 ထွက်မှာပါ
print(smc2.id)      # 2
print(smc3.id)      # 3

```

Class attribute တွေကို အော်ဂျက် (ပိုတိကျအောင်ကြေရင် အော်ဂျက်ကို ကိုယ်စားပြုတဲ့ မေရ့ရောဘဲလ်) နဲ့ ရော ကလပ်စံနံပါတ်နဲ့ပါ အသုံးပြုလို ရပေမဲ့ instance attribute တွေကိုတော့ ကလပ်စံနံပါတ်နဲ့ အသုံးပြုလို မရနိုင်ပါဘူး။ SomeClass ဥပမာမှာ အခုလို စမ်းကြည့်ရင်

```
print(SomeClass.id)  
print(SomeClass._id)
```

`var_name.attribute_name` ပုံစံနဲ့ class attribute အသုံးပြုတဲ့အခါ လက်ရှိအော်ဘက်ရဲ့ attribute လား class attribute လား မကဲ့ဟြားတော့ဘူး။ ဥပမာ

```
>>> dt1 = date(1990, 11, 30)
>>> print(dt1.min)
0001-01-01
```

ဒါကို ဖတ်တဲ့သူအနေနဲ့ `min` က class attribute လားဆိုတာ ရှုတ်တရက် မသိနိုင်ပါဘူး။ `date` ရဲ့ documentation မှာ ကြည့်မှ `min` ဟာ class attribute လို့ ဖော်ပြထားတာ တွေ့ရမှုပါ။ ကလပ်စံ နဲ့မည့်နဲ့ `date.min` ဆိုရင်တော့ class attribute ပဲ ဖြစ်ရပါမယ်။

Class attribute `self`

ကလပ်စံ ကော်ဒီထဲမှာ ငါးကလပ်စံ ကိုယ်တိုင်ရဲ့ class attribute တွေကို self.attribute_name နဲ့ ရည်ညွှန်း အသုံးပြန်ပါတယ်။ ဒါပေမဲ့ attribute တန်ဖိုးကို သုံးလိုပဲ ရုံးဖြစ်ပြီး အဆိုင်းမန်လုပ် လို့ မရပါဘူး။ တကယ်လို့ အဆိုင်းမန် လုပ်ရင် လက်ရှိအော်ဘက်မှာ (class attribute နဲ့ နံမည် တဲ့) ဖေရာ်ရော်လဲ အသစ်တစ်ခု ရှိသွားမှာဖြစ်တယ်။

```
class SomeClass:
    # class attributes
    MAX_ID = 9999
    cur_id = 0

    def __init__(self):
        # class attribute
        self._id = self.cur_id + 1
        # လက်ရှိအော်ဂျက်မှ cur_id ကြော်သလို ဖြစ်သွားမှာ၊ class attribute
        # ကို အဆိုင်းမနဲ့ လုပ်တော်မဟုတ်ဘူး
        self.cur_id = self._id
```

Class Methods

Class attribute နဲ့ ဆက်စပ်နေတဲ့ class method တွေကတော့ မက်သဒ်ကို ခေါ်ဖို့အတွက် အော့သ် ဂျက်ရှိစရာ မလိုဘူး။ တစ်နည်းအားဖြင့် class method တွေက ကလပ်စိတစ်ခုရဲ့ အခြေအနေအောက် မှာ လုပ်ဆောင်တာ။ အဲဒီ class instance ရဲ့ အခြေအနေအောက်မှာ လုပ်ဆောင်တာ မဟုတ်ဘူး။ မက် သဒ်ခေါ်တဲ့ အခါ *ClassName.method_name()* ပုံစံနဲ့ ခေါ်လို့ရတယ်။

```

class Joker:
    # class attributes
    CLASS_NICK_NAME = 'NaiveIdiot'
    joker_ids = []
    last_id = 0

    # class method
    @classmethod
    def get_info(cls):
        return {'CLS_NICK_NAME': cls.CLASS_NICK_NAME,
                'NXT_JOKER_ID': cls.last_id + 1,
                'CUR_JOKERS_IDS': cls.joker_ids}

    def __init__(self, name):
        self._id = Joker.last_id + 1
        Joker.joker_ids.append(self._id)
        Joker.last_id = self._id
        self._name = name

jk1 = Joker('Deadly Bee')
jk2 = Joker('Happy Hi')

print(Joker.get_info())

```

Output:

```
{'CLS_NICK_NAME': 'NaiveIdiot', 'NXT_JOKER_ID': 3, 'CUR_JOKERS_IDS': [1, 2]}
```

get_info က ကလပ် မက်သဒ် ဖြစ်ပါတယ်။ **@classmethod** decorator က မက်သဒ် တစ်ခုကို 'ကလပ် မက်သဒ် ဖြစ်ပါတယ်လို့' ကြော်တဲ့ အဓိပါယ်ပါ။ ပုံမှန် မက်သဒ်ဆိုရင် ပထမဆုံး ပါရေမီတာ က လက်ရှိအော်ဘက်ကို ရည်ညွှန်းတဲ့ **self** ဖြစ်တယ်။ ကလပ် မက်သဒ် ဆိုရင်တော့ အဲဒီအား ပထမဆုံး ပါရေမီတာက 'လက်ရှိကလပ်' ကို ရည်ညွှန်းတဲ့ **cls** ဖြစ်ရမှာပါ။ **cls** ပါရေမီတာနဲ့ class attribute တွေကို အသုံးပြုလို ရတယ်။ Joker ကလပ် get_info မှာ class attribute တွေထံး ထားတာကို တွေ့ရမှာပါ။ Class method ခေါ်တဲ့အခါ **cls** နေရမှာ လက်ရှိအသုံးပြုနေတဲ့ ကလပ် ဝင်လာမှာပါ။ Python က အလိုအလျောက် ထည့်ပေးတာဖြစ်တယ်။

၉.၆ Static Methods

Static method တွေကတော့ ကလပ်ထဲမှာ သတ်မှတ်ထားပေမဲ့ လက်ရှိအော်ဘက် **self** ကိုရော လက်ရှိကလပ် **cls** ကိုပါ အသုံးပြုစိုး မလိုအပ်တဲ့ မက်သဒ်တွေပါ။ **@staticmethod** decorator နဲ့ ကြော်ရတယ်။ ရှိခိုး top level ဖန်ရှင်လိုပဲ **cls** (သို့) **self** ပါရေမီတာ မလိုဘူး။ ငွေလွှာ ကိစ္စအတွက် transfer မက်သဒ်ကို static method အနေနဲ့ ဥပမာ ပြထားတာကြည့်ပါ။

```

class Account:
    def __init__(self, holder, acc_number, balance, branch):
        # ...

```

Built-in Decorators

Python မှာ `@property`, `@staticmethod`, `@classmethod` စတဲ့ built-in ပါပြီးသား decorator တွေပါရှိတယ်။ Decorator အမျိုးအစားအလိုက် လိုအပ်တဲ့ အသွင်ပြောင်းလဲမှု (transformation) တွေ Python က လုပ်ပေးသွားမှာ ဖြစ်တယ်။ ဥပမာ `@property` decorator က မက သုတေသနတဲ့ attribute လို့ သုံးလို့ရအောင် လုပ်ပေးတယ်။ `@staticmethod` နဲ့ `@classmethod` ကလည်း မက်သုတေသနခုကို class method နဲ့ static method အနေနဲ့ အသုံးပြုလို့ရအောင် လိုအပ်တဲ့ ပြောင်းလဲမှုတွေ လုပ်ပေးမှာ ဖြစ်တယ်။

Decorator တွေရဲ့ နောက်ကွယ်မှာ အလုပ်လုပ်ပေးတာက ဖန်ရှင်တွေပါပဲ။ ကိုယ်ပိုင် decorator သတ်မှတ်လို့လည်း ရပါတယ်။ အတွေ့အကြား ပရိုဂရမ်မာ ဖြစ်လာတော့မှ လေ့လာကြလေ့ရှိပြီး ဒီ စာအပ်မှာတော့ ငှါးတိုက် သီးသန့်ဖော်ပြမှာ မဟုတ်ပါဘူး။

```
        self._balance = balance
        ...
# ...
@staticmethod
def transfer(from_acc, to_acc, amt):
    if amt > from_acc._balance:
        raise ValueError('Not enough balance!')
    from_acc._balance -= amt
    to_acc._balance += amt
```

ဒီ မက်သုတေသနတဲ့ ခေါ်တဲ့ အခါ class method လိုပဲ ကလပ်နံမည်နဲ့ ခေါ်လို့ရသလို instance နဲ့ ခေါ်ချင်လည်း ရှုတယ်။

```
Account.transfer(acc1, acc2, Decimal('50_000.00'))
acc1.transfer(acc1, acc2, Decimal('50_000.00'))
```

ကလပ်နံမည်နဲ့ ခေါ်တာ ပိုရှင်းပါတယ်။

မှတ်ချက်။ `transfer` မက်သုတေသနကို instance method အနေနဲ့ပဲ ရေးထားတာကို လာမဲ့ အသုံးချေပော်မှာ တွေ့ရပါမယ်။ Instance method ဆိုရင် ခေါ်တဲ့ အခါ

```
acc1.transfer(acc2, Decimal('50_000.00'))
```

ဖြစ်ပါမယ်။

Static method ဟာ ဖိုင်တစ်ခုရဲ့ top level မှာ ကြော်တဲ့ ပုံမှန်ဖန်ရှင်တွေလိုပဲ `cls` (သီး) `self` မပါတဲ့ အတွက် ကလပ်နံတဲ့ အပြင်ကို ထုတ်လိုက်ရင် ရီးရိုးဖန်ရှင် ဖြစ်သွားမှာပါ။

```
# @staticmethod decorator လည်း မလိုတော့ဘူး
def transfer(from_acc, to_acc, amt):
    if amt > from_acc._balance:
        raise ValueError('Not enough balance!')
    from_acc._balance -= amt
    to_acc._balance += amt
```

```

class Account:
    def __init__(self, holder, acc_number, balance, branch):
        # ...
        self._balance = balance
        # ...
    # transfer ကို ကလပ်စဲက ထုတ်လိုက်တယ်

```

ဒါပေမဲ့ ငွေလွှာတယ်ဆိတာ ဘက်အကောင့်နဲ့ သက်ဆိုင်တဲ့ ကိစ္စဖြစ်တဲ့ အတွက် Account ကလပ်စဲမှာ static method အနေနဲ့ ရှိတာက ပိုပြီး သဘာဝကျပ်တယ်။

မက်သဒ်နှင့် attribute အမျိုးအစားများ အကြား အပြန်အလှန် အသုံးပြုနိုင်မှု

Instance method ၏ instance/class attribute နဲ့ instance/class/static method တွေကို အသုံးပြုလိုရတယ်။ Class method နဲ့ static method ကတေသာ့ instance attribute/method တွေကို သုံးလိုရမှာ မဟုတ်ပါဘူး။ အောက်ပါ ပေါ်မှုနှင့် အမျိုးအစားအလိုက် အသုံးပြုလို ရ/မရ ခွဲခြားပြထားတာကို ကြည့်ပါ။

Method Type	Class Attribute	Instance Attribute
Static	Yes (class name)	No
Class	Yes (<code>cls</code> /class name)	No
Instance	Yes (<code>self</code> /class name)	Yes (<code>self</code>)

တေဘာ် ၉.၁ မက်သဒ်အမျိုးအစားနှင့် attribute အမျိုးအစား အကြား အသုံးပြုနိုင်မှု

Method Type	Static Method	Class Method	Instance Method
Static	Yes (class name)	Yes (class name)	No
Class	Yes (<code>cls</code> /class name)	Yes (<code>cls</code> /class name)	No
Instance	Yes (<code>self</code> /class name)	Yes (<code>self</code> /class name)	Yes (<code>self</code>)

တေဘာ် ၉.၂ မက်သဒ်အမျိုးအစားများ အကြား အသုံးပြုနိုင်မှု

ပေါ်မှုနှင့် အလုံးမှာ class name/`cls`/`self` ဘာနဲ့ အသုံးပြုလိုရလဲ ကိုလည်း ပြထားတယ်။ ဥပမာ instance method ဆိုရင် class attribute ကို class name (သို့) `self` နဲ့ သုံးလိုရမှာပါ (ပထမ ပေါ်မှု အောက်ခုံး row ပထမ column)။ Class method ၏ static method ကို `cls` (သို့) class name နဲ့ သုံးလိုရပါမယ် (ဒုတိယ ပေါ်မှု row ပထမ column)။ အခုလို ကုဒ်ရေးပြီး စမ်းသပ်စစ်ဆေး ထားတာကိုလည်း လေ့လာကြည့်ပါ။

```

# File: accessibility_test.py
# can be used inside the class
top_level_var = 'top level var'

```

```

# can be called inside the class
def top_level_fun():

```

```
pass

class AccessibilityTest:

    class_attr = 'class attr'

    def __init__(self):
        self.instance_attr = 'instance attr'
        self.instance_attr2 = 'instance attr2'

    @staticmethod
    def static_method():
        # print(self.instance_attr)  # cannot use
        print(AccessibilityTest.class_attr)
        # can only call static and class method types
        AccessibilityTest.static_method2()
        AccessibilityTest.class_method2()
        # self.instance_method()      # cannot use

    @staticmethod
    def static_method2():
        pass

    @classmethod
    def class_method(cls):
        # print(self.instance_attr)  # cannot use
        print(cls.class_attr)
        print(AccessibilityTest.class_attr)
        # can only call static and class method types
        AccessibilityTest.static_method()
        cls.class_method2()
        AccessibilityTest.class_method2()
        # self.instance_method()      # cannot use

    @classmethod
    def class_method2(cls):
        pass

    def instance_method(self):
        print(self.instance_attr)
        print(AccessibilityTest.class_attr)
        # can call all type of methods
        AccessibilityTest.static_method()
        AccessibilityTest.class_method()
        self.instance_method2()
```

```

def instance_method2(self):
    print(self.instance_attr2)

# just test
avar1 = AccessibilityTest()
AccessibilityTest.static_method()
print()
AccessibilityTest.class_method()
print()
avar1.instance_method()

```

၉.၇ Object-Oriented Programming

ကလပ်နဲ့ အော်ဂျက်တွေ အကြောင်း အထိုက်အလျောက် လေ့လာပြီးနောက် ငါးတိုကို ပရိုကရမ် တည် ဆောက်ရုံး၊ အသုံးချကတ်အောင် ဆက်လက် လေ့လာကြရပါမယ်။ လက်တွေရေး လေ့ကျင့်စိုး လိုပါ တယ်။ အော်ဂျက်သောတရား အခြေပြု ပရိုကရမ်ရေးသား တည်ဆောက်တာကို *Object-oriented Programming* (OOP) လိုခေါ်ပါတယ်။ ဒီနည်းနဲ့ တည်ဆောက်ထားတဲ့ ပရိုကရမ်တွေကို object-oriented program/software လို့ ဆိုနိုင်ပါတယ်။ စလေ့လာသူတွေ အတွက် အထောက်အကူဖြစ်စေ မဲ့ ရိုးရှင်းတဲ့ နှမူနာ object-oriented program လေးတစ်ခုကို လေ့လာကြည့်ပါမယ်။

A Simple Banking Application

S Bank ဆိုတဲ့ ဘဏ်အတွက် ပရိုကရမ်တစ်ခု ရေးပေးရမယ်လို့ စိတ်ကူးကြည့်ပါ။ ဘဏ်အကောင့် အသစ် ဖွင့်တဲ့ အခါ အောက်ပါ အချက်အလက်တွေ လိုအပ်တယ်။ S Bank က လောလောဆယ် ရှုံးကန်၊ မန္တလေး နဲ့ နေပြည်တော် ဘဏ်ခဲ့ သုံးခဲ့ ရှိတယ်။

Open Account

Name: Kathy

NRIC: 12/ThaGaKa (N) 283772

Gender (F/M): F

Date of Birth: 12/12/1990

Address Line 1: 45A, Marlar St., 16 Ward

Address Line 2: South Okkalapa

City: Yangon

State/Division: Yanon

Initial deposit: 350000.00

Branch: Yangon

Please confirm (Y/N): Y

Account No: 1000000000012

Successfully created

အကောင့်နံပါတ်က ကဏ်း (၁၂) လုံး ဖြစ်ရမယ်။ 100000000001 က ပထမဆုံးဖွင့်တဲ့ အကောင့်နံပါတ် ဖြစ်ပြီး အကောင့်သစ် တစ်ခုတိုင်းအတွက် နံပါတ်စဉ် တိုး သွားရမှာပါ။

အကောင့်ထဲကို ငွေသွင်းမယ်ဆိုရင် အကောင့်နံပါတ်နဲ့ သွင်းမဲ့ ငွေပမာဏ ထည့်ပေးရပါမယ်။ အကောင့်နံပါတ် ထည့်ပေးတာ မှားခဲ့ရင် အခြားအကောင့်တစ်ခုထဲကို ငွေသွင်းမိနိုင်တယ်။ အကောင့်ပိုင်ရှင်နံမည်ကို ပြပေးပြီး ကွန်ဖန်းလုပ်ရင် အမှားဖြစ်နိုင်ခြား ပိုပြီးနည်းသွားမှာပါ။ ငွေထုတ်ရင်လည်း ဒီလိပ် ကွန်ဖန်းလုပ်သင့်တယ်။

Deposit

Account No: 1000000000012

Amount: 250000

Please check and confirm the following details:

Holder: Kathy

Account No: 1000000000012

Amount to deposit: 250000

Please confirm (Y/N): Y

...

Account transaction successfully completed.

Withdrawal

Account No: 1000000000012

Amount: 50000

Please check and confirm the following details:

Holder: Kathy

Account No: 1000000000012

Amount to withdraw: 50000

Please confirm (Y/N): Y

...

Account transaction successfully completed.

ငွေလွှဲမယ်ဆိုရင် လွှဲမဲ့အကောင့် လက်ခံမဲ့အကောင့် နှစ်ခုလို့မယ်။ လွှဲမဲ့အကောင့်နဲ့ လက်ခံအကောင့်တစ်ခုတည်း ဖြစ်မနေသင့်ပါဘူး။

Transfer

From Account No: 1000000000012

To Account No: 1000000000015

Amount: 50000

Please check and confirm the following details:

From

Holder: Kathy

Account No: 1000000000012

To

Holder: Sandar

Account No: 1000000000015

Amount: 50000

Please confirm (Y/N): Y

...

Account transaction successfully completed.

ဒီပြုရှိရမ်းလေးက အပြင်မှာ တကယ်သံးလို့မရပေမဲ့ တတ်နိုင်သမှုံ ယူတွေ့ကျအောင်၊ အပြင်နဲ့ နီးစပ်အောင် စဉ်းစားပြီး ရေးသင့်ပါတယ်။ ဥပမာ အကောင့်နံပါတ်မှားရင် ဘယ်လိုပြပေးသင့်သလဲ၊ ထုတ်မဲ့ ငွေ

ပမာဏက လက်ကျွန်ုင်ထွက် ပိုများနေတဲ့အခါ ဘယ်လိုဖြစ်သင့်လဲ စတာတွေကို သေသေချာချာ စဉ်းစား ပေးသင့်တယ်။ အခြေခံအဆင့် ဥပမာမို့လို့ အလွန်အကျိုး ရှုပ်ထွေးလို့တော့လည်း မရသေးဘူးပေါ့။ ဘိုင်နာအဆင့်နဲ့ သင့်တော်လောက်မဲ့ ပုံစံမျိုးလေးနဲ့ တစ်ပိုင်းချင်း တည်ဆောက်သွားရအောင်။

ပရိုဂရမ်တစ်ခု စတည်ဆောက်ဖို့ ကြိုးစားတဲ့အခါ ဘယ်ကစရမယ်မှန်းမသိ အစရှာလို့မရ ဖြစ်နေတတ်ပါတယ်။ ဒီလိုအခြေအနေမျိုးမှ problem decomposition ကို အသုံးချုပ်မှုပါ။ ဖြေရှင်းမဲ့ ကိစ္စကို အပိုင်းတွေ ခဲ့ကြည့်ပြီး ဘယ်တစ်ခုကို အရင်ဆုံး တည်ဆောက်ရမလဲ ရွေးချယ်ရပါမယ်။ ဖော်ပြထားချက်တွေအရ ဘဏ်ပရိုဂရမ်က အကောင့်ဖွံ့ဖြိုးတာ၊ ငွေသွင်း၊ ငွေထုတ်နဲ့ ငွေလွှာ ကိစ္စတွေ ဆောင်ရွက်ပေးရမှာပါ။ အကောင့်ဖွံ့ဖြိုးတဲ့ အပိုင်းကို ပထမဆုံး တည်ဆောက်ပါမယ်။ ဘာလို့ ဒါနဲ့စတင်မယ်လို့ ဆုံးဖြတ်တာလဲ ခိုင်လုံတဲ့ အကြောင်းပြချက် သိပ်တော့ကြီးကြီးမားမားရှယ် မရှိပါဘူး။ (ငွေသွင်း/ထုတ် တွေ လုပ်နိုင်ဖို့က အကောင့်အရင် ဖွံ့ဖြိုးထားမှ လုပ်လို့ရမှာဖြစ်လို့ ဒီကနေ စသင့်တယ်လို့တော့ မြင်မိတယ်)။ ဒါပေမဲ့ ငွေသွင်းတဲ့အပိုင်းနဲ့ စမယ်ဆုံးရင်လည်း ရတာပါပဲ။

Object-oriented Programming နည်းနဲ့ ပရိုဂရမ်ရေးတဲ့အခါ ဖော်ပြထားချက်ထဲမှာ ပါဝင်တဲ့အကြောင်းအရာတွေကို လေ့လာသုံးသပ်ပြီး ဘယ်အရာတွေကို အော့ဘ်ဂျက်တွေအနေနဲ့ ဖော်ပြရမလဲ ဆုံးဖြတ်ရတယ်။ အကောင့်၊ အကောင့် ပိုင်ရှင်၊ လိပ်စာ၊ ဘဏ် နဲ့ ဘဏ်ခဲ့တို့ကို အော့ဘ်ဂျက်အနေနဲ့ ရှိမြင်နိုင်တယ်။ ဒါကြောင့် Account, Holder, Address, Bank နဲ့ Branch ကလပ်တွေ သတ်မှတ်ပါမယ်။ Account ကို အရင်ကြည့်ရအောင် ...

Account Class

Account ကလပ်စာ ရှေ့ပိုင်းက ဥပမာတွေမှာလည်း တွေ့ခဲ့ကြပြီးပါပြီ။ ဘဏ်ပရိုဂရမ်အတွက် လိုအပ်တာ တချို့ကို ထပ်ဖြည့်ထားပါတယ်။

```
# File: bank_app.py
class Account:
    _next_acc_no = 100_000_000_001

    def __init__(self, holder, balance, branch):
        self._holder = holder
        self._acc_number = Account._next_acc_no
        Account._next_acc_no += 1
        self._balance = balance
        self._branch = branch

    def deposit(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount!')
        self._balance += amt

    def withdraw(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount!')
        if amt > self._balance:
            raise ValueError('Not enough balance!')
        self._balance -= amt
```

```

def transfer(self, to_acc, amt):
    if self.acc_number == to_acc._acc_number:
        raise ValueError('From-account and to-account should not be '
                         'the same!')
    if amt <= Decimal(0.00):
        raise ValueError('Invalid amount!')
    if amt > self._balance:
        raise ValueError('Not enough balance!')
    self._balance -= amt
    to_acc._balance += amt

@property
def holder(self):
    return self._holder

@property
def acc_number(self):
    return self._acc_number

@property
def balance(self):
    return self._balance

@property
def branch(self):
    return self._branch

```

ရှေ့မှာဖော်ပြခဲ့တဲ့ လိုအပ်ချက်အရ အကောင့်နံပါတ်မှာ ကတန်း (၁၂) လုံး ရှိရမယ်။ ပထမဆုံး ဖွင့်တဲ့အကောင့် နံပါတ်က 1000000000001 ဖြစ်ရမယ်။ နောက်ထပ် အကောင့်သစ်တစ်ခု ဖွင့်တိုင်း နံပါတ်စဉ် တစ်ခုချင်း တိုးသွားရမှာပါ။ ဒီလိုအပ်ချက်အတွက် class attribute ကို အသုံးပြုနိုင်ပါတယ်

_next_acc_no = 100_000_000_001

Initializer ထဲမှာ ဒီ attribute ကို ဘယ်လို သုံးထားလဲ လေ့လာကြည့်ပါ။ လက်ရှိအော့သ်ဂျက်ရဲ့ အကောင့်နံပါတ်အဖြစ် သုံးပြီးတာနဲ့ နောက်ဖန်တီးမဲ့ အော့သ်ဂျက်အတွက် အသင့်ဖြစ်အောင် တစ်တိုးပေးထားတော့ရှိရမှာပါ။ မက်သမ်တွေနဲ့ instance attribute တွေကတော့ ရှေ့က ဥပမာတွေမှာ တွေ့ပြီး ဖြစ်တဲ့အတွက် သိပ်တွေတွေထူးထား ရှင်းပြစိုး မလိုလောက်တော့ဘူး။ transfer မက်သမ်ကတော့ နည်းနည်းထူးတာလေးတွေ ပါနေတယ်

```

def transfer(self, to_acc, amt):
    # ... (တစ်ချင်းတွေ မပြထား)
    self._balance -= amt
    to_acc._balance += amt # မသုံးစေချင်တာကို ယူသုံးထားတာ ဖြစ်မနေဘူးလား

```

self._balance က ပြသုနာမျိုးပေမဲ့ to_acc ရဲ့ _balance ကို တိုက်ရှိက် ယူသုံးထားတာ access control စည်းကမ်းဖောက်သလို ဖြစ်မနေဘူးလား မေးစရာရှိပါတယ်။ အခု ဒီလိုမျိုး သုံးထားတာက Account ကလပ်စိမှာပါ။ to_acc ကလည်း အခြား Account အော့သ်ဂျက် တစ်ခုဖြစ်တယ်။ လက်ရှိ

ကလပ်စ်သတ်မှတ်သူဟာ အဲဒီကလပ်စ်အမျိုးအစား အော့သ်ဂျက်တွေရဲ့ attribute တွေကို တိုက်ရှိက သုံးတာက ပြဿနာမရှိဘူးလို့ ယူဆရပါမယ်။ အကြောင်းအရင်းက ဒီလိုပါ။ ပိုင်ရှင်ဟာ သူကလပ်စ် attribute (ပုံတိကျအောင် ပြောရင် သူကလပ်စ်ကနဲ့ ဖန်တီးယူလိုက်တဲ့ အော့သ်ဂျက်ရဲ့ attribute) တွေကို အခြားသူတွေကိုပဲ တိုက်ရှိက အသုံးမပြုစေချင်တာ။ သူကိုယ်တိုင်အနေနဲ့တွေ့ သုံးလို့ရသင့်ပါတယ်။ ဘာဖြစ်လိုလဲဆိုတော့ သူကိုယ်တိုင်က ပိုင်ရှင် ဖြစ်တဲ့အတော် ကလပ်စ်ကို ပြင်ဆင်ပြောင်းလဲတဲ့အခါ ဒီလို တိုက်ရှိက်ယူသုံးထားတဲ့အပေါ် ပြဿနာ ရှိ/မရှိ သူ့အနေနဲ့ သုံးသပ်နိုင်ပါတယ် (သူရေးထားတဲ့ ကလပ်စ်ပဲလေ၊ ဘာလုပ်ရင် ဘာဖြစ်မယ် သူသိရမှာပေါ့)။ ဒီအတွက် တိုက်ရှိက်ယူသုံးထားတဲ့ နေရာတွေကိုလည်း လိုအပ်သလို အလိုက်သင့် ပြင်ဆင်ပြောင်းလဲ ပေါ်နိုင်ပါတယ်။ ဒီလောက်ဆိုရင် ကလပ်စ်ပိုင်ရှင် အနေနဲ့ အဲဒီကလပ်စ် အော့သ်ဂျက်တွေရဲ့ attribute တွေကို တိုက်ရှိက အသုံးပြုလို့ ရသင့်တယ်ဆိုတဲ့ အကြောင်းပြုချက်ဟာ ကျိုးကြောင်းဆီလျှော့ ရှိတယ်လို့ စာဖတ်သူ လက်ခံနိုင်မယ် မျှော်လင့်ပါတယ်။

ကလပ်စ်တစ်ခုမှာ ပါဝင်သင့်တဲ့ မက်သဒ်နဲ့ attribute တွေကို တစ်ခါတည်းနဲ့ အားလုံးပြီးပြည့်စုံအောင် ကြိုးတင်စဉ်းထားထားဖို့ မလိုအပ်ပါဘူး။ စဉ်းထားလို့ရတာတွေနဲ့ပဲ စသတ်မှတ်ထားနိုင်တယ်။ ပရိုဂရမ်တည်ဆောက်ရင်းနဲ့ လိုအပ်တာတွေ ထပ်ပြီး တွေ့လာ၊ မြင်လာတဲ့ အခါကျမှ ပြင်ဆင်ဖြည့်စွက်လို့ရပါတယ်။ ကလပ်စ်တစ်ခုမှာပဲ လည်နေဖြီး ရှုံးမဆက်နိုင်တာဟာ စလေ့လာသူ ဖြစ်လေ့ရှုံးတဲ့ အမှားတစ်ခုပါ။

Address and Holder Classes

Address နဲ့ Holder ကို အခုလို သတ်မှတ်ထားပါတယ်။ Account နဲ့ နှီးယူဥ်ရင် အတော်လေး ရှိုးရှင်းတာ တွေရမှာပါ။

```
class Address:
    def __init__(self, line_1, line_2, city, division):
        self._address_line_1 = line_1
        self._address_line_2 = line_2
        self._city = city
        self._division = division

    @property
    def address_line_1(self):
        return self._address_line_1

    @property
    def address_line_2(self):
        return self._address_line_2

    @property
    def city(self):
        return self._city

    @property
    def division(self):
        return self._division

class Holder:
    def __init__(self, name, dob, nric, gender, address):
```

```

if name.strip() == '':
    raise ValueError('Name cannot be empty!')
if age_today(dob) < 18:
    raise ValueError('Too young to open a bank account!')
if nric.strip() == '':
    raise ValueError('Name cannot be empty!')
if not (gender.upper() == 'F'
        or gender.upper() == 'M'):
    raise ValueError('Unknown gender!')
self._name = name
self._dob = dob
self._nric = nric
self._gender = gender.upper()
self._address = address

@property
def age(self):
    return age_today(self._dob)

@property
def name(self):
    return self._name

@name.setter
def name(self, name):
    if name.strip() == '':
        raise ValueError('Name cannot be empty!')
    self._name = name

@property
def dob(self):
    return self._dob

@dob.setter
def dob(self, value):
    self._dob = value

@property
def gender(self):
    return 'FEMALE' if self._gender == 'F' else 'MALE'

```

Holder ရဲ့ age property မှာ age_today ဖန်ရှင် သုံးထားတယ်။ Insurance premium ပုံးကြော်လည်း ခီဖန်ရှင်ကို သုံးထားတယ်။ ပရိုဂရမ်တွေမှာ ဒီလိုမျိုး မကြာခဏ ပြန်အသုံးပြုရတဲ့ ဖန်ရှင် တွေကို မော်ဒူး (သို့) လိုက်ဘရိတစ်ခုအနေနဲ့ ထုတ်ထားလိုရတယ်။ လိုတဲ့အခါ import လုပ်ပြီး သုံးရုံပဲ။ ပြန်ရေးဖို့ မလိုတော့ဘူး။ မော်ဒူးအကြောင်းကို နောက်အခန်းတွေမှာ ဆက်လက်လေ့လာကြမှုပါ။ လောလောဆယ်တော့ ဘဏ်ပရိုဂရမ်ဖို့ (bank_app.py) ရဲ့ top level မှာ ပြန်ရေးပါမယ်။ (Copy

and paste လုပ်လည်းရတယ်)။

```
# File: bank_app.py
# top level ဖန်ရှင်ပါ၊ ကုဒ်ဖိုင်ရဲ အောက်မှာ တိုက်ရှိက်ရှိပါတယ်၊ ကလပ်စံထဲမှာမဟုတ်
def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1
```

Branch and Bank Classes

Bxanch ကလပ်စ်ကတော့ ရှင်းပါတယ်။ နံမည်နဲ့ လိပ်စာပဲ ပါတယ်။ ပရိဂုရမဲ လိုအပ်ချက်အရ နောက်ပိုင်း ထပ်ဖြည့်ရတာတွေတော့ ရှိလာနိုင်ပါတယ်။ ဥပမာ ဘက်ခွဲ စဖွင့်တဲ့ရက်၊ ဘက်ခွဲမှာ ရရှိနိုင်တဲ့ ဝန်ဆောင်မှု တွေ စသည်ဖြင့်။ အခါတော့ ဒါတွေ မလိုသေးပါဘာ။ ဖော်ပြထားချက်ကို ပြန်ကြည့်ရင် ဘက်ခွဲလိပ်စာကို လည်း သုံးဖို့ မလိုသေးတာ တွေ့ရှုပါ။ အခုပုရှိရမဲ လိုအပ်ချက်အရဆိုရင် ဘက်ခွဲကို သီးခြား အော့သ ဂျက်အနေနဲ့ မယူဆလည်း ရတယ်။ ဘက်ခွဲနံမည်ကို စာသားအနေနဲ့ သိမ်းထားရင် ရပြီ။ ဒါပေမဲ့ ဘက်ခွဲ ဟာ အပြင်မှာတကယ်ရှိတဲ့ အရာဖြစ်တယ်။ နံမည်အပြင် ဘက်ခွဲပါတယ်။ အေး အချက်အလက် တွေကိုလည်း နောက်ပိုင်းမှာ လိုအပ်လာနိုင်တယ်လို့ မျှော်မှန်းရပါတယ်။ ဒါကြောင့် အော့သဂျက်ဖြစ်သင့် တယ်လို့ ယူဆပါမယ်။

```
class Branch:
    def __init__(self, name, address):
        self._name = name
        self._address = address

    @property
    def name(self):
        return self._name
```

မှတ်ချက်။ ။ ဒီနေရာမှာ ဘဏ်ခွဲသည် အော်ဂျက် ဖြစ်ကိုဖြစ်ရမယ်လို တစ်ထောက်ချမှုသာ လိုအပ် ချက်ပေါ်တည်ပြီး ဆုံးဖြတ်နိုင်တယ် ဆိတာကို အခိုက်ပြေချင်တာပါ။ မလိုအပ်ဘူး ယူဆရင် ဘဏ်ခွဲ မံမည်ကို စာသားအနေဖြင့် ထားပြီး နောင်လိုအပ်လာတော့မှ ကလပ်စ် သတ်မှတ်လည်း ရတယ်။

Bank ကလပ်စိုက် ကြည့်ရအောင်။ ဘဏ်မှာ အကောင့်တွေ ရှိပါမယ်။ အကောင့် ဖွင့်ထားတဲ့ ကာစတမ်းမာတော့ ရှိပါမယ်။ ဘဏ်ခဲ့သုံးခဲ့ရမယ်။ ဆက်စပ်မှုက has-a relationship ပါ။

```

Branch('Mandalay', Address('65B, 63 St.',
                            'Chanayetharsi',
                            'Mandalay',
                            'Mandalay')),
Branch('Naypyidaw', Address('332C, Main Rd.',
                            'Ottarathiri Township',
                            'Naypyidaw',
                            'Naypyidaw Union'))]

def open_acc(self, holder, init_bal, branch_name):
    branch = self.retrieve_branch(branch_name)
    account = Account(holder, init_bal, branch)
    self._accounts.append(account)
    self._acc_holders.append(holder)
    return account

def withdraw(self, acc_no, amt):
    acc = self.retrieve_account(acc_no)
    acc.withdraw(amt)

def deposit(self, acc_no, amt):
    acc = self.retrieve_account(acc_no)
    acc.deposit(amt)

def transfer(self, from_acc_no, to_acc_no, amt):
    from_acc = self.retrieve_account(from_acc_no)
    to_acc = self.retrieve_account(to_acc_no)
    from_acc.transfer(to_acc, amt)

def retrieve_account(self, acc_no):
    for acc in self._accounts:
        if acc.acc_number == acc_no:
            return acc
    raise ValueError(f'Account with {acc_no} not found!')

def retrieve_branch(self, name):
    for branch in self._branches:
        if branch.name == name:
            return branch
    raise ValueError(f'{name} branch not found!')

```

ဘဏ်ခွဲသုံးခုကို ပြိုပြီး ထည့်ထားတယ်။ လက်ရှိမှာ သုံးခုပဲရှိပေး ဘဏ်ခွဲ အသစ်တွေ ထပ်ဖွဲ့ခိုင်တယ်။ ပရိုဂရမ် နောက်ပိုင်း ဗားရှင်းတွေမှာ ဘဏ်ခွဲအသစ် ထပ်ထည်းပိုင်ဖို့ လိုအပ်ချက်ရှိလာမယ်လို့ မျှော်မျှန်း ရပါတယ်။ အားလုံး သိပြီးဖြစ်တဲ့အတိုင်း ပရိုဂရမ်တွေက သူတိုကို အသုံးပြုနေတဲ့ ကာလက်လျှောက် အမြဲပိုင်ဆင်ဖြည့်စွာက်တာ၊ ပြုပိုင်ထိန်းသိမ်းတာတွေ လုပ်နေရမှာပါ။ ဗားရှင်းအသစ်တွေ မကြေခာကျ ပျော်ပေး လုပ်ကြရပါတယ်။ တကယ့် အသုံးချု ဆော်ဖို့တွေ တည်ဆောက်တဲ့အခါ ပြုပိုင်ထိန်းသိမ်းရ လွှာယ်ကူ

အောင် ဖီချာအသစ်တွေ ထပ်ထည့်စို့ မက်ခဲအောင် ဒီဇိုင်းပြုလုပ်ထားရပါတယ်။ အခြေခံ ပရီဂါရမ်းမင်း တတ်ကျမ်းပြီးတဲ့နောက်မှာ ဒီလို ဒီဇိုင်းပိုင်းဆိုင်ရာတွေ နားလည်းအသုံးချုတ်လာအောင် ဆက်လက် လေ့လာဖို့ လိုအပ်ပါလိမ့်မယ်။

Account မှာ ပါတဲ့ မက်သဒ်တွေကို Bank မှာလည်း ပြန်တွေ့ရပါတယ်။ transfer ကို ကြည့်ရင် အခုလိုရေးထားတာ တွေ့ရတယ်

```
def transfer(self, from_acc_no, to_acc_no, amt):
    from_acc = self.retrieve_account(from_acc_no)
    to_acc = self.retrieve_account(to_acc_no)
    from_acc.transfer(to_acc, amt)
```

အခု Bank ကလည်း transfer က အကောင့် နံပါတ်နှစ်ခုနဲ့ ငွေပောက်ကို လက်ခံထားတယ်။ အဲဒီနံပါတ်နဲ့ အကောင့်နှစ်ခုကို ဘဏ်ရဲ့ အကောင့်တွေအားလုံးထဲကနေ ဆဲထဲတ်တယ် (retrieve_account မက်သဒ်ကို ကြည့်ပါ)။ ရလာတဲ့ အော့ဘ်ဂျက်ရဲ့ transfer ကို တစ်ဆင့်ပြန်ခေါ်ထားတယ်။ ဘာလို ဒီလို အဆင့်ဆင့် လုပ်နေရလဲ မေးစာရှိလာပါတယ်။ တစ်ဆင့်တည်းနဲ့ရော ရအောင်လုပ်လို့ မရနိုင်ဘူးလား။ ဘာလို အခုလို နှစ်ဆင့်လုပ်သင့်လဲ design principle တရာ့နဲ့ ရှင်းပြလို့ ရနိုင်ပေမဲ့ အခြေခံအဆင့်အတွက် အချိန်မကျသေးဘူးလို့ ယူဆရပါတယ်။ လက်ရှိအဆင့်မှာ အခုပြထားတဲ့ နည်းလမ်းအပြင် မိမိ စဉ်းစားလို့ရတဲ့ အခြားနည်းလမ်းနဲ့ ပြောင်းလဲ စမ်းသပ်ရေးကြည့်ပါ။ ဘယ်နည်းလမ်းက ရေးရတာ (သို့) နားလည်းရတာ ပိုလွှာယ်လဲ ပိုရှိုးရှင်းလဲ စဉ်းစားချင့်ချိန်ကြည့်ခြင်းအားဖြင့် အကျိုးရှိနိုင်တယ်။ ပိုပြီးတော့ ထိုးထွေးသိမြင်လှဖို့လည်း အထောက်အကူ ဖြစ်နိုင်ပါတယ်။

မှတ်ချက်။ ။ Bank ဟာ ငှါးရဲ့ အကောင့် အော့ဘ်ဂျက်တွေ အားလုံးကို စုစည်းသိမ်းထားတယ်။ ဒါကြောင့် အကောင့်နံပါတ်နဲ့ အကောင့်တွေကို ရှာတဲ့အလုပ်ကို Bank ကပဲ လုပ်ဆောင်ပေးတာ သဘာဝကျတယ်။ ငွေလွှာတဲ့အခါ အကောင့်နှစ်ခုရဲ့ လက်ကျွန်းငွေကို ပြောင်းလဲစေမှာပါ။ လက်ကျွန်းငွေ balance က Account နဲ့ဆိုင်တဲ့ အချက်အလက် ဖြစ်တာကြောင့် transfer ဟာ Account မှာပဲရှိသင့်တယ်လို့ ယူဆနိုင်တယ် (တစ်နည်းအားဖြင့် ဒေတာကို ပိုင်ဆိုင်ထားတဲ့ ကလပ်စ်နဲ့ ငှါးဒေတာအပေါ် မြှို့လုပ်ဆောင်ရမဲ့ မက်သဒ်ကို တပေါ်းစာစည်းတည်း ရှိစေချင်တာ)။ ငွေလွှာကိုစွာဟာ ဘဏ်နဲ့ ငှါးပိုင်ဆိုင်ထားတဲ့ အကောင့်နှစ်ခု ပူးပေါင်းလုပ်ဆောင်ရတဲ့ အလုပ်လို့ ရှိမြင်ကြည့်နိုင်ပါတယ်။

BankApp Class

Input/Output နဲ့ ဆိုင်တာတွေကို လုပ်ငန်းပိုင်းဆိုင်ရာတွေနဲ့ ခွဲေားရေးသင့်တယ်လို့ အတွေ့အကြံရှိပရီဂါရမ်းတည်ဆောက်သူတွေ အားလုံးက လက်ခံထားကြပါတယ်။ လုပ်ငန်းပိုင်းဆိုင်ရာကို domain လို ခေါ်ပါတယ်။ ဘဏ်လုပ်ငန်းမှာဆိုရင် အကောင့်ဖွံ့ဖြိုးခြင်း၊ ငွေ သွင်း/လွှာ/ထိတ် လုပ်ခြင်း၊ အတိုးပေးခြင်း၊ ဈေးထွေ့ထွေ့ပေးခြင်း စတာတွေဟာ banking domain နဲ့ သက်ဆိုင်တဲ့ အရာတွေပါ။ ရှေ့မှာဖော်ပြဲတဲ့ ကလပ်စ်တွေဟာ banking domain နဲ့သက်ဆိုင်ပါတယ်။

Input/Output ဆိုတာ ပရီဂါရမ်းပြုပြင်ပေလောက ဆက်သွယ်လုပ်ဆောင်တဲ့ ကိစ္စတွေလို့ အကြမ်းဖျုပ်း ပြောလို့ရပါတယ်။ ကိုဘုံးကနေ ရှိရှိထည့်လိုက်တာကို ပရီဂါရမ်းက ဖတ်တာဟာ input ဖြစ်တယ်။ မော်နှီးတာမှာ စာသား/ရုပ်ပုံ ပြပေးတာဟာ output ဥပမာ တစ်ခုဖြစ်တယ်။ Input/output အပိုင်းမှာ ပါဝင်တဲ့ အခြား ဥပမာတွေလည်း အများကြီးရှိပါတယ်။

BankApp က input/output ကို အစိုက် တာဝန်ယူလုပ်ဆောင်ပေးတဲ့ ကလပ်စ်ပါ။ ဒီကလပ်စ်က နေ အော့ဘ်ဂျက်တွေ ဖန်တီးယူဖို့ မလိုတဲ့အတွက် initializer နဲ့ instance attribute/method တွေ မပါတော့ဘူး။ ကလပ်စ်အနေဖြင့် ထားပြီး ဘာလို အော့ဘ်ဂျက်ဖန်တီးဖို့ မလိုအပ်တာလဲ။ BankApp ဟာ ဘဏ်ပရီဂါရမ်းတစ်ခုလုံးကို ကိုယ်စားပြုတယ်။ ပရီဂါရမ်းကတစ်ခုပဲ ရှိမှာပါ။ ဒီတော့ အော့ဘ်ဂျက်တစ်ခုချင်း

ဒီ ပိုင်ဆိုင်ထားရမဲ့ attribute တွေလည်း သူမှာ မရှိဘူး။ ဒါကြောင့် ကလပ်စာနှင့် ထားပြီး အသုံးပြုလိုရတယ်။

Class attribute နှစ်ခု တွေရတယ်။ 'S Bank' အတွက် Bank အော်ဂျက်တစ်ခုနဲ့ ဘဏ်ခွဲမည်တော့ ပါတဲ့ list တစ်ခုတို့ ဖြစ်တယ်။ ဘဏ်ခွဲမည် list ကို read_branch မှာ သုံးထားတယ်။ ဒါမက်သခ်က ဘဏ်ခွဲမည် မှားနေရင် ပြန်ထည့်ခိုင်းမှပါ။

class BankApp:

```

    _bank = Bank('S Bank')
    _branches = ['Yangon', 'Mandalay', 'Naypyidaw']

    @classmethod
    def open_account(cls):
        name = read_nonempty_text('Name: ')
        nric = read_nonempty_text('NRIC: ')
        gender = read_gender('Gender (F/M): ')
        dob = read_date('Date of Birth: ')

        address_line1 = read_nonempty_text('Address Line 1: ')
        address_line2 = read_nonempty_text('Address Line 2: ')
        city = read_nonempty_text('City: ')
        division = read_nonempty_text('State/Division: ')

        init_deposit = read_decimal('Initial deposit: ')

        address = Address(address_line1, address_line2, city, division)
        holder = Holder(name, dob, nric, gender, address)
        branch = cls.read_branch()
        has_confirmed = read_confirmation('Please confirm (Y/N): ')
        if has_confirmed:
            try:
                acc = cls._bank.open_acc(holder, init_deposit, branch)
                print(f'A new account with account number {acc.acc_number} '
                      f'successfully created')
            except ValueError as err:
                print(err)
            except RuntimeError as err:
                print(err)

    @classmethod
    def read_branch(cls):
        while True:
            branch_name = read_nonempty_text('Branch: ')
            if branch_name in cls._branches:
                return branch_name
            print(f'{branch_name} not found!')

```

`open_account` ၂။ class method ပါ။ `read_nonempty_text` က စာသား input ဖတ်ပေးတဲ့ ဖန်ရှင်ဖြစ်တယ်။ `bank_app.py` ဖို့ top level မှာ သတ်မှတ်ထားတာပါ။ အခြား `read_decimal`, `read_gender` စတဲ့ input ဖတ်ပေးတဲ့ ဖန်ရှင်တွေကိုလည်း top level မှာပဲ သတ်မှတ်ထားတယ်။ ဘက်ခဲ့ နံမည်ဖတ်တဲ့ `read_branch` ကိုတော့ `BankApp` class method အနေနဲ့ သတ်မှတ်တယ်။ ဘက်ခဲ့ နံမည်မှားရင် ပြန်ထည့်ပေးခိုင်း ရပါမယ်။ ဘက်ခဲ့နံမည်ဖတ်တာက ဘက်ပရိုက်ရမ်နဲ့ ဆက်စပ် အသုံးပြုရမှာပါ။ ဒါအပြင် ဘက်ခဲ့နံမည်တွေကိုလည်း `BankApp` class attribute အနေနဲ့ ထားသင့်တယ်။ `BankApp` ကလပ်စား ဆက်စပ်မှုရှိတဲ့ ဖန်ရှင်တွေနဲ့ ဒေတာတွေကို စုစည်းပေးထားတာပါ။ ဒီက လပ်စားမှာ ပါရှိတာတွေကို top level ကို ထုတ်ရေးလည်း ရပေးမဲ့ အခုလိုစွဲစည်းထားခြင်းအားဖြင့် ဘက်ပရိုက်ရမ်နဲ့ ဆက်စပ်အသုံးပြုရတဲ့ အရာတွေကို ပိုပြီး နီးနီးကပ်ကပ် တပေါင်းတစည်းတည်း ရှိစေတယ်။ Top level ဖန်ရှင်တွေကတော့ အခြား ပရိုက်ရမ်တွေမှာလည်း အသုံးတည့်မယ်။ နောက်ပိုင်းမှာ မော်ဒါးများ တစ်ခုအနေနဲ့ ထုတ်ထားရင် ပုံကောင်းတယ်။

`deposit`, `withdraw`, `transfer`

Input/output အတွက် `deposit`, `withdraw` နဲ့ `transfer` တို့ဟာလည်း class method တွေ ပါ။ `open_account` နားလည်ရင် ဒီမက်သင်တွေကိုလည်း နားလည်မှာပါ။ သဘောပေါက်အောင် လွှဲလာကြည့်မို့ တိုက်တွေ့န်းပါတယ်။

```
@classmethod
def withdraw(cls):
    acc_no = read_nonempty_text('Account no: ')
    try:
        acc = cls._bank.retrieve_account(int(acc_no))
    except ValueError as err:
        print(err)
        return
    amt = read_decimal('Amount: ')
    print('Please check and confirm the following details:')
    print(f'Holder: {acc.holder.name}')
    print(f'Account no: {acc.acc_number}')
    print(f'Amount to withdraw: {amt}')
    has_confirmed = read_confirmation('Please confirm (Y/N): ')
    if has_confirmed:
        try:
            cls._bank.withdraw(int(acc_no), amt)
            print('Operation successfully completed.')
        except ValueError as err:
            print(err)

@classmethod
def deposit(cls):
    acc_no = read_nonempty_text('Account no: ')
    try:
        acc = cls._bank.retrieve_account(int(acc_no))
    except ValueError as err:
        print(err)
```

```

        return
amt = read_decimal('Amount: ')
print('Please check and confirm the following details:')
print(f'Holder: {acc.holder.name}')
print(f'Account no: {acc.acc_number}')
print(f'Amount to deposit: {amt}')
has_confirmed = read_confirmation('Please confirm (Y/N): ')
if has_confirmed:
    try:
        cls._bank.deposit(int(acc_no), amt)
        print('Operation successfully completed.')
    except ValueError as err:
        print(err)

@classmethod
def transfer(cls):
    from_acc_no = read_nonempty_text('From account number: ')
    to_acc_no = read_nonempty_text('To account number: ')
    try:
        from_acc = cls._bank.retrieve_account(int(from_acc_no))
        to_acc = cls._bank.retrieve_account(int(to_acc_no))
    except ValueError as err:
        print(err)
    return
amt = read_decimal('Amount: ')
print('Please check and confirm the following details:')
print('From:')
print(f'    Holder: {from_acc.holder.name}')
print(f'    Account no: {from_acc.acc_number}')
print('To:')
print(f'    Holder: {to_acc.holder.name}')
print(f'    Account no: {to_acc.acc_number}')
print(f'Amount to deposit: {amt}')
has_confirmed = read_confirmation('Please confirm (Y/N): ')
if has_confirmed:
    try:
        cls._bank.transfer(int(from_acc.acc_number),
                           int(to_acc.acc_number),
                           amt)
        print('Operation successfully completed.')
    except ValueError as err:
        print(err)

```

Application Loop

အခြေနာက်ဆုံးမှာ အကောင့်ဖွံ့ဌေး၊ ငွေသွေး၊ ငွေထုတ်၊ ငွေလွှဲ ဘယ် transaction လုပ်မလဲ ရွှေးလိုရအောင် application loop ကို ရေးရပါမယ်။

```
@classmethod
def app_loop(cls):
    print('O (to open a new account)')
    print('D (to deposit)')
    print('W (to withdraw)')
    print('T (to transfer)')
    while True:
        menu_choice = input('Please enter O/D/W/T to start '
                            'a transaction: ')
        if menu_choice == 'O':
            cls.open_account()
        elif menu_choice == 'D':
            cls.deposit()
        elif menu_choice == 'W':
            cls.withdraw()
        elif menu_choice == 'T':
            cls.transfer()
        else:
            print(f'{menu_choice} is not valid!')
```

ပရိုဂရမ်က ဒီ loop ကနေ စအလုပ်လုပ်ရမှာဆိုတော့ အခုလို ခေါ်ပေးရုပါပဲ

```
BankApp.app_loop()
```

ဒီလိုပြပေးပါလိမ့်မယ်။ O/D/W/T ထဲက တစ်ခုရွှေးရင် လုပ်ဆောင်မဲ့ ကိစ္စအလိုက် လိုအပ်တဲ့ input တွေ ဆက်တောင်းမှာပါ။ နောက်ဆုံးမှာ Y/N ကွန်ဖန်းလုပ်ရပါမယ်။

```
O (to open a new account)
D (to open a new account)
W (to open a new account)
T (to open a new account)
Please enter O/D/W/T to start a transaction:
```

Top Level Common Functions

Top level ဖန်ရှင်တွေကို လေ့လာလိုရအောင် ဖော်ပြပေးထားပါတယ်။ အလားတူ ဖန်ရှင်တွေကို အခန်း (၈) မှာ အသေးစိတ် ရှင်းပြထားတယ်။

```
FEMALE = 'F'
MALE = 'M'

MONTHS = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
          'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
```

```

'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

def parse_date_str(dt_str):
    parts = dt_str.replace('/', '-').split('-')
    if parts[0].isdigit():
        return date(int(parts[2]), int(parts[1]), int(parts[0]))
    else:
        return date(int(parts[2]), MONTHS[int(parts[0])], int(parts[1]))

def read_date(prompt):
    while True:
        try:
            return parse_date_str(input(prompt))
        except Exception as err:
            print("Incorrect date format. Please enter the date again.")
            print('Error is probably: ' + str(err))

def read_gender(prompt):
    while True:
        gender = input(prompt)
        if gender in ['F', 'M']:
            return gender
        else:
            print('Invalid gender! Accept only F or M!')

def read_confirmation(prompt):
    while True:
        choice = input(prompt)
        if choice == 'Y':
            return True
        elif choice == 'N':
            return False
        else:

```

```

        print('Invalid gender! Accept only Y or N!')

def read_decimal(prompt):
    while True:
        value = input(prompt)
        try:
            return Decimal(value)
        except InvalidOperation:
            print('Invalid number!')


def read_nonempty_text(prompt):
    while True:
        txt = input(prompt)
        if txt.strip() != '':
            return txt
    print(f'{prompt} cannot be empty!')

```

၃၃။

အခြောက်ပြခဲ့တဲ့ ပရိုကရမ်ကို လေ့လာခြင်းအားဖြင့် အော့ဘ်ဂျက်နဲ့ ကလပ်စောင့်တွေကို ပရိုကရမ်တစ်ခု တည်ဆောက်ရမှာ ဘယ်လိုအသုံးချလိုရလဲ သဘောပေါက် နားလည်မယ်လို့ မျှော်လင့်ပါတယ်။ ဒီပရိုကရမ်မှာ အခြားဖိချာတွေ ထပ်ဖြည့်ကြည့်ပါ။ ဥပမာ လက်ကျွန်ငွေ စစ်လို့ရအောင်၊ ဘတ်ခဲ့ အသစ်ထည့်လို့ရအောင် ချဲတွင်ကြည့်ရင် ပိုပြီး နားလည်လာမှာပါ။

အခန်း ၁၀

Inheritance and Polymorphism

Inheritance ဟာ ရှိထားပြီး ကလပ်စာစုံခု မှာ attribute နဲ့ မက်သဒ္ဓတ္ထကို အားကလပ်စာနဲ့ ဆက်ခံယူဖို့ဖြစ်နိုင်တယ်။ ကလပ်စာစုံခု ရေးထားတဲ့ ပရိုကရမဲ့ ကုဒ်တွေကို ထပ်ခါထပ်ခါ ရေးစရာ မလိုဘဲ ပြန်လည် အသုံးပြုလိုရစေတဲ့ နည်းလိုလည်း ယူဆနိုင်တယ်။ ဆက်ခံရရှိတဲ့ attribute နဲ့ မက်သဒ္ဓတ္ထအပြင် အသစ် ထပ်ထည့်တာ၊ နိဂုံရှင်းကို ပြင်ဆင်တာလည်း ကလပ်စာစုံအသစ်က လုပ်ဆောင်လိုရမှာ ဖြစ်တယ်။ Object-oriented programming မှာ inheritance ဟာ အရေးအပါဆုံး သဘောတရား တစ်ခု ဆိုရင်လည်း မမှားဘူး။ ဂိမ်းရေးတဲ့ လိုက်ဘရိတ္ထ၊ Graphical User Interface (GUI) အတွက် လိုက်ဘရိတ္ထ အသုံးပြုမယ်ဆိုရင် inheritance သဘောတရားက မသိမဖြစ်ပါဘူး။ ဒီအခန်း အတွက် inheritance အသုံးချ ဥပမာအနေနဲ့ ‘Breakout’ လိုခေါ်တဲ့ ဂိမ်းလေးတစ်ခုကို Arcade လိုက်ဘရိနဲ့ ရေးထားတာကို နောက်ပိုင်းမှာ လေ့လာကြရမှာပါ။

ရှုံးအခန်းမှာ ဖော်ပြခဲ့တဲ့ has-a အပြင် object-oriented programming မှာ အရေးပါတဲ့ နောက် relationship တစ်မျိုးက *is-a* ဖြစ်ပါတယ်။ *Is-a* relationship ရှိတဲ့ အရာတွေကို inheritance နဲ့ ထင်ဟပ်ဖော်ပြနိုင်တယ်။ ဘတ်စာကားသည် ကားဖြစ်သည်။ ကားသည် ယာဉ်ဖြစ်သည်။ ယာဉ်၊ ကားနဲ့ ဘတ်စာကားတို့အကြား ဆက်စပ်မှုဟာ *is-a* relationship ဖြစ်တယ်။ *Vehicle*, *Car* နဲ့ *Bus* ကလပ်တွေဟာ အစဉ်အလိုက် inherit လုပ်ယူနိုင်တာကို တွေ့ရမှာပါ။

၁၀.၁ Inheritance ဥပမာ ‘Account Class Hierarchy’⁹

ဘဏ်အကောင့် အမျိုးအစား အမျိုးမျိုး ရှိပါတယ်။ အတိုင်းနှင့် တစ်နေ့တာ လုပ်ဆောင်နိုင်တဲ့ transaction အကြိမ်အရေအတွက် အနည်းဆုံး ထားရှိရမဲ့ ငွေပေးကြား လက်ရှိ လက်ကျန်ငွေထက် ပိုထုတ်လို ရ/ မရ (overdraft) စတဲ့အချက်တွေ အကောင့်တစ်မျိုးနဲ့တစ်မျိုး မတူကြပါဘူး။

ဥပမာ savings account (ငွေစွာအကောင့်) ဆိုရင် ဘဏ်တိုးရမယ်။ တစ်နေ့တာ transaction အကြိမ်အရေအတွက်ရော ထုတ်ယူနိုင်တဲ့ ငွေပေးကြား အကောင့်အသတ်ရှိတယ်။ ဒါကြောင့် ဒီအကောင့် အမျိုးအစားက လုပ်နေးသုံးအတွက် အဆင်မပြောဘူး။ Current account ကတော့ စီးပွားရေး လုပ်ငန်း တွေအတွက် အဓိက ရည်ရွယ်တယ်။ Transaction အကောင့်အသတ်မရှိဘူး။ လိုသလောက် ထတ်ယူလို ရတယ်။ Overdraft လိုခေါ်တဲ့ ကိုယ့်မှာရှိတာထက် (အကောင့်အသတ်တော့ရှိတာပေါ့) ပိုထုတ်လို ရတယ်။ Current account ကို ဘဏ်က အတိုင်းပေးလေ့မရှိဘူး။

SavingAccount နဲ့ CurrentAccount ကလပ်စာကို အောက်ပါအတိုင်း သတ်မှတ်နိုင်ပါတယ်။ ကလပ်နှစ်ခုကို နှင့်ယုံကြည့်ရင် တူညီတဲ့အပိုင်းတွေ ပါဝင်နေတာ တွေ့ရမှာပါ။ နှစ်ခုလုံးမှာ deposit မက သဒ်က တူတူပါဘူး။ _holder, _balance, _acc_number မေးရိုရော့လဲတွေ ပါဝင်တာချင်းလည်း တူ

တယ်။ ကွားချက်တခိုက်လည်း တွေ့ရပါတယ်။ withdraw မက်သင် နှစ်ခု မတူကြဘူး။ _txn_cnt_tot, _txn_amt_tot နဲ့ provide_interest တို့ကို SavingAccount မှာပဲ တွေ့ရတယ်။ တစ်ဖက် မှာလည်း _overdraft_amt နဲ့ is_overdrafted တို့ကိုတော့ CurrentAccount မှာ တွေ့မှုဖြစ်ပြီး SavingAccount မှာ မပါပြန်ဘူး။

```

class SavingAccount:
    def __init__(self, holder, acc_number, balance):
        self._holder = holder
        self._acc_number = acc_number
        self._balance = balance
        self._txn_cnt_tot = 0
        self._txn_amt_tot = Decimal(0.00)

    def deposit(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount for deposit!')
        self._balance += amt

    def withdraw(self, amt):
        if self._txn_amt_tot >= Decimal(500_000.00):
            raise ValueError('Daily withdraw limit exceeds!')
        if self._txn_cnt_tot >= 5:
            raise ValueError('Daily transaction count exceeds!')
        if amt > self._balance:
            raise ValueError('Not enough balance!')
        self._balance -= amt

    def provide_interest(self):
        """complex logic for interest calculation"""
        pass

class CurrentAccount:
    def __init__(self, holder, acc_number, balance):
        self._holder = holder
        self._acc_number = acc_number
        self._balance = balance
        self._overdraft_amt = Decimal(1_000_000.00)

    def deposit(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount for deposit!')
        self._balance += amt

    def withdraw(self, amt):
        if amt > (self._balance + self._overdraft_amt):
            raise ValueError('Not enough balance!')

```

```

    self._balance -= amt

    def is_overdrafted(self):
        return self._overdraft_amt < Decimal(0.00)

```

ဒီကလပ်နှင့်ခုမှာ ထပ်နေတဲ့ တူညီတဲ့ကုဒ်တွေကို တစ်ခါပဲ ရေးဖိုလိုမယ်ဆိုရင် ပိုပြီးအဆင်ပြေမှာ ပါ။ ဒီလိုအခြေအနေမျိုးမှာ inheritance က ဘယ်လို အထောက်အကူဗြို့ပြုလဲ ကြည့်ရအောင်။ ထပ်နေတဲ့ တူညီတဲ့ကုဒ်တွေကို ဘုံထုတ်လိုက်ပြီး ကလပ်တစ်ခု သတ်မှတ်ပါမယ်။

```

from decimal import Decimal

class Account:
    def __init__(self, holder, acc_number, balance):
        self._holder = holder
        self._acc_number = acc_number
        self._balance = balance

    def deposit(self, amt):
        if amt <= Decimal(0.00):
            raise ValueError('Invalid amount for deposit!')
        self._balance += amt

```

CurrentAccount နဲ့ SavingAccount က တူညီတဲ့ အပိုင်းတွေ Account ကလပ်မှာ ခဲ့ထုတ်ထား တဲ့ သတ်မှတ်ပါ။ ဒီအပိုင်းတွေကို inherit လုပ်ယူပါမယ်။

```

class CurrentAccount(Account):
    def __init__(self, holder, acc_number, balance, overdraft_amt):
        super().__init__(holder, acc_number, balance)
        self._overdraft_amt = overdraft_amt

    def withdraw(self, amt):
        if amt > (self._balance + self._overdraft_amt):
            raise ValueError('Not enough balance!')
        self._balance -= amt

    def is_overdrafted(self):
        return self._balance < Decimal(0.00)

```

CurrentAccount ၏ Account ကို inherit လုပ်မှာဖြစ်တာကြောင့် ကလပ်သတ်မှတ်တဲ့အခါ အခုလို ရေးရပါမယ်

```
class CurrentAccount(Account):
```

Account ကို superclass လိုက်ပြီး သူဆီကနေ ဆက်ခံယူမဲ့ CurrentAccount ကို subclass လို ခေါ်ပါတယ်။ ပေးမဲ့ကလပ်က superclass ။ ယူမဲ့ ကလပ်က subclass ပေါ့။ parent နဲ့ child လို လည်းခေါ်တယ်။

Subclass initialize လုပ်တဲ့အခါ superclass အပိုင်းကိုလည်း initialize လုပ်ဖို့လိုတယ်။ ဒါ အတွက် superclass `__init__` ကို အခုလို ခေါ်ပါတယ်

```
# call __init__ method of the superclass
super().__init__(holder, acc_number, balance)
```

`_holder`, `_acc_number`, `_balance` instance variable တွေက ဆက်ခံရရှိထားတာပါ။ ငှါးတို့ initialize လုပ်ဖို့အတွက် superclass initializer ကို ခေါ်ပါမယ်။ Subclass က တိုက်ရှိက်မလုပ်ရပါဘူး။ Subclass ကိုယ်ပိုင် instance variable တွေကိုတော့ ပုံမှန်အတိုင်း initialize လုပ်နိုင်တယ်။

```
self._overdraft_amt = overdraft_amt
```

`deposit` မက်သဒ်ကို superclass ကနေ `CurrentAccount` က ဆက်ခံရရှိမယ်။ `withdraw` နဲ့ `is_overdrafted` က ကိုယ်ပိုင်ရှိမယ်။ ဒီမက်သဒ်နှစ်ခုမှာ `self._balance` ကို သုံးထားတာ တွေရမှာပါ။ ဆက်ခံရရှိတဲ့ instance variable တွေကို subclass မက်သဒ်ထဲမှာ `self` နဲ့ ရည်ညွှန်းအသုံးပြုလို ရတယ်။

`SavingAccount` ကိုအောက်ပါအတိုင်း သတ်မှတ်ပါတယ်။ တစ်နေတာ `transaction` အကြိမ် အရေအတွက်နဲ့ ထုတ်ယူတဲ့ ပမာဏအတွက် `_txn_cnt_tot` နဲ့ `_txn_amt_tot` instance variable တွေ တပ်ဖြည့်ထားတယ်။ (`_txn_cnt_tot` နဲ့ `_txn_amt_tot` ကို တစ်နေတာကုန်ဆုံးတိုင်း သူညျဖြစ်အောင် ပြန်လုပ်ထားဖို့ လိုပါလိမ့်မယ်။ အခုံပမားမှ ဒီကိစ္စအတွက် ထည့်မစဉ်းစားထားပါဘူး)။

```
class SavingAccount(Account):
    def __init__(self, holder, acc_number, balance):
        super().__init__(holder, acc_number, balance)
        self._txn_cnt_tot = 0
        self._txn_amt_tot = Decimal(0.00)

    def withdraw(self, amt):
        if self._txn_amt_tot >= Decimal(200_000.00):
            raise ValueError('Daily withdraw limit exceeds!')
        if self._txn_cnt_tot >= 30:
            raise ValueError('Daily transaction count exceeds!')
        if amt > self._balance:
            raise ValueError('Not enough balance!')
        self._balance -= amt
        self._txn_cnt_tot += 1
        self._txn_amt_tot += amt

    def provide_interest(self):
        """complex logic for interest calculation"""
        pass
```

`Saving` နဲ့ `current` အပိုင် သတ်မှတ်ကာလအတွင်း မထုတ်ဘဲထားရတဲ့ `fixed deposit` ၊ အသေးစား လုပ်ငန်းတွေအတွက် `SME` စတဲ့ အိုးအကောင် အမျိုးအစားတွေလည်း ရှိတယ်။ ဒီအကောင့်တွေ နဲ့ သက်ဆိုင်တဲ့ ကလပ်စွဲတွေကလည်း `Account` ကို inherit လုပ်နိုင်ပါတယ်။

၁၁၂ Overriding

Superclass ကနေ ဆက်ခံရရှိထားတဲ့ မက်သာဒ်ကို subclass မှာ ပြင်ဆင်သတ်မှတ်တာကို *method overriding* လို့ ခေါ်ပါတယ်။ Deposit လုပ်တဲ့အခါ current account က အနည်းဆုံး ဆယ်သိန်း ဖြစ်မှ လက်ခံတယ်ဆိုပါစို့။ ဆက်ခံ ရထားတဲ့ မူလ deposit မက်သာဒ်က ဒီလိုအပ်ချက်နဲ့ မကိုက်ညီတော့။ အခုလို အခြေအနေမျိုးမှာ ဆက်ခံယူမဲ့ CurrentAccount ဟာ deposit မက်သာဒ်ကို လက်ရှိလိုအပ်ချက်နဲ့ ကိုက်ညီအောင် override လုပ်နိုင်ပါတယ်။

```
class CurrentAccount(Account):
    def __init__(self, holder, acc_number, balance):
        super().__init__(holder, acc_number, balance)
        super._holder = holder
        self._overdraft_amt = Decimal(1_000_000.00)

    # override deposit method
    def deposit(self, amt):
        if amt <= Decimal(1_000_000.00):
            raise ValueError('Deposit at least 1,000,000 to current acc!')
        self._balance += amt

    # other methods
```

မက်သာဒ်တစ်ခုကို ဆက်ခံရရှိတဲ့အတိုင်း အသုံးပြုလို အဆင်မပြတဲ့အခါ subclass က override လုပ်နိုင်ပါတယ်။ Subclass အများစုက ဆက်ခံရရှိတဲ့အတိုင်း အသုံးပြုလိုရပြီး တချို့ အနည်းစုကပဲ override ရတဲ့အခါ ကုဒ်တွေထပ်နေတာ သိသိသာသာ လျော့သွားမှုပါ။

ဆက်ခံရရှိတဲ့အတိုင်း အသုံးပြုလိုရနိုင်ပေမဲ့ optimization အတွက် override လုပ်ရတာလည်း ရှိတယ်။ Polygon ပါတ်လည်နားရာတဲ့ နည်းလမ်းကို rectangle အတွက်လည်း သုံးလို့ရပေမဲ့ rectangle အတွက် သီးသန်ဖော်မြှုံးလာ $2 \times (width + height)$ က ပိုရှင်းလင်းပြီး တွက်ချက်ရပို့ဖြန်မှာ အမှန်ပါပဲ။ ဒါကြောင့် စွမ်းဆောင်ရည်အတွက် Rectangle ကလပ်စုံမှာ perimeter ကို override လုပ်ဖို့ ဆုံးဖြတ်နိုင်ပါတယ်။

```
import math

class Polygon:
    def __init__(self, vertices):
        self.vertices = vertices

    def perimeter(self):
        perimeter = 0
        num_vertices = len(self.vertices)
        for i in range(num_vertices):
            x1, y1 = self.vertices[i]
            x2, y2 = self.vertices[(i + 1) % num_vertices]
            perimeter += math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
        return perimeter
```

```

class Rectangle(Polygon):
    def __init__(self, width, height):
        # Define vertices for a rectangle assuming bottom-left corner at (0, 0)
        vertices = [(0, 0), (width, 0), (width, height), (0, height)]
        super().__init__(vertices)
        self.width = width
        self.height = height

    def perimeter(self):
        # Overriding the perimeter method for performance
        return 2 * (self.width + self.height)

```

Inheritance နဲ့ method overriding ကို ပရိဂရမ်တစ်ခုရဲ့ စထရက်ချာကို ထိန်းကွပ်ပေးဖို့ အသုံးပြုတာကိုလည်း တချို့လိုက်ဘရီတွေမှာ တွေ့ရတယ်။ ဥပမာ Arcade လိုက်ဘရီနဲ့ ဂိမ်းရေးတဲ့အခါ သူမှာပါတဲ့ ကလပ်စွေတွေကို inherit လုပ်ပြီး ကိုယ်လိုချင်တဲ့ ပုံစံနဲ့ အလုပ်လုပ်အောင် မက်သွ်တွေကို လိုအပ်သလို override လုပ်ပေးရတယ်။ ဒီအခန်း နောက်ပိုင်းမှာ Arcade လိုက်ဘရီနဲ့ ဂိမ်းလေးတစ်ခု တည်ဆောက်တဲ့အခါ တွေ့ရမှာပါ။

၁၁.၃ Multilevel Inheritance

ကလပ်စွေ A ကို B က ဆက်ခံထားမယ်။ တစ်ခါ B ကို C က ထပ်ဆင့် ဆက်ခံယူမယ်။ ဒီလို အဆင့်ဆင့် inherit လုပ်တာကို multilevel inheritance လို့ခေါ်တယ်။ ကလပ်စွေ C ဟာ B နဲ့ A နှစ်ခုလုံးရဲ့ attribute နဲ့ မက်သွ်တွေကို ရရှိမှာပါ (A မှာ ရှိတော့တွေကို B ကနေတစ်ဆင့် ရတာ)။ ရှေ့က ဥပမာ မှာ inheritance level နှစ်ခုတွေရှိတယ်။ အပေါ်အဆင့်မှာ Account, အောက်တစ်ဆင့်မှာ သူ့ကိုဆက်ခံယူထားတဲ့ SavingAccount နဲ့ CurrentAccount ရှိတယ်။ SavingAccount နဲ့ CurrentAccount ကို ဆက်ခံထားတဲ့ ကလပ်စွေ နောက်တစ်ဆင့်ရှိလာရင် သုံးဆင့်ဖြစ်သွားမှာပါ။

```

class SavingAccount(Account):
    def __init__(self, holder, acc_number, balance):
        super().__init__(holder, acc_number, balance)
        self._txn_cnt_tot = 0
        self._txn_amt_tot = Decimal(0.00)

    def provide_interest(self):
        """complex logic for interest calculation"""
        pass

class SuperSavingAccount(SavingAccount):
    def __init__(self, holder, acc_number, balance):
        super().__init__(holder, acc_number, balance)

    def withdraw(self, amt):
        if self._txn_amt_tot >= Decimal(200_000.00):
            raise ValueError('Daily withdraw limit exceeds!')
        if self._txn_cnt_tot >= 30:

```

```

        raise ValueError('Daily transaction count exceeds!')
    if amt > self._balance:
        raise ValueError('Not enough balance!')
    self._balance -= amt
    self._txn_cnt_tot += 1
    self._txn_amt_tot += amt

class FamilySavingAccount(SavingAccount):
    def __init__(self, holder, acc_number, balance):
        super().__init__(holder, acc_number, balance)

    def withdraw(self, amt):
        if self._txn_amt_tot >= Decimal(600_000.00):
            raise ValueError('Daily withdraw limit exceeds!')
        if self._txn_cnt_tot >= 30:
            raise ValueError('Daily transaction count exceeds!')
        if amt > self._balance:
            raise ValueError('Not enough balance!')
        self._balance -= amt
        self._txn_cnt_tot += 1
        self._txn_amt_tot += amt

```

၁၁.၄ Class Hierarchy and UML

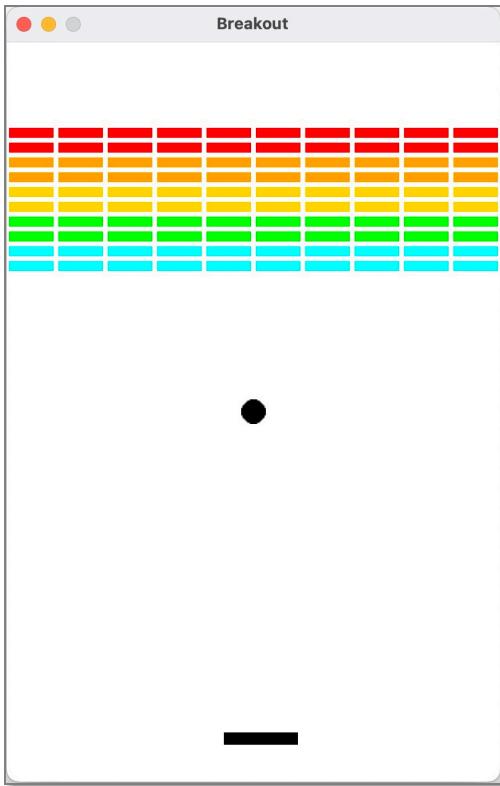
【ဖြည့်စွက်ရန်】

၁၁.၅ Is-A Relationship and Inheritance

【ဖြည့်စွက်ရန်】

၁၁.၆ အသုံးချုပ်မာ (၁) Breakout Game

ဒီအခန်းအတွက် အသုံးချုပ်မာက Breakout လိုပေါ်တဲ့ ရှိုးရှင်းတဲ့ ဂိမ်းလေးတစ်ခု တည်ဆောက်ပါမယ်။ Breakout ဂိမ်းဟာ ကမ္ဘာကျော် Apple ကုမ္ပဏီ ပူးတွဲတည်တောင်သူ Steve Wozniak ဒီနိုင်းလုပ် ခဲ့တဲ့ ကြွောင်ဂိမ်းတစ်ခု ဖြစ်ပါတယ်။ ဗားရှင်းတွေအမျိုးမျိုး ရှိုပေါ့ အစိကအနှစ်သာရကတော့ ရွှေနောတဲ့ ဘေးလုံးကို paddle ပြားနဲ့ လိုက်ဖမ်းပြီး အစိအရှိရှိနေတဲ့ အုတ်ခဲတွေ အားလုံးကို ကုန်တဲ့အထိ ဖျက်ရတာပါ။ ဂိမ်းစစ်ချင်း အနေအထားကို ပုံ (၁၀.၁) မှာ တွေ့ရပါမယ်။ အပေါ်ပိုင်းမှာက အုတ်နံရုပါ့။ ဘေးလုံးက အောက်ကိုကျေလာမယ်။ paddle ပြားနဲ့ လိုက်ဖမ်းရမယ်။ ဘေးလုံးက paddle, အုတ်ခဲ၊ ဘယ်/ညာ/အပေါ် ဘောင်တွေနဲ့ ဝင်တိုက်ရင် ပြန်ကန်ထွက်တယ်။ အုတ်ခဲတွေက ဘေးလုံးနဲ့ တိုက်မိရင် ပျက်သွားမယ်။ ဘေးလုံးကို မဖမ်းလိုက်နိုင်လို့ paddle ပြားအောက်ဘက် ကျသွားရင် ရှိုးမယ် (အောက်ဘက်ကျသွားရင် ပြန်ကန်မထွက်ဘူး)။ အုတ်ခဲတွေ ကုန်တဲ့ထိ ဘေးလုံးမကျသွားအောင် ထိန်းထားနိုင်ရင် အနိုင်ရုံးမှာဖြစ်တယ်။



ပုံ ၁၀.၁

ဒီဂိမ်းရေးဖို့ Arcade လိုက်ဘရီကို အသုံးပြုမှာပါ။ Arcade လိုက်ဘရီဟာ inheritance ကို အခြေခံထားတယ်။ `Window`, `View`, `Sprite` စတဲ့ ကလပ်စွဲတွေကို လိုက်ဘရီကပေးထားတယ်။ ဂိမ်းတစ်ခုရေးတဲ့အခါ ဒီကလပ်စွဲတွေကို inherit လုပ်ပြီး လိုအပ်တဲ့ မက်သဒ်တွေကို override လုပ်ပေးရုပ်။ လိုက်ဘရီက ပရိုကရမ်ကို အကြမ်းထည် စထရက်ချာ ချထားပေးတယ်။ အဲဒါ စထရက်ချာထဲမှာ ကိုယ်တိုင် စိတ်ကြိုက် ဖန်တီးချင်တဲ့နေရာတွေကို ပြင်ဆင်ဖြည့်စွက်လို့ရအောင် လုပ်ပေးထားတာလို့ ယူဆနိုင်တယ်။ ကြိုတင်သိထားဖို့ လိုအပ်တာတွေ တစ်ဆင့်ချင်း အရင်ကြည့်ရအောင်။

arcade.Window

`Window` ကလပ်စွဲဟာ ကရပ်စစ် `window` တစ်ခုကို ကိုယ်စားပြုတယ်။ ကရပ်စစ်ရုပ်ပုံတွေ ဆွဲဖို့နဲ့ အန်နီ မေးရှင်း လုပ်ဖို့ လိုအပ်တာတွေ ဒီကလပ်စွဲမှာ ထောကပံပေးထားတယ်။ ဒါအပြင် မောက်စိုး၊ ကီးဘုဒ် စတဲ့ `input device` တွေနဲ့ ဂိမ်းကို ကွန်ထရီးလုပ်လို့ရအောင် နည်းလမ်းတွေလည်း ပါတယ်။

Arcade ပရိုကရမ်တစ်ခုကို `run` တဲ့အခါ `Window` မှာ ပါတဲ့ `on_draw` မက်သဒ်ကို တစ်စက်နဲ့ အကြိမ် ခြောက်ဆယ် ခေါ်ပေးတယ်။ သူနှင့်အတိုင်း `on_draw` မက်သဒ်က ဘာမှ လုပ်ဆောင်မပေးဘူး။ ကိုယ့်လို့ချင်တဲ့ ကရပ်စစ်ပုံ ဖော်ဖို့ `Window` ကို inherit လုပ်ပြီး override လုပ်ပေးရမယ်။ အပြာရောင် ဘောလုံးတစ်ခု အလယ်ဗဟိုမှာ ဆွဲမယ်ဆိုပါစဲ့ ...

```
# File: arcade_blue_circle.py
import arcade
from arcade.color import *
```

```

WIN_WIDTH = 640
WIN_HEIGHT = 480

class MyGame(arcade.Window):

    def __init__(self, width, height, title):
        super().__init__(width, height, title)
        arcade.set_background_color(arcade.color.ALMOND)

    def on_draw(self):
        self.clear()
        arcade.draw_circle_filled(WIN_WIDTH / 2, WIN_HEIGHT / 2, 20, BLUE)

window = MyGame(WIN_WIDTH, WIN_HEIGHT, "Show Blue Circle")
arcade.run()

```

on_draw ကို အခုလို override လုပ်နိုင်ပါတယ်။ ဒီပုဂ္ဂိုရမ် run တဲ့အခါ on_draw ကို တစ်စက်နှစ်တိုင်း တစ်စက်နှစ်တိုင်းမှာ အကြိမ်ခြေက်ဆယ် (60 frames per second) တောက်လျောက် ခေါ်နေမှုပါ။ ပရိုဂ္ဂရမ window မပိတ်မချင်းပေါ့။ ဒီလိုဖြစ်အောင် Arcade လိုက်ဘရိုက လုပ်ပေးထားတာ။ ဒီအတွက် သီးခြားရေးဖို့ မလိုဘူး။ Window ကလပ်စ်ကို inherit လုပ်ပြီး on_draw ကို override လုပ်ပေးရင် ရပြီ။ clear မက်သဒ်ကိုလည်း superclass Window ကနေ ဆက်ခံရထားတာ။ Window ပေါ်မှာ ရှိတဲ့ ဂရပ်ဖစ်အားလုံးကို ရှင်းပေးပြီး သတ်မှတ်ထားတဲ့ နောက်ခံရောင် ဖြည့်ပေးတဲ့ မက်သဒ်ပါ။ အနုနီမေးရှင်း အတွက် ဒီမက်သဒ်က အရေးကြီးတယ်ဆိုတာ တွေ့ရပါမယ်။

မောက်စ် (သို့) ကိုဘုံးဖို့ ဘောလုံးကို ရွှေ့လို့ရအောင်လည်း လုပ်လို့ရတယ်။ ခက်ခဲတဲ့ ကိစ္စတွေကို Window က အဆင်သင့် လုပ်ထားပေးတာပါ။ ဒါဟာ အထူးအဆန်း ဖြစ်နေနိုင်ပါတယ်။ ဘယ်လိုများလုပ်ထားလဲ အကြိမ်းဖျဉ်း သဘောတရား နားလည်ချင်ရင် ဒီဥပမာကို လေ့လာကြည့်ပါ။

```

class ConsolePrinter:
    def __init__(self, times):
        self._times = times

    def do_before(self):
        print("Checking if everything is ready.")

    def do_after(self):
        print("Doing cleaning up.")

    def do_my_task(self):
        self.do_before()
        for i in range(self._times):
            self.my_task()
        self.do_after()

    # do nothing
    def my_task(self):

```

```
pass
```

```
class MyConsolePrinter(ConsolePrinter):
    def __init__(self, times, text):
        super().__init__(times)
        self._text = text

    # to print as you want
    def my_task(self):
        print(self._text)

printer = MyConsolePrinter(15, 'Hello')
printer.do_my_task()
```

ConsolePrinter မှာ my_task ကလဲလို့ ကျန်မက်သိတွေအားလုံး အပြည့်အစုံရေးထားတာ တွေ့ရမှာပါ။ do_my_task က my_task ကိုခေါ်ထားတယ်။ MyConsolePrinter က ထုတ်ပေးချင်တဲ့ စာသားအတွက် my_task ကို override လုပ်ထားတာကို ရရှိပြုပါ။ Arcade လိုက်ဘရဲ့ ကလပ်စ်တွေကလည်း ဒီသဘောတရားပဲ။ ဂိမ်းတစ်ခုအတွက် လိုအပ်ချက်တွေကို ပြင်ဆင်ပေးထားတယ်။ ပုံစံချပေးထားတယ်။ အသုံးပြုသူက ဒီကလပ်စ်တွေကို inherit လုပ်ပြီး ဂိမ်းရဲ့ လိုအပ်ချက်အတိုင်း ဖြစ်အောင် မက်သိတွေကို override လုပ်ပြီး စိတ်ကြိုက်ပုံဖော်ယူရတာ။

အနိန်မေးရင်း

စောစောက ပရိုဂရမ်မှာ စက်ပိုင်း ညာဘက်ဘက်အပေါ်ကို ရွှေသွားအောင် အန်နီမေးရှင်း လုပ်မယ်ဆိုပါတဲ့။ on_draw လုပ်တဲ့ အခါတိုင်းမှာ လက်ရှိ x နဲ့ y တန်ဖိုးကို နည်းနည်းချင်း တိုးပေးနိုင်ပါတယ်။ နမူနာအနေနဲ့ x ကို 2 pixels နဲ့ y ကို 1 pixel တိုးပေးပါမယ်။

```
# File: arcade_moving_circle.py
import arcade
from arcade.color import *

WIN_WIDTH = 640
WIN_HEIGHT = 480
RADIUS = 15

class MyGame(arcade.Window):

    def __init__(self, width, height, title):
        super().__init__(width, height, title)
        self._circle_x = WIN_WIDTH / 2
        self._circle_y = WIN_HEIGHT / 2
        arcade.set_background_color(ALMOND)
```

```

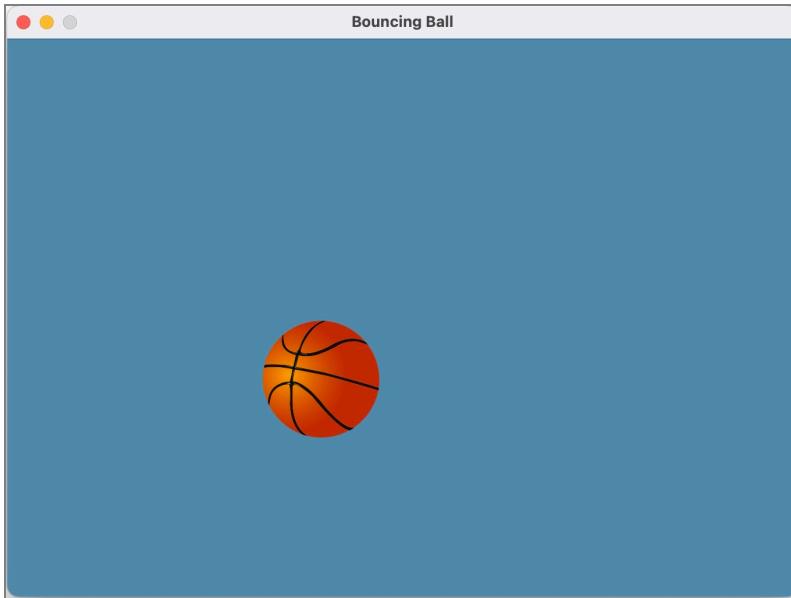
def on_draw(self):
    self.clear()
    self._circle_x += 2      # move 2 pixels to the right
    self._circle_y += 1      # move 1 pixels upwards
    arcade.draw_circle_filled(self._circle_x,
                              self._circle_y,
                              RADIUS,
                              BLUE)

window = MyGame(WIN_WIDTH, WIN_HEIGHT, "Moving Circle")
arcade.run()

```

Sprite

Sprite ကလပ်က ရုပ်ပုံ (image) တွေကို ဖော်ပြနိုင်တယ်။ ရုပ်ပုံအတွက် .jpg, .png, .bmp အစရိတဲ့ image file ဖော်ပြန်ဖိုး ထောက်ပံ့ပေးတယ်။ ဂိမ်း အတ်ကောင်တွေ၊ အရာဝတ္ထုတွေ ပုံဖော်ပြနိုင်ဖို့ အဓိကကျတဲ့ ကလပ်ပါ။ တစ်ခုနဲ့တစ်ခု ဝင်တိုက်တာ (collision)၊ အပေါ်/အောက် (သို့) ဘယ်/ညာ လှည့်တာ၊ ပစ္စည်းတွေကို အပ်စုလိုက် ရွှေလျားစေတာ စတာတွေကို အလွယ်တကူ လုပ်ဆောင်လို့ရအောင် Sprite ကလပ်က ထောက်ပံ့ပေးထားတယ်။ Window ဘောင် တစ်ဖက်ဖက်နဲ့ ဝင်တိုက်ရင် ဘောလုံး က ပြန်ကန်ထွက်တဲ့ အနုနီမေးရှင်းဥပမာကို ကြည့်ရအောင် ...



ပုံ ၁၁၂ Ball Sprite ပေါ်

```

# File: arcade_bouncing_ball.py
import arcade
SCREEN_WIDTH = 680
SCREEN_HEIGHT = 480

```

```

class Ball(arcade.Sprite):

    def update(self):
        # rotate the sprite
        self.angle += 1
        # move the sprite
        self.center_x += self.change_x
        self.center_y += self.change_y

        if self.left < 0:
            self.change_x *= -1

        if self.right > SCREEN_WIDTH:
            self.change_x *= -1

        if self.bottom < 0:
            self.change_y *= -1

        if self.top > SCREEN_HEIGHT:
            self.change_y *= -1

```

center_x, center_y ကဲ Sprite ဗဟိုမှတ် တည်နေရပါ။ change_x, change_y က အနုစိ မေးရှင်းလုပ်ရင် နည်းနည်းချင်း ရွှေ့ပေးရမဲ့ x နဲ့ y ပမာဏ။ Sprite အမြန်နှင့်နဲ့ ဦးတည်ရာကို ဒီနှစ်ခု ရဲ့ တန်ဖိုးနဲ့ ထိန်းလှုပေးနိုင်တယ်။ left, right, top, bottom attribute တွေကတော့ Sprite ရဲ့ ဘယ်/ညာ x တန်ဖိုးနဲ့ အထက်/အောက် y တန်ဖိုးတွေကို ဖော်ပြတယ်။ ဒါ ပေါ်ရောဘဲလ်တွေ အားလုံး ကို superclass Sprite ကနေ ဆက်ခံရရှိထားတာ။

ဘယ် (သို့) ညာဘက် ဘောင်နဲ့ ထိရင် change_x ကို အပေါင်းအနှံတ် ပြောင်းပြန် လုပ်ပေးရပါ မယ်။ အထက် (သို့) အောက် ဘောင်နဲ့ ထိရင် change_y ကို ပြောင်းပြန် လုပ်ပေးရပါမယ်။ ပုံ (၁၀.၃) မှာ ကြည့်ပါ။

Ball ကလည်းမှာ __init__ မက်သွေ့ မပါဘူး။ (Subclass မှာ __init__ မပါရင် အော့ဂျက် ဖန်တီးတဲ့အခါ ဆက်ခံရရှိထားတဲ့ superclass __init__ ကိုပဲခေါ်ပါတယ်)။

ဒါ Sprite ကို ဘယ်လို အသုံးပြုလဲ ဆက်ကြည့်ရအောင်

```

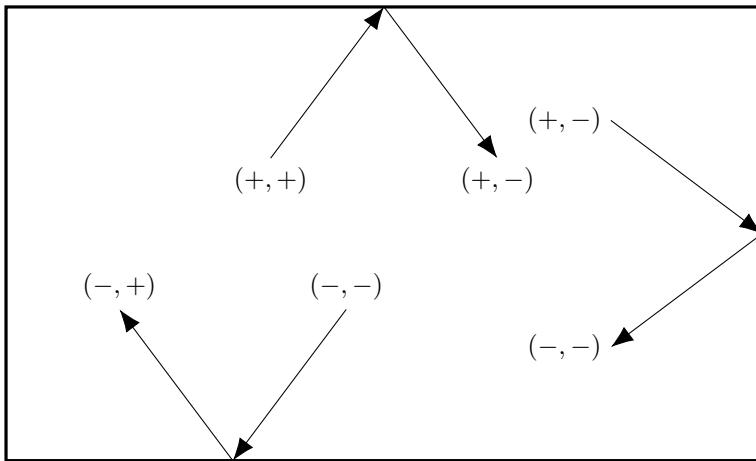
# File: arcade_bouncing_ball.py
class MyGame(arcade.Window):

    def __init__(self, width, height, title):
        super().__init__(width, height, title)

        self._ball = Ball('ball.png', 0.2)

        self._ball.center_x = SCREEN_WIDTH / 2
        self._ball.center_y = SCREEN_HEIGHT / 2

```



ပုံ ၁၀.၃ `change_x` နှင့် `change_y` လက္ခဏာ ပြောင်းလဲပုံ

```

    self._ball.change_x = 2
    self._ball.change_y = 1
    arcade.set_background_color(arcade.color.AIR_FORCE_BLUE)

    def on_draw(self):
        self.clear()
        self._ball.draw()

    def on_update(self, delta_time):
        self._ball.update()
  
```

```

window = MyGame(SCREEN_WIDTH, SCREEN_HEIGHT, "Bouncing Ball")
arcade.run()
  
```

Sprite အော်ကျက် ဖန်တီးတဲ့အခါ ရုပ်ပုံဖိုင် နံမည်နဲ့ ချုံ/ခဲ့ တန်ဖိုး (scale factor) ထည့်ပေး ရပါတယ်။ နှင့် ball.png ဖိုင် ဆယ်ပုံနှစ်ပုံအရွယ်အစားနဲ့ ကောလုံးပုံ Sprite ကို အခုလို

```

    self._ball = Ball('ball.png', 0.2)
  
```

ဖန်တီးတယ်။ ရုပ်ပုံဖိုင်က ကုဒ်ဖိုင်နဲ့ ဖို့အတွက်ခုထဲမှာ ရှိရပါမယ်။ ဒါက အလွယ်နည်းကို ပြောတာပါ။ အခြား ဖို့ဒါမှာ ထားလိုရပေမဲ့ နည်းနည်း ပိုရှုပ်လို့။

Sprite ရွှေလျားတာနဲ့ collision ဖြစ်တဲ့ ကိစ္စတွေအတွက် အဓိက ဂိမ်းလော့ဂျွစ်ကို on_update မက်သဒ်မှာ ရေးပေးရမယ်။ ဒါလည်းပဲ တကောက် ဆက်ခံထားတဲ့ on_update ကို override လုပ်တာပါ။ ဒီမက်သဒ်ကိုလည်း တစ်စက်နဲ့ အကြိမ်ခြောက်ဆယ် အလိုအလျောက် ခေါ်ပေးပါတယ်။ ဒါပေမဲ့ on_draw နဲ့က ရည်ရွယ်ချက်ချင်း မတော်ဘူး။ on_draw က စာရင်ကိုပဲ refresh လုပ်ပေးတာ။ တစ်နည်းအားဖြင့် window ပေါ်မှာ ဂရပ်ဖစ်ပုံပဲ ဖော်ပေးတာ။ ဂိမ်းလော့ဂျွစ်နဲ့ သက်ဆိုင်တာတော့ မလုပ်ဘူး။ on_update ကျတော့ ပုံဆွဲတဲ့ကိစ္စကို မလုပ်ဘူး။ ဂိမ်းလော့ဂျွစ်သို့သနဲ့ လုပ်ဆောင်တယ်။ ဒီမက်သဒ်နှစ်ခု လုပ်ဆောင်ပေးတဲ့ တာဝန်ကို ခွဲခြားထားရပါမယ်။ ဂိမ်းမှာပါဝင်တဲ့ Sprite အားလုံးရဲ့ state ကို update လုပ်ပေး

ခြင်းဟာ `on_update` ရဲ့ အဓိကတာဝန်တွေထဲက တစ်ခုအပါအဝင် ဖြစ်တယ်။ အချုပ်မာမှာ ဘောလုံးရဲ့ state ကို

```
| self._ball.update()
```

ခေါ်ပြီး `update` လုပ်ပေးထားတယ်။

`on_update` မက်သဒ် `delta_time` ပါရာမီတာကို ရှင်းပြန့် ကျန်ပါသေးတယ်။ `on_update` ကို ခေါ်တဲ့အခါ နောက်ဆုံးခုခဲ့တဲ့ အချိန်နဲ့ အချုပ်ချိန် အချိန်ကြား ကွဲဟန်ကို `delta_time` အနောက် ထည့်ပေးတယ်။ Arcade လိုက်ဘရဲ့ နောက်ကွဲယ်က အလုပ်လုပ်ပုံ သဘောတရားအရ ထည့်ထားတဲ့ ပဲရာမီတာဖြစ်ပြီး လိုက်ဘရဲ့ သုံးတဲ့သူအနေနဲ့ အသေးစိတ်သိမ့် မလိုအပ်ပါဘူး။ `on_update` ကို `override` လုပ်တဲ့အခါ `delta_time` ပါရာမီတာ မကျွန်ခဲ့ရင် ရပါပြီ။

Velocity, `change_x` နှင့် `change_y` ဆက်စပ်ပုံ



ပုံ ၁၀၉

မူလ p_1 အမှတ်ကနေ ညာဘက်ကို Δx , အထက်ကို Δy (နှစ်ခုလုံး အပေါင်းတန်ဖိုး) ရွှေရင် p_2 အမှတ်ကို ရောက်သွားမှာပါ (ဘယ် ပုံ)။ မူလ p_1 အမှတ်ကနေ ညာဘက်ကို Δx (အပေါင်း), အောက် တည့်တည့်ကို Δy (အနှုတ်) ရွှေရင် p_2 အမှတ်ကို ရောက်သွားမှာပါ (ညာ ပုံ)။ Sprite မှာ `change_x`, `change_y` ဟာ $\Delta x, \Delta y$ ပါပဲ။ p_1 က (x_1, y_1) , p_2 က (x_2, y_2) ဖြစ်တယ်ဆိုရင်

$$x_2 = x_1 + \Delta x$$

$$y_2 = y_1 + \Delta y$$

Frames per second (တစ်စက်နှစ် `update/refresh` လုပ်တဲ့ အကြိမ်) နဲ့ velocity ဆက်စပ် နေမယ်ဆိုတာ သိသာပါတယ်။ $\Delta x = 2, \Delta y = 1$ ဆိုပါစို့

$$\begin{aligned} \Delta s &= \sqrt{\Delta x^2 + \Delta y^2} \\ &= \sqrt{2^2 + 1^2} \\ &= \sqrt{5} \\ &\approx 2.24 \end{aligned}$$

တစ်စက်နှစ် အကြိမ်ပြောက်ဆယ် `update` လုပ်မယ်ဆိုရင် တစ်စက်နှစ်အတွင်းမှာ စုစုပေါင်း အကွာအဝေး $60 \times 2.24 = 134.4$ pixels ရွှေမှာပါ။ မျဉ်းပြောင့်အတိုင်း ရွှေမှာ။ ဦးတည်ရာက $\Delta x, \Delta y$ အပေါင်း/အနှုတ် ပေါ် မူတည်ပါမယ်။

Event Handling

မောက်စ်၊ ကီးဘုဒ်၊ joystick စသုံး input device တွေနဲ့ ဂိမ်းကို တိန်းချုပ်ဖို့အတွက်လည်း Windows က လပ်စ်က လုပ်အောင်လုပ်ထားတယ်။ `on_mouse_motion`, `on_mouse_press`, `on_key_press` စ တဲ့ မက်သဒ်တွေကို override လုပ်ရှုပါပဲ။ အဖြစ်အပျက် တစ်စုံတစ်ခု (event) ကို ပရိုကရမ်က တုပြန် လုပ်ဆောင်ပေးတာကို *event handling* လို့ ခေါ်တယ်။ ကီးဘုဒ် ကိုနှိပ်/လွတ် လိုက်တာ၊ မောက်စ် ကလစ်နှုပ်/လွတ် လိုက်တာ၊ မောက်စ်ရွှေတာ စတာတွေဟာ event ဥပမာတချို့ ဖြစ်တယ်။ ဒါတွေကို ပရိုကရမ်က တုပြန်လုပ်ဆောင်ချင်တဲ့ အခါ event handling ကို သုံးရပါတယ်။

ဒါက ဘေးလုံးကို မောက်စ်နဲ့ ရွှေတဲ့ နမူနာပါ။ မောက်စ်ကို နှိပ်ထားပြီး ရွှေရတာ (dragging) မဟုတ်ဘူး။ ဒီတိုင်း ပိုင်တာရွှေတဲ့ နောက်ကို ဘေးလုံးက လိုက်နေမှာပါ။

```
# File: arcade_m_move.py
class MyGame(arcade.Window):

    def __init__(self, width, height, title):
        super().__init__(width, height, title)
        self._ball = arcade.Sprite('ball.png', 0.2)
        self._ball.center_x = SCREEN_WIDTH / 2
        self._ball.center_y = SCREEN_HEIGHT / 2
        arcade.set_background_color(arcade.color.AIR_FORCE_BLUE)

    def on_draw(self):
        self.clear()
        self._ball.draw()

    def on_mouse_motion(self, x: int, y: int, dx: int, dy: int):
        self._ball.center_x = x
        self._ball.center_y = y
```

မောက်စ်ကို ရွှေရင် မောက်စ်ပိုင်တာရွှေနေသ၍ `on_mouse_motion` ကို တစ်ခါပြီးတစ်ခါ အဆက်မပြတ် ခေါ်နေမှာပါ။ မောက်စ် ပုံပိုင်ရင် ဒီမက်သဒ်ခေါ်တာလည်း ရပ်သွားမယ်။ ဒီလိုဖြစ်အောင် Windows က လုပ်ပေးထားတာ။ x နဲ့ y က လက်ရှိ မောက်စ်ပိုင်တာရဲ့ တည်နေရာ။ dx က ဒီမက်သဒ်ကို နောက်ဆုံး ခေါ်ခဲ့တဲ့ အချိန်နဲ့ အချိန်အတွင်း ပြောင်းလဲသွားတဲ့ x ကွာဟာချက်။ dy က ဒီမက်သဒ်ကို နောက်ဆုံး ခေါ်ခဲ့တဲ့ အချိန်နဲ့ အချိန်အတွင်း ပြောင်းလဲသွားတဲ့ y ကွာဟာချက်။ နောက်ဆုံး ခေါ်ခဲ့တဲ့ မောက်စ်က (x_1, y_1) မှာ ရှိခဲ့တယ်၊ အခုခေါ်ခဲ့အချိန် (x_2, y_2) မှာ ဆိုပါစို့။ $dx = x_2 - x_1$, $dy = y_2 - y_1$ ဖြစ်တယ်။

မောက်စ် ကလစ်နှုပ်တဲ့ အခါ တစ်ခုခု လုပ်မယ်ဆိုရင် `on_mouse_press` ကို override လုပ်ရပါ မယ်။ ဒီမက်သဒ်ကိုတော့ ကလစ်နှုပ်တော့မှုပဲ ခေါ်မှာပါ။ မောက်စ် ဘယ်ဘက် ကလစ်နှုပ်လိုက်တဲ့ နေရာကို ဘေးလုံး (ချက်ချင်း) ရောက်စေချင်ရင် အခုလုံး ...

```
# File: arcade_m_press.py
def on_mouse_press(self, x, y, button, modifiers):
    if button == arcade.MOUSE_BUTTON_LEFT:
        self._ball.center_x = x
        self._ball.center_y = y
```

ကိုးဘုဒ္ဒနဲ့ ထိန်းချင်ရင် `on_key_press` ကို override လုပ်ရပါမယ်။ `arcade_k_press.py` မှာ လေ့လာကြရှုပါ။ မောက်စ် နိုင်ထားပြီး ဘေးလုံးကိုရွှေ့ (dragging) တာကို `arcade_m_drag.py` ဖိုင် မှာ ကြည့်နိုင်ပါတယ်။

arcade.View

`arcade.View` ဟာ `Window` နဲ့ သဘောတရား အတော်လေးဆင်တူပါတယ်။ `Window` လိုပဲ စခရင် မှာ ဂရပ်ဖစ်ပုံဖော်ဖို့ `View` ကို သုံးလို့ရတယ်။ Event handling အတွက် override လုပ်ရတာတွေ ကလည်း `Window` နဲ့ တူတူပါပဲ။ ဒါပေမဲ့ `View` က သူချဉ်း မရှုတည်နိုင်ပါဘူး။ `View` ကို ပြေားစီး အတွက် `Window` တစ်ခုလိုပါတယ်။ `Window` တစ်ခုတည်းဟာ `View` အမျိုးမျိုးကို အလှည့်ကျ လဲပြီးပဲ လို့ရတယ်။ ဂိမ်းတစ်ခုမှာ welcome screen, game over screen, pause screen စသည်ဖြစ် ပြေား စိုလိုပါတယ်။ ဒီလို လိုအပ်ချက်မျိုးအတွက်ဆိုရင် Arcade မှာ `View` ကို အသုံးပြုရမှာပါ။ နမူနာကြည် ရင် ပိုရှင်းသွားပါလိမ့်မယ်။

အောက်ပါဥပမာက ပထမ စစချင်း Click to Start! လို့ ပြနေမှာပါ။ ကလစ်နိုင်လိုက်ရင် ဘေးလုံး က စရွေပါမယ်။ ကလစ်ထပ်နိုင်လိုက်ရင် ဘေးလုံးရွှေ့နေတာ ပျောက်သွားပြီး The End စသားပြပါ တယ်။ `MsgView` က စသားပြေားစီးအတွက်။ `MainView` က ဘေးလုံး အနုနီမေးရှင်းအတွက်။ နှစ်ခုလုံး `arcade.Window` အတား `arcade.View` ကို inherit လုပ်ထားတာ သတိပြုပါ။

```
# File: arcade_view_switch.py
import arcade
from arcade.color import *

WIN_WIDTH = 400
WIN_HEIGHT = 600

class MainView(arcade.View):
    def __init__(self):
        super().__init__()
        self._circle_x = WIN_WIDTH // 2    # move 2 pixels to the right
        self._circle_y = WIN_HEIGHT // 2   # move 1 pixels upwards
        self._next_view = None

    def on_draw(self):
        self.clear()
        self._circle_x += 1.3  # move 2 pixels to the right
        self._circle_y += 2    # move 1 pixels upwards
        arcade.draw_circle_filled(self._circle_x,
                                  self._circle_y,
                                  20,
                                  BLUE)

    def on_mouse_press(self, _x, _y, _button, _modifiers):
        """ If the user presses the mouse button, start the game. """
        self.window.show_view(self._next_view)
```

```

class MsgView(arcade.View):
    def __init__(self, msg):
        super().__init__()
        self._msg = msg
        self._next_view = None

    def on_draw(self):
        self.clear()
        arcade.draw_text(self._msg,
                         WIN_WIDTH / 2,
                         WIN_HEIGHT / 2,
                         RED,
                         font_size=20,
                         anchor_x="center")

    def on_mouse_press(self, _x, _y, _button, _modifiers):
        if self._next_view:
            self.window.show_view(self._next_view)

def main():
    window = arcade.Window(WIN_WIDTH, WIN_HEIGHT, "View Switch")
    end_view = MsgView("The End")
    start_view = MsgView("Click to Start!")
    main_view = MainView()

    start_view._next_view = main_view
    main_view._next_view = end_view
    window.show_view(start_view)
    arcade.run()

if __name__ == "__main__":
    main()

```

စော့စောက်ပြောခဲ့သလို View ကို ပြနိုင် Window ရှိရပါမယ်။ Arcade ပရိုဂရမ်တစ်ခု run တဲ့အခါ Window တစ်ခုကတော့ ရှိကိုရှိရမှုပါ။ အဲဒီ Window ကို View က self.window နဲ့ ရည်ညွှန်းအသံးပြု နိုင်တယ် (View ကို ဆက်ခံထားတဲ့ subclass မှာလည်း သုံးလို့ရတယ်)။ Window မှာ ပြချင်တဲ့ View ကို window.set_view မက်သွန့် သတ်မှတ်ရတယ်။ MsgView နဲ့ MainView မှာ on_mouse_press ကို ဒီလို့ override လုပ်ထားတယ်

```

def on_mouse_press(self, x, y, button, modifiers):
    if self._next_view:
        self.window.show_view(self._next_view)

```

ကလစ်နိုပ်ရင် self._next_view ကို ပြောင်းပေးမှာပါ။

နောက်တစ်ခုက View အကျယ်နဲ့ အမြင့်ဟာ ငြင်းကိုပြပေးတဲ့ Window အပေါ် မှတည်တယ်။ ဒါ ကြောင့် View တစ်ခုချင်းအတွက် အကျယ် အမြင့် မသတ်မှတ်ဘူး။ Window အကျယ်နဲ့ အမြင့် သတ်မှတ်ရင် ရပြီ။

main မက်သဒ်မှာ Window တစ်ခု နဲ့ View သုံးခု ဖန်တီးထားတယ်။ View တစ်ခုကနေ တစ်ခု ပြောင်းလဲဖို့ အခုလို ချိတ်ဆက်ပေးထားတာ

```
start_view._next_view = main_view
main_view._next_view = end_view
```

တွေ့ရမှာပါ။ start_view ပေါ်မှာ ကလစ်နိုပ်ရင် main_view, main_view ပေါ်မှာ နိုပ်ရင် end_view ကို ပြပေးမှာပါ (on_key_press မက်သဒ်နဲ့ ဆက်စပ်ကြည့်ပြီး နားလည်အောင်လုပ်ပါ)။

၁၀.၄ Breakout တည်ဆောက်ခြင်း

ဂိမ်း မတည်ဆောက်ခင် ကြိုတင်နားလည်ထားရမဲ့ Arcade သဘောတရား အတော်များများ ရှင်းပြဲခဲ့ပြီ။ တကယ် လက်တွေ ဂိမ်းရေးဖို့ပဲ ကျွန်ုပ်ပါတွေတယ်။ ပရိုဂရမ် အပြည့်အစုံကို စာမျက်နှာ (၂၁၉) မှာ ကြည့်ပါ။ အခု တစ်စိုင်းချင်း ခွဲထုတ် ရှင်းပြပါမယ်။ ဂိမ်းအတွက် လိုအပ်တဲ့ constant တွေကို ပထမဆုံး သတ်မှတ်ထားတယ်။ အများစုံက တည်နေရာ၊ အရွယ်အစားနဲ့ သက်ဆိုင်တာတွေ။

```
WIN_WIDTH = 400
WIN_HEIGHT = 600

# paddle size
PDL_WIDTH = 60
PDL_HEIGHT = 10
PDL_Y_OFFSET = 30      # distance between paddle and lower edge of window

BALL_RADIUS = 10
BRICKS_PER_ROW = 10
BRICK_ROWS = 10        # number of brick rows
BRICK_GAP = 4           # gap between bricks

# calculate and define constants for brick width and height
BRICK_WIDTH = ((WIN_WIDTH - (BRICKS_PER_ROW - 1) * BRICK_GAP)
                // BRICKS_PER_ROW)
LEFT_MARGIN = (WIN_WIDTH - (BRICK_WIDTH * BRICKS_PER_ROW +
                            BRICK_GAP * (BRICKS_PER_ROW - 1))) // 2
BRICK_HEIGHT = 8

# Window အောက်ဘက် ဘောင်နဲ့ နံရံအောက်မြေ အကွာအင်း
BRICK_Y_OFFSET = 414
# Window ပတ်ဝန်
CENTER_X = WIN_WIDTH // 2
CENTER_Y = WIN_HEIGHT // 2
```

အုတ်ခဲခီခြင်း

setup_bricks က အုတ်ခဲတွေ နေရာတကျ စီပေးတဲ့ ဖန်ရှင်။ အုတ်ခဲကို SpriteSolidColor နဲ့ ဆဲ လိုရတယ်။ ဒီကလပ်စ်က အရောင်အပြည့် ထောင့်မှန်စတုဂံပဲ စီတော်လိုပဲ။ image ဖိုင် ရှိဖို့ မလိုဘူး။ စီထားတဲ့ အုတ်ခဲအားလုံးကို SpriteList နဲ့ သိမ်းထားပြီး return ပြန်ပေးထားတယ်။

```
def setup_bricks():
    colors = [CYAN, CYAN, GREEN, GREEN, GOLD, GOLD,
              ORANGE, ORANGE, RED, RED]
    # အုတ်ခဲအားလုံး ထည့်ထားဖို့ SpriteList
    brick_lst = arcade.SpriteList()
    y = BRICK_Y_OFFSET + BRICK_HEIGHT // 2
    # 10 x 10 bricks wall
    for i in range(BRICKS_PER_ROW):
        x = LEFT_MARGIN + BRICK_WIDTH // 2
        for j in range(BRICK_ROWS):
            brick = arcade.SpriteSolidColor(BRICK_WIDTH,
                                              BRICK_HEIGHT,
                                              colors[i])
            brick.center_x = x
            brick.center_y = y
            # အုတ်ခဲတော်ချင်း SpriteList ထည့်
            brick_lst.append(brick)
            x += (BRICK_WIDTH + BRICK_GAP)
        y += (BRICK_HEIGHT + BRICK_GAP)
    return brick_lst
```

SpriteList ထဲမှာ Sprite တွေ တစ်စုတစ်စည်းတည်း သိမ်းထားတာဟာ ဘောလုံးနဲ့ ဝင်တိုက်မိတဲ့ အုတ်ခဲတွေ (တစ်ခုထက်ပိုနိုင်တယ်) ကို စစ်ထုတ်ဖို့ လွယ်ကူစေတယ်ဆိုတာ ခကေနေ တွေ့ရမှာပါ။ 『တည်နေရာ အတွက်အချက် အသေးစိတ်နားလည်ချင်ရင် စာမျက်နှာ (၁၀၈) မှ ကျားကွဲက်ခံ ဥပမာကို ကြည့်ပါ။』

ဘောလုံးနှင့် တာထွက် velocity

ဘောလုံးအတွက် Ball ကလပ်စိုး အခဲလို သတ်မှတ်ပါမယ်။ ဘောင်တွေကို တိုက်တဲ့အခါ ပြန်ကန်ထွက်အောင် လုပ်တဲ့နည်းလမ်းက ရှေ့မှာတွေ့ခဲတဲ့ ဥပမာကလိုပါပဲ။ self.left < 0 ဖြစ်ရင် ဘယ်ဘက်ဘောင်နဲ့ တိုက်တာ၊ self.right < WIN_WIDTH ခါးရင် ညာဘက်နဲ့တိုက်တာ စသည်ဖြင့်ပေါ့။ အထက်/အောက် ဘောင်နဲ့တိုက်တာလည်း ဒီသဘောတရားပါပဲ။

```
class Ball(arcade.SpriteCircle):
    def update(self):
        self.center_y += self.change_y
        self.center_x += self.change_x

        if self.left < 0:
            self.change_x *= -1
```

```

if self.right > WIN_WIDTH:
    self.change_x *= -1

# hitting bottom edge doesn't bounce
if self.bottom < 0:
    pass

if self.top > WIN_HEIGHT:
    self.change_y *= -1

```

ဒီကလပ်စ်က arcade.SpriteCircle ကို inherit လုပ်ထားတယ်။ SpriteCircle က အရောင်ဖြည့် စက်ပိုင်းပဲ Sprite အတွက်။ image ဖိုင် မလိုဘူး။

setup_ball ကတော့ ဘေးလုံးရဲ့ ကန်ပိုးတည်နေရာနဲ့ တာထွက် velocity ကို အဓိက တွက်ချက်သတ်မှတ်ပေးတယ်။ ဖန်တီးထားတဲ့ ဘေးလုံးကိုလည်း return ပြန်ပေးတယ်။

```

def setup_ball():
    ball = Ball(BALL_RADIUS, arcade.color.BLACK)
    ball.center_x = WIN_WIDTH // 2
    ball.center_y = WIN_HEIGHT // 2
    ball.change_y = -6
    random.uniform(1.0, 3.0)
    ball.change_x = random.uniform(1.0, 3.0) \
        if random.uniform(0.0, 1.0) <= 0.5 \
        else random.uniform(-1.0, -3.0)
    return ball

```

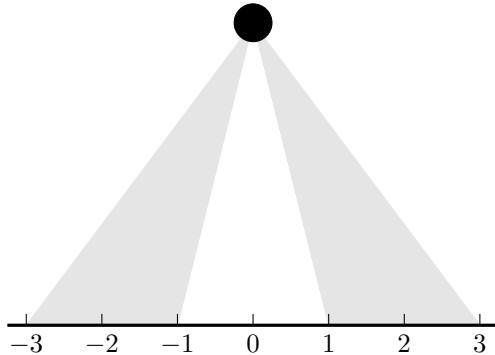
အစွဲ ဘေးအောက်ကို ကျလာတဲ့အခါ ဦးတည်ရာက အောက်တည့်တည့်ကိုပဲ အမြဲတစ်သမှတ်တည်းဖြစ်နေရင် သိပ်မကောင်းဘူး။ ဤော်မြောင်းရှုံးရေဖြစ်နေမယ်။ ဘယ်ဘက်ကို ဦးတည် ဆင်းလာမှာလဲ ခန့်မှန်းလို့မရရင် ပိုမိုက်တယ်။ စိတ်လှပ်ရားဖို့ ပိုကောင်းတယ်။ ဒီအတွက် change_x ကို 1.0 ကနေ 3.0 အတွင်းနဲ့ -1.0 ကနေ -3.0 အတွင်း random ထုတ်ထားတယ်။ random.uniform ဖန်ရှင် သုံးပါတယ်။ change_x အနှစ်တန်ဖိုးဆိုရင် ဘယ်ဘက် အပေါင်းဆိုရင် ညာဘက်ကို ဦးတည်တယ်။ ဘယ်ဘက်လားလှာ ညာဘက်လားကိုလည်း ငါးဆယ် ငါးဆယ် ဖြစ်ချင်တယ်။ ဒါကြောင့်

```
random.uniform(0.0, 1.0) <= 0.5
```

ဖြစ်ရင် အပေါင်းတန်ဖိုး ဖြစ်အောင် random.uniform(1.0, 3.0) နဲ့ ထုတ်ပေးတယ်။ မဟုတ်ရင်တော့ အနှစ်တန်ဖိုး ထွေကောင်လုပ်ထားတယ်။ ဒါကြောင့် စစ်ချင်းမှာ ဘေးလုံးကျလာတဲ့ လားရာက ပဲ (၁၀.၅) မှာ ပြထားတဲ့ နယ်ပယ်အတွင်းမှာပဲ ရှိပါမယ်။

Breakout Class

Breakout ကလပ်စ်ကို ဆက်ကြည့်ရအောင်။ ဂိမ်းရဲ့ အဓိက View ဖြစ်ပြီး အရှုံးအခိုင်ဆုံးဖြတ်တာ ဘေးလုံးနဲ့ အုတ်ခဲ့၊ ဘေးလုံးနဲ့ paddle တိုက်မိတာ စတာတွေကို ဒီကလပ်စ်မှာ ရေးထားတာပါ။ paddle ကို မောက်စံနဲ့ ရွှေလှ့ရအောင် on_mouse_motion ကလည်း ဒီထဲမှာပဲ။ 〔Breakout ကလပ်စ် အပြည့်အစုံကို စာမျက်နှာ (၂၁၉) မှာ ကြည့်ပါ။〕



ပုံ ၁၀.၅ ဘေးလုံးကျလာနိုင်တဲ့နေရာ

`__init__` မက်သဒ်က သိပ်ရှုပ်ရှုပ်ထွေးထွေး မရှိဘူး။ မောက်စိုင်တာ ပေါ်နေအောင် `self.window.set_mouse_visible(True)` နဲ့ လုပ်ထားတယ်။ ဖျောက်ထားချင်ရင် `False` ထည့်ပေး။ Instance variable တွေ အားလုံးကို `None` ထည့်ထားတယ်။

```
def __init__(self):
    super().__init__()

    arcade.set_background_color(WHITE)
    self.window.set_mouse_visible(True)
    self._ball = None
    self._paddle = None
    self._brick_lst = None
    self._brick_remains = None
    self._next_view = None
```

Initialization ကို `setup` မက်သဒ်မှာ အဓိက လုပ်ထားတယ်။ `__init__` မှာ ဘာလို့ မလုပ်လဲ မေးစရာရှိတယ်။ `__init__` မက်သဒ်က အော့သ်ဂျက် ဖန်တီးတဲ့အခါမှာပဲ အလိုအလျောက်ခေါ်ပေးတာ။ တိုက်ရှိက်ခေါ်ရတဲ့ မက်သဒ်မဟုတ်ဘူး။ ရုံးသွားလို့ (သို့) နိုင်သွားလို့ နောက်တစ်ခါ ပြန်ဆော့ချင်ရင် အစအနေအထား ဖြစ်အောင် `setup` မက်သဒ်ကို ခေါ်လို့ရမယ်။ (ဒီတော့ Breakout အော့သ်ဂျက် အသစ်တစ်ခု ဖန်တီးတာ၊ `__init__` ကို တိုက်ရှိက်လည်း မခေါ်ဘဲ ပြန်စချင်ရင် `setup` ကို ခေါ်လို့ရမယ်)။ ရော့မာတွေ့ခဲ့တဲ့ `setup_ball`, `setup_paddle`, `setup_bricks` တို့ကို တစ်ဆင့် ပြန်ခေါ်ထားတာ တွေ့ရမယ်။ `_brick_remains` က လက်ကျွန်းမှတ်တဲ့ အရေအတွက် မှတ်ထားဖို့ အုတ်ခဲ တစ်လုံးပျက်သွားတိုင်း တစ်လျှော့ပေးရမယ်။

```
def setup(self):
    self._paddle = arcade.SpriteSolidColor(PDL_WIDTH,
                                            PDL_HEIGHT,
                                            BLACK)
    self._paddle.bottom = PDL_Y_OFFSET
    self._paddle.center_x = CENTER_X

    self._ball = setup_ball()
    self._brick_lst = setup_bricks()
```

```
    self._brick_remains = 20
```

draw မက်သွက ဂိမ်းမှုပါတဲ့ အုတ်နံရုံ ဘေးလုံးနဲ့ paddle ပြားတိုကို ဆွဲတယ်။ ဒီမက်သွက Breakout ကလပ်စွာ ထပ်ဖြည့်ထားတာ။ View ကလပ်စွဲ တစ်စက်နှုံး အကြိမ်ခြောက်ဆယ် ခေါ်နေ့ on_draw ကို override လုပ်တာ မဟုတ်ဘူး။ draw ကို ခွဲထုတ်ထားရတာက လိုတဲ့အချိန်မှာ ကိုယ်တိုင် ခေါ်လိုရအောင် ရည်ရွယ်တာပါ။

```
def on_draw(self):
    self.draw()

def draw(self):
    self.clear()
    self._paddle.draw()
    self._brick_lst.draw()
    self._ball.draw()
```

ဂိမ်းရဲ့ အဓိကလေ့ဂျွဲကို on_update မက်သွကမှာ တွေ့ရမှုပါ။ View ကလပ်စွဲ on_update ကို override လုပ်ပေးတာပါ။

```
def on_update(self, delta_time):
    self._ball.update()
    self._paddle.update()
    self._brick_lst.update()
    bricks_hit = arcade.check_for_collision_with_list(self._ball,
                                                       self._brick_lst)

    if len(bricks_hit) > 0:
        self._ball.change_y *= -1
    for brick in bricks_hit:
        brick.remove_from_sprite_lists()
        self._brick_remains -= 1

    if self._brick_remains == 0:
        self._next_view._msg = "You Win!"
        self.window.show_view(self._next_view)

    if self._ball.center_y <= PDL_Y_OFFSET:
        self._next_view._msg = "You Lost!"
        self.window.show_view(self._next_view)

    if (arcade.check_for_collision(self._ball, self._paddle)
        and self._ball.change_y < 0):
        self._ball.change_y *= -1
```

on_update ကို တစ်စက်နှုံး အကြိမ်ခြောက်ဆယ် အဆက်မပြတ် အလိုအလျောက် ခေါ်ပေးတယ်လို့ ပြော ခဲ့တာ ပြန်အမှတ်ရမယ် ထင်ပါတယ်။ ဂိမ်းရဲ့ state ကို ဒီမက်သွက တစ်ကြိမ်ခေါ်တိုင်း update လုပ်ပေး ရပါမယ်။ ဒါဟာ ဂိမ်းတစ်ခုရဲ့ အချိန်နဲ့အမျှ ပြောင်းလဲနေတဲ့ အရာအားလုံးအတွက် အဓိကသော့ချက်ပဲ။

၂၃

ပထမဆုံး Breakout မှာပါတဲ့ ဘေးလုံး၊ paddle နဲ့ အုတ်ခဲအားလုံးရဲ့ လက်ရှိအခြေအနေကို update လုပ်ရမယ်။ ဒီအတွက် ဂိုမ်းမှာပါဝင်တဲ့ သက်ဆိုင်ရာ Sprite (သို့) SpriteList အားလုံးရဲ့ update မက်သို့ကို ခေါ်ပေးရမှာပါ။

```
self._ball.update()  
self._paddle.update()  
self._brick_lst.update()
```

check_for_collision_with_list က ဝင်တိုက်မိတဲ့ Sprite တွေကို စစ်ထုတ်ပေးတဲ့ မက သို့။ ဘေးလုံးနဲ့ တိုက်မိတဲ့ အုတ်ခဲတွေကို လိုချင်တာ။ ဒီတော့ အခုလို ခေါ်ရမယ်

```
bricks_hit = arcade.check_for_collision_with_list(self._ball,  
                                                 self._brick_lst)
```

အုတ်ခဲ (တော့) နဲ့ ဝင်တိုက်ရင် ဘေးလုံးကို အထက် (သို့) အောက် ပြန်ကန်ထွက်အောင် change_y ကို လက္ခဏာ ဆန့်ကျင်ဘက် ပြောင်းပေးပါတယ်။

```
if len(bricks_hit) > 0:  
    self._ball.change_y *= -1
```

ဝင်တိုက်တဲ့ အုတ်ခဲတွေကို ဖျက်ပစ်ရပါမယ်။ ဖျက်လိုက်ရင် လက်ကျွန်းအုတ်ခဲလည်း လျှော့သွားရမယ်။ ဒီ အတွက် အခုလို

```
for brick in bricks_hit:  
    brick.remove_from_sprite_lists()  
    self._brick_remains -= 1
```

တိုက်မိတဲ့ အုတ်ခဲတစ်ခဲချင်း ဖော်ထုတ်ပါတယ်။

Start, Main and End Views

ပထမ စစ်ဆေးမှာ ပုံ (၁၀.၁) မှာ တွေ့ရတဲ့ အနေအထားအတိုင်း ရှိနေမယ်။ ဒါပေမဲ့ စတာနဲ့ ဘေးလုံးက ချက်ချင်းထွက်ရင် ဆော့ရတာ သိပ်အဆင်မပြောဘူး။ ကလစ်နှင့်လိုက်မှ ဘေးလုံးစထွက်မယ်ဆိုရင် ပို့ကောင်းမယ်။ StartView ကို အခုလိုသတ်မှတ်ထားတယ်

```
class StartView(arcade.View):  
    def __init__(self):  
        super().__init__()  
        self._next_view = None  
  
    def on_draw(self):  
        self.clear()  
        # Breakout ရဲ့ setup နဲ့ draw ကိုခေါ်တာ  
        self._next_view.setup()  
        self._next_view.draw()  
  
    def on_mouse_press(self, x, y, button, modifiers):  
        # ကလစ်နှင့် Breakout View ကို ပြောင်းပေးတာ
```

```

if self._next_view:
    self._next_view.setup()
    self.window.show_view(self._next_view)

```

တေတေကရှင်းပြဲတဲ့ Breakout ဟာ ဂိမ်းရဲ့ အဓိက View ။ ဒါပေမဲ့ ဒီ View က စတာနဲ့ တောလုံးက ရွှေမှာ on_draw နဲ့ on_update က ရပ်ထားလို့မရဘူး။ View ကို Window မှာ ပြပြီဆိတာနဲ့ တောက်လျောက် ခေါ်နေမှာ။ StartView က ရပ်ပုံကို အပြိုမ်ပဲ ပြပေးရမယ်။ ဖြစ်နိုင်တဲ့ နည်းလမ်းတစ်ခုက Breakout ရဲ့ setup နဲ့ draw ကို StartView ရဲ့ on_draw ကနေ ခေါ်လို့ရပါတယ်။ StartView နဲ့ Breakout ကို main မက်သင်ထဲမှာ အခုလို ချိတ်ပေးထားပါမယ်။

```

start_view = StartView()
game_view = Breakout()
# ...
start_view._next_view = game_view
window.show_view(start_view)
# ...

```

အခြားအနှင့် You Win!/You Lose! ပြန့် EndView ရဲ့ တာဝန်။ ရုံး (သို့) နိုင်ရင် မက်ဆွဲချုပ်ပြပေးပြီး တစ်ခါတပ်ဆော့ချင်ရင် ကလစ်နှုပ်ရပါမယ်။

```

class EndView(arcade.View):
    def __init__(self):
        super().__init__()
        self._next_view = None
        self._msg = None

    def on_draw(self):
        self.clear()
        arcade.draw_text(self._msg,
                         WIN_WIDTH / 2,
                         WIN_HEIGHT / 2,
                         RED,
                         font_size=20,
                         anchor_x="center")

    def on_mouse_press(self, x, y, button, modifiers):
        if self._next_view:
            self.window.show_view(self._next_view)

```

main မက်သင်ထဲမှာ အခုလို ချိတ်ပေးထားပါတယ်

```

end_view = EndView()
# ...
end_view._next_view = start_view

```

၃၀။

ဒီအခန်း၏ Breakout ဂိုးကို အသုံးချ ဥပမာအနေနဲ့ ထည့်ပေးထားတဲ့ ရည်ရွယ်ချက်က Arcade လိုက်ဘရိနဲ့ ဂိမ်းတွေ ဖန်တီးခိုင်ဖို့ အဓိက မဟုတ်ပါဘူး။ Inheritance ကို လိုက်ဘရိတွေမှာ အသုံးချပေါ်ရှိတဲ့ ပုံစံတရှိကို နားလည်သဘောပေါက်အောင်၊ သတိပြုမိအောင်၊ ဆက်စပ်မိအောင် အဓိက ရည်ရွယ်တာပါ။ စလေ့လာသူတွေအတွက် လွယ်လွယ်နဲ့ နားလည်နိုင်မယ်လို့တော့ မမျှော်လင့်နိုင်ဘူး။ စိတ်ရည်ရည်ထားအချိန်ပေးပြီး နားလည်သဘောပေါက်အောင် လေ့လာဖို့ လိုပါလိမ့်မယ်။ ပရိုကရမ်အပြည့်အစုံကို အောက်မှာဖော်ပြပေးထားပါတယ်။

တည်နေရာ အတွက်အချက်တွေ နားမလည်လို့လည်း သချာကြောက်သူတွေ စိတ်ခါတ်ကျစရာ မလိုပါဘူး။ အတွက်အချက်တွေ အကြမ်းဖျဉ်းလောက် နားလည်အောင် ကြည့်၊ ကျန်တဲ့ ပရိုကရမ်းမင်းနဲ့ဆိုင်တဲ့ သဘောတရားတွေ အဓိကထားကြည့်မယ်ဆိုရင်လည်း အတိုင်းအတာတစ်ခုထိ အကျိုးရှိမှာပါပဲ။

```
import random
import arcade
from arcade.color import *

WIN_WIDTH = 400
WIN_HEIGHT = 600

PDL_WIDTH = 60
PDL_HEIGHT = 10
PDL_Y_OFFSET = 30      # distance between paddle and lower edge of window

BALL_RADIUS = 10
BRICKS_PER_ROW = 10
BRICK_ROWS = 10      # number of brick rows
BRICK_GAP = 4          # gap between bricks
BRICK_WIDTH = ((WIN_WIDTH - (BRICKS_PER_ROW - 1) * BRICK_GAP)
               // BRICKS_PER_ROW)
LEFT_MARGIN = (WIN_WIDTH - (BRICK_WIDTH * BRICKS_PER_ROW +
                            BRICK_GAP * (BRICKS_PER_ROW - 1))) // 2
BRICK_HEIGHT = 8
BRICK_Y_OFFSET = 414
CENTER_X = WIN_WIDTH // 2
CENTER_Y = WIN_HEIGHT // 2

class Breakout(arcade.View):
    def __init__(self):
        super().__init__()

        arcade.set_background_color(WHITE)
        self.window.set_mouse_visible(True)
        self._ball = None
        self._paddle = None
```

```
self._brick_lst = None
self._brick_remains = None
self._next_view = None

def setup(self):
    self._paddle = arcade.SpriteSolidColor(PDL_WIDTH,
                                            PDL_HEIGHT,
                                            BLACK)
    self._paddle.bottom = PDL_Y_OFFSET
    self._paddle.center_x = CENTER_X

    self._ball = setup_ball()
    self._brick_lst = setup_bricks()
    self._brick_remains = 20

def on_draw(self):
    self.draw()

def draw(self):
    self.clear()
    self._paddle.draw()
    self._brick_lst.draw()
    self._ball.draw()

def on_update(self, delta_time):
    self._ball.update()
    self._paddle.update()
    self._brick_lst.update()
    bricks_hit = arcade.check_for_collision_with_list(self._ball,
                                                       self._brick_lst)

    if len(bricks_hit) > 0:
        self._ball.change_y *= -1
    for brick in bricks_hit:
        brick.remove_from_sprite_lists()
        self._brick_remains -= 1

    if self._brick_remains == 0:
        self._next_view._msg = "You Win!"
        self.window.show_view(self._next_view)

    if self._ball.center_y <= PDL_Y_OFFSET:
        self._next_view._msg = "You Lost!"
        self.window.show_view(self._next_view)

    if (arcade.check_for_collision(self._ball, self._paddle)
        and self._ball.change_y < 0):
```

JJº

```
        self._ball.change_y *= -1

    def on_mouse_motion(self, x, y, dx, dy):
        self._paddle.center_x = x

def setup_bricks():
    colors = [CYAN, CYAN, GREEN, GREEN, GOLD, GOLD,
              ORANGE, ORANGE, RED, RED]
    brick_lst = arcade.SpriteList()
    x = LEFT_MARGIN + BRICK_WIDTH // 2
    y = BRICK_Y_OFFSET + BRICK_HEIGHT // 2
    # 10 x 10 bricks wall
    for i in range(BRICKS_PER_ROW):
        for j in range(BRICK_ROWS):
            brick = arcade.SpriteSolidColor(BRICK_WIDTH,
                                              BRICK_HEIGHT,
                                              colors[i])
            brick.center_x = x + (j * (BRICK_WIDTH + BRICK_GAP))
            brick.center_y = y + (i * (BRICK_HEIGHT + BRICK_GAP))
            brick_lst.append(brick)
    return brick_lst

class Ball(arcade.SpriteCircle):
    def update(self):
        self.center_y += self.change_y
        self.center_x += self.change_x

        if self.left < 0:
            self.change_x *= -1

        if self.right > WIN_WIDTH:
            self.change_x *= -1

        # hitting bottom edge doesn't bounce
        if self.bottom < 0:
            pass

        if self.top > WIN_HEIGHT:
            self.change_y *= -1

def setup_ball():
    ball = Ball(BALL_RADIUS, arcade.color.BLACK)
    ball.center_x = WIN_WIDTH // 2
```

```
ball.center_y = WIN_HEIGHT // 2
ball.change_y = -6
random.uniform(1.0, 3.0)
ball.change_x = random.uniform(1.0, 3.0) \
    if random.uniform(0.0, 1.0) <= 0.5 \
    else random.uniform(-1.0, -3.0)
return ball

class StartView(arcade.View):
    def __init__(self):
        super().__init__()
        self._next_view = None

    def on_draw(self):
        self.clear()
        self._next_view.setup()
        self._next_view.draw()

    def on_mouse_press(self, x, y, button, modifiers):
        if self._next_view:
            self._next_view.setup()
            self.window.show_view(self._next_view)

class EndView(arcade.View):
    def __init__(self):
        super().__init__()
        self._next_view = None
        self._msg = None

    def on_draw(self):
        self.clear()
        arcade.draw_text(self._msg,
                        WIN_WIDTH / 2,
                        WIN_HEIGHT / 2,
                        RED,
                        font_size=20,
                        anchor_x="center")

    def on_mouse_press(self, x, y, button, modifiers):
        if self._next_view:
            self.window.show_view(self._next_view)

def main():
```

JJR

```
window = arcade.Window(WIN_WIDTH, WIN_HEIGHT, "Breakout")
start_view = StartView()
game_view = Breakout()
end_view = EndView()

start_view._next_view = game_view
game_view._next_view = end_view
end_view._next_view = start_view

window.show_view(start_view)
arcade.run()

if __name__ == "__main__":
    main()
```


နောက်ဆက်တဲ့ က

လိုအပ်သည့် ဆောဖံ့ဌများ ထည့်သွင်းခြင်း

အခြား အင်ဂျင်နီယာ/သိပ္ပံာ ပညာရပ်တွေလိုပဲ ပရိုကရမ်မင်းလေ့လာတဲ့အခါ လက်တွေ့လုပ်ကြည့်ဖို့ အင်မတန်အရေးကြိုးပါတယ်။ လက်တွေ့ရေးမကြည့်ဘဲ စမ်းသပ်မကြည့်ဘဲ သဘောတရားပိုင်းဆိုင်ရာတွေကို အမှန်တကယ်နားလည်ဗုံး မဟုတ်ပါဘူး။ အခြားပညာရပ်တွေထက် ကွန်ပျိုးတာ ပရိုကရမ်မင်းရဲ့ အားသာချက်တစ်ခုကတော့ လက်တွေ့စမ်းသပ်ခန်းကြိုးတွေ ရှိစရာမလိုတာပါပဲ။ စမ်းသပ်ပစ္စည်းတွေ လည်း များများစားစား မလိုအပ်ဘူး။ ကွန်ပျိုးတာ တစ်လုံးနဲ့ လိုအပ်တဲ့ ဆောဖံ့ဌပဲတဲ့ ရှိရင်ရပြီ။ ဆောဖံ့ဌတွေကလည်း ပိုက်ဆံကုန်စရာမလိုဘူး။ မပေးဘဲ သုံးလို့ရတာ။

ပရိုကရမ်လက်တွေ့ရေး လေ့ကျင့်ဖို့အတွက် လိုအပ်တဲ့ ဆောဖံ့ဌတွေ ထည့်ထားရပါမယ်။ Python programming language အတွက် Python ဆောဖံ့ဌ ထည့်ရပါမယ်။ Python ဆောဖံ့ဌမရှိဘဲ Python ကုဒ်တွေ၊ Python ပရိုကရမ်တွေ run လို မရပါဘူး။ Python အပြင် ပရိုကရမ်ကုဒ် ရေးဖို့ အတွက် အထောက်အကူဗြာ ဆောဖံ့ဌပဲတစ်ခုလည်း လိုအပ်တယ်။

ပရိုကရမ် ကုဒ်ရေးဖို့အတွက် PyCharm သိမဟုတ် Visual Studio Code (VS Code) ကို အသုံးပြုနိုင်ပါတယ်။ အက်ဆေးတစ်ပုဒ်ရေးတဲ့အခါ ပိုက်ခရိုဆောဖံ့ဌ Word နဲ့ရေးလိုရသလို ရိုးရိုးရှင်းရှင်း Notepad လောက်နဲ့ ရေးလိုလည်း ရတာပါပဲ။ အက်ဆေးရဲ့ အဓိပ္ပာယ်ကသာ အစိုကပါ။ ပရိုကရမ်ကုဒ် ရေးတဲ့အခါမှာလည်း ခီးသောပါပဲ။ ဆောဖံ့ဌပဲ ပရောဂျက်ကြိုးတွေ တည်ဆောက်တဲ့အခါ လိုအပ်မဲ့ ဖီချာတွေ အားလုံး စုံစုံလင်လင်ပါပြီးသား။ PyCharm လို Integrated Development Environment (IDE) ဆောဖံ့ဌပဲနဲ့ ကုဒ်တွေရေးလိုရသလို ပါပေါ့ပါးပါးနဲ့ လိုအပ်မှုပဲ လိုတဲ့ဖီချာအတွက် extensions (plugin လိုလည်းခေါာက်) ထည့်သွင်းရတဲ့ Visual Studio Code (VS Code) လို ကုဒ်အသိဒ်ဘာ (Code Editor) ဆောဖံ့ဌပဲ့ဗို့ သုံးပြီး ရေးရင်လည်း ရတာပါပဲ။ နောက်ဆုံး ကုဒ်နည်နည်း၊ ဖိုင်နည်းနည်း ရိုးရှင်းတဲ့ ပရိုကရမ်လေးတွေဆိုရင် Notepad နဲ့ ရေးလိုတောင် ရပါတယ်။ ကုဒ်တွေများမယ် ဖိုင်တွေ များမယ်၊ အသင်းအစွဲလိုက် ပူးပေါင်းရေးရတဲ့ ပရိုကရမ်မျိုးတွေ ဆိုရင်တော့လည်း Notepad လောက်နဲ့ အဆင်မပြနိုင်တော့ဘူးပေါ့။

PyCharm ရော VS Code ထည့်သွင်းနည်းပါ ဖော်ပြပေးပါမယ်။ မိမိ နှစ်သက်ရာ အဆင်ပြောရာ သို့မဟုတ် နီးစပ်ရာ ပရိုကရမ်ဟာ အသိမိတ်ဆွေ အကြုပြုတဲ့ တစ်ခုကို ရွေးချယ်သုံးပါ။ စလေ့လာသူအနေနဲ့ PyCharm ကို အသုံးပြုတာ ပိုအဆင်ပြောမယ်လို ထင်တယ်။ PyCharm သုံးကြည့်လို မိမိကွန်ပျိုးတာ မှာ နေးလွန်းတယ်ဆိုရင် VS Code ကိုစမ်းကြည့်ပါ။ နှစ်ခုလုံး စက်အရမ်းကောင်း/မြင် ဖို့ မလိုပါဘူး။ တော်ရုံး အတန်အသင်းကောင်းတဲ့ စက်လောက်နဲ့ အဆင်ပြောပါတယ်။

မိမိကိုယ်တိုင်က ကွန်ပျိုးတာ အသုံးပြုပဲ အခြေခံ အားနည်းပြီး ဖော်ပြပေးထားတဲ့ အတိုင်း တစ်ဆင့် ချင်း အင်စတောလ် လုပ်တာလည်း အဆင်မပြောစွာရင် ဒီဘာအုပ်ရဲ့ အောက်ပါ ဖွေ့စွဲဘွဲ့ခံနဲ့ ယူကျ။

ချုပ်နယ်တွေမှာ ကြည့်ရှုမေးမြန်း အကူအညီ တောင်းနိုင်ပါတယ်။ ဒါမှမဟုတ် အတွေးအကြိုးတဲ့ နီးစပ်ရာ အသိမိတ်ဆေး/ညီကိုမောင်နှစ်မဲ တစ်ယောက်ယောက်ရဲ့ အကူအညီယူပြီး အင်စတောလ်လုပ်ပါ။

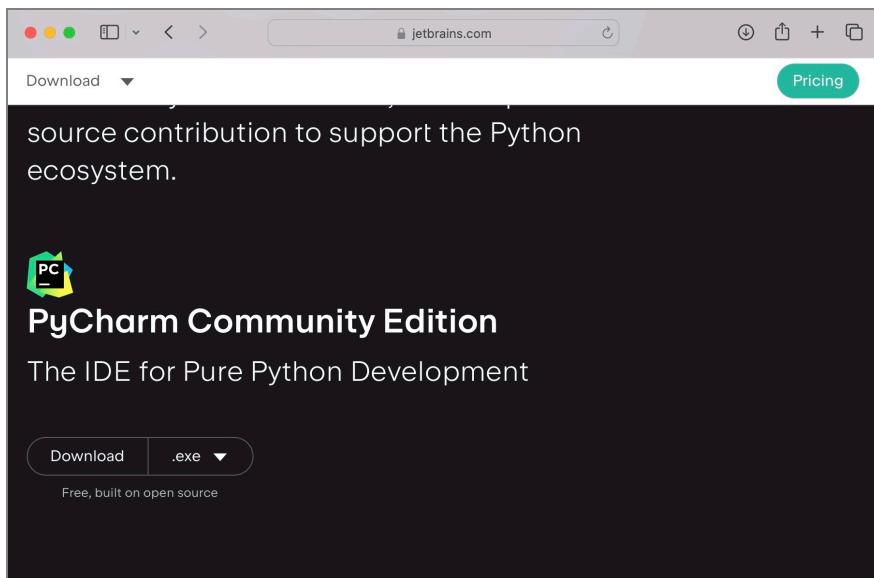
<https://www.facebook.com/bpwp>

<https://www.youtube.com/bpwp>

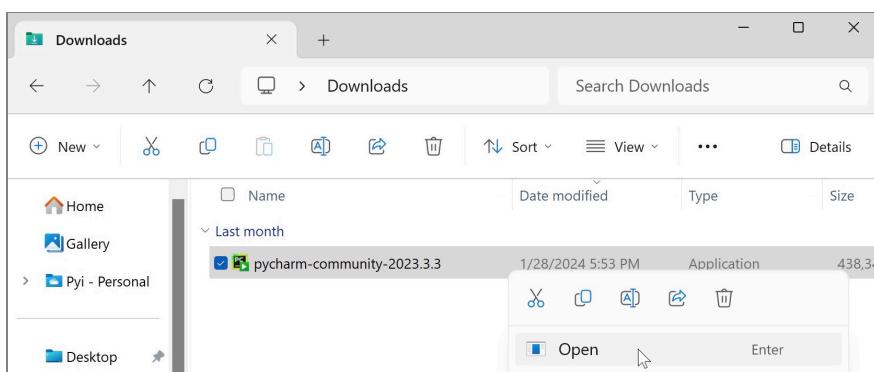
PyCharm အတွက် အရင်ဖော်ပြပေးပါမယ်။ VS Code အတွက် စာမျက်နှာ ၂၄၀ မှာ ကြည့်ပါ။

Python နှင့် PyCharm IDE ထည့်သွင်းခြင်း

<https://www.jetbrains.com/pycharm/download/> လင့်ကိုဖွင့်ပါ။ ဝဘ်စာမျက်နှာ အောက်ဘက် နည်းနည်း ဆွဲချုပ်ရင် PyCharm Community Edition ခေါင်းလုပ်ခလုတ်ကို တွေ့ရပါမယ်။ ပုံ (က/ခ) ကိုကြည့်ပါ။



ပုံ ၃/၁၁



ပုံ ၃/၂၂

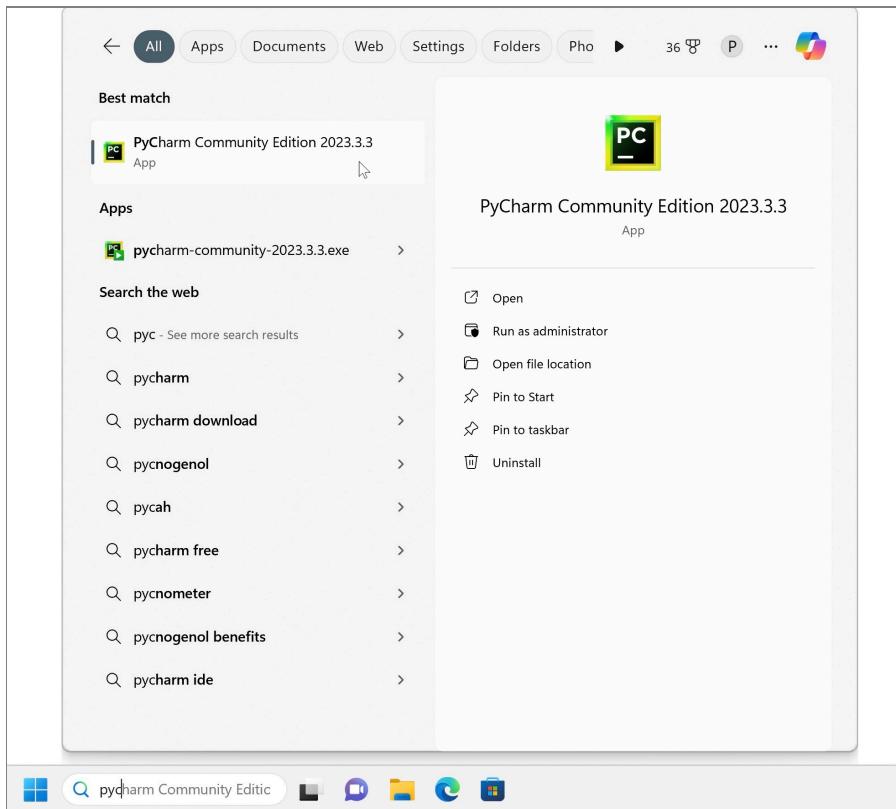
PyCharm Community Edition ကို အောင်လုပ်ပါ။ (ဝဘ်စာမျက်နှာ အပေါ်ပိုင်းက ၀၂ သုံးရတဲ့ PyCharm Professional ကို အောင်လုပ် မှားမလုပ်မိန့် သတိပြုပါ)။ အင်စတော်လာဖိုင်ကို ညာကလစ်နှုပ်ပြီး Open လုပ်ပါ (ပုံ ၂/၂ ကိုကြည့်ပါ)။ Yes/No မေးတဲ့အခါ Yes နှုပ်ပါ။ Next > ကို နှုပ်၍ ရှေ့ကိုဆက်သွားပြီး နောက်ဆုံးမှာ Install နှုပ်ပြီး ကွန်ဖန်လုပ်ပါ။ အင်စတော်လာပြီးသွားရင် Finish နှုပ်ပါ။

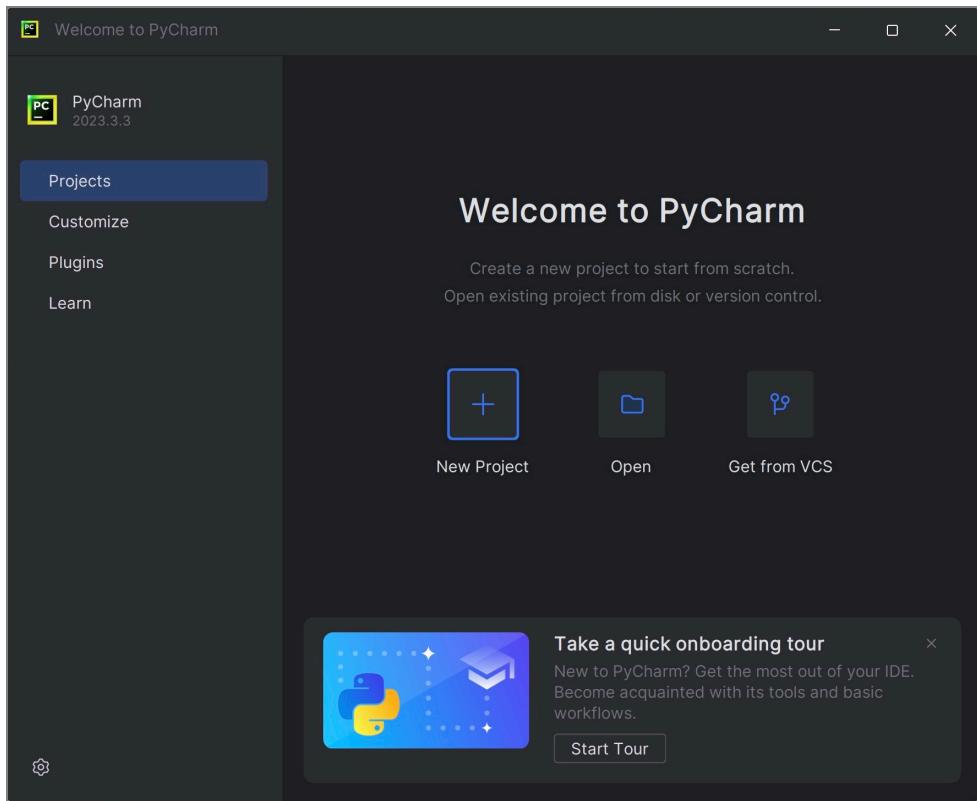
၀၂:၃၀: Taskbar Search ကနေ PyCharm ကိုရှာပြီးဖွင့်ပါ (ပုံ ၂/၃ ကို ကြည့်ပါ)။ သဘောတူ ကြောင်း ကွန်းဖန်းလုပ်ခိုင်းရင် ချက်ချက်ဘောက်စ် ချက်ချက်လုပ်ပြီး Continue နှုပ်ပါ။ ဒေတာပုံချင်လား ထပ် မေးပါလိမ့်မယ်။ Don't Send နှုပ်ပါ။ Welcome စခရင်ကိုပေါ်လာမယ်။ ပုံ (၂/၄) မှာ ကြည့်ပါ။

ဒီစာရေးနေချိန် လက်ရှိ PyCharm ဗားရှင်းက ၂၀၂၃ ပါ။ သိပ်မကြာခင် ၂၀၂၄ ထွက်ပါတော့မယ်။ အကယ်၍ လက်ရှိဗားရှင်းထက် နိမ့်တဲ့ဗားရှင်းတွေကို အောင်လုပ် လုပ်ချင်ရင် အောက်ပါ လင့်ကို သွားပါ။

<https://www.jetbrains.com/pycharm/download/other.html>

ဗားရှင်း ၂၀၂၄/၂၅ ထွက်ပြီးတဲ့ အချိန်မှာ ၂၀၂၃ ဗားရှင်းကို လိုချင်ရင် ဝဘ်စာမျက်နှာမှ ရှာပြီး အောင်လုပ် လုပ်ပါ။ ၂၀၂၃ မှာလည်း ဗားရှင်းအခွဲတွေ ရှိပါသေးတယ်။ လက်ရှိအမြဲ့အမွှေ့ဗားရှင်းအခွဲ (ဥပမာ ၂၀၂၃.၃.၃) ကို သုံးလို့ရပါတယ်။ ၂၀၂၄/၂၅ သုံးမယ်ဆိုရင်လည်း ပြဿနာတော့ မရှိပါဘူး။ Update ဗားရှင်းဖြစ်တဲ့အတွက် ပုံတွေမှာ ပြထားတာနဲ့တော့ ကွားခြားချက်တရာ့၍ ရှိကောင်းရှိနိုင်ပါတယ်။

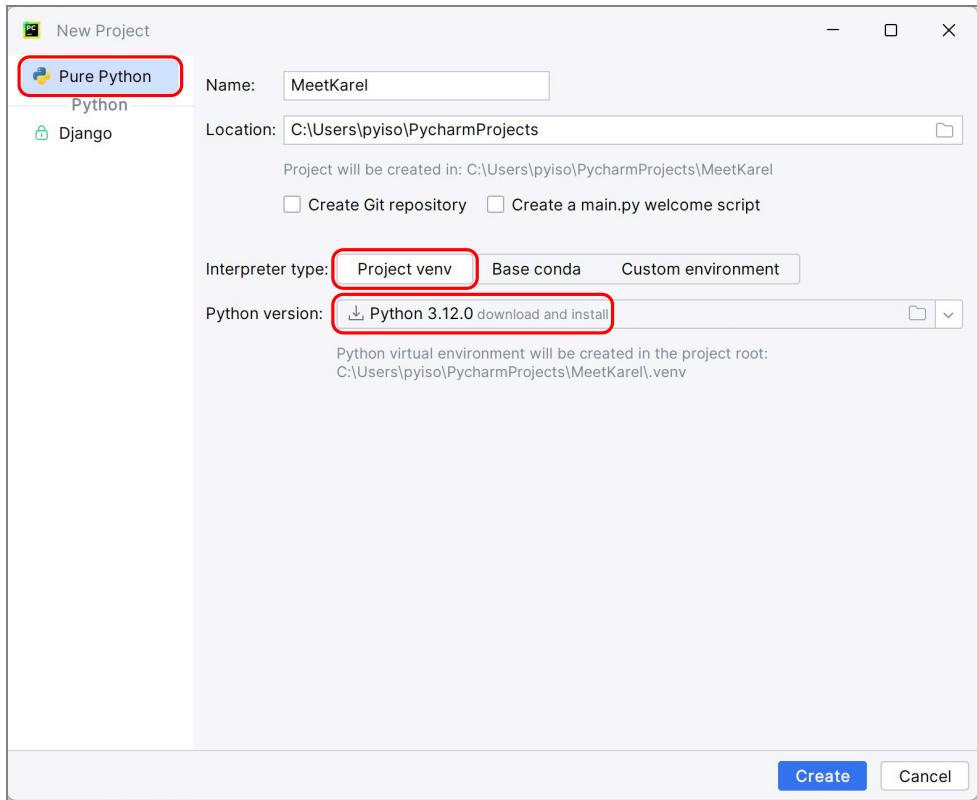




ဗုံး ၂/၄

PyCharm IDE ဆိတာဘာလ

PyCharm သာ Python နဲ့ ဆော်ဖို့အတွက် အထောက်အကူဗြိုံ Integrated Development Environment(IDE) ဆော်ပြုပြုပါတယ်။ စာစီတရာ်လုပ်တဲ့အခါ Microsoft Word ကို အသုံးပြုကြသလိုပဲ Python ကုဒ်ရေးနဲ့ PyCharm ကိုသုံးတဲ့ သဘောပေါ့။ PyCharm IDE ၏ Python ပရောဂျက်တွေ အတွက် အခိုက်ရည်ရွယ်တယ်။ အထောက်အဦးတစ်ခု၊ တံတားတစ်ခု ဆောက်လုပ်တာ ကို ပရောဂျက်လို့ ပြောလေ့ရှိသလို ပရိုကရမ်/ဆော်ပဲတစ်ခု တည်ဆောက်တာကိုလည်း ပရောဂျက်လိုပဲ သုံးနှုန်းပါတယ်။ တစ်ဦးတစ်ယောက်တည်း ရေးတဲ့ ပရိုကရမ်အသေးလေးတွေ အတွက် PyCharm ကို အသုံးပြုနိုင်သလို ပရိုကရမ်မာတွေ အဖွဲ့လိုက်နဲ့ တည်ဆောက်ရတဲ့ ပရောဂျက်ကြီးတွေ အတွက်လည်း သုံးပါတယ်။ ဒီစာအုပ်မှာတော့ PyCharm ရဲ့ အဆင့်မြင့်ဖို့ချာတွေကို အသုံးပြုမှာ မဟုတ်ပါဘူး။ ပရိုကရမ်းမင်း စလေ့လာသူတွေကို လွှာယ်ကူးအဆင်ပြေစေတဲ့ အခြေခံ ဖီချာတွေလောက်ပဲ အသုံးပြုမှုပါ။



ပုံ ၁/၁ ပရောဂျက် အသစ်ယူခြင်း

PyCharm ပရောဂျက်ဆောက်ခြင်း

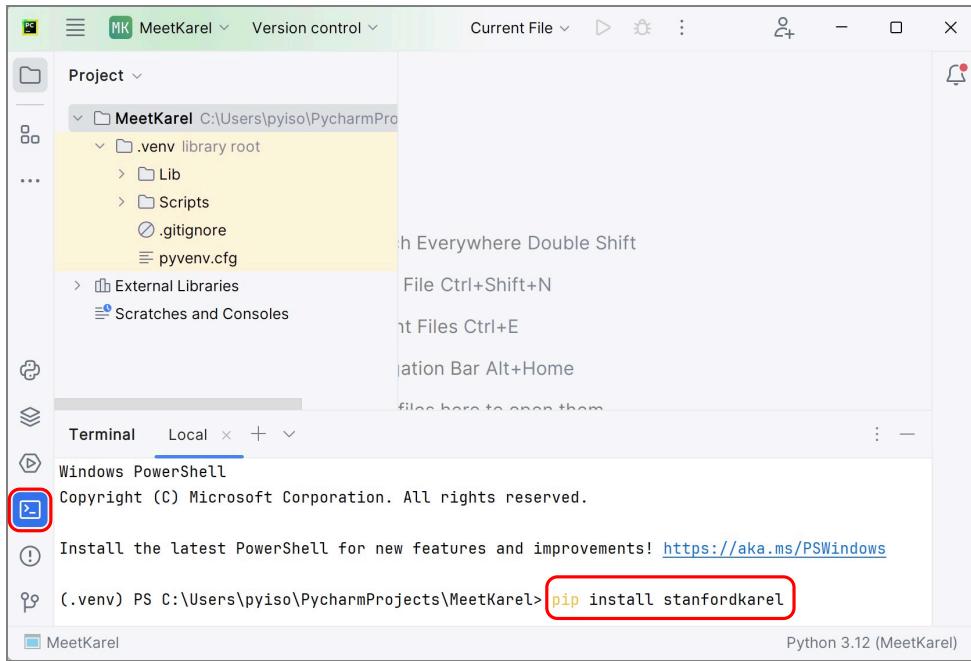
အင်စတောလုပ်ပြီးရင် PyCharm IDE ကိုဖွေ့ပြီး Welcome စခရင်မှ သိမဟုတ် File မိန္ဒီးမှ New Project နှင့်ပြီး ပရောဂျက် အသစ်ယူပါ (ပြီးခဲတဲ့ ပုံ ၁/၄ ကို ပြန်ကြည့်ပါ)။ နံမည်ကို MeetKarel သေးပါ။ ပုံ (၁/၅) မှာ တွေ့ရတဲ့အတိုင်း Pure Python, Proj venv နဲ့ Python ဗားရှင်း 3.12.xx ကို ရွေးပါ။ xx က အခြားကဏ္ဍား ဖြစ်နေနိုင်တယ်။ အခိုက် ဗားရှင်း 3.12 သာ ဖြစ်ပါခေါ် Create ခလုတ်နှင့်ပါ။ ပရောဂျက်အတွက် Python ကို အင်စတောလ် လုပ်ပါလိမ့်မယ်။

မိမိလေ့လေနေတဲ့ အခန်းတစ်ခုချင်းစီအတွက် ပရောဂျက်တစ်ခု ဆောက်နိုင်ပါတယ်။ အခန်း (၂) အတွက် Chapter02 ၊ အခန်း (၃) အတွက် Chapter03 စသည်ဖြင့်။ သက်ဆိုင်ရာအခန်းအလိုက် ကုဒ်ဖိုင်တွေကို ပရောဂျက်တစ်ခုစီမှာ ထားတဲ့အတွက် ဖိုင်တွေများပြီး ရှုပ်ထွေဖောင်းပွဲနေတဲ့ ပြဿနာ မရှိတော့ဘူး။ ကုဒ်ဖိုင်တွေကို ပရောဂျက်တစ်ခုခဲ့မှာပဲ ဖိုဒ်အလိုက်ခဲ့ထားလို့ရပေမဲ့ စလေ့လာသူအနေနဲ့ ပရောဂျက်တစ်ခုချင်း ခဲ့ထားတာလောက် မလွယ်ကူဘူး။

ပရောဂျက် အသစ်ယူတဲ့အခါ တည်နေရာ Location: ကို သူ့ရှိခိုက်အတိုင်း ထားနိုင်သလို မိမိထားချင်တဲ့နေရာကို ညာဘာက်စွန်း ဖိုဒ်အရိုင်ကွန်လေးနှင့်ပြီး ပြောင်းလို့ရတယ်။

stanfordkarel လိုက်ဘရီ အင်စတောလ်လုပ်ခြင်း

ပုံ (၂/၆) မှာ အနိရောင် ဝိုင်းပြထားတဲ့ အိုင်ကွန်ကို နှင့်ပြီး Terminal ကိုဖွေ့ပြီး Terminal မှာ အောက်ပါ ကွန်မန်းဖြင့်



ပုံ က/၆ Karel လိုက်ဘရီ အင်စောင်လုပ်ခြင်း

```
pip install stanfordkarel
```

ကားရဲလ်လိုက်ဘရိကို အင်စတောလ်လုပ်ပါ။ ပုံ (က/၆) မှာ အနီရောင် ဂိုင်းပြထားပါတယ်။ ခက်ကြာတဲ့ အခါ အခုလို မက်ဆွဲချုတ် ကျေလာပါလိမ့်မယ်။

Collecting stanfordkarel

Downloading stanfordkarel-0.2.7-py3-none-any.whl (51 kB)

- 51.9/51.9 kB 443.1 kB/s

eta 0:00:00

Installing collected packages: stanfordkarel

Successfully installed stanfordkarel-0.2.7

[notice] A new release of pip is available: 23.2.1 -> 24.0

[notice] To update, run: `python.exe -m pip install --upgrade pip`

(.venv) PS C:\Users\pyiso\PycharmProjects\MeetKarel>

ဟိုက်လိုက်ပြထားတဲ့ မက်ဆောချိ တော်ရှင် အင်စတောလ်လုပ်တာ အောင်မြင်လိုပါ။

Page 11 of 11

မှတ်ချက်။ ။ ပရောဂျက် အသစ်ဆောကတဲး လုအပ်တဲ့ လက်ဘရဲ

လုပ်ရပါမယ်။ ကားရဲ့လအခန်း ပရောဂျက်တစ်ခုစွဲအတွက် stanfordkarel လုပ်ရပါမယ်။

မှတ်ချက်။ ပရောဂျက် အသစ်ဆောက်တိုင်း လိုအပ်တဲ့ လိုက်ဘရိကို တစ်ခါထပ်ပြီး အင်စတေလုပ်ရပါမယ်။ ကားရဲလ်အခန်း ပရောဂျက်တွင်ခုစ္စအတွက် stanfordkarel လိုက်ဘရိကို အထက်ဖော်ပြပါအတိုင်း အင်စတေလုပ်စိုက်ပေးပါ။

နမူနာ ကားရဲလ် ကဗ္ဗာနှင့် ပရိုကရမ်ကုဒ် ဖိုင်များထည့်ခြင်း

meet_karel.zip ဖိုင်ကို ဒီလင့် <http://tinyurl.com/3mmmm9c7j> ကနေ ဒေါင်းလုပ်လုပ်ပါ။ ငှါး zip ဖိုင်ကို extract လုပ်ပါ။ meet_karel နံမည်နဲ့ ဖိုဒါတစ်ခု ရလာပါမယ်။ ငှါးဖိုဒါထဲမှ အောက်ပါ worlds ဖိုဒါနှင့် .py ဖိုင်အားလုံးကို ကော်ပီလုပ်ပါ။

- worlds
 - ▶ meet_karel.w
 - ▶ move_beeper_to_other_side.w
- meet_karel.py
- move_beeper_to_other_side.py
- world_editor.py

MeetKarel ပရောဂျက်ထဲတွင် ကူးထည့်ပါ။ ပင်မ ပရောဂျက် MeetKarel (ပုံ က/၂ မှာ မြှားပြထား) ပေါ်မှာ ညာကလစ်နှင့်ပြီး Paste လုပ်ရမှာပါ။ ကော်ပီကူးထည့်လိုက်တဲ့ ဖိုင်တွေက ပုံမှာ တွေ့ရတဲ့ အတိုင်း MeetKarel ဖိုဒါအောက်မှာ ရှိသင့်ပါတယ်။

အခန်းအလိုက် နမူနာ ကုဒ်ဖိုင်တွေ ထည့်ပေးထားတဲ့ .zip ဖိုင်တွေကိုလည်း အထက်ပါအတိုင်း အလားတူ လုပ်ရပါမယ်။ ပရောဂျက်အသစ်ဆောက်၊ .zip ဖိုင်ကို ဖြည့်၊ ရလာတဲ့ ဖိုဒါထဲက ဖိုင်တွေကို ပင်မ ပရောဂျက် ဖိုဒါထဲ ကော်ပီကူးထည့် ရုံပါပဲ။

meet_karel.py ဖိုင်ကို ကလစ်နှစ်ချက်နှင့်ပါ။ ပုံ (က/၂) မှာ အနီးစိုင်းထားတဲ့ ကုဒ်အယ်ဒီတာ (code editor) ပွင့်လာပါမယ်။ အဲဒီ မူရင်းဖိုင်မှာ အောက်ပါအတိုင်း ဆက်လက် ပြင်ဆင်ဖြည့်စွက်ပါ။

```
from stanfordkarel import *

def main():
    """Karel code goes here!"""
    move()
    move()
    move()
    pick_beeper()
    turn_left()
    move()
    move()

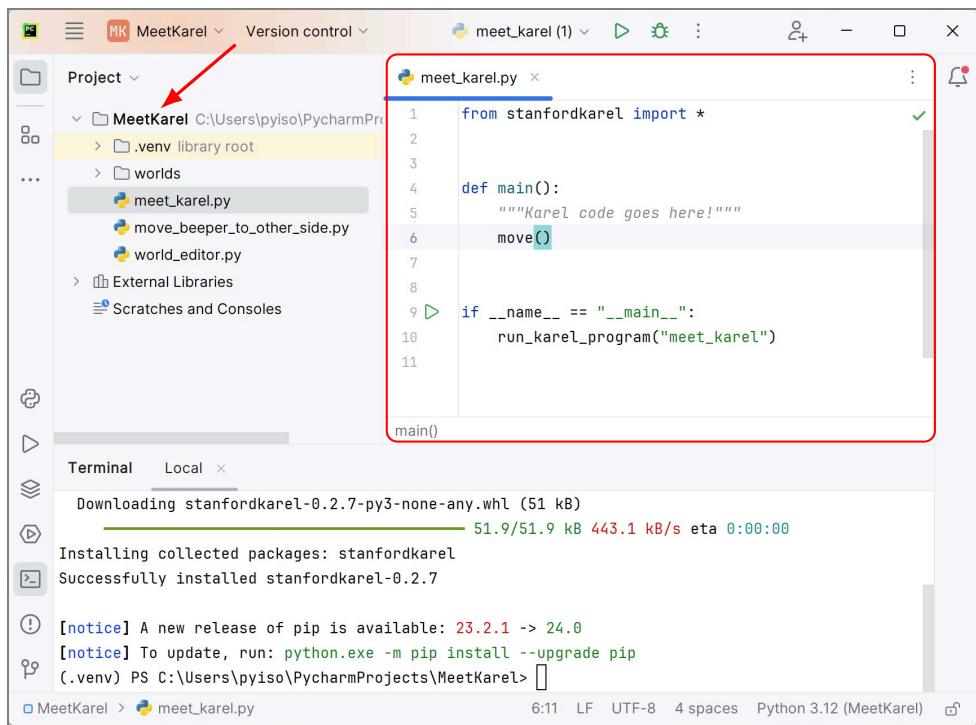
    turn_left()
    turn_left()
    turn_left()

    move()
    put_beeper()

if __name__ == "__main__":
    run_karel_program("meet_karel")
```

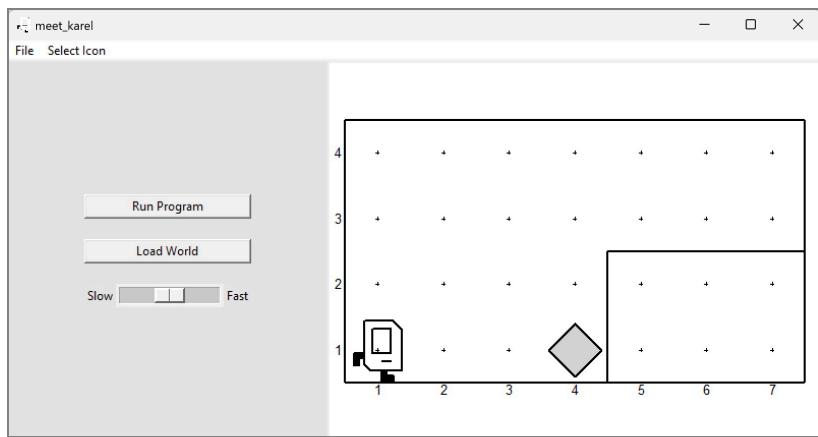
run ထားတဲ့ ပရိုကရမ်တစ်ခုကို တစ်ခါထပ် run ရင် Cancel (သို့) Stop and Rerun လုပ်မှုလား မေးတယ်။ Stop and Rerun လုပ်ရပါတယ်။

meet_karel.py ဖိုင်ကို ညာကလစ်နှုမြို့း Run 'meet_karel' လုပ်ပါ (အယ်ဒီတာ ဧရိယာမှာ ညာကလစ်နှုမြို့း Run 'meet_karel' လုပ်လိုလည်း ရပါတယ်)။ ပုံ (က/စ) က ကားရဲ့ပဲ ပရိုကရမ် တက်လာရင် Run Program ခလုတ်ကိုနှိပ်ပါ။ ဘိပါကို ကားရဲ့ပဲက ခွဲေပေးပါလိမ့်မယ်။



ပုံ ၃/၄

JR2



ବୁଦ୍ଧି

ဆင်းတက်စ်အမှားများ

အကယ်၍ ပရိုကရမ ရှုံးလို့မရရင် ကုဒ်ရေးတာမှားနေလို့ ဖြစ်နိုင်တယ်။ မမိရေးထားတာကို စာမျက်နှာ (J20) က ပရိုကမ်းကုဒ်နဲ့ နှင့်ယူဉ် စစ်ဆေးကြည့်ပါ။ PyCharm အယ်ဒီတာမှာ အနီလိုင့်တွန်လေး တွေ (ပုံ ၂/၉) ပြတဲ့နေရင် အဲဒီနေရာတွေမှာ ဆင်းတက်စ်မှားနေလို့ (သို့) လိုက်ဘရီမထည့်ရသေးလို့ပဲ။

လိုက်ကွင်းကျို့နေတာက အဖြစ်များတဲ့ အမှားပါ။ ကျို့ခဲ့လို့ မရပါဘူး။ အင်ဒန်တေးရှင်း (indentation) လုပ်ရမဲ့နေရာမှာ မလုပ်ထားရင်လည်း ပြဿနာဖြစ်တယ်။ `move`, `turn_left` တွေကို ဘေးမျဉ်း ညာဘက်ဆွဲပြီး အင်ဒန်လုပ်ပေးရမယ်။ အဲဒီတွေ ကရုမစိုက်မိရင် ဆင်းတက်စ်အမှားဖြစ်ပြီး ပရိုကရမ ရှုံးလို့ မရနိုင်ဘူး။

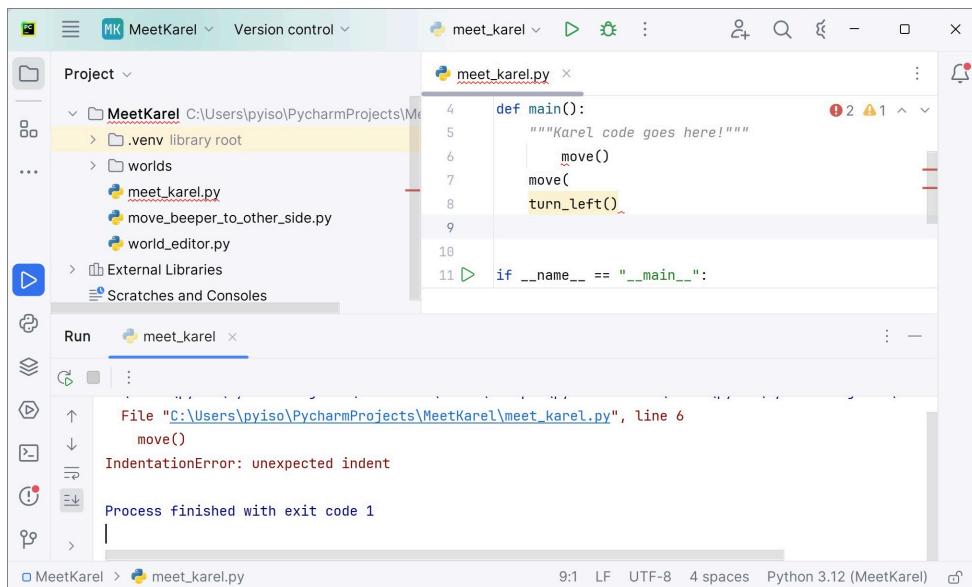
Terminal မှ ထုတ်ပေးတဲ့ မက်ဆွဲချုတ္တာကို ကြည့်ပြီးတော့လည်း ဘာပြဿနာဖြစ်နေလဲ မှန်းဆ လို့ရနိုင်တယ်။ ဘာကြောင့်ဖြစ်နိုင်လဲ ဆက်စပ်စဉ်းစားလို့ ရတယ်။ ဥပမာ ဖြစ်တဲ့ပြဿနာအလိုက် အခုလို တွေ့ရပါမယ်။

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 6
    move(
        ^
SyntaxError: '(' was never closed
```

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 7
    move()
    ^
IndentationError: unexpected indent
```

Traceback (most recent call last):

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 1,
    in <module>
```



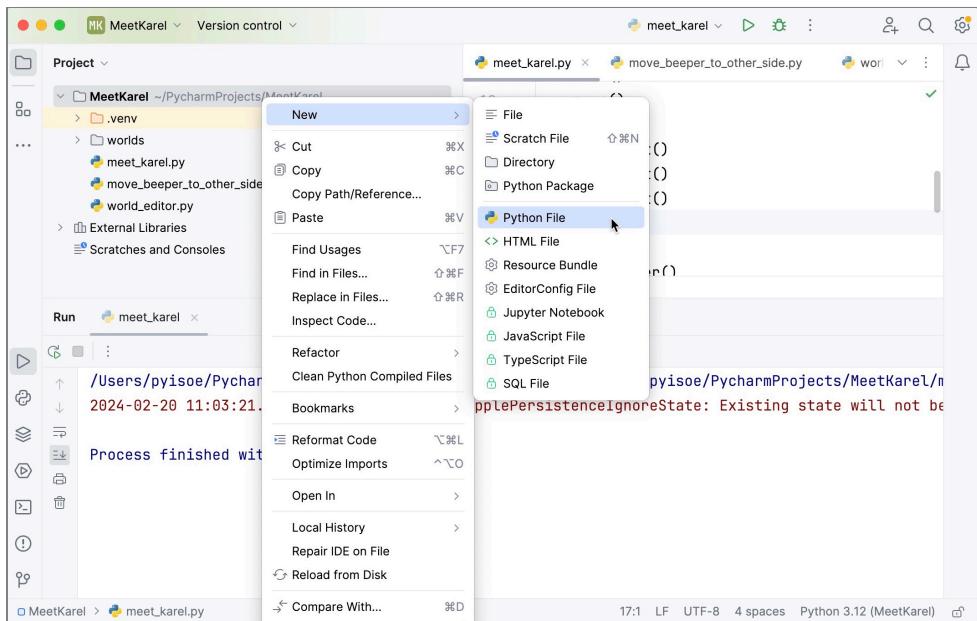
JRJ

```
from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
```

Python ဖိုင် အသစ်ယူခြင်း

MeetKarel ပင်မ ပရောဂျက်ဖိုဒ်အပေါ်မှာ ညာကလစန်ပြီး Python ဖိုင် အသစ်ယူနိုင်ပါတယ်။ Python ဖိုင်တွေက .py အိုင်စာန်းရှင်းနဲ့ပါ။ ကားရဲ့လ်ပရိုဂရမ်တစ်ခုကို Python ဖိုင်တစ်ခု ထားပါမယ်။ ပင်မ ပရောဂျက်ဖိုဒ်အောက်မှာပဲ တိုက်ရိုက်ရှိရပါမယ်။

နောက်ပိုင်း အဆင့်မြင့်လာရင် ပရိုဂရမ်တစ်ခုအတွက် ပရောဂျက်တစ်ခု ထားနိုင်တွေ အပြင် ပရိုဂရမ်အတွက် လိုအပ်တဲ့ ရုပ်ပုံတွေ၊ အခြားဖိုင်တွေ (config ဖိုင်၊ setting ဖိုင် စသည်ဖြင့်) လည်း ပါနိုင်တယ်။ ပင်မပရောဂျက် အောက်မှာပဲ ဖိုင်တွေက တိုက်ရိုက်ရှိဖို့လည်း မလိုတော့ဘူး။ ဆက်စပ်ရာ ဖိုင်တွေကို အမျိုးအစားအလိုက် ဖန်ရှင်အလိုက် ဖို့ဒါတွေခဲ့ပြီး စနစ်ကျ စီစဉ်ဖွဲ့စည်း ထားရမှာပါ။ ပရောဂျက်တစ်ခုမှာ ဖိုင်တွေကို စနစ်တကျ စုဖွဲ့ထားဖို့ အရေးကြီးပါတယ်။



ပုံ ၂/၁၀

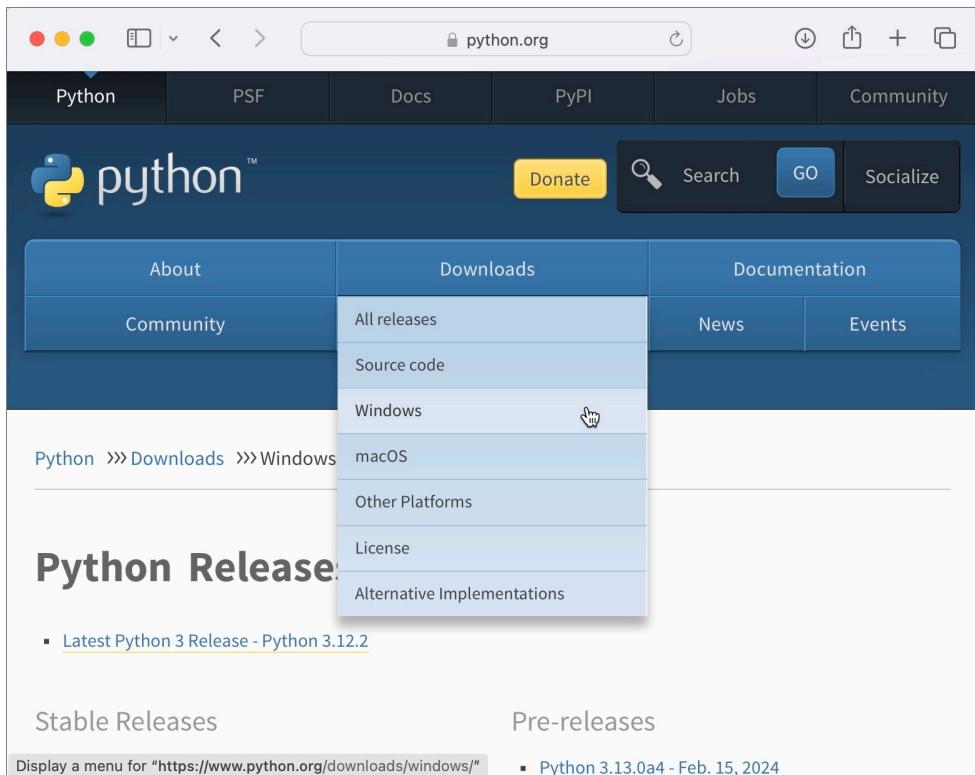
Visual Studio Code နှင့် Python အင်စတောလ်လုပ်ခြင်း

Visual Studio Code(VS Code) ဟာ ပရိုဂရမ်မာအများစုံ ကြိုက်နှစ်သက်တဲ့ မောဒန် ကုဒ်အယ်ဒီတာ တစ်ခုပါ။ Python, JavaScript, C++ စုတဲ့ programming language အမျိုးမျိုးအတွက် အသုံးပြု နိုင်ပါတယ်။

PyCharm မှာတော့ ပရောဂျက်အသစ်ဆောက်ရင် Python ပါ တစ်ခါတည်း ဒေါင်းလုဒ်လုပ်ပြီး အင်စတောလ် လုပ်လိုရတယ်။ VS Code နဲ့ Python ရေးမယ်ဆိုရင် Python Programming Language ကို သိုံးခြား ဒေါင်းလုဒ်လုပ်ပြီး အင်စတောလ် လုပ်ရပါမယ်။

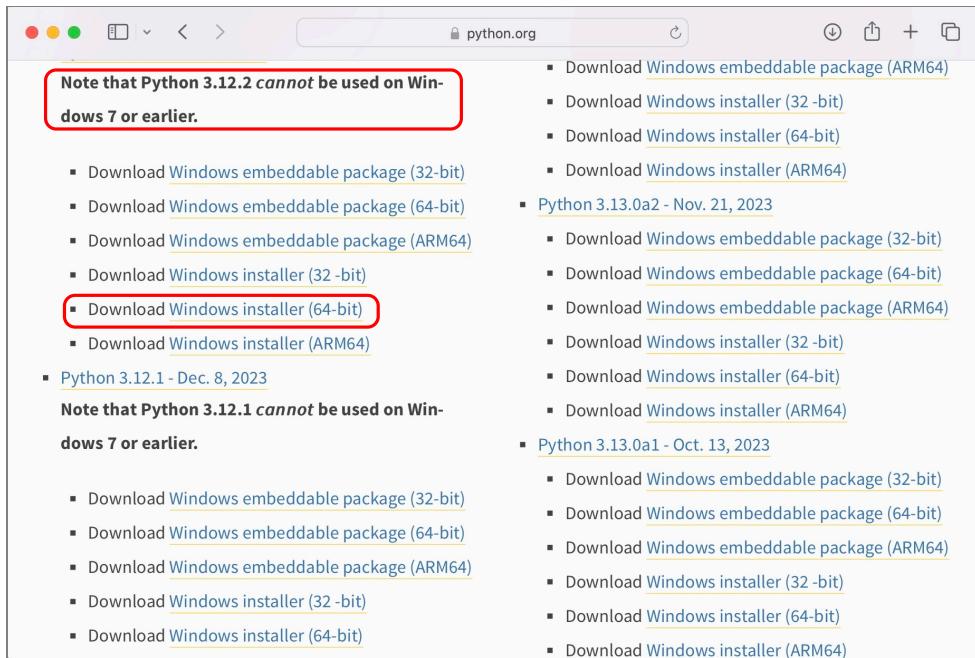
Python အင်စတောလ်လုပ်ခြင်း

ဒီလုပ် <https://www.python.org/> ကိုဖွေ့စ်ပါ။ Download မိန္ဒီးမှ Windows နိုပ်ပါ (ပုံ က/၁၁ ကို ကြည့်ပါ)။ Apple ကွန်ပျိုးတာအတွက် ဆိုရင် macOS ရွှေးရပါမယ်။ လူများစုသုံးတဲ့ မိုက်ခရီးဆော် ဝင်းခွီးအတွက် အင်စတောလ်လုပ်နည်းကို အခိုက်ပြေားမှာပါ။ ပုံ က/၁၂ မှာလို တွေ့ရပါလိမ့်မယ်။ အင်စတောလ်လာ ဗားရှင်း 3.12 ထဲက လက်ရှိအမြင့်ဆုံး ကိုရွေးပါ။ ဒီစာရေးချိန်မှာ 3.12.2 ဟာ အမြင့်ဆုံးဗားရှင်းပါ။ Windows Installer (64-bit) ကိုနိုပ်ပြီး ဒေါင်းလုဒ်လုပ်ပါ။



ပုံ က/၀၁

ဒေါင်းလုဒ်ပြီးရင် အင်စတောလ်လာဖိုင်ကို ညာကလစ်နိုပ်ပြီး Run as administrator လုပ်ပါ။ ပုံ (က/၁၃) ကိုကြည့်ပါ။ ပုံ (က/၁၄) မှာလို ဒိုင်ယာလော် ဆောက်စ် ပွင့်လာပါမယ်။ အနိဂုင်းထားတဲ့ ချက်ချေ ဆောက်စ်နှစ်ခုကို ချက်ချေလုပ်ပြီး Install Now နိုပ်ပါ။ အင်စတောလ် ပြီးလို Setup was successful



Note that Python 3.12.2 cannot be used on Windows 7 or earlier.

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)**
- Download Windows installer (ARM64)

Python 3.12.1 - Dec. 8, 2023

Note that Python 3.12.1 cannot be used on Windows 7 or earlier.

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)

Python 3.13.0a2 - Nov. 21, 2023

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)
- Download Windows installer (ARM64)

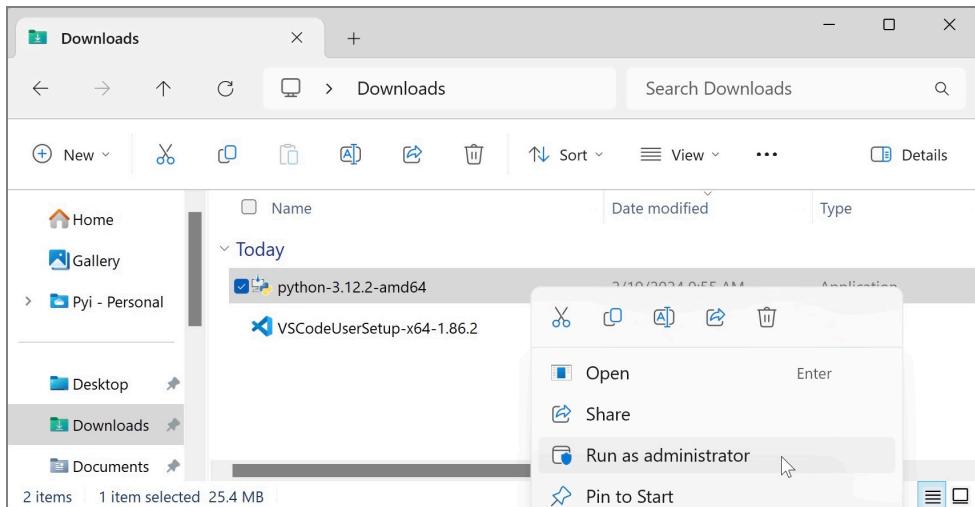
Python 3.13.0a1 - Oct. 13, 2023

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)
- Download Windows installer (ARM64)

ဗိ က/၁၂

ပေါ်လာရင် Close ခလုတ်နိုပ်ပြီး ပိတ်ပါ။

ဝင်းဒီး command prompt (cmd) မှာ `python --version` run ရင် ဖုံး (၁/၁၅) မှာလို အင် စတော်လုပ်ထားတဲ့ ဗားရှင်းကို ပြပေးသင့်ပါတယ်။



Downloads

Name Date modified Type

python-3.12.2-amd64

VSCodeUserSetup-x64-1.86.2

Open Enter

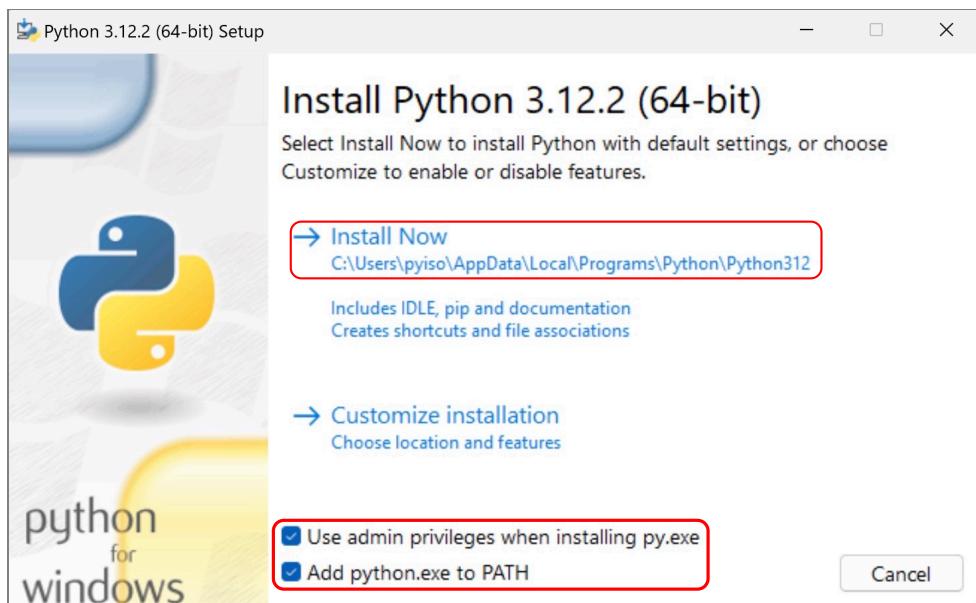
Share

Run as administrator

Pin to Start

ဗိ က/၁၃

JRC



◊ ፩/፭፭

```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pyiso>python --version
Python 3.12.2

C:\Users\pyiso>
```

◊ ፩/፭፭

VS Code အင်စတေလုပ်ခြင်း

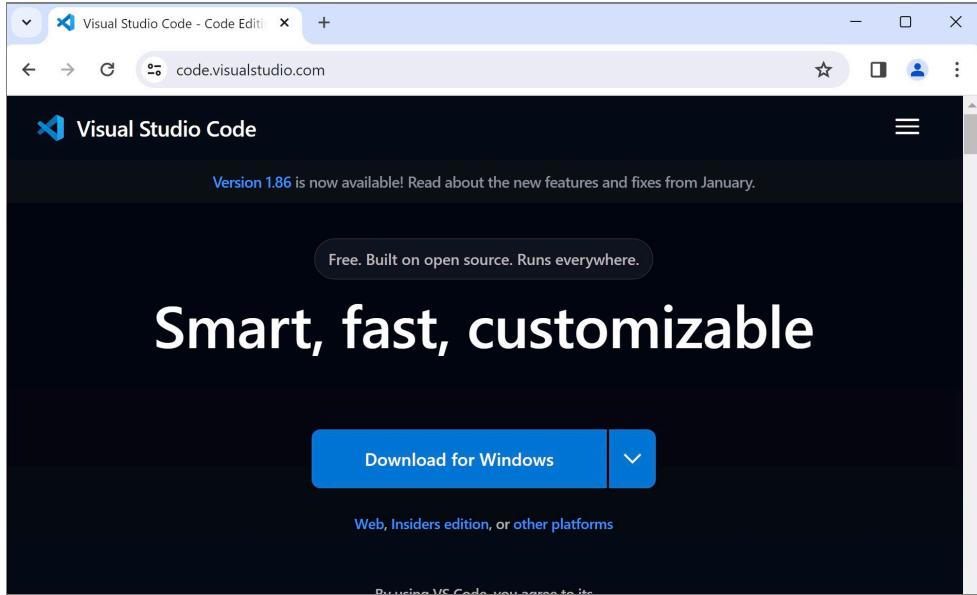
အခုနိရင် Python programming language အောင်မြင်စွာ ထည့်ပြီးသွားပါပြီ။ VS Code အင်စတေလုပ်ခြင်း ဆက်လုပ်ပါမယ်။ အင်စတေလုပ်ခြင်း အောင်မြင်စွာ ထည့်ပြီးသွားပါပြီ။

<https://code.visualstudio.com>

မှတစ်ဆင့် သွားပါ။ ပုံ (က/၁၆) ဝဘ်စာမျက်နှာကို တွေ့ရပါမယ်။ [Download for Windows](#) နှင့် ဒေါင်းလုပ်လုပ်ရန် ဝဘ်စာမျက်နှာကို အောက်ပါလုပ်ပါ။

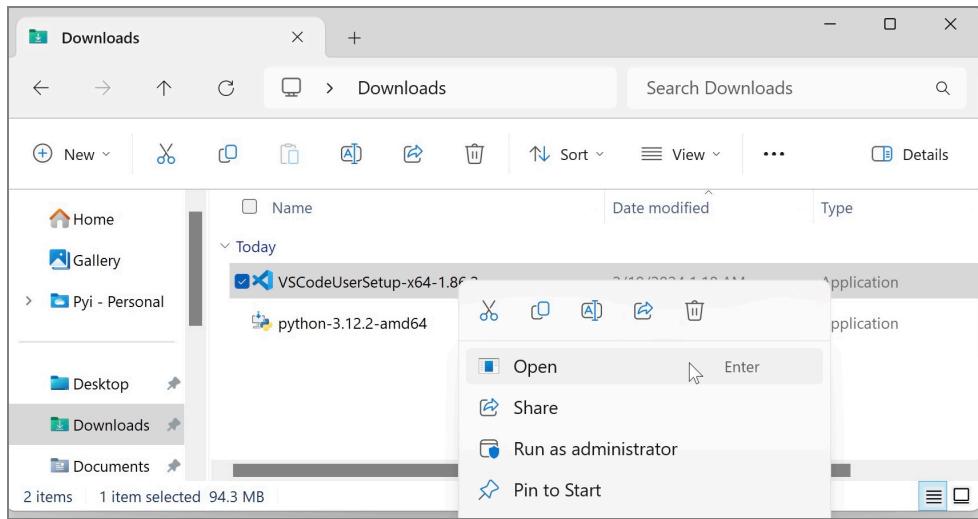
ပြီးတဲ့အခါ အင်စတေလုပ်ခြင်းကို ညာကလစ်နှင့်ပြီး ဖွင့်ပါ (ပုံ က/၁၇ မှာ ပြထားပါတယ်)။ အင်စတေလုပ်ခြင်းကို အောင်မြင်စွာ ထည့်ပြီး ပွင့်လာရင် [I accept the agreement](#) ကို ချက်ချွဲလုပ်၍ [Next >](#) တစ်ခြားတစ်ခု ဆက်နှိပ်သွားပြီး နောက်ဆုံးမှာ [Install](#) နှင့်ပါ။ အင်စတေလုပ်ခြင်းနောက် ခေါ်စောင့်ပြီး၊ ပြီး သွားရင် [Finish](#) နှင့်ပါ။

Welcome စခရင်ကို ပုံ (က/၁၈) လို တွေ့ရပါမယ်။ [Dark Modern](#) (သို့) [Light Modern](#) နှစ်သက်ရာ သီးမ်စ် ရွှေးပါ။ ဒါဆိုရင် VS Code လည်း အင်စတေလုပ်ခြင်းနှင့်ပါ။ ကားရဲ့လုပ်ရှိရမဲ့ ရေးဖို့ ဆက်လုပ်ရပါမယ်။

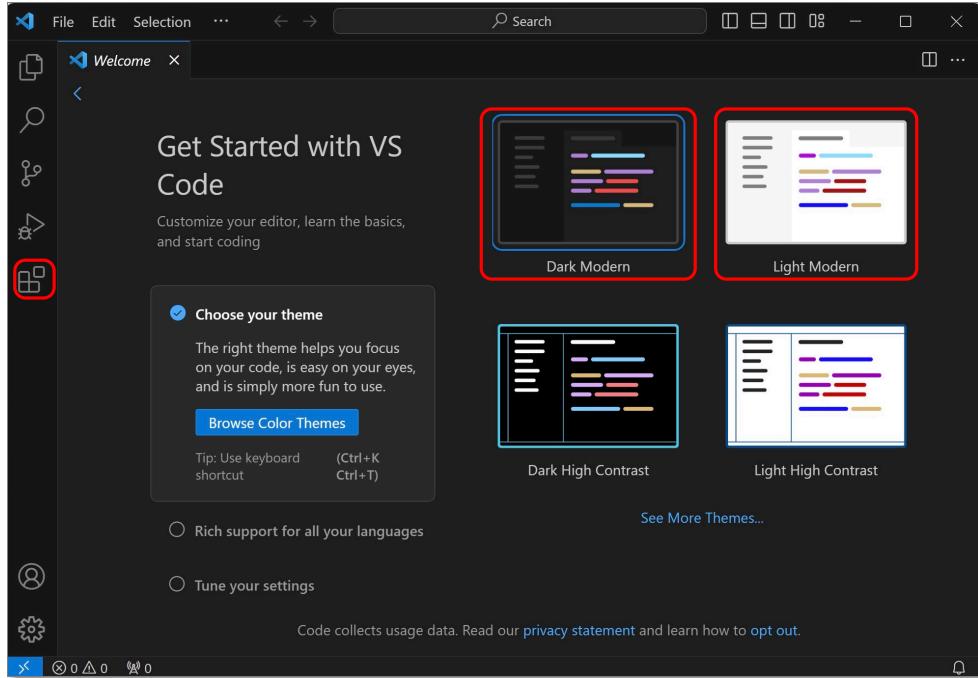


ပုံ က/၁၆

၂၅



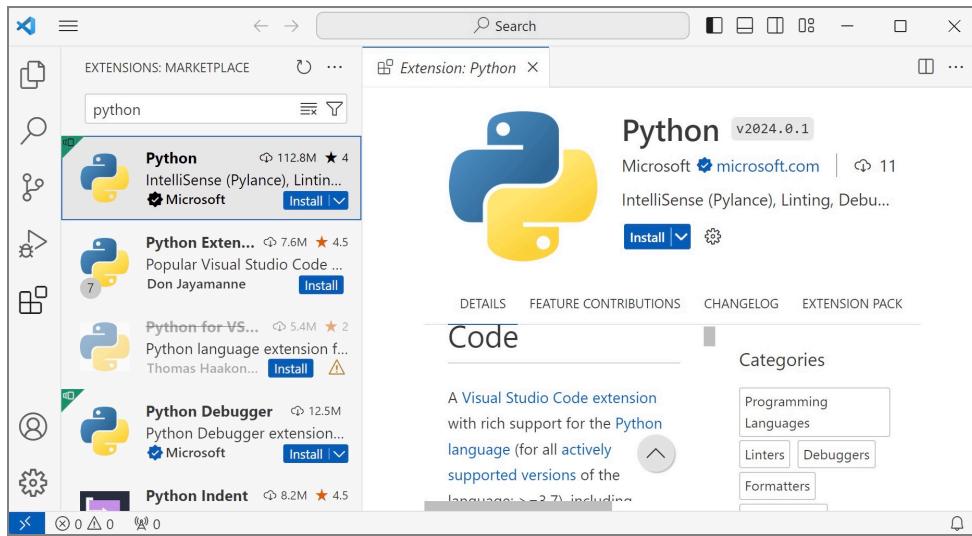
ပုံ ၂/၀၇



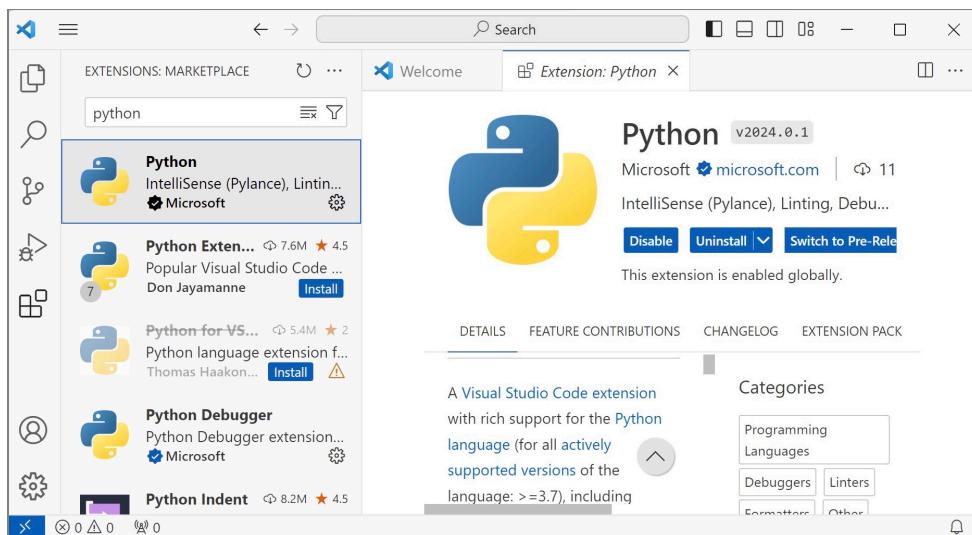
ပုံ ၂/၀၈

VS Code Python Extension တည်ခြင်း

Python ပရိဂုဂ်ရမ်ရေးဖို့အတွက် VS Code က ဒီအတိုင်းဆိုရင် သိပ်အဆင်မပြသေးပါဘူး။ Python အတွက် extension အင်စတောလ် လုပ်ပေးရပါအိုးမယ်။ ပုံ (က/၁၈) ဘယ်ဘက်သောင်နားမှာ အနီးပိုင်းထားတဲ့ အိုင်ကွန်လေးကို နှိပ်ပါ။ Python extension ကိုရှာပါ။ ပုံ (က/၁၉) မှာ ဘယ်လိုရှာရမလဲ ပြထားပါတယ်။ ပုံမှာတွေ့ရတဲ့ Microsoft က ထုတ်တဲ့ extension ကို အင်စတောလ်လုပ်ပါ။ အောင်မြင်ရင် ပုံ (က/၂၀) မှာလို ဖြစ်သွားပါမယ်။ Python နဲ့ VS Code ကိစ္စတော့ ပြီးသွားပြီ။ ကားရဲ့လဲ example run ဖို့ ဆက်လုပ်ရမယ်။



ပုံ က/၁၉



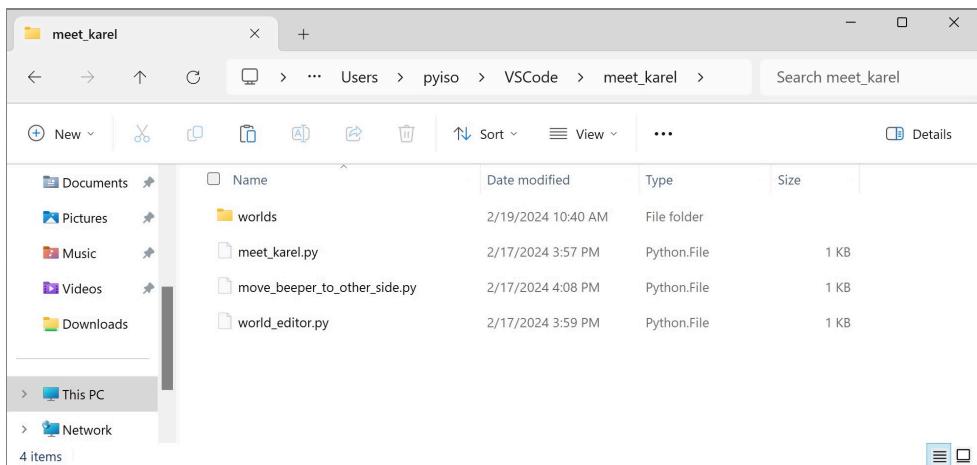
ပုံ က/၂၀

နမူနာ ကားရဲလ် ကဗ္ဗာနှင့် ပရိုဂရမ်ကို ပိုင်များထည့်ခြင်း

meet_karel.zip ဖိုင်ကို ဒီလင့် <http://tinyurl.com/3mm9c7j> ကနေ ဒေါင်းလုပ်လုပ်ပါ။ ငါး zip ဖိုင်ကို extract လုပ်ပါ။ meet_karel နံမည်နဲ့ ဖို့ဒါတစ်ခု ရလာပါမယ်။ ဖို့ဒါထဲ ဝင်ကြည့်ရင် အောက်ပါ အတိုင်း ရှိသင့်ပါတယ်။

■ worlds

- ▶ meet_karel.w
 - ▶ move_beeper_to_other_side.w
- meet_karel.py
 - move_beeper_to_other_side.py
 - world_editor.py



ပုံ က/ဂ

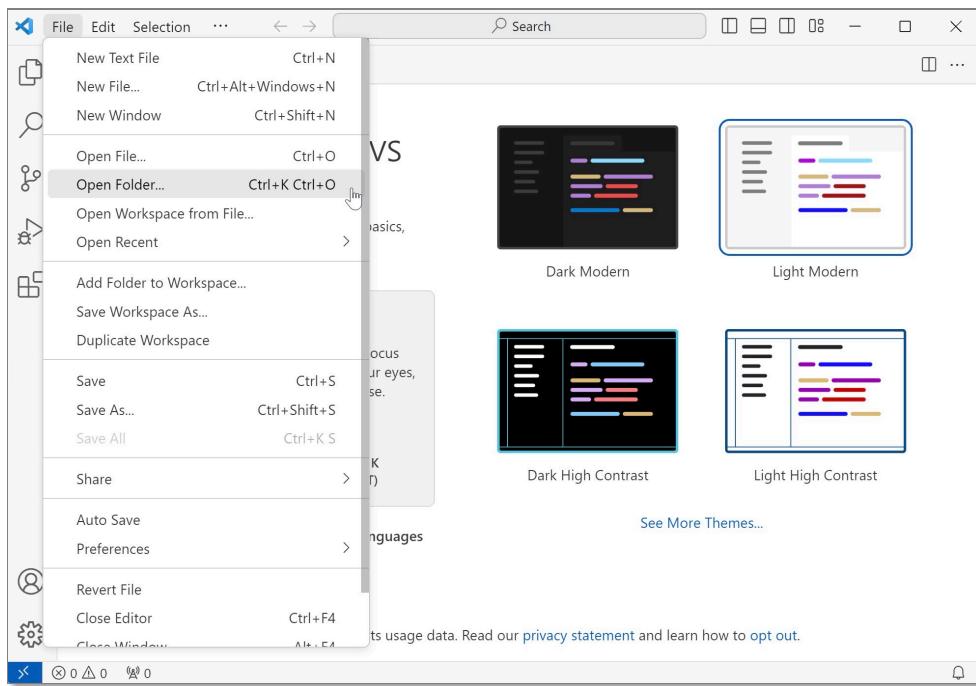
VS Code အတွက် ဖို့ဒါတစ်ခုကို မိမိအတွက် အဆင်ပြေမဲ့နေရာမှာ သီးသန့်တည်ဆောက် ထားသင့်တယ်။ ဥပမာ

C:\Users\yourname\VS Code

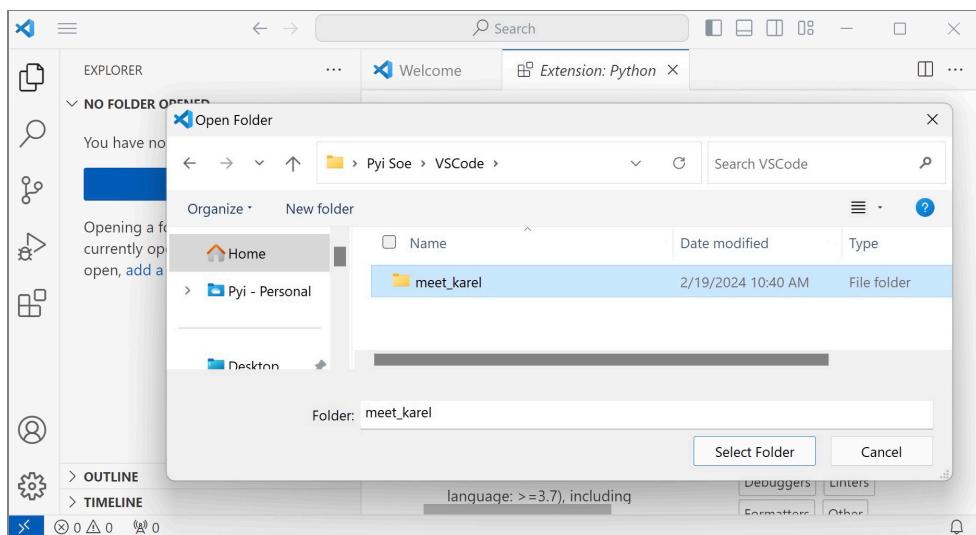
မိမိ လက်ရှိ Home ဖို့ဒါကို C: drive ရဲ့ Users ဖို့ဒါထဲမှာ တွေ့နိုင်ပါတယ်။ Win + R ကြောက်နှင့်ပြီး %userprofile% ရိုက်ထည့်၍ Ok လုပ်ပြီး Home ဖို့ဒါကို သွားနိုင်ပါတယ်။ အကယ်၍ မသွားတတ်ရင်လည်း ပြဿနာမရှိပါဘူး။ ကိုယ့်အတွက် လွယ်ကူမဲ့ Desktop, Downloads, Documents တစ်ခုခုထဲမှာ VS Code အတွက် ဖို့ဒါတစ်ခု ထားလည်းရတယ်။

meet_karel ဖို့ဒါ (.zip ဖိုင်မဟုတ်ပါ) ကို အထက်ပါအတိုင်း အသစ် ဆောက်ထားတဲ့ VS Code သီးသန့်ဖို့ဒါထဲကို ကော်ပိကူးထည့်ပါ။ ငါး meet_karel ဖို့ဒါကို VS Code File မိန္ဒာမှ Open Folder နှင့်ပြု၍ ဖွင့်ပါ။ ပုံ (က/ဂ) တွင်ကြည့်ပါ။

အခန်းအလိုက် နမူနာ ကုဒ်ဖိုင်တွေ ထည့်ပေးထားတဲ့ .zip ဖိုင်တွေကိုလည်း အထက်ပါအတိုင်း လုပ်ရပါမယ်။ .zip ဖိုင်ကို ဖြည့်၊ ရလာတဲ့ ဖို့ဒါကို သီးသန့်ဖို့ဒါတစ်ခုထဲကို ကော်ပိကူးထည့်၊ VS Code နဲ့ အဲခီဖို့ဒါကို ဖွင့်ရုံပါပဲ။



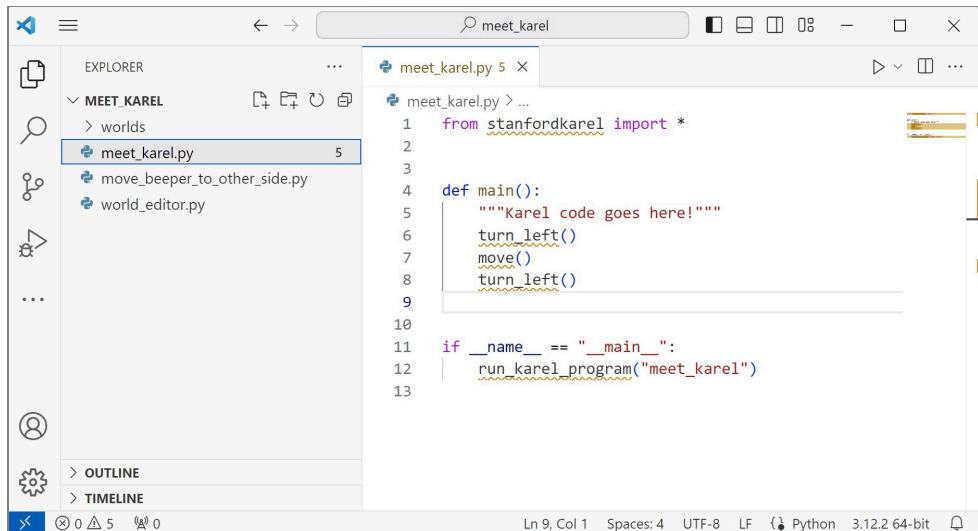
နိုင် ကြ/JJ



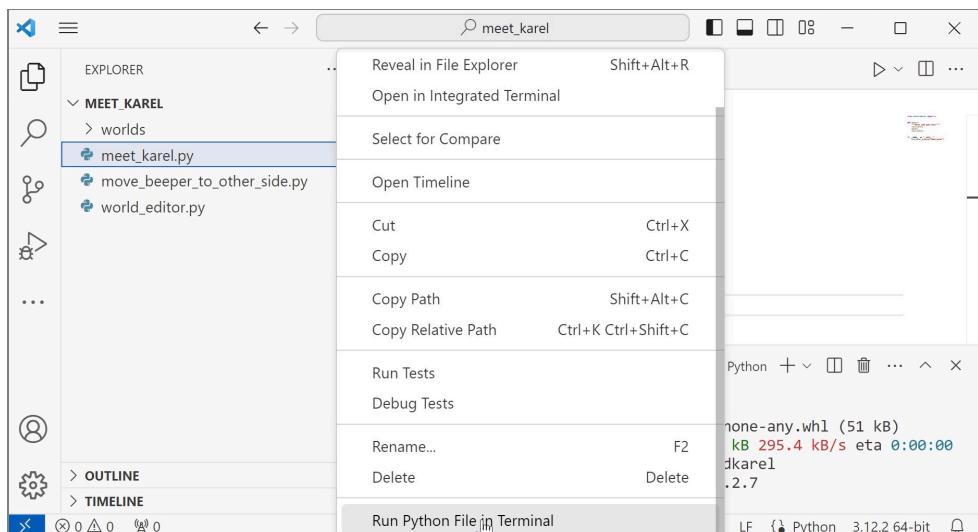
နိုင် ကြ/JR

stanfordkarel လိုက်ဘရီ အင်စတောလ်လုပ်ခြင်း

`meet_karel.py` ဖိုင်ကို ကလစ်နှစ်ချက်နှင့် ဖွင့်ပါ။ ကုဒ်အယ်ဒီတာ ပွင့်လာမယ် (ပုံ ၂/၂၃)။ အဲဒီ ကုဒ် အယ်ဒီတာပေါ် (သို့) `meet_karel.py` ဖိုင်ကို ညာကလစ်နှင့်ပြီး `Run Python File in Terminal` လုပ်ပါ (ပုံ ၂/၂၃)။ Terminal ပွင့်လာပြီး အယ်ရာမက်ဆောင်တွေ ပြလိမ့်မယ်။ ပုံ (၂/၂၄) မှာ ကြည့်ပါ။ ကားရုံးပရှုံးရပ်အတွက် လိုအပ်တဲ့ `stanfordkarel` လိုက်ဘရီ အင်စတောလ် မလုပ်ရသေးပါဘူး။ ဒါ ကြောင့် အယ်ရာဖြစ်နေတာ။



ပုံ ၂/၂၄



ပုံ ၂/၂၅

ခုနကပွင့်လာတဲ့ Terminal မှာပဲ အောက်ပါကွန်မန်းကို run ပြီး `stanfordkarel` လိုက်ဘရီကို အင်စတောလ်လုပ်ပါ။



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure with a folder named "MEET_KAREL" containing files: "worlds", "meet_karel.py" (selected), "move_beeper_to_other_side.py", and "world_editor.py".
- Code Editor:** The file "meet_karel.py" is open, showing the following code:

```
1  from stanfordkarel import *
2
3
4  def main():
5
```
- Terminal:** The terminal shows a command-line session:

```
PS C:\Users\pyiso\VSCode\meet_karel> & C:/Users/pyiso/AppData/Local/Programs/Python/Python312/python.exe c:/Users/pyiso/VSCode/meet_karel/meet_karel.py
Traceback (most recent call last):
  File "c:/Users/pyiso/VSCode/meet_karel/meet_karel.py", line 1, in <module>
    from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
PS C:\Users\pyiso\VSCode\meet_karel> pip install stanfordkarel
```

ଓ ତ/ଜ୍ଞ

```
pip install stanfordkarel
```

ပုံ (က/ဂ္ဂ) မှာ အနိဂုံးထားတာကို ကြည့်ပါ။ အဲဒီအတိုင်းရိုက်ထည့်ပြီး Enter ကီးနှုပ်ပါ။ ခက်ကြာတဲ့ အခါ အခုလို မက်ဆွဲချုပ်တဲ့ ကျေလာပါလိမ့်မယ်။

Installing collected packages: stanfordkarel

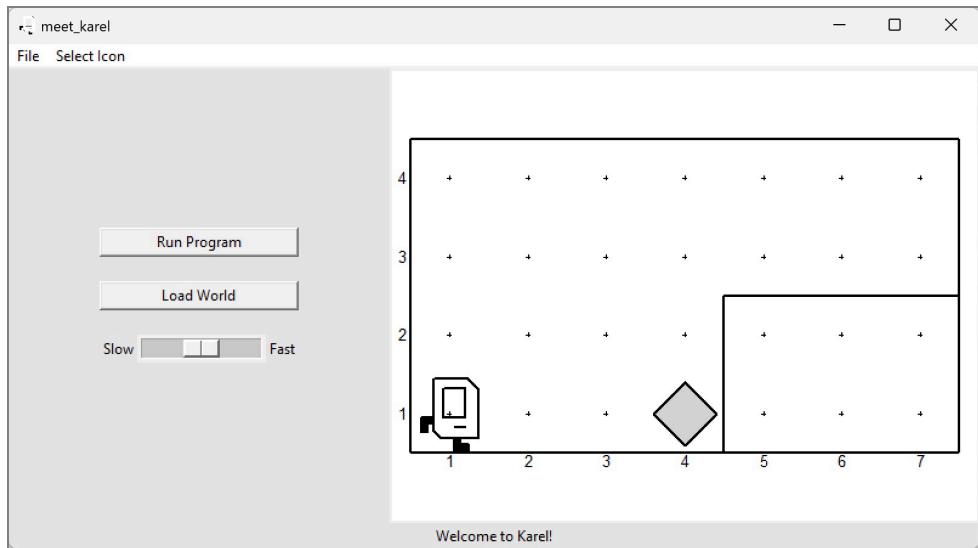
Successfully installed stanfordkarel-0.2.7

PS C:\Users\pyiso\VSCode\meet_karel>

ဟိုက်လိုက်ပြထားတဲ့ မက်ဆော်ချုပ် တွေ့ရရင် အင်စတောလ် အောင်မြင်လိုပါ။ ပုံ (က/၂၄) အယ်ဒီတာမှုလို သတိပေး လိုင်းတွန်များတွေ့ မရှိသင့်တော့ဘူး။ stanfordkarel လိုက်ဘရီ အင်စတောလ် လုပ်ပြီးပြီမြတ်လို သတိပေးတာတွေ့ ပျောက်သွားသင့်တယ်။

meet_karel.py ဖိုင်ကို ညာကလ်နှင့်ပြုပြီး Run Python File in Terminal ပြန်လုပ်ကြည့်ပါ။ ပုံ (က/ဂျ) မှာပြထားတဲ့ ကားရဲလ်ပရိုကရမ် ပွုံးလာသင့်ပါတယ်။ Run Program နှင့်ကြည့်ပါ။ စက်ရှုပ်လေး ကားရဲလ် နေရာရွှေသွားတဲ့ တော့ရမယ်။ meet_karel.py ကို အောက်ပါအတိုင်း ဖြည့်စက်ရေးပါ။

```
from stanfordkarel import *
def main():
    """Karel code goes here!"""
    move()
    move()
```



ဦး ၂/၂၇

```

move()
pick_beeper()
turn_left()
move()
move()

turn_left()
turn_left()
turn_left()

move()
put_beeper()

if __name__ == "__main__":
    run_karel_program("meet_karel")

```

`meet_karel.py` ဖိုင်ကို ညာကလစ်နိုပ်ပြီး Run Python File in Terminal ပြန်လုပ်ကြည့်ပါ။ ကား ရဲလ်ပရိုကရမ် ပွင့်လာရင် `Run Program` နှိပ်ကြည့်ပါ။ ဘိပါလေးကို နေရာခွဲ့ပေးပါလိမ့်မယ်။ အကယ်၍ ကားရဲလ်ပရိုကရမ် မတက်လာရင် ကုဒ်ရေးတာမှားနေလို့ ဖြစ်နိုင်တယ်။ အပေါ်ကုဒ်နဲ့ ဦးယူဉ်ပြီး ကြည့်ပါ။ VS Code အယ်ဒီတာမှာ အနိုလိုင်တွေ့လေးတွေ ပြတဲ့နေရင် အဲဒီနေရာတွေမှာ ဆင်းတက်မှုးနေတာ ဖြစ်နိုင်တယ်။

VS Code အယ်ဒီတာမှာ ပရိုကရမ်ကုဒ်ပြင်ပြီး ပြန် run တဲ့အခါ ပထမ run ထားတဲ့ ပရိုကရမ်ကို အရင်ပိတ်ဖို့လိုပါတယ်။ ဆိုလိုတာက meet_karel.py ကို run ထားတယ်ဆိုပါစူး။ ပုံ (က/ဂျ) က ဝင်းဒီးပင့်လာမယ်။ meet_karel.py ကုဒ်ကို ပြင်ပြီး ပြန် run ချင်ရင် အဲဒီ ဝင်းဒီးကို အရင်ပိတ်ရမယ်။ မဟုတ်ရင် ပြင်ထားတဲ့ ပရိုကရမ်က ချက်ချင်း ပွင့်မလာဘူး။ ပထမ ဟာကို ပိတ်တော့မှုပဲ နောက် run တဲ့ဟာ ပွင့်လာမှာ။

ပိုက်ကွင်းကျို့နေတာက အဖြစ်များတဲ့ အမှားပါ။ ကျို့ခဲ့လို မရပါဘူး။ အင်ဒန်တေးရှင်း (indentation) လုပ်ရမဲ့နေရာမှာ မလုပ်ထားရင်လည်း ပြဿနာဖြစ်တယ်။ move, turn_left တွေကို ဘေးမျဉ်းညာဘက်ဆွဲပြီး အင်ဒန်လုပ်ပေးရမယ်။ အဲဒါတွေ ကရာဇ်စိုက်မိရင် ဆင်းတက်စ်အမှားဖြစ်ပြီး ပရိုကရမ် run လို မရနိုင်ဘူး။

Terminal မှာ ထုတ်ပေးတဲ့ မက်ဆွဲချုပ်တွေကို ကြည့်ပြီးတော့လည်း ဘာပြဿနာဖြစ်နေလဲ မှန်းဆလိုရနိုင်တယ်။ ဘာကြောင့်ဖြစ်နိုင်လဲ ဆက်စပ်စဉ်းတားလို ရတယ်။ ဥပမာ ဖြစ်တဲ့ပြဿနာအလိုက် အခုလို တွေ့ရပါမယ်။

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 6
    move(
        ^
SyntaxError: '(' was never closed

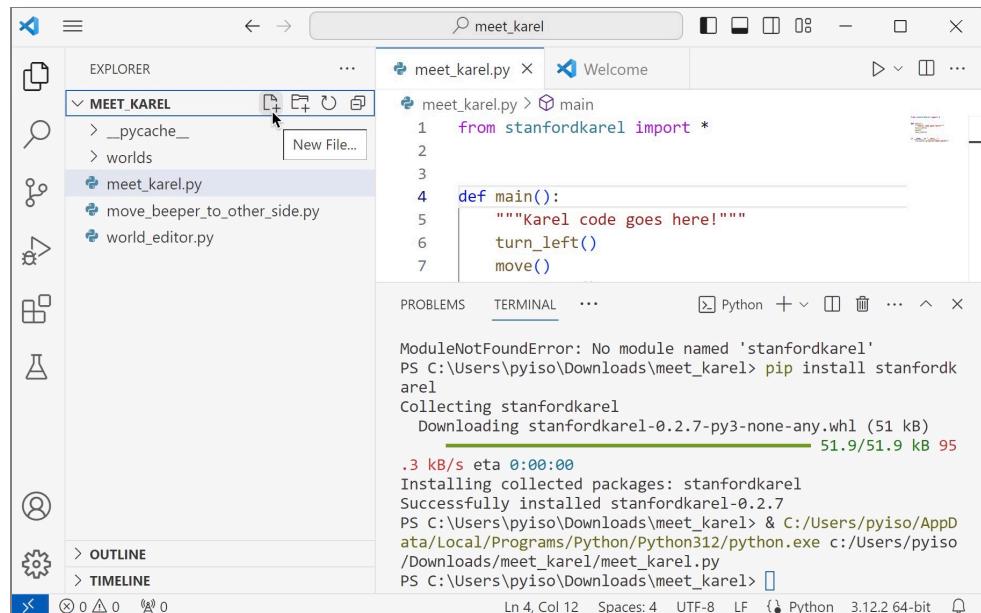
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 7
    move()
        ^
IndentationError: unexpected indent

Traceback (most recent call last):
  File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 1,
    in <module>
      from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
```

VS Code တွင် Python ဖိုင် အသစ်ယူခြင်း

MEET_KAREL ပင်မ ပရောဂျက်ဖို့ဒါပေါ်မှာ ကလစ်နိုင်ပါ။ ပုံမှာ ပြထားတဲ့ **New File** အိုင်ကိုနိုင်ပါ။ ဖိုင်နံပည်ဖြည့်တဲ့ ဘောက်စံလေး ပေါ်လာမယ်။ Python ဖိုင်တွေက .py အိုင်စံနံပည်နဲ့ဖြစ်ရပါမယ်။ ဒါကြောင့် နံပည် ဖြည့်တဲ့အခါ .py နဲ့ အဆုံးသတ်ပေးရပါမယ် (ဥပမာ hello.py)။ ကားချုပ်ပရိုက်ရမ်တစ်ခုကို Python ဖိုင်တစ်ခု ထားပါမယ်။ ပင်မ ပရောဂျက်ဖို့ဒါ အောက်မှာပဲ တိုက်ရှိကြရပါမယ်။

နောက်ပိုင်း အဆင့်မြင့်လာရင် ပရိုက်ရမ်တစ်ခုအတွက် ပရောဂျက်တစ်ခု ထားနိုင်တယ်။ ကုဒ်ဖိုင်တွေ အပြင် ပရိုက်ရမ်အတွက် လိုအပ်တဲ့ ရုပ်ပုံတွေ၊ အခြားဖိုင်တွေ (config ဖိုင်၊ setting ဖိုင် စသည်ဖြင့်) လည်း ပါနိုင်တယ်။ ပင်မပရောဂျက် အောက်မှာပဲ ဖိုင်တွေက တိုက်ရှိကြရဖို့လည်း မလိုတော့ဘူး။ ဆက်စပ်ရာ ဖိုင်တွေကို အမျိုးအစားအလိုက်၊ ဖန်ရှင်အလိုက် ဖို့ဒါတွေခဲ့ပြီး စနစ်ကျ စီစဉ်ဖွဲ့စည်း ထားရမှာပါ။ ပရောဂျက်တစ်ခုမှာ ဖိုင်တွေကို စနစ်တကျ စွဲထားဖို့ အရေးကြီးပါတယ်။



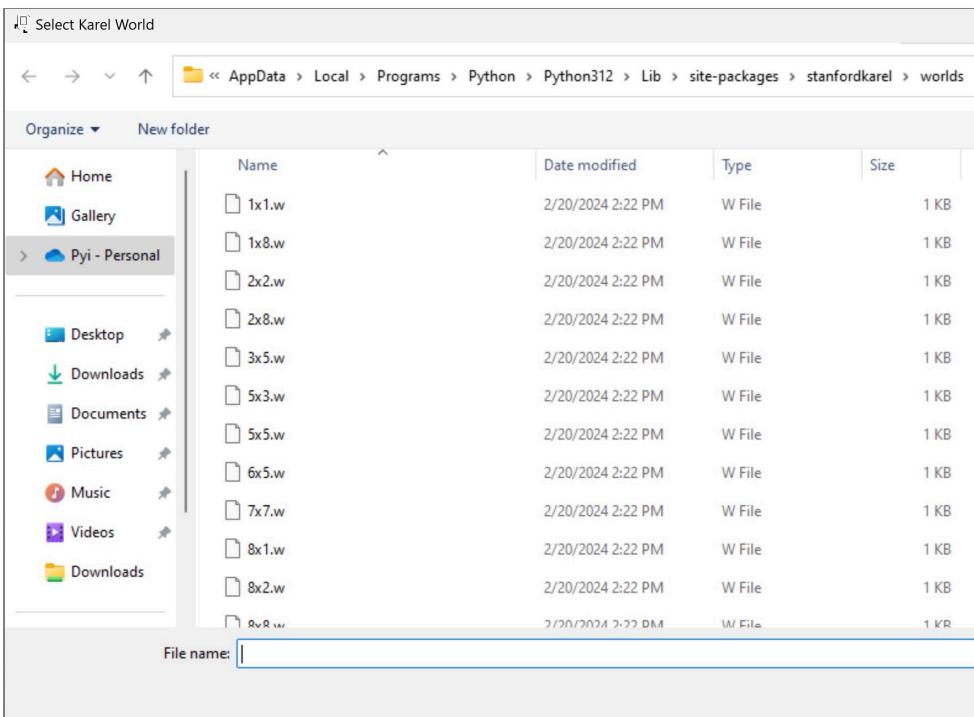
ပုံ က/၂

နောက်ဆက်တွဲ ခ

ကားရဲလ်ပရီဂရမ် ဖီချာများ

ကားရဲလ် ကဗျာဖိုင်များ

ကားရဲလ်ပရီဂရမ် ဝင်းခွဲးမှာ **Load World** ခလုတ်နှုပ်ပြီး ကဗျာဖိုင်အသစ် တင်လိုရတယ်။ ခလုတ်နှုပ်လိုက် ရင် အခုလို ဖိုင် ခိုင်ယာလော့ဂုံးပွင့်လာမယ်။



ပုံ ၉/၁

ဒါက stanfordkarel လိုက်ဘရဲ သူနိုင်အရို့အတိုင်း ပါတဲ့ worlds ဖို့ပါ။ ဖိုင်တွေက **.w** နဲ့ ဆုံးပါတယ်။ ကဗျာဖိုင်တွေကို ပင်မ ပရောဂျက်အောက် worlds ဖို့ပါထဲမှာ ထားလိုလည်းရတယ်။ အခြားနေရာတွေမှာ ထားလိုတော့ မရဘူး။

စာအုပ်ပါ ဥပမာတွေ၊ လောကျင့်ခန်းတွေအတွက် ကမ္မာဖိုင်တွေကို ပရောဂျက် တစ်ခုချင်းအလိုက် သီးခြား worlds ဖို့ဒါထဲမှာ ထည့်ပေးထားမှာပါ။ ပရိုဂ်ရမ်တစ်ခုဟာ ကမ္မာတစ်ခုတည်းမှာပဲ အလုပ်လုပ်တာမဟုတ်ဘဲ အလားတူ ကမ္မာအမျိုးမျိုးအတွက် အလုပ်လုပ်အောင် ရေးပေးရတာ။ အခုပြောတာကို နား မလည်ရင် အခန်း (၂) ဖတ်ပြီးရင် နားလည်သွားမှာပါ။

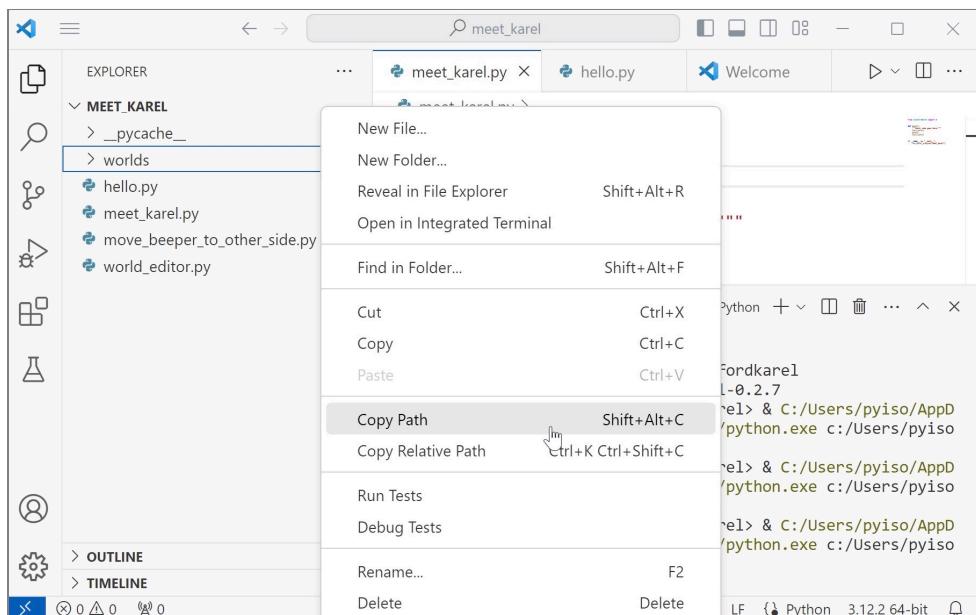
Load World လုပ်တဲ့အခါ ပွင့်လာတဲ့ ဖိုင် ခိုင်ယာလော်ကာ ကိုယ်လိုချင်တဲ့ worlds ဖို့ဒါ မဖြစ် နေဘူး။ သူနိုင်ပါတဲ့ worlds ဖို့ဒါ ဖြစ်နေတယ်။ ကိုယ်ခေါ်တင်ချင်တဲ့ ဖိုင်တွေရှိတဲ့ လက်ရှိပရောဂျက်ရဲ့ worlds ဖို့ဒါကို သွားရမယ်။ ဥပမာ PyCharm/VS Code အတွက် MeetKarel/meet_karel ပရောဂျက် worlds ဖို့ဒါ လမ်းကြောင်း အပြည့်အစုံက

C:\Users\yourname\VSCode\meet_karel\worlds

C:\Users\yourname\PycharmProjects\MeetKarel\worlds

ဖြစ်ပါမယ်။ ဖိုင်ခိုင်ယာလော်ကနေ ဖော်ပြပါ လက်ရှိပရောဂျက် worlds ဖို့ဒါကို တစ်ဆင့်ချင်း သွားပြီး တင်ချင်တဲ့ ကမ္မာဖိုင် (.w ဖိုင်) ကို ရွေးရမှာပါ။

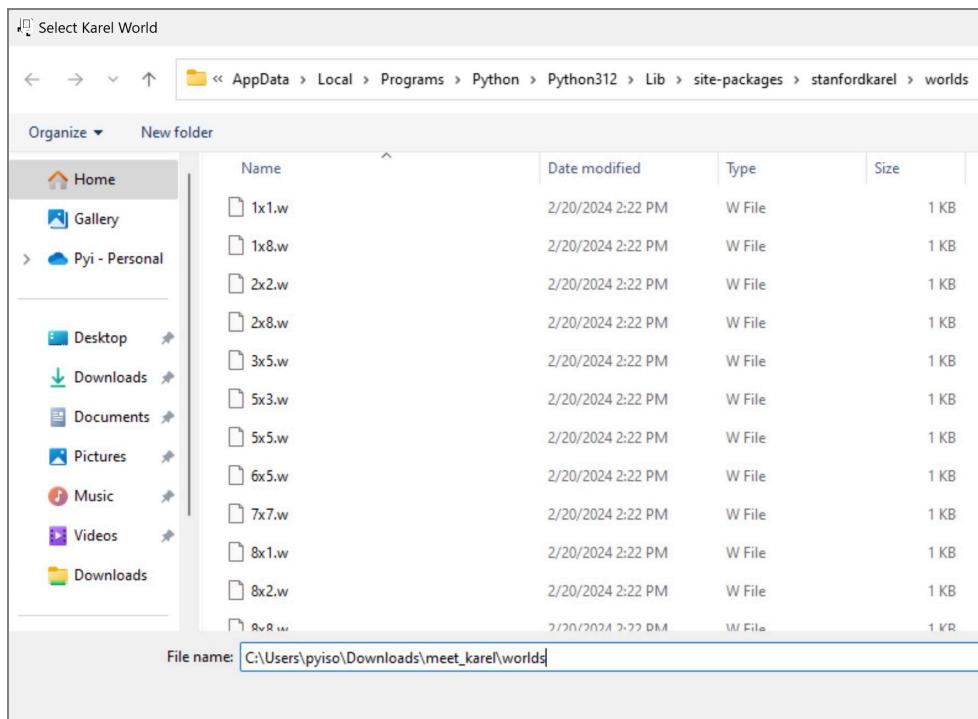
အပေါ်ကနည်းနဲ့ အဆင်မပြော့ရင် အခုလိုစမ်းကြည့်ပါ။ လက်ရှိပရောဂျက် worlds ဖို့ဒါကို ညာ ကလစ်နိုပ်ပြီး Copy Path လုပ်ပါ (ပုံ ခ/၂)။ ဖိုင် ခိုင်ယာလော် **File name** မှာ ကူးထည့်ပါ (ပုံ ခ/၃)။ **Enter** ကိုနိုပ်ပါ။ ပရောဂျက် worlds ဖို့ဒါကိုရောက်သွားပါမယ်။ လိုတဲ့ကမ္မာဖိုင် ရွေးတင်ရုံပါပဲ။ ပုံ (ခ/၄) မှာ meet_karel worlds ကို နှုနာပြထားပါတယ်။



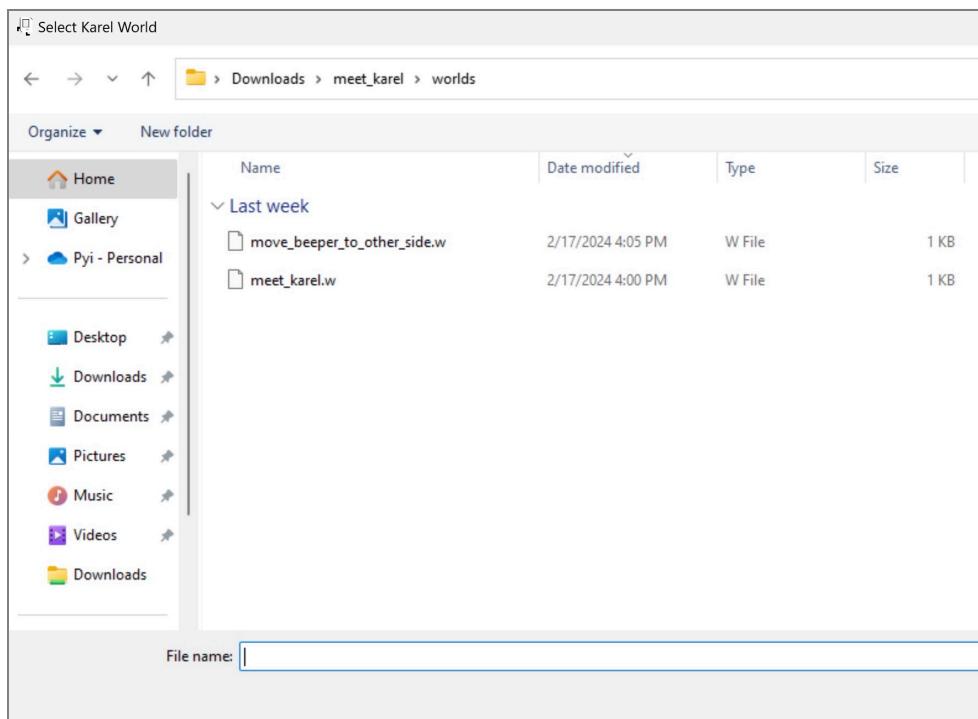
ပုံ ခ/၂

အကယ်၍ အထက်ဖော်ပြပါနည်းတွေက ရှုပ်နေတယ်ထင်ရင် မှတ်ရ/သွားရ လွယ်တဲ့ Desktop, Downloads, Documents လို နေရာတစ်ခွဲမှာ သီးသန်ဖို့ဒါတစ်ခု ဆောက်ပြီး ပရောဂျက်အားလုံး ထည့်ထားတာ အရှင်းဆုံးပါပဲ။ ပရောဂျက်ဖို့ဒါနေရာ သိရင် ဖိုင်ခိုင်ယာလော်ကနေ ဘယ်လိုမဆို ရောက်အောင် သွားလို့ရပါတယ်။

J9R



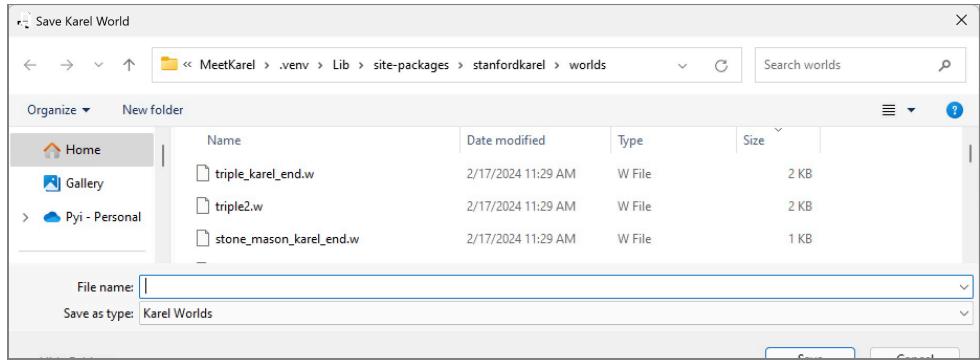
ڦ ۾/ڦ



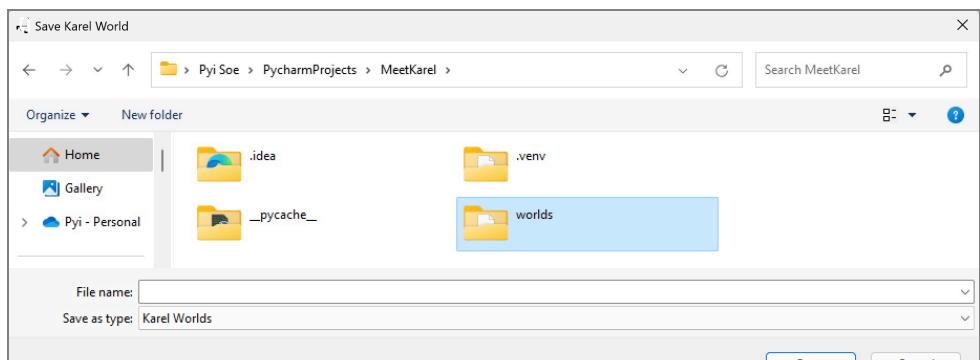
ڦ ۾/ڦ

ကိုယ်ပိုင် ကားရဲလ်ကမ္ဘာ ဆောက်ခြင်း

ကားရဲလ်ရဲ ကမ္ဘာ အသစ်တစ်ခုဆောက်မယ်ဆိုရင် world_editor.py ဖိုင်ကို ညာကလစ်နှင့် Run ပါ။ Would you like to load an existing world? လို ပေါ်လာပြီး Yes/No ရွှေးချင်ပါလိမ့်မယ်။ No ကို နှိပ်ပါ။ ကမ္ဘာအရွယ်အစားအတွက် ကော်လံ ဘယ်နှစ်ခုလဲ row ဘယ်နှစ်ခုလဲ ထည့်ပေးပါ။ ကိုယ်ပိုင် ကားရဲလ်ကမ္ဘာ တည်ဆောက်လိုရတဲ့ ဝင်းဒီးပွင့်လာပါလိမ့်မယ်။ ကားရဲလ် မျက်နှာမူရာအရပ်၊ ဘိပါအိတ်ထဲရှိ ဘိပါအရေအတွက်၊ နံရုံဆောက်/ဖျက်တာ၊ ဘိပါထည့်/ဖယ်ထုတ်တာ စတာတွေ လုပ်နှင့်ပါတယ်။ Save World နှိပ်ပြီး သိမ်းခိုင်ပါတယ် ဖိုင်ကိုသိမ်းတဲ့အခါ သုန္တရိ သိမ်းခိုင်းတဲ့ဖို့ (default world folder) ထဲမှာ သို့မဟုတ် ပင်မ ပရောဂျက်ဖို့တဲ့က worlds ဖို့ဒါထဲမှာ သိမ်းရပါမယ်။



ပုံ ၉/၅



ပုံ ၉/၆

