Begin Modern Programming with Python

Pyi Soe

မာတိကာ

အခန်း ၁

စက်ရုပ်ကားရဲလ်ဖြင့် ပရိုဂရမ်းမင်းမိတ်ဆက်

ကွန်ပျူတာတွေဟာ သက်မဲ့ စက်ပစ္စည်းတွေပါပဲ။ ကားတို့၊ လေယာဉ်တို့နဲ့ မတူတာက ကွန်ပျူတာတွေဟာ စက်ချည်းသက်သက် ဘာအစွမ်းမှ မယ်မယ်ရရ မရှိဘူး။ ဒါပေမဲ့ ဆောင်ရွက်လိုတဲ့ ကိစ္စအဝဝအတွက် ပရိုဂရမ်အမျိုးမျိုး ထည့်ပေးလိုက်တဲ့အခါမှာ သူ့ရဲ့အစွမ်းက အတိုင်းအဆမဲ့ပဲ။ နေရာမျိုးစုံ၊ နယ်ပါယ်မျိုးစုံမှာ အကူအညီပေးနိုင်တဲ့ စွယ်စုံသုံး ပစ္စည်းတစ်ခုဖြစ်သွားတယ်။ ဂီတသံစဉ်တွေကို ဖွင့်ပေးနိုင်သလို အသံလည်းသွင်းပေးနိုင်တယ်။ ရုပ်ရှင်တည်းဖြတ် လုပ်ချင်တာလား။ ပြဿနာမရှိဘူး၊ ကူညီပေးနိုင်တယ်။ နျူကလီးယား ဓါတ်ပေါင်းဖိုတွေကို စီမံနိုင်သလို မောင်းသူမဲ့ ဒုံးပျံတွေကိုလည်း ပဲ့ထိန်းပေးနိုင်တယ်။

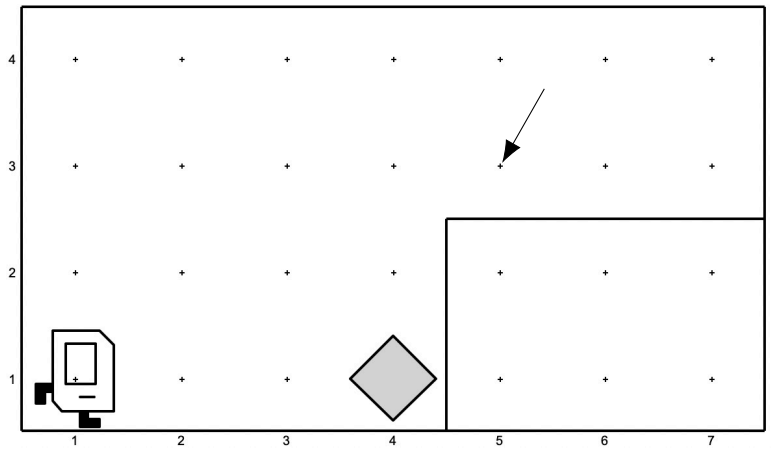
ကျွန်တော်တို့တွေ နိစ္စဓူဝ အသုံးပြုနေကြတဲ့ ကား၊ စမတ်ဖုန်း၊ လက်ပါတ်နာရီ၊ မိုက်ခရိုဝေဖ်မီးဖို၊ အဝတ်လျှော်စက် စတဲ့ စက်ပစ္စည်း အမျိုးမျိုးဟာလည်း ကွန်ပျူတာတွေနဲ့ မကင်းပြန်ပါဘူး။ "ကွန်ပျူတာနည်းပညာ အကူအညီမပါတဲ့ ခေတ်မီဆန်းသစ်တီထွင်မှုဆိုတာ မရှိဘူး" လို့ ဆိုနိုင်ပါတယ်။

တစ်ချက်တစ်ချက် ရိုက်ခတ်လိုက်တဲ့ ကွန်ပျူတာနည်းပညာ လှိုင်းလုံးကြီးတွေဟာ ကမ္ဘာတစ်ဝှမ်းလုံး ပုံစံပြောင်းသွားလောက်အောင် အဟုန်ပြင်းထန်လှတယ်။ ဘီလီယံနဲ့ချီတဲ့ လူတွေ ဆိုရှယ်မီဒီယာတွေပေါ်က နေ ရုပ်သံတွေနဲ့ ချိတ်ဆက်ပြောဆိုဆက်သွယ်လို့ ရစေတာဟာလည်း ကွန်ပျူတာစနစ်တွေပါပဲ။ Artificial Intelligence (AI) နည်းပညာကြောင့် သက်ရှိတွေမှာပဲတွေ့ရတဲ့ ညာဏ်ရည်မျိုးကို ကွန်ပျူတာတွေမှာလည်း တွေ့လာရပါပြီ။ သခ်ၤာပုစ္ဆာတွေ ဖြေရှင်းခြင်း၊ စစ်တုရင်ထိုးခြင်း စတဲ့ကိစ္စမျိုးတွေအပြင် ပန်းချီဆွဲခြင်း၊ ကဗျာရေးစပ်ခြင်း၊ သီချင်းရေးဖွဲ့ခြင်း ကဲ့သို့ အနုပညာဖန်တီးမှုတွေကိုပါ AI က လုပ်ဆောင်ပေးနိုင်ပါတယ်။ နှစ်ဆယ့်တစ်ရာစုရဲ့ အထူးခြားဆုံး AI နည်းပညာလှိုင်းဟာ အရှိန်အဟုန်ပြင်းပြင်း ရိုက်ခတ်ဖို့ အားယူစ ပြုနေပါပြီ။

'ကွန်ပျူတာ' လို့ပြောတဲ့အခါ စက်ပစ္စည်းသက်သက် မဟုတ်ဘဲ ကွန်ပျူတာမှတ်ညာဏ်ထဲက ပရိုဂရမ်တွေလည်း ပါဝင်တယ်ဆိုတာ သတိချပ်ရပါမယ်။ ကွန်ပျူတာတွေ တစ်စုံတစ်ရာ စွမ်းဆောင်နိုင်စေတဲ့ ပရိုဂရမ်တွေ ရေးတဲ့အလုပ်ကို ပရိုဂရမ်းမင်း (Programming) လို့ခေါ်တယ်။

၁.၁ စက်ရုပ် ကားရဲလ်

ပရိုဂရမ်းမင်းဆိုတာ ဘယ်လိုမျိုးလဲ သဘောပေါက်အောင် စာတွေတစ်သီကြီးရေး ရှင်းပြတာထက် ပရိုဂရမ်လေးတွေ လက်တွေ့ ရေးကြည့်လိုက်တာ ပိုပြီးထိရောက်ပါတယ်။ ဒါကြောင့် စက်ရုပ်ကားရဲလ်ကို ပရိုဂရမ်လေးတွေရေးပြီး အလုပ်တွေခိုင်းကြည့်ကြမယ်။ ပုံ (၁.၁) မှာ တွေ့ရတာက ကားရဲလ် ရောက်ရှိနေတဲ့ နမူနာကမ္ဘာတစ်ခုပါ။ မီးခိုးရောင် မှန်ကူကွက်ပုံလေးကို ဘီပါ (beeper) လို့ ခေါ်တယ်။ အဲဒီဘီပါကို မြှားပြထားတဲ့ နေရာကို ရွှေ့ခိုင်းချင်တယ်။ မျဉ်းမည်းအထူတွေက နံရံတွေပါ။ ကားရဲလ်ကို ကိစ္စတစ်ခု ဆောင်ရွက်စေ



ပုံ ၁.၁ စက်ရုပ်လေး ကားရဲလ်

ချင်တဲ့အခါ အခြေခံ ကားရဲလ်ကွန်မန်းတွေကို အသုံးပြုရပါတယ်။ ကွန်မန်းတွေကို နူတ်နဲ့ပြောပြီး ခိုင်းရတာ မဟုတ်ဘဲ ပရိုဂရမ်ရေးပြီး ခိုင်းရတာပါ။ ကားရဲလ်နားလည်တဲ့ ကွန်မန်းတွေကို ကြည့်ကြရအောင်။

ကားရဲလ်ကွန်မန်းများ

မဖြစ်မနေ သိထားရမဲ့ အခြေခံ ကားရဲလ်ကွန်မန်း လေးခုပဲ ရှိတယ်။ `move`, `turn_left`, `put_beeper` နဲ့ `pick_beeper` တို့ဖြစ်တယ်။ အခြား ကားရဲလ်ကွန်မန်း တွေလည်း ရှိပါသေးတယ်။ ဒါပေမဲ့ ကားရဲလ် ပရိုဂရမ်းမင်း စလေ့လာဖို့ ဒီလေးခုနဲ့ပဲ လုံလောက်ပါပြီ။

`move` ကွန်မန်းက ကားရဲလ်ကို ရှေ့တစ်ကွန်နာကို ရွှေ့ခိုင်းတာ။ ကားရဲလ်ကမ္ဘာထဲမှာ တစ်ခုနဲ့တစ်ခု အကွာအဝေးတူ ခြားထားတဲ့ အတန်းလိုက် အတန်းလိုက် အစက်ကလေးတွေဟာ ကွန်နာ (corner) တွေ ဖြစ်တယ်။ ကမ္ဘာကို မျဉ်းမည်းအထူ နံရံတွေနဲ့ ထောင့်မှန်စတုဂံပုံ ပါတ်လည် ဘောင်ခတ်ထားတယ်။ ကွန်နာတွေကြားမှာလည်း နံရံတွေရှိနိုင်တယ်။ နမူနာကမ္ဘာမှာ ဘေးတိုက် နံပါတ်စဉ် ၄ နဲ့ ၅ ကြား ထောင်လိုက် နံရံတစ်ခု၊ အထက်အောက် နံပါတ်စဉ် ၂ နဲ့ ၃ ကြား အလျားလိုက် နံရံတစ်ခုကို တွေ့ရပါမယ်။ ကွန်နာရှေ့မှာ နံရံကာနေရင် ကားရဲလ်ကို `move` ခိုင်းလို့မရပါဘူး။

`put_beeper` က ကားရဲလ် လက်ရှိ ရှိနေတဲ့ ကွန်နာမှာ 'ဘီပါတစ်ခုချ' ထားခိုင်းတာ၊ `pick_beeper` က ရပ်နေတဲ့ ကွန်နာမှာ 'ဘီပါတစ်ခုကောက်' ခိုင်းတာပါ။ ကွန်နာမှာ ဘီပါရှိနေမှ ကောက်ခိုင်းလို့ရမှာပါ။ မရှိရင် ကောက်ခိုင်းလို့ မရဘူး။ ဘီပါချခိုင်းရင်လည်း ကားရဲလ်မှာ ဘီပါရှိမှ ချခိုင်းလို့ရတယ်။ ကားရဲလ်ကို ဘီပါတွေ လိုသလောက် ဖြည့်ပေးထားတယ်လို့ ယူဆပါ။ `turn_left` က 'ဘယ်လှည့်' ခိုင်းတာ။

ဘီပါကို ဘယ်လိုရွှေ့ခိုင်းမလဲ

ပုံ (၁.၁) အနေအထားကနေ ရှေ့ကို သုံးနေရာရွှေ့၊ ဘီပါကောက်၊ ဘယ်ဘက်လှည့်၊ အပေါ် နှစ်နေရာရွှေ့၊ ညာဘက်လှည့်၊ ရှေ့တစ်နေရာထပ်ရွှေ့ပြီး ဘီပါချထားခိုင်းလိုက်ရင် အလုပ်ပြီးသွားပါပြီ။

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

၁.၂ Meet Karel ပရိုဂရမ်

ပရိုဂရမ် ရေးတယ်ဆိုတာ ကွန်ပျူတာကို ကိစ္စတစ်ခုခု ဆောက်ရွက်ပေးဖို့ ခိုင်းစေတဲ့ ညွှန်ကြားချက်တွေ ရေးတာပါပဲ။ ဒီလို ညွှန်ကြားချက်တွေကို ပရိုဂရမ်ကုဒ် (*program code*) လို့ ခေါ်တယ်။ ပရိုဂရမ်ကုဒ်တွေကို ကွန်ပျူတာနားလည်တဲ့ programming language တစ်ခုခုနဲ့ ရေးရတယ်။ ဒီစာအုပ်မှာ အသုံးပြုမဲ့ programming language ကတော့ Python ပါ။ Programming language တစ်မျိုးပဲ ရှိတာ မဟုတ်ပါဘူး။ ရာနဲ့ချီပြီး ရှိတာပါ။ လူဘာသာစကားတွေ အမျိုးမျိုးရှိသလိုပဲပေါ့။ Python ဟာ ဒီလို ရာနဲ့ချီတဲ့ထဲက လက်ရှိအသုံးအများဆုံး ထိပ်ဆုံးဆယ်ခု ထဲမှာ ပါဝင်တယ်။ Python နဲ့ ဘိပါရွှေခိုင်းတဲ့ ပရိုဂရမ်ကို လေ့လာကြည့်ရအောင်။ ကားရဲလ်နဲ့ ပထမဆုံး မိတ်ဆက်ပေးတဲ့ ပရိုဂရမ်မို့လို့ ဒီပရိုဂရမ် နံမည်ကို 'Meet Karel' လို့ 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' လို့ အတိုချုံးပဲ ပြောတာများတယ်။ Python 'စာ/စကား' မတတ်ရင် ဒီ 'Python ကုဒ်' တွေကိုလည်း နားလည်မှာ မဟုတ်ဘူး။ ဒီတော့ Python 'စာ/စကား' အခြေခံက စပြီး လေ့လာဖို့ လိုပါမယ်။

ကွန်းမန့် (Comment)

ပထမဆုံး # သင်္ကေတနဲ့ စတဲ့ စာကြောင်းတွေက ကွန်းမန့်တွေပါ။ ကွန်းမန့်တွေက ကွန်ပျူတာ ဆောင်ရွက်ပေးရမဲ့ ညွှန်ကြားချက်တွေ မဟုတ်ဘူး။ ပရိုဂရမ်ကုဒ်နဲ့ ပါတ်သက်ပြီး ကုဒ် ဖတ်ရှုသူ အတွက် မှတ်ချက်

ရေးတာ သို့မဟုတ် ရှင်းပြထားတာပါ။ တနည်းအားဖြင့် ဖတ်ရှုသူ (လူ) ပရိုဂရမ်မာအတွက် ရည်ရွယ်တာ။ ကွန်ပျူတာ (စက်) အတွက် ရည်ရွယ်တာ မဟုတ်ဘူး။ ကွန်ပျူတာက ကုဒ်ထဲက ကွန်းမန့်တွေ အားလုံးကို လစ်လျူရှုမှာပါ။ ဒါပေမဲ့ ပရိုဂရမ်ကုဒ်ကို ဖတ်တဲ့လူ နားလည်ဖို့ အထောက်အကူ ဖြစ်စေတဲ့အတွက် ကွန်းမန့်ရေးတာကို ပေါ့ပေါ့တန်တန် အရေးမပါသလို သဘောထားလို့ မရပါဘူး။ မိမိရေးတဲ့ ကုဒ်ကို ရှင်းပြဖို့ လိုအပ်ရင် ကွန်းမန့်ရေးရပါမယ်။ ရေးသင့်တဲ့ နေရာတွေကိုလည်း မကြာခင်တွေ့ရမှာပါ။

import စတိတ်မန့်

```
from stanfordkarel import *
```

ကတော့ အင်ပို့စတိတ်မန့် ဖြစ်ပါတယ်။ "stanfordkarel လိုက်ဘရီမှ အားလုံးကို ထည့်သွင်းပေးပါ" လို့ တောင်းဆိုတဲ့ အဓိပ္ပါယ်။ * သင်္ကေတကို 'အားလုံး' လို့ ယူဆပါ။ stanfordkarel လိုက်ဘရီမှာ ကား ရဲလ်ပရိုဂရမ်အတွက် လိုအပ်တာအားလုံး ပါဝင်တယ်။ ဒီလိုက်ဘရီကို အင်ပို့လုပ်ထားမှ ကားရဲလ်ကွန်မန်း တွေ သုံးလို့ရမှာပါ။ သီးခြား ကားရဲလ်ပရိုဂရမ် တစ်ခုစီတိုင်းအတွက် အင်ပို့လုပ်ရမှာ ဖြစ်တယ်။

လိုက်ဘရီ

လိုက်ဘရီ (*library*) ဆိုတာ ပညာရပ်နယ်ပယ် တစ်ခုအတွက် ရည်ရွယ်ရေးထားတဲ့ ပရိုဂရမ်ကုဒ်တွေ ပါပဲ။ သင်္ချာအတွက်အချက် လိုက်ဘရီ၊ ဂိမ်းရေးဖို့ လိုက်ဘရီ၊ 2D/3D ဂရပ်ဖစ်ဆွဲဖို့ လိုက်ဘရီ၊ အေ အိုင်အတွက် လိုက်ဘရီ စသည်ဖြင့် နယ်ပယ်အသီးသီး၊ ကိစ္စရပ်အဖုံဖုံအတွက် သက်ဆိုင်ရာ ကျွမ်းကျင် ပညာရှင်တွေ ထုတ်လုပ်ဖြန့်ချီပေးထားတဲ့ လိုက်ဘရီတွေရှိတယ်။ Matrix အော်ပရေးရှင်းတွေအတွက် numpy ၊ ဂရပ်ဖစ်ဆွဲမယ်ဆိုရင် matplotlib စတဲ့ လိုက်ဘရီတွေကို အင်ပို့လုပ် အသုံးပြုနိုင်ပါတယ်။ မေ့ ထရစ် A ကို B နဲ့ မြှောက်ရင် 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)
```

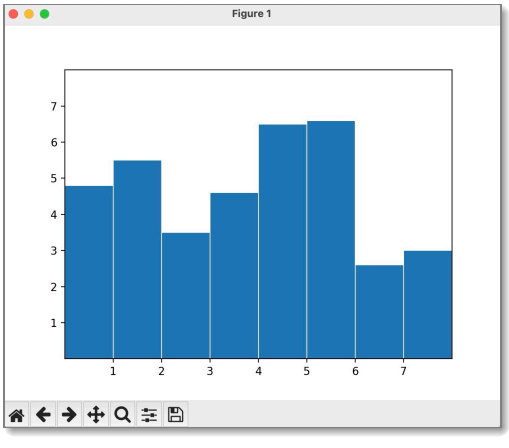
ရလဒ် အခုလိုထွက်ပါမယ်။

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

ဒါကတော့ ဘားချတ် 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()
```

ဘားချတ်ကို ဒီလို ထုတ်ပေးပါတယ်။



ပုံ ၁.၂

လိုက်ဘရီတွေဟာ ပရိုဂရမ်တွေ တည်ဆောက်ရာမှာ အင်မတန်မှ အရေးပါတယ်။ ဖော်ပြထားတဲ့ မှေ့ထရစ် နဲ့ ဘားချတ် ကုဒ်တွေကို (အခုတော့) နားလည်မှာ မဟုတ်သေးဘူးပေါ့။ ဒါပေမဲ့ သက်ဆိုင်ရာ လိုက်ဘရီတွေနဲ့ ဒီလိုကိစ္စတွေကို သိပ်မခက်ခဲဘဲ လုပ်လို့ရနိုင်တယ်ဆိုတာ မြင်မယ် ထင်ပါတယ်။ လိုက်ဘရီတွေ သာမရှိရင် ပရိုဂရမ်တွေကို အခုထက် အဆပေါင်းများစွာ အချိန်ပေးပြီး ရှုပ်ရှုပ်ထွေးထွေး ခက်ခက်ခဲခဲ တည်ဆောက်ကြရမှာပါ။

တုံကင်၊ စတိတ်မန့်နှင့် ဆင်းတက်စ်

ဂျန်ပန်စာ၊ ပြင်သစ်စာ စတဲ့ လူဘာသာစကားတွေဟာ စကားလုံးတွေ ဝါကျတွေနဲ့ ဖွဲ့စည်းထားသလို ပရိုဂရမ်ကုဒ်တွေဟာလည်း စကားလုံးတွေ၊ ဝါကျတွေနဲ့ ဖွဲ့စည်းထားတာပါပဲ။ Programming language တွေမှာ စကားလုံးတွေကို တုံကင် (*token*) လို့ခေါ်ပြီး ဝါကျတွေကိုတော့ စတိတ်မန့် (*statement*) လို့ ခေါ်ပါတယ်။ ဝါကျတွေကို စကားလုံးတွေနဲ့ ဖွဲ့စည်းထားသလို စတိတ်မန့်တွေကတော့ တုံကင်တွေနဲ့

ဖွဲ့စည်းထားတာပါ။ စတိတ်မန့် ပုံစံတစ်မျိုးကို တွေ့ခဲ့ပြီးပါပြီ။ အဲဒါကတော့ ရှေ့စာမျက်နှာက အင်ပို့စတိတ်မန့်ပဲ ဖြစ်ပါတယ်။

လူ့ဘာသာစကားတွေမှာ ဖွဲ့စည်းတည်ဆောက်ပုံ စထရက်ချာရှိသလို programming language တွေမှာလည်း စထရက်ချာရှိဖို့ လိုအပ်တာပေါ့။ ဖွဲ့စည်းပုံ စထရက်ချာ မှန်/မမှန်ကို သဒ္ဒါစည်းမျဉ်တွေနဲ့ ထိန်းကွပ်ထားတာပါ။ ပရိုဂရမ်ကုဒ် စထရက်ချာ မှန်/မမှန် ထိန်းကွပ်ပေးတဲ့ သဒ္ဒါစည်းမျဉ်တွေကိုတော့ ဆင်းတက်စ် (*syntax*) လို့ခေါ်တယ်။

မြန်မာလိုရေးရင် မြန်မာသဒ္ဒါကို လိုက်နာရသလို Python နဲ့ ရေးရင် Python ဆင်းတက်စ်ကို လိုက်နာရမှာပေါ့။ မြန်မာသဒ္ဒါမှားရင် ဖတ်တဲ့သူက သည်းခံနားလည် ပေးပေမဲ့ ဆင်းတက်စ်မှားရင်တော့ Python က လုံးဝ လက်ခံမှာ မဟုတ်ပါဘူး။ ဆင်းတက်စ် စည်းကမ်းတွေဟာ ပိုပြီး တင်းကျပ်တယ်။ လွဲချော်လို့ မရဘူး။ ဆင်းတက်စ်မှားနေတဲ့ ပရိုဂရမ်ကို Python က run ခွင့် ပြုမှာမဟုတ်ဘဲ အမှားနဲ့ သက်ဆိုင်တဲ့ အယ်ရာမက်ဆေ့ချ်တွေ ပြပေးမှာပါ။ ဖြစ်လေ့ရှိတဲ့ ဆင်းတက်စ်အမှားတွေကို မကြာခင် တွေ့ရပါမယ်။

Keywords

`from`, `import`, `def` , `if` စတာတွေဟာ *keyword* တွေဖြစ်တယ်။ Python ရေးတဲ့အခါ သူ့နေရာနဲ့ သူ အဓိပ္ပါယ်ကိုယ်စီနဲ့ အသုံးပြုရတဲ့ စကားလုံးတွေဖြစ်တယ်။ `from` နဲ့ `import` ကို လိုက်ဘရီ အင်ပို့လုပ်ဖို့ သုံးတယ်။ `def` ကို ဖန်ရှင် သတ်မှတ်တဲ့အခါ သုံးတယ်။ Python က သတ်မှတ်ထားတဲ့ နေရာတွေက လွဲလို့ အခြားကိစ္စတွေအတွက် keyword တွေကို အသုံးပြုလို့ မရပါဘူး။ ဒါကြောင့် keyword တွေကို reserved word လို့လည်း ခေါ်တယ်။

`main` ဖန်ရှင်

'Meet Karel' ပရိုဂရမ် အင်ပို့စတိတ်မန့် အပြီးမှာ တွေ့ရတာကတော့ `main` ဖန်ရှင်သတ်မှတ်ချက်ပါ။ ကြည့်ရအဆင်ပြေအောင် သူ့ချည်းသီးသန့် 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 ပရိုဂရမ်ကုဒ် အစုအဝေးတစ်ခုပါပဲ။ ဖန်ရှင်ကို အသုံးပြုတဲ့အခါ ၎င်းရဲ့ လုပ်ငန်းတာဝန်အတိုင်း ဖန်ရှင်က လုပ်ဆောင်ပေးမှာ ဖြစ်တယ်။ ဖန်ရှင်အသုံးပြုတာကို 'ဖန်ရှင်ကောလ်' (*function call*) လုပ်တာလို့ ပြောတယ်။

`main` ဖန်ရှင်သတ်မှတ်ချက်ကို အပိုင်းနှစ်ပိုင်းခွဲ ကြည့်နိုင်တယ်။ ပထမတစ်ပိုင်း

```
def main():
```

ကို ဖန်ရှင်ခေါင်းစီး (function header) လို့ခေါ်တယ်။ ဖန်ရှင်ခေါင်းစီးမှာ ဖန်ရှင်နံမည်နဲ့ ဖန်ရှင်ပါရာမီတာတွေကို ဝိုက်ကွင်းထဲမှာ သတ်မှတ်ပေးရပြီး colon ':' နဲ့ အဆုံးသတ်တယ်။ ဥပမာ `x, y` ပါရာမီတာ နဲ့ `myfun` ဖန်ရှင် အတွက်

```
def myfun(x, y):
```

ပါရာမီတာမပါရင်လည်း ဝိုက်ကွင်းအလွတ် တစ်စုံ '()' တော့ပါရမယ်။ `main` ဖန်ရှင်မှာ ပါရာမီတာ မပါဘူး။ ပါရာမီတာတွေအကြောင်း နောက်ပိုင်းအခန်းတွေမှာ အသေးစိတ် လေ့လာရမှာပါ။ ကားရဲလ်ပရိုဂရမ်တွေမှာ ပါရာမီတာအကြောင်း သိဖို့မလိုသေးပါဘူး။ ပါရာမီတာ မလိုတဲ့ ဖန်ရှင်တွေပဲ တွေ့ရမှာပါ။

ဖန်ရှင်ခေါင်းစည်းအောက် ဒုတိယပိုင်းကတော့ `main` ဖန်ရှင် ကုဒ်ဘလောက် (*code block*) ပါ။ ကုဒ်ဘလောက် ဆိုတာ ကုဒ်တွေကို အုပ်စုတစ်စု အဖြစ် ဖွဲ့စည်းထားတာကို ဆိုလိုတာပါ။ ဘလောက်လို့လည်း ခေါ်တယ်။ ဘလောက်တစ်ခုမှာ ပါဝင်တဲ့ ကုဒ်လိုင်းတွေကို ညာဘက် ခွာရေးရပါမယ်။ အင်ဒန့်ထ် (indent) လုပ်တာလို့ ခေါ်တယ်။ တက်ဘ်ကီးတစ်ချက် (သို့) စပေ့စ်လေးခုစာ ခွာလေ့ရှိတယ်။ ဘော်ဒီ ပထမတစ်ကြောင်း

```
"""Karel code goes here!"""
```

ကို *docstring* လို့ ခေါ်တယ်။ quote သုံးခုတွဲ '"""' တစ်စုံကြား ညှပ်ရေးတဲ့ ကွန်းမန့်တစ်မျိုးလို့ ယူဆနိုင်တယ်။ ဖန်ရှင်နဲ့ ပါတ်သက်တဲ့ ရှင်းလင်းဖော်ပြချက်တွေ ရေးဖို့အတွက် သုံးတာပါ။

Docstring အောက်မှာ တွေ့ရတာကတော့ ကားရဲလ်ကွန်မန်းတွေဆိုတာ သိပါလိမ့်မယ်။ ကားရဲလ် ကွန်းမန်းတွေဟာ `stanfordkarel` လိုက်ဘရီ ဖန်ရှင်တွေပါ။ တနည်းအားဖြင့် `stanfordkarel` လိုက်ဘရီမှာ ကားရဲလ်ကွန်းမန်းတွေအတွက် ဖန်ရှင်သတ်မှတ်ချက်တွေ ပါဝင်တယ်။ ဖန်ရှင်တစ်ခုကို အသုံးပြုဖို့ အတွက် အဲဒီဖန်ရှင်ကို ခေါ်ရပါတယ်။ ဒါကို ဖန်ရှင်ကောလ် (*function call*) လုပ်တယ်လို့ ပြောတယ်။ ကားရဲလ်ကို ဘယ်ဘက်လှည့်စေချင်ရင် `turn_left` ဖန်ရှင်ကောလ် လုပ်ရပါမယ်။ ဘီပါကောက်ခိုင်းချင်ရင် `pick_beeper` ဖန်ရှင်ကောလ် လုပ်ရပါမယ်။ ဖန်ရှင်ကောလ် လုပ်တဲ့ ပုံစံက

```
turn_left()
pick_beeper()
```

စသည်ဖြင့် ဖြစ်တယ်။

Python မှာ အင်ဒန့်ထ်ကို ဖြစ်ကတတ်ဆန်း လုပ်လို့မရဘူး။ ဘေးမျဉ်းကနေ ခွာတဲ့ အကွာအဝေး မညီတာနဲ့ ဆင်းတက်စ်အမှား ဖြစ်တယ်။ မလိုတဲ့နေရာမှာလည်း ခွာရေးလို့ မရဘူး။ ခေါင်းစီးကို ဘေးမျဉ်းနဲ့ ခွာထားကြည့်ပါ။ အယ်ရာဖြစ်တာကို တွေ့ရမယ်။ အင်ပို့စတိတ်မန့်လည်း ဘေးမျဉ်းနဲ့ ကွာနေလို့မရဘူး။ အခြား language တွေမှားလည်း အင်ဒန့်ထ်လုပ် ရေးကြပေမဲ့ Python မှာလောက် မတင်းကျပ်ဘူး။ အင်ဒန့်ထ်မလုပ်လည်း ဆင်းတက်စ်မှားတာ မဖြစ်ဘူး။

ကားရဲလ်ပရိုဂရမ်တစ်ခုမှာ `main` ဟာ အထူးတာဝန်တစ်ခု လုပ်ဆောင်ပေးရတယ်။ အဲဒါကတော့ ပရိုဂရမ်ဝင်းဒိုးမှာ `Run Program` ခလုတ် (ပုံ ၁.၅ မှာကြည့်ပါ) နှိပ်လိုက်ရင် တုံ့ပြန် လုပ်ဆောင်ပေးရတာပါ။ ဒါကြောင့် ကွန်မန်းတွေဟာ အဲဒီခလုတ် နှိပ်တော့မှပဲ စအလုပ်လုပ်တာ ဖြစ်တယ်။

Entry Point

'Meet Karel' ပရိုဂရမ်မှာ `main` ဖန်ရှင်နောက် အောက်ဆုံးအပိုင်းဟာ ပရိုဂရမ် run တဲ့အခါ ပထမဆုံး စတင်လုပ်ဆောင်ပေးရမဲ့ ဖန်ရှင်ကို ဖော်ပြပေးတာပါ။ အန်ထရီပွိုင့် (*entry point*) လို့ခေါ်တယ်။

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

`run_karel_program` ဖန်ရှင်ဟာ ကားရဲပရိုဂရမ် တစ်ခုအတွက် အန်ထရီပွိုင့် ဖြစ်တယ်။ ပရိုဂရမ် တက်လာတာနဲ့ တစ်ပါတည်း ခေါ်တင်ချင်တဲ့ ကမ္ဘာကို ဒီဖန်ရှင်မှာ ထည့်ပေးတယ်။ meet_karel.w ကမ္ဘာကို စခချင်းခေါ်တင်ထားချင်ရင် `"meet_karel"` ထည့်ပေးရမယ်။ ကမ္ဘာဖိုင်မရှိရင် အယ်ရာတက်ပြီး ပရိုဂရမ်ပွင့်လာမှာ မဟုတ်ဘူး။ ကမ္ဘာမထည့်ပေးထားဘဲ ဒီလို

```
run_karel_program()
```

ဆိုရင် 8×8 အရွယ် default ကမ္ဘာကို တင်ပေးပါတယ်။

ကားရဲလ်ကမ္ဘာတစ်ခုကို လိုချင်တဲ့ပုံစံ ဒီဇိုင်းဆွဲပြီး ဖိုင်နဲ့ သိမ်းထားရတာပါ။ ကမ္ဘာ ပုံစံချတဲ့ ပရိုဂရမ်လည်း ရှိတယ်။ ကမ္ဘာဖိုင်တွေက .w ဖိုင် အိပ်စ်တန်းရှင်းနဲ့ ဖြစ်တယ်။ ဒီစာအုပ်မှာပါတဲ့ ဥပမာတွေ၊ လေ့ကျင့်ခန်းတွေ အားလုံးအတွက် လိုအပ်တဲ့ ကမ္ဘာတွေကို အဆင်သင့်ပေးထားမှာပါ။ ကိုယ့်ဟာကို လုပ်ဖို့ မလိုဘူး။ စိတ်ဝင်စားရင် စမ်းကြည့်လို့ရအောင် စာမျက်နှာ (**??**) နောက်ဆက်တွဲ (**??**) မှာ အကျဉ်းဖော်ပြပေးထားပါတယ်။

၁.၃ ကားရဲလ် ပရိုဂရမ် run ခြင်း

လိုအပ်တဲ့ဆော့ဖ်ဝဲတွေ ထည့်သွင်းနည်းကို စာမျက်နှာ (**??**) နောက်ဆက်တွဲ (**??**) မှာ တစ်ဆင့်ချင်း ဖော်ပြပေးထားပါတယ်။ အခုက Python ပရိုဂရမ်တစ်ခုကို အရိုးရှင်းဆုံး (လွယ်တယ်လို့ မဆိုလို) run လို့ရတဲ့ နည်းကိုဖော်ပြပေးမှာပါ။ သဘောတရားပိုင်း နားလည်ဖို့ အထောက်အပံ့ဖြစ်မယ်။ အခုနည်းလမ်းကို အကြမ်းဖျဉ်း နားလည်အောင် ဖတ်ပြီးမှ နောက်ဆက်တွဲ (**??**) ကို ဖတ်စေချင်ပါတယ်။

မိုက်ခရိုဆော့ဖ် ဝင်းဒိုးမှာ Notepad ၊ အက်ပဲလ် မက်ခ်အိုအက်စ်မှာ TextEdit စတဲ့ တက်စ်အယ်ဒီတာတစ်ခုခုနဲ့ ပရိုဂရမ်ကုဒ်ရေးလို့ရတယ်။ ကုဒ်ဖိုင်ကို .py အိပ်စ်တန်းရှင်းနဲ့ သိမ်းရပါမယ်။ ပလိန်းတက်စ် (plain text) ဖိုင် ပါပဲ။ Python ကုဒ်ဖိုင်မို့လို့ .txt အစား .py နဲ့ သိမ်းတာပါ။ Python ဖိုင်နံမည်ကို စာလုံးအသေးနဲ့ပဲ ပေးတဲ့ ထုံးစံရှိတယ်။ စပေ့စ်နေရာမှာ _ (underscore) သုံးတဲ့ ထုံးစံရှိတယ်။ ဒါကြောင့် 'Meet Karel' ပရိုဂရမ်ကုဒ်ကို meet_karel.py ဖိုင်မှာ သိမ်းသင့်တယ်။

Python နဲ့ ရေးထားတဲ့ ပရိုဂရမ်ကို run မယ်ဆိုရင် Python ဆော့ဖ်ဝဲရှိရမှာပါ။ ဒီဆော့ဖ်ဝဲ အင်စတောလ် လုပ်နည်းကို နောက်ဆက်တွဲ (**??**) စာမျက်နှာ (**??**) မှာ ဖော်ပြပေးထားပါတယ်။ Python ကုဒ်တွေကို ကွန်ပျူတာက တိုက်ရိုက် နားမလည်ပါဘူး။ Python ဆော့ဖ်ဝဲဟာ Python ကုဒ်တွေကို တိုက်ရိုက် နားလည်ပြီး ကွန်ပျူတာပေါ်မှာ run လို့ရအောင် ကြားခံဆောင်ရွက်ပေးတဲ့ ဆော့ဖ်ဝဲလို့ ယေဘုယျအားဖြင့် ယူဆနိုင်တယ်။

Python ထည့်ပြီးရင် `stanfordkarel` လိုက်ဘရီကို အောက်ပါကွန်မန်းနဲ့ အင်စတောလ် လုပ်ရပါမယ်။ အင်တာနက်ပေါ်ကနေ ဒေါင်းလုဒ် လုပ်ရတာမို့လို့ ကွန်နက်ရှင်ရှိရမယ်။

```
pip install stanfordkarel
```

ကားရဲလ်ပရိုဂရမ်မှာ ခေါ်တင်ချင်တဲ့ ကမ္ဘာဖိုင်တွေလည်း ရှိရပါမယ်။ meet_karel.w ကမ္ဘာဖိုင်က meet_karel.zip ဖိုင် worlds ဖိုဒါထဲမှာ ရှိပါတယ်။ `http://tinyurl.com/3mmm9c7j` လင့်ကနေ meet_karel.zip ဖိုင်ကို ဒေါင်းလုဒ်လုပ်ပါ။ ဒီ zip ဖိုင်ထဲက worlds ဖိုဒါကို meet_karel.py ဖိုင်ရှိတဲ့ နေရာမှာ ကော်ပီကူးထည့်ပါ။ ဝင်းဒိုးမှာ Command Prompt ၊ မက်ခ်အိုအက်စ်မှာ Terminal ဖွင့်ပြီး cd ကွန်မန်းနဲ့ ကုဒ်ဖိုင်ထားတဲ့ ဖိုဒါထဲကို သွားပြီး အောက်ပါအတိုင်း python ကွန်မန်းနဲ့ ကုဒ်ဖိုင်ကို run ပေးရပါမယ် (လာမဲ့စာမျက်နှာမှာ နမူနာပြထားတာ ကြည့်ပါ)။

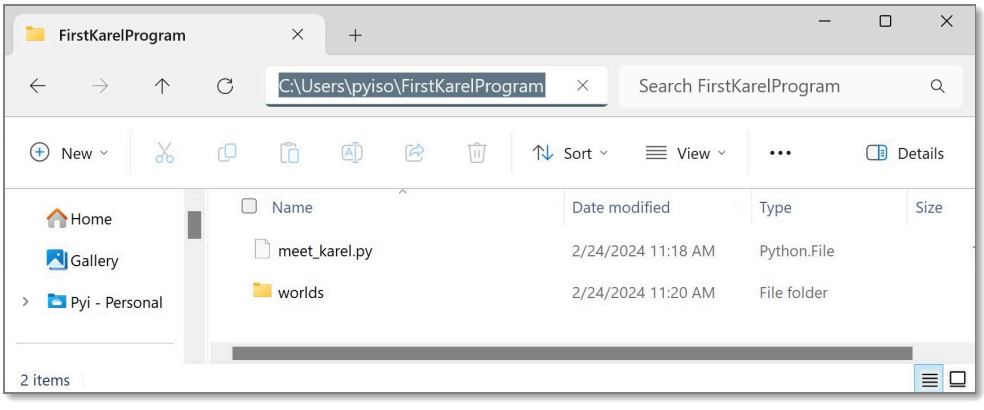
```
python meet_karel.py
```

ကုဒ်ရေးထားတာ ဆင်းတက်စ်အမှား မရှိဘူးဆိုရင် ကားရဲပရိုဂရမ် ပွင့်လာမှာပါ။

အခုဖော်ပြခဲ့တာက ကားရဲလ်ပရိုဂရမ်တစ်ခု run ဖို့ မဖြစ်မနေလုပ်ရမဲ့ အနည်းဆုံးလိုအပ်ချက်ပါ။ Python ဆော့ဖ်ဝဲ ရှိရမယ်။ `stanfordkarel` လိုက်ဘရီ အင်စတောလ် လုပ်ရမယ်။ ကမ္ဘာဖိုင်ပါတဲ့ worlds ဖိုဒါရှိရမယ်။ .py ဖိုင် တစ်ခုနဲ့ ပရိုဂရမ်ကုဒ်ကို သိမ်းရမယ်။ worlds ဖိုဒါနဲ့ ကုဒ်ဖိုင်ကို တစ်နေရာတည်းမှာ ထားရမယ်။ ပြီးရင် ကွန်မန်းလိုင်းမှာ

```
python your_karel_program.py
```

run ရုံပါပဲ။ C:\Users\pyiso\FirstKarelProgram ဖိုဒါထဲမှာ ကုဒ်ဖိုင်နဲ့ worlds ဖိုဒါကို ထားပြီး ဘယ်လို run ရလဲ နမူနာပြထားတာကို ကြည့်ပါ။



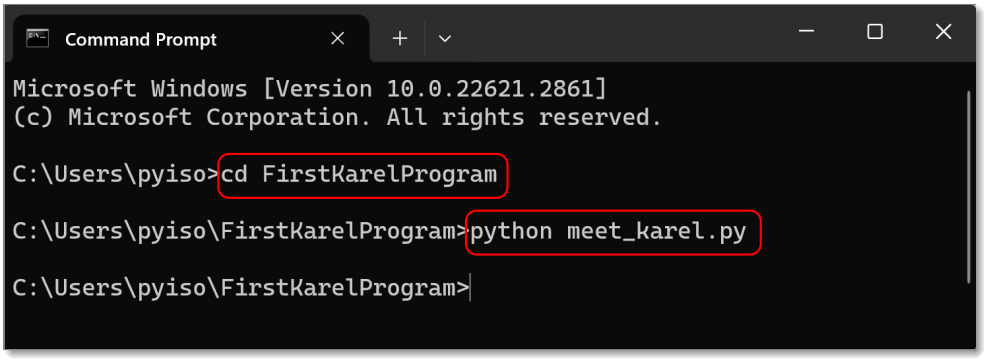
ပုံ ၁.၃

၁.၄ Python အင်စတောလ်လုပ်တဲ့အခါ ဘာတွေပါလဲ

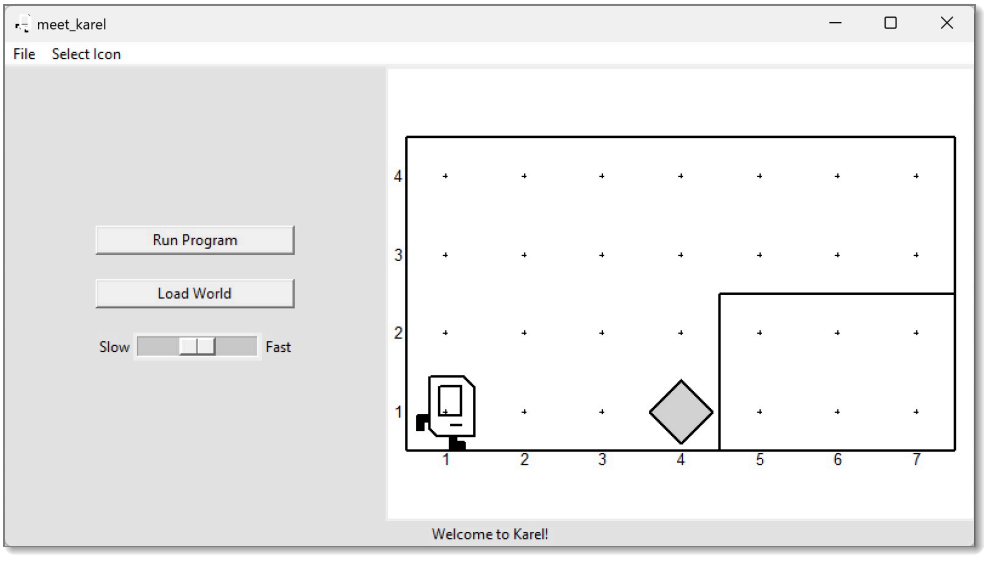
Python အင်စတောလ်လုပ်တယ်လို့ ယေဘုယျ ပြောပေမဲ့ အင်စတောလ်လုပ်တဲ့အခါ တကယ်တမ်းက ပရိုဂရမ်တစ်ခုတည်း ထည့်သွင်းပေးသွားတာ မဟုတ်ပါဘူး။ Python *interpreter* ပရိုဂရမ်၊ Python *standard library* နဲ့ အခြားလိုအပ်တဲ့ ပရိုဂရမ်တွေကို ထည့်ပေးသွားတာပါ။

Python Interpreter

အင်တာပရက်တာ ဆိုတာ ပရိုဂရမ်ကုဒ်တွေကို ဖတ်ပြီး ပရိုဂရမ်ကုဒ်ထဲက ညွှန်ကြားချက်တွေအတိုင်း လုပ်ဆောင်ပေးတဲ့ ပရိုဂရမ်ပါ။ Python အင်တာပရက်တာကတော့ Python နဲ့ရေးထားတဲ့ ကုဒ်တွေကို ဖတ်နိုင်တဲ့အပြင် ညွှန်ကြားထားတဲ့အတိုင်းလည်း လုပ်ဆောင်ပေးနိုင်ပါတယ်။ Python ကုဒ်တွေကို



ပုံ ၁.၄



ပုံ ၁.၅

ကွန်ပျူတာက တိုက်ရိုက် နားလည်တာ မဟုတ်ပါဘူး။ အင်တာပရက်တာကသာ တိုက်ရိုက်နားလည်တာပါ။ ၎င်းက တစ်ဆင့်ခံ၍ Python ကုဒ်တွေကို ကွန်ပျူတာပေါ်မှာ run ပေးရတာဖြစ်တယ်။ အင်တာပရက်တာကို python ကွန်မန်းနဲ့ အသုံးပြု run ရပါတယ်။

```
python meet_karel.py
```

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

Python Standard Library

Python အင်စတောလ်လုပ်တဲ့အခါ standard library လည်း တစ်ပါတည်း ထည့်သွင်းပေးသွားတယ်။ Standard library မှာ ကွဲပြားခြားနားတဲ့ ရည်ရွယ်ချက်အမျိုးမျိုးအတွက် ကွန်ပိုးနန့်တွေ အမြောက်အများ ပါဝင်ပါတယ်။ Standard library ကွန်ပိုးနန့်တွေကို နောက်ပိုင်းမှာ အသုံးပြုကြရမှာပါ။ ၎င်းတို့ကို အလျဉ်းသင့်သလို ဖော်ပြပေးသွားပါမယ်။

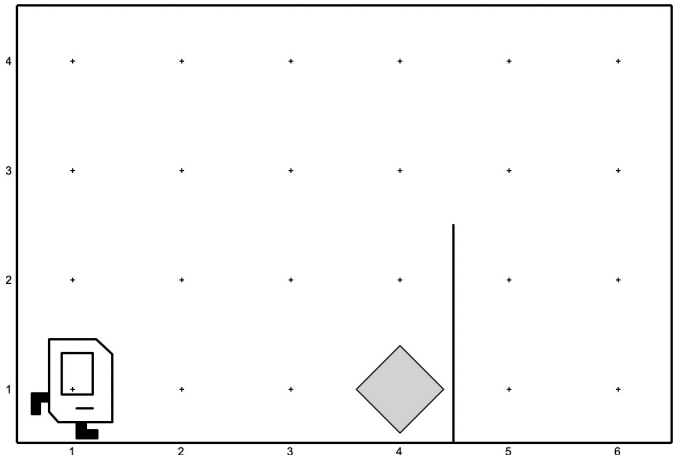
Standard အပြင် အခြားရရှိနိုင်တဲ့ လိုက်ဘရီတွေလည်း အများအပြားရှိပါတယ်။ **pip** ဟာ လိုက်ဘရီတွေ အင်စတောလ်လုပ်ဖို့ အသုံးပြုတဲ့ ပရိုဂရမ်တစ်ခုပါပဲ။ Python documentation မှာတော့ **pip** ကို Python *package* တွေ အင်စတောလ် လုပ်ဖို့ အသုံးအများဆုံး tool (ထောက်ကူပြုပစ္စည်း) လို့ ဆိုထားပါတယ်။ Programming language တွေနဲ့ပါတ်သက်ပြီး မော်ဂျူး (module)၊ ပက်ကေ့ချ် (package)၊ လိုက်ဘရီ (library) စတဲ့ အသုံးအနှုန်းတွေ တွေ့ရကြားရပါတယ်။ ဒီစကားလုံးတွေ၊ အခေါ်အဝေါ်တွေရဲ့ အဓိပ္ပါယ်သတ်မှတ်ချက်ကလည်း သက်ဆိုက်ရာ programming language အလိုက် ကွာခြားလေ့ရှိတယ်။ ဘီဂင်နာအနေနဲ့ ဒါနဲ့ပါတ်သက်ပြီး သိပ်ပြီးခေါင်းရှုပ်ခံစရာ မလိုသေးပါဘူး။ ပရိုဂရမ်ကုဒ် သိမ်းဆည်း၊ ဖွဲ့စည်း၊ ထားသို၊ ဖြန့်ဖြူးတဲ့ ကိစ္စတွေနဲ့ သက်ဆိုင်တဲ့ အခေါ်အဝေါ်တွေလို့ သိထားရင် လုံလောက်ပါပြီ။ နောက်ပိုင်းမှာ ဒီအခေါ်အဝေါ်တွေနဲ့ သက်ဆိုင်တဲ့ အဓိပ္ပါယ်သတ်မှတ်ချက်တွေကို အလျဉ်းသင့်ရင် သင့်သလို ဖော်ပြပေးပါမယ်။

၁.၅ Move Beeper to Other Side

ပရိုဂရမ်းမင်း လေ့လာတဲ့အခါ စာချည်းပဲ ဖတ်နေပြီး အမှန်တကယ် နားလည်သွားဖို့ဆိုတာ မဖြစ်နိုင်ပါဘူး။ လက်တွေ့ စမ်းသပ်ကြည့်၊ ရေးကြည့်မှပဲ တကယ် နားလည်လာမယ်။ တကယ်လည်း ကျွမ်းကျွမ်းကျင်ကျင် ရေးတတ်လာမှာပါ။ ဒါကြောင့် လက်တွေ့ရေးကြည့်ပါ။ များများ လေ့ကျင့်ပါ။ ဥပမာတွေကိုလည်း နားလည်အောင် ဖတ်ပြီးရင် မိမိဘာသာ အလွတ် ပြန်ရေးကြည့်ပါ။

ပုံ (၁.၆) မှ ဘိပါကို နံရံအခြားတစ်ဘက် အောက်ခြေကို ရွှေ့ပေးတဲ့ ပရိုဂရမ် ရေးကြည့်ပါ။ Python ထုံးစံအရ move_beeper_to_other_side.py ဖိုင်နဲ့ သိမ်းသင့်ပါတယ်။meet_karel.zip ဖိုင်ထဲမှာပါတဲ့ worlds ဖိုဒါမှာပဲ အခု ကမ္ဘာဖိုင် ထည့်ပေးထားပါတယ်။ move_beeper_to_other_side.w နံမည်နဲ့ပါ။ အန်ထရီပို့ဂ်အတွက် အခုလိုရေးရပါမယ်။

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



ပုံ ၁.၆

အခန်း ၂

ကွန်ထရိုးလ် စတိတ်မန့်များ

ကွန်ထရိုးလ် စတိတ်မန့်တွေက ကားရဲလ်မှာ တွေ့တော့ တွေ့ခဲ့ပြီးသားပါ။ ဒါပေမဲ့ ကားရဲလ်ပရိုဂရမ်မင်း အတွက် လိုသလောက် အခြေခံကိုပဲ ကန့်သတ်ဖော်ပြခဲ့တာပါ။ ဒီအခန်းမှာ ပြည့်စုံအောင် အသေးစိတ် ဆက်လက် လေ့လာကြပါမယ်။ လက်တွေ့အသုံးချ ဥပမာတွေ၊ လေ့ကျင့်ခန်းတွေ ဂရုတစိုက် စီစဉ်ပေးထား တယ်။ စတိတ်မန့် အသစ်တချို့လည်း တွေ့ရမယ်။ စာအုပ်တွေမှာ ဖော်ပြတာ သိပ်မတွေ့ရပေမဲ့ ဘီဂင်နာ အများစု အခက်အခဲတွေ့ကြတဲ့ နေရာတွေ၊ တိတိကျကျ နားလည်ဖို့ လိုတဲ့ ပို့ဝိုင့်တွေကိုလည်း အလေးပေး ရှင်းပြထားတယ်။ အထူးခြားဆုံးကတော့ ရုပ်ပုံတွေဆွဲတာနဲ့ အန်နီမေးရှင်း အခြေခံကို Arcade ဂိမ်းလိုက် ဘရီ အသုံးပြုပြီး စတင်မိတ်ဆက်ထားတာပဲ ဖြစ်တယ်။ စာသားတွေချည်းပဲထက် စိတ်လှုပ်ရှားဖို့ ပိုကောင်း မယ်ထင်ပါတယ်။

၂.၁ `if` စတိတ်မန့်

စားသောက်ဆိုင် တစ်ဆိုင်က ကျပ်ငွေ $50,000$ နဲ့ အထက် သုံးတဲ့ ကတ်စတမ်မာတွေကို 10% လျှော့ပေး ပြီး ပရိုမိုးရှင်း လုပ်တယ် ဆိုပါစို့။ ကီးဘုဒ်ကနေ ကျသင့်ငွေ ရိုက်ထည့်ပေးရမယ်။ ဒစ်စကောင့် ရမဲ့ ကတ်စ တမ်မာအတွက်ပဲ 'Get 10 % discount.' ပြပေးပြီး လာရောက် စားသုံးတဲ့အတွက် ကျေးဇူးတင်ကြောင်း 'Thanks for coming!' ကိုတော့ ကတ်စတမ်မာတိုင်းကို ပြပေးချင်ပါတယ်။

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

`amt > 50_000` ဘူလီယန် အိပ်စ်ပရက်ရှင် `true` ဖြစ်မှပဲ `if` ဘလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဒစ် စကောင့် 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 ဗားရှင်း 3.6 ကစပြီး f-string (formatted string) ခေါ်တဲ့ string အသစ်တစ်မျိုး ပါလာပါတယ်။ String ရှေ့မှာ `f` နဲ့ စရင် 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 ဗေရီရေဘဲလ် (သို့) အိပ်စ်ပရက်ရှင် တွေကို တွန့်ကွင်းထဲမှာ ထည့်ရေးလို့ရတယ်။ ၎င်းတို့ကို f-string က တန်ဖိုးရှာပြီး အစားထိုးပေးမှာပါ။ ဒါကြောင့် `{2 + 3}` က 5 ဖြစ်သွားတာပါ။ ရိုးရိုး string နဲ့ဆို အခုလို

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

ရေးနေရမယ်။ F-string နဲ့ဆို ပိုအဆင်ပြေတယ်။ နမူနာတချို့ကို 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 ပရိုမိုးရှင်းလုပ်တဲ့အခါ သတ်မှတ်ပမာဏ မပြည့်သေးရင် ဘယ်လောက်ဖိုးထပ်သုံးတာနဲ့ ဒစ်စကောင့် ရမှာဖြစ်ကြောင်းပြောပြီး ဆွဲဆောင်လေ့ရှိတယ်။ ဒစ်စကောင့်ရအောင် ဘယ်လောက်ထပ်သုံးရမလဲ ပရိုဂရမ်က ပြပေးချင်တယ် ဆိုပါစို့။ `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` ယေဘုယျပုံစံကို ကြည့်ရင် အခုလိုရှိပါတယ်

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

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

test ဟာ ဘူလီယန် အိပ်စ်ပရက်ရှင်း ဖြစ်ရပါမယ်။ (ကားရဲလ် ကွန်ဒီရှင်တွေဟာ ဘူလီယန်တန်ဖိုး ပြန်ပေးတဲ့ predicate မက်သဒ်တွေပါ။ predicate မက်သဒ်တွေကို ဘူလီယန် အိပ်စ်ပရက်ရှင်းလို့ ယူဆနိုင်တယ်။)

အခုဆက်ကြည့်ကြမဲ့ `if...elif...else` ပုံစံကတော့ ရှေ့ပိုင်းမှာ မတွေ့ဖူးသေးဘူး။ "Cascading `if` statement" လို့ခေါ်တယ်။ အောက်ပါ ဇယားအရ စာမေးပွဲရမှတ် ကနေ grading ထုတ်ပေးမယ် ဆိုပါစို့။

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

တေဘဲလ် ၂.၁ Score and Grading

ကျောင်းသူ/သား နံမည်နဲ့ ရမှတ်ကို ထည့်ပေးရင် ပရိုဂရမ်က အခုလို ပြပေးရပါမယ်။

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

ဒီပရိုဂရမ် 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 အတွဲလိုက် တွေ့ရပါမယ်။ နောက်ဆုံးမှာ `else` အပိုင်းကို တွေ့ရတယ် (ဒီအပိုင်းက optional ပဲ၊ မပါလို့လဲရတယ်။ ခဏနေ ရှင်းပြပါမယ်)။ 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 မပါလို့လည်းရတယ်။ အပေါ်မှာ တစ်ခုမှ `True` မဖြစ်တော့မှ `else` ဘလောက်ကို လုပ်ဆောင်တယ်။ နောက်ဆုံးမှာ `else` အပိုင်းမပါဘူး၊ အပေါ်မှာလည်း ဘယ်တစ်ခုကမှ `True` မဖြစ်ဘူး ဆိုရင်တော့ လုပ်ဆောင်ပေးစရာ ဘလောက်လည်း မရှိဘူးပေါ့။ ဒီတော့ အားလုံး `False` ဖြစ်ခဲ့ရင် လုပ်ချင်တဲ့ကိစ္စ ရှိ/မရှိ အပေါ်မူတည်ပြီး `else` အပိုင်း လို/မလို ဆုံးဖြတ်ရတယ်။

အခု grading ပရိုဂရမ်မှာ ရမှတ်ဟာ 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 တစ်ခုမှ မမှန်နိုင်ဘူး။ ဒါကြောင့် `else` အပိုင်းနဲ့ မှားထည့်ထားတယ်လို့ ပြပေးလိုက်တယ်။

အခုပရိုဂရမ်မှာ သိပ်စိတ်တိုင်းကျစရာ မကောင်းတဲ့ ပြဿနာတစ်ခုတွေ့ရပါတယ်။ 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 <= 79:
        grade = 'C'
    elif 60 <= score <= 69:
        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.')
```

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

Cascading `if` မှာ ဘူလီယန် အိပ်စ်ပရက်ရှင် တစ်ခုချင်းကို အထက်အောက် အစဉ်အတိုင်း တန်ဖိုးရှာတယ်၊ 'ပထမဆုံး `True` ဖြစ်တဲ့ ဘလောက် တစ်ခုကိုပဲ လုပ်ဆောင်ပေးတယ်' ဆိုတဲ့အချက်ကို နားလည်ဖို့ အရေးကြီးတယ်။ အောက်ကို ရောက်လာတာဟာ အပေါ်မှာ မှားခဲ့လို့ပဲ။ တစ်ခုမှန်ပြီဆိုတာနဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို လုပ်ဆောင်ပြီး cascading `if` တစ်တွဲလုံး ပြီးဆုံးသွားမှာ ဖြစ်တယ်။ အောက်ပိုင်းက `elif` (သို့) `else` တွေကို မရောက်လာတော့ဘူး။ ဒီအကြောင်းကြောင့် grading အတွက် အခုလိုလည်း ရေးလို့ရတယ်

```
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 ကိုးဆယ်အောက် ဖြစ်မှာတော့ သေချာတယ် (`score >= 90` မဟုတ်လို့ ဒီကို ရောက်လာတာ)၊ ဒါကြောင့် ရှစ်ဆယ်နဲ့အထက် (`score >= 80`) ဖြစ်လား စစ်ရင်ရပြီ။ နောက်တစ်ဆင့်ကို ရောက်လာရင် ရှစ်ဆယ်အောက် မို့လို့ သေချာတယ်၊ `score >= 70` ဖြစ်လားစစ်ရုံပဲ။ စသည်ဖြင့် အောက်အဆင့်တွေ အတွက်လည်း ထိုနည်းတူစွာ စဉ်းစားနိုင်တယ်။

Cascading `if` မသုံးဘဲ `if...else` တွေနဲ့လည်း ရေးလို့တော့ ရပါတယ်။ Nesting လုပ်တာတွေ အရမ်းများပြီး 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.')
```

၂.၂ `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 ပြန်ကျော့မလဲ အတိအကျ ကြိုသိရင် *definite loop* လို့ သတ်မှတ်တယ်။ `for` loop ဟာ definite loop ဖြစ်ပါတယ်။ 'အဆင့်မြင့် အက်ဘ်စရက်ရှင်း' လို့ ပြောရတာက list, dictionary, set, range စတဲ့ စထရက်ချာ အမျိုးမျိုးနဲ့ အသုံးပြုလို့ရတဲ့ အတွက်ကြောင့်ပါ။

`for` loop နဲ့ list ထဲက အိုက်တမ်တစ်ခုချင်း ထုတ်ယူအသုံးပြုနိုင်ပါတယ် ...

```
fruits = ['Orange', 'Kiwi', 'Banana', 'Papaya', 'Apple', 'Plum', 'Mango']
for itm in fruits:
    print(itm)
```

Output:

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

Loop တစ်ခေါက်ပြန်ကျော့တိုင်း itm ဗေရီရေဘဲလ်ထဲမှာ list အိုက်တမ်တစ်ခုချင်း အစဉ်အတိုင်း ထည့်ပေးမှာပါ။ အိုက်တမ်တွေကို နံပါတ်စဉ်နဲ့ တွဲချင်ရင် enumerate လုပ်ပြီး အခုလို ထုတ်လို့ရတယ်

```
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 ကာရက်တာတစ်လုံးချင်း လိုချင်ရင်လည်း ရတာပဲ

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

Output:

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

List နှစ်ခုရဲ့ cartesian product ပါ (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 ဟာ list, dictionary, string စတဲ့ စထရက်ချာအမျိုးမျိုးနဲ့ အလုပ်လုပ်နိုင်တာ တွေ့ရမှာပါ။ တစ်ခါကျော့တိုင်း အိုက်တမ်တစ်ခုကို loop ဗေရီရေဘဲလ်ထဲမှာ ထည့်ပေးထားတယ်။ အိုက်တမ်တွေအားလုံး ပြီးတဲ့အခါ `for` loop ရပ်သွားမှာ ဖြစ်တယ်။ ပြန်ကျော့တဲ့ အကြိမ်အရေအတွက်ဟာ အိုက်တမ်အရေအတွက်ပဲ ဖြစ်တယ်။

`range` ဖန်ရှင်နှင့် `for` loop

သုညကနေ တစ်ဆယ်ထိ အစဉ်အတိုင်း ရေတွက်ချင်ရင် နည်းလမ်းတစ်ခုက

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

ဒီလိုသာ ဂဏန်းတစ်လုံးချင်း ရိုက်ထည့်ရရင် အဆင်မပြေပါဘူး။ ပိုကောင်းတဲ့ နည်းလမ်း ရှိရမှာပါ။ `range` ဖန်ရှင်ဟာ ဒီလိုနေရာမျိုးအတွက် အသင့်တော်ဆုံးပါပဲ။ `range(0,11)` က သုညကနေ တစ်ဆယ်ထိ အစဉ်အတိုင်း ထုတ်ပေးမဲ့ `range` အော့ဘ်ဂျက်ကို ပြန်ပေးတယ်။ ဒီဂဏန်းတွေကို တစ်ခါတည်း ကြို ထုတ်ထားတာ မဟုတ်ပါဘူး။ လိုအပ်မှ တစ်ခုချင်း ထုတ်ပေးတာပါ။ (ဒီအတွက်ကြောင့် `range(0, 11)` နဲ့ `range(0, 1_000_000)` နှစ်ခုလုံး မမ်မိုရီသုံးစွဲတာအရ သိပ်မကွာခြားပါဘူး)။

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

$2, 4, 6, \ldots, 12$ စုံကိန်းတွေ လိုချင်ရင် `range(2, 13, 2)`၊ $5, 8, 11, \ldots, 98$ လိုချင်ရင် `range(5, 99, 3)` စသည်ဖြင့် သုံးခုထည့်ပြီး ဖန်ရှင်ခေါ်ရပါမယ်။ အစ၊ အဆုံးနဲ့ နောက်ဆုံးတစ်ခုကတော့ ကိန်းတန်းမှာပါတဲ့ ကပ်လျက်ကိန်းနှစ်ခုရဲ့ ကွာခြားချက်ပါ။

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

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

အစဂဏန်း မသတ်မှတ်ပေးရင် သုညလို့ ယူဆတယ်။ ဒါကြောင့် သုညကနေ တစ်ဆယ်ထိကို `range(11)` နဲ့ ယူလို့ရတယ်။ ကွာခြားချက်က အနှုတ်ကိန်း ဖြစ်လို့ရတယ်

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

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

မှတ်ချက်။ ။ ကွာခြားချက် တစ်မဟုတ်ရင် အစ၊ အဆုံး၊ ကွာခြားချက် သုံးခုလုံး လိုပါမယ်။

for loop အသုံးချ ဥပမာများ

စတိတ်မန့်တစ်စုကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ပြန်ကျော့ဖို့ for loop ကို အသုံးပြုနိုင်ပါတယ်။ ကီးဘုဒ်ကနေ ထည့်ပေးတဲ့ ဂဏန်း ဆယ်လုံးကို ပေါင်းမယ်ဆိုပါစို့။

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

print(f"Total: {tot}")
```

ပရိုဂရမ် run တဲ့ အချိန်ကျတော့မှ အကြိမ်အရေအတွက် သတ်မှတ်လို့လည်း 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 တစ်ကျော့ပြီး တစ်ကျော့ ဗေရီရေဘဲလ် တန်ဖိုးတွေ ဘယ်လောက်ဖြစ်နေလဲ အခုလို လိုက်ကြည့်နိုင်ပါတယ်။

```
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_total

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

Nested `for` loop သုံးထားတယ်။ ပေါင်းလဒ်ကို ထည့်ထားဖို့ ဗေရီရေဘဲလ် နှစ်ခု သုံးထားတာ သတိထားကြည့်ပါ။ ဒီနေရာမှာ ဗေရီရေဘဲလ် စကုပ် (*scope*) သဘောတရားကို နားလည်ဖို့ လိုအပ်လာပါတယ်။ ဗေရီရေဘဲလ်တစ်ခုကို ဘယ်နေရာကနေ သုံးလို့ရလဲဆိုတာဟာ ၎င်းဗေရီရေဘဲလ်ရဲ့ စကုပ်နဲ့ သက်ဆိုင်ပါတယ်။ `grand_tot` ဟာ top level ဗေရီရေဘဲလ် ဖြစ်တယ်။ ၎င်းကို ကြေငြာတဲ့ နေရာကစပြီး အောက်ပိုင်းတလျှောက်လုံး သုံးလို့ရတယ်။ `grand_tot` ရဲ့ စကုပ်ဟာ ၎င်းကို ကြေငြာထားတဲ့ ဖိုင်အဆုံးထိ ဖြစ်တယ်။ `rot_tot` ကတော့ block level ဗေရီရေဘဲလ်ပါ။ Block level ဗေရီရေဘဲလ်ကိုတော့ ၎င်းကို ကြေငြာထားတဲ့ ဘလောက်အတွင်းမှာပဲ သုံးလို့ရပါမယ်။ Block level ဗေရီရေဘဲလ်တစ်ခုရဲ့ စကုပ်ဟာ ၎င်းကို ကြေငြာထားတဲ့ ဘလောက် အဆုံးထိ ဖြစ်တယ်။

`row` တစ်ခုချင်း ပေါင်းလဒ်ကို `grand_tot` မှာ ပေါင်းထည့်ပေးဖို့ လိုတယ်။ အောက်ဆုံးမှာလည်း `grand_tot` ကို `print` ထုတ်ပေးရမယ်။ အကယ်၍ block level မှာ ထားလိုက်ရင် အောက်ဆုံးမှာ သုံးလို့ရမှာ မဟုတ်တော့ဘူး။ `row` တစ်ခု ပေါင်းပြီးရင် `rot_tot` ကို ထုတ်ပေးဖို့ လိုတယ်။ ဒါကြောင့် ၎င်းကို အပြင်ဘလောက်မှာ ကြေငြာရမယ်။ အတွင်းဘလောက်မှာဆိုရင် အပြင်ကနေ သုံးလို့ရမှာ မဟုတ်တော့ဘူး။ (အတွင်း `for` loop ဟာ အပြင် `for` loop ရဲ့ ဘလောက် အတွင်းမှာ ပါဝင်တဲ့အတွက် `rot_tot` ကို သုံးလို့ရပါတယ်)။

Loop တစ်ကျော့ပြီး တစ်ကျော့ ဗေရီရေဘဲလ် တန်ဖိုးတွေ ပြောင်းလဲသွားတာကို ပြထားတယ်။ တစ်ဆင့်ချင်း ဂရုစိုက်ပြီး လိုက်ကြည့်ပါ။ (အကြိမ်အရေအတွက် သိပ်မများအောင် အိုက်တမ် နည်းတဲ့

list နဲ့ ဥပမာ ပြထားတာပါ)။

```
# ဒီ list ထဲက ဂဏန်းတွေ ပေါင်းမှာပါ
rows = [[1, 3, 5],
        [2, 4, 6]]
```

```
grand_tot = 0

    # အပြင် for loop
    1st iter:
    row = [1, 3, 5]
    row_tot = 0

        # အတွင်း 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
    # အပြင်ဘလောက် တစ်ကျော့ပြီး row_tot စကုပ် အဆုံး

    # အပြင် for loop
    2nd iter:
    row = [2, 4, 6]
    row_tot = 0 # ဗေရီရေဘဲလ် တစ်ခါထပ်ကြေငြာတယ်

        # အတွင်း 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
    # အပြင်ဘလောက် နောက်တစ်ကျော့ပြီး row_tot စကုပ် အဆုံး

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

ခရစ်စမတ် သစ်ပင်လေး ဆွဲကြည့်ရအောင်။ တစ်တန်းမှာ စပေ့စ်ဘယ်နှစ်ခုပါလဲ ကြည့်ရလွယ်အောင် □ 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 တြိဂံမှာ အတန်း ရှစ်ခု (LEAF_ROWS)၊ ပင်စည်မှာ သုံးခု (TRUNK_ROWS) သတ်မှတ်

ထားတယ်။ ပင်စည် အကျယ် (`TRUNK_SZ`) ကိုလည်း `*` သုံးခု ထားတယ်။ အပေါ်ဆုံးကို row နံပါတ် သုည၊ ဒုတိယကို တစ် စသည်ဖြင့် ယူဆပါမယ်။ row နံပါတ်စဉ်ကို `r` ဗေရီရေဘဲလ်က ဖော်ပြတယ်။ တြိဂံပုံမှာ စပေ့စ်အရေအတွက်နဲ့ row နံပါတ်စဉ်ဟာ `(LEAF_ROWS - 1 - r)` ဖော်မြူလာနဲ့ ဆက်စပ် နေတယ်။ `*` အရေအတွက်ကတော့ နှစ်ခုစီတိုးသွားပြီး `(r * 2 + 1)` ဖြစ်တယ်။ ပင်စည်ပိုင်း စပေ့စ် အရေအတွက်ကတော့ တစ်တန်း ခြောက်ခုပါ `(LEAF_ROWS - 2)` ။

၂.၃ Python Arcade Library

Python Arcade (ဝက်ဘ်ဆိုက် `https://api.arcade.academy`) ဟာ အရည်အသွေးကောင်းတဲ့ ထိပ်ဆုံး Python ဂိမ်းလိုက်ဘရီတွေထဲက တစ်ခုဖြစ်တယ်။ Arcade အပြင် အသုံးများတဲ့ အခြားတစ် ခုက pygame (ဝက်ဘ်ဆိုက် `https://www.pygame.org`) ပါ။ ဒီစာအုပ်မှာ Arcade ကို သုံးပါ မယ်။ Arcade ပိုကောင်းတယ်လို့ မဆိုလိုပါဘူး။ ဘီဂင်နာတွေအတွက် ပိုသင့်တော်မယ် ယူဆတဲ့အတွက် အသုံးပြုတာပါ။ အောက်ပါအတိုင်း အင်စတောလ်လုပ်နိုင်ပါတယ်

```
pip install arcade
```

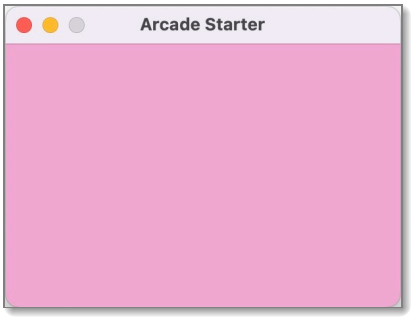
Arcade နဲ့ ပုံဆွဲဖို့အတွက် ဂရပ်ဖစ် ဝင်းဒိုးတစ်ခုကို အောက်ပါအတိုင်း ယူရပါတယ်။ Run လိုက်ရင် ပုံ (၂.၁) မှာ တွေ့ရတဲ့ နောက်ခံရောင်နဲ့ ဝင်းဒိုးအလွတ် တစ်ခု တက်လာမှာပါ။

```
# 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` လိုက်ဘရီ အင်ပို့လုပ်ထားတာပါ။ ရှေ့ပိုင်းမှာသုံးတဲ့ `from` *lib* `import *` ပုံစံ နဲ့ ကွာခြားတာက အခုနည်းနဲ့ အင်ပို့လုပ်ထားရင် လိုက်ဘရီမှာ ပါတဲ့ အစိတ်အပိုင်းတွေကို ဒေါ့ထ် အမှတ်အသားနဲ့ အသုံးပြုရပါမယ်။ ဥပမာ `open_window` ဖန်ရှင်ကို

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

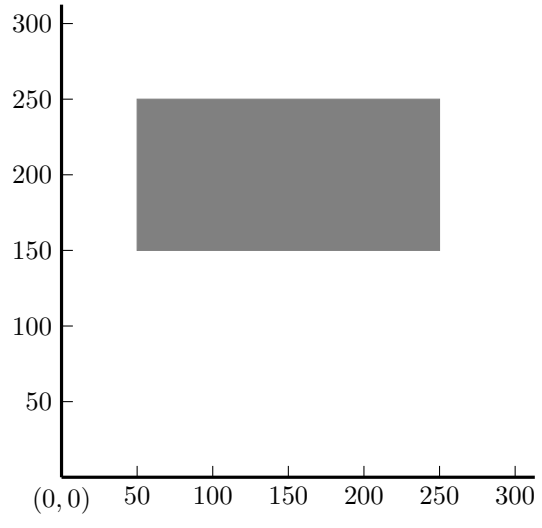
လိုက်ဘရီ နံမည်နောက်မှာ (`.`) အမှတ်အသား ခံပြီး ခေါ်ရမှာပါ။ ဒီဖန်ရှင်မှာ လိုချင်တဲ့ဝင်းဒိုး အကျယ်၊ အမြင့်၊ ခေါင်းစည်းစာသား ထည့်ပေးထားတယ်။



ပုံ ၂.၁

ကိုဩဒိနိတ်စနစ်နှင့် ဝင်းဒိုးပေါ်တွင် တည်နေရာ သတ်မှတ်ခြင်း

ဝင်းဒိုး အောက် ဘယ်ဘက်ထောင့်စွန်းကို origin အမှတ် $(x = 0, y = 0)$ အဖြစ် သတ်မှတ်ပါ တယ်။ ညာဘက်သွားရင် x တန်ဖိုး များလာမယ်။ အပေါ်ဘက်တက်ရင် y တန်ဖိုး တိုးလာမှာပါ။ အခြား ကွန်ပျူတာ ဂရပ်ဖစ်စနစ်တွေမှာလိုပဲ x, y တန်ဖိုးကို ယူနစ်အားဖြင့် pixel နဲ့ ဖော်ပြတယ်။ ပုံ (၂.၂) မှာ rectangle ဘယ်ဘက်အောက် ထောင့်စွန်း (x, y) တန်ဖိုး $(50, 150)$, ညာဘက် အပေါ်ထောင့်ဟာ $(250, 250)$ ဖြစ်တယ်။



ပုံ ၂.၂

အောက်ပါ ဖန်ရှင်ကတော့ ဝင်းဒိုးရဲ့ နောက်ခံရောင် သတ်မှတ်တာပါ။ လိုက်ဘရီရဲ့ `color` မော်ဂျူး (*module*) မှာ အရောင်တန်ဖိုးတွေ အဆင်သင့် သတ်မှတ်ပေးထားတယ်။ (မော်ဂျူးဆိုတာ လိုက်ဘရီရဲ့ အစိတ်အပိုင်းတစ်ခုလို့ အကြမ်းဖျဉ်း ယူဆနိုင်တယ်)။

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

အခုလို အင်ပို့လုပ်ထားရင် အရောင်တွေ သုံးရတာ ပိုအဆင်ပြေတယ်

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

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

PINK_PEARL, RED စသည်ဖြင့် အရောင်နံမည် တမ်းရေးလို့ရတယ်။ ရှေ့မှာ arcade.color. ထည့်ဖို့ မလိုတော့ဘူး။

ပုံဆွဲဖို့ အဆင်သင့်ဖြစ်အောင် start_render ခေါ်ပေးရမယ်။ ဆွဲပြီးရင်လည်း finish_render ခေါ်ဖို့လိုတယ်။ ပုံဆွဲတဲ့ကိစ္စကို ၎င်းတို့နှစ်ခုကြားမှာ လုပ်ရမှာပါ။

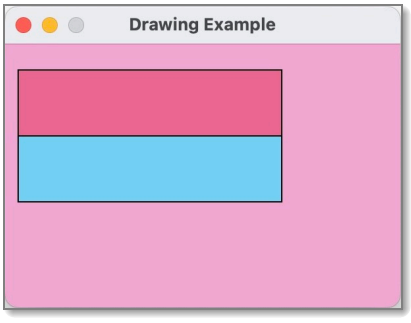
```
arcade.start_render()
# call drawing functions here
...
...
arcade.finish_render()
```

```
arcade.run()
```

ဝင်းဒိုးကို မပိတ်မချင်း ပေါ်နေအောင် run ဖန်ရှင် ခေါ်ပေးရတာပါ။ မခေါ်ထားဘဲ ပရိုဂရမ်ကို run ရင် ဝင်းဒိုးပွင့်လာပြီး ဖျတ်ခနဲ ပြန်ပိတ်သွားမှာပါ။ မကျန်ခဲ့ဖို့ သတိပြုရပါမယ်။

Arcade မှာ ပါတဲ့ အခြေခံ ပုံဆွဲဖန်ရှင် တချို့ကို ဆက်ကြည့်ရအောင်။ ထောင့်မှန်စတုဂံဆွဲတဲ့ ဖန်ရှင်တွေထဲက နှစ်ခု သုံးပြထားတယ်။ နှစ်ခုလုံးက ထောင့်မှန်စတုဂံရဲ့ ဘယ်ဘက်အောက် ထောင့်စွန်းနဲ့ တည်နေရာကို သတ်မှတ်ပြီး အရွယ်အစားကို အကျယ်၊ အမြင့်နဲ့ သတ်မှတ်ပေးရတာပါ။ 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 ၂.၃

```
# 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 (၂.၃)]]။ ပါရာမီတာတွေက $x, y, width, height, color$ 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။

ကျားကွက်ခုံ (သို့) စစ်တုရင်ခုံ ဆွဲခြင်း

ကျားကွက် (သို့) စစ်တုရင်ခုံ ပုံဖော်တဲ့ checkerboard ဥပမာကို အောက်မှာလေ့လာကြည့်ပါ။ ပုံ (၂.၄) က ဒီပရိုဂရမ်နဲ့ ဆွဲထားတာပါ။

```
# 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()
```

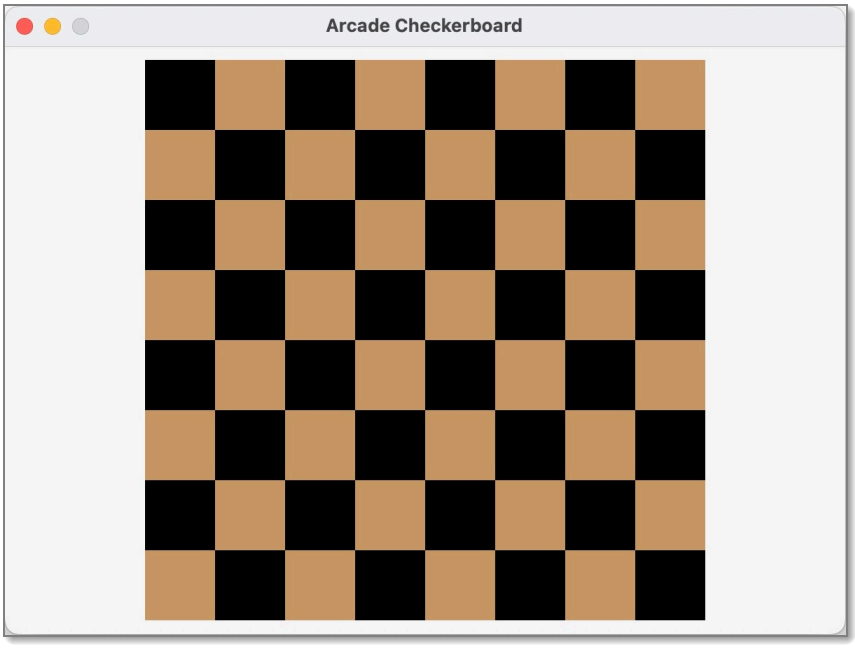
```
# width/height of the whole board
BOARD_SIZE = 400
COLS = 8
ROWS = 8
# size of each square of the board
SQ_SIZE = BOARD_SIZE // ROWS
# x of left side of the board
X_LFT = (WIN_WIDTH - BOARD_SIZE) // 2
# y of the bottom side of the bord
Y_BTM = (WIN_HEIGHT - BOARD_SIZE) // 2

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 အရွယ်အစားကို (၄၀၀) pixels ထားတယ်။ စတုရန်းပုံဆိုတော့ အကျယ်/အမြင့် နှစ်ခုလုံး (၄၀၀)။ `X_LFT` က row တစ်ခုချင်းရဲ့ စမှတ် x တန်ဖိုး၊ `Y_BTM` ကတော့ အောက်ဆုံး row y စမှတ် တန်ဖိုး ဖြစ်ပါတယ်။

အတွင်း `for` loop က တစ်တန်းမှာပါတဲ့ ကျားကွက် (၈) ခုကို ဘယ်ကနေညာ တစ်ကွက်ချင်း ဆွဲသွားမှာဖြစ်ပြီး အပြင် `for` loop ကတော့ row တစ်ခုပြီး တစ်ခု အောက်ဆုံးကစပြီး ပုံဖော်ပေးမှာပါ။ Row တစ်တန်းမှာ ပါဝင်တဲ့ အကွက်တွေအားလုံး `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 ဂရုစိုက်ကြည့်ပါ)။ Row တစ်ခုအတွက် ကျားကွက်တစ်ကွက်ချင်းကို ဘယ်



ပုံ ၂.၄

ဘက်ကနေစပြီး ဆွဲမှာပါ။ တစ်ကွက်ပြီး နောက်တစ်ကွက် ကူးတဲ့အခါ လက်ရှိ x တန်ဖိုးကို ကျားကွက် တစ်ကွက်စာ ပေါင်းပေးရပါမယ်။ ဒီအတွက်

```
x += SQ_SIZE
```

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

```
x = X_LFT
```

ပြန် လုပ်ပေးရပါမယ်။

ကျားကွက်ပတ်တန် တစ်လှည့်စီ အရောင်ခြယ်ဖို့ ဘယ်လိုလုပ်ရမလဲ။ တစ်ကွက်ချင်းရဲ့ row နံပါတ် နဲ့ column နံပါတ်ကို ပေါင်းကြည့်ပါ။ ပေါင်းလဒ် စုံဂဏန်း ဖြစ်တဲ့ အကွက်တွေ အရောင်တူတယ်။ မ ဂဏန်းဖြစ်တဲ့ အကွက်အားလုံး 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 တလှည့်စီခြယ်တယ်။ ဒီလောက်ဆိုရင် ကျားကွက်ပရိုဂရမ်ကို နားလည်မယ် ထင်ပါတယ်။

၂.၄ `while` loop

`while` loop ဟာ *indefinite loop* ဖြစ်ပါတယ်။ Loop ကို စတင် လုပ်ဆောင်တဲ့အချိန်မှာ ဘယ်နှစ်

ကြိမ် ပြန်ကျော့မလဲ အတိအကျ ကြို 'မသိ' ရင် indefinite loop လို့ သတ်မှတ်တယ်။ တစ်နဲ့ တစ်ဆယ် ကြား (အပါအဝင်) ကိန်းပြည့်တစ်လုံးကို ကျပန်း (random) ထုတ်ထားမယ်။ မမှန်မချင်း အဲဒီဂဏန်းကို ခန့်မှန်းပေးရမယ်။ ခန့်မှန်းတာ မှန်သွားရင် ဘယ်နှစ်ခါ ခန့်မှန်းရလဲနဲ့ မှန်တဲ့ဂဏန်းကို ပြပေးတဲ့ ပရိုဂရမ်မှာ 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 (ဘူလီယန် အိပ်စ်ပရက်ရှင်) မမှားမချင်း loop က ပြန်ကျော့ပေးနေမှာပါ။ မှားတဲ့အခါ ထပ်မကျော့တော့ဘဲ ရပ်သွားမှာဖြစ်တယ်။ ကျပန်းဂဏန်းက ခုနှစ်ကျတယ် ဆိုပါစို့။ တစ်၊ ကိုး၊ သုံး၊ တစ်ဆယ်နဲ့ ခုနှစ် တို့ကို အစဉ်အတိုင်း ခန့်မှန်း ထည့်ပေးမယ်ဆိုရင် တစ်ကျော့ချင်း 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 စတိတ်မန့်တွေ ဆက်လုပ်ပါတယ်
```

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

Loop က ထပ်မကျော့တော့ဘဲ ရပ်သွားတာကို *loop exits* ဖြစ်သွားတယ်လို့ ပြောတယ်။ မြန်မာလိုတော့ loop ကနေ ထွက်သွားတယ်ပေါ့။ အကယ်၍ စောစောက ဥပမာမှာ ပထမဆုံးတစ်ခါမှာပဲ ခုနှစ်ထည့်လိုက်ရင်တော့ ဘလောက်ကို တစ်ခါမှ မလုပ်ဆောင်ဘဲ loop က ချက်ချင်းထွက်သွားမှာပါ။

Sentinel Controlled Loop

ဂဏန်းတွေ ဘယ်နှစ်လုံး ပေါင်းမှာလဲ ကြိုသိရင် `for` loop နဲ့ အလုပ်ဖြစ်တယ်။ ဂဏန်း ဘယ်နှစ်လုံးရှိလဲ ကြိုမသိထားဘဲ ရှိသလောက် တစ်လုံးချင်းရိုက်ထည့်ပြီး အားလုံးပေါင်းချင်တာဆိုရင်တော့ `for` loop နဲ့ အဆင်မပြေဘူး (မရနိုင်ဘူးလို့ မဆိုလိုပါ၊ မရမက လုပ်တဲ့နည်းတွေလည်း ရှိပါတယ်)။ ဒီလိုအခြေအနေမျိုးမှာ *sentinel controlled loop* ကို သုံးပါတယ်။

Sentinel controlled loop မှာ loop ကနေ ထွက်ဖို့အတွက် အသုံးပြုတဲ့ သီးသန့်တန်ဖိုးတစ်ခု ရှိရပါမယ်။ ဒီတန်ဖိုးကို *sentinel value* လို့ ခေါ်တယ်။ ဂဏန်းတွေပေါင်းတဲ့ ကိစ္စအတွက် *sentinel value* ရွေးချယ်သတ်မှတ်တဲ့အခါ ပေါင်းရမဲ့ဂဏန်း မဖြစ်နိုင်တဲ့ တစ်ခုကို ရွေးရပါမယ်။ သုညဟာ အပေါင်းထပ်တူရ ဂုဏ်သတ္တိရှိတဲ့အတွက် ၎င်းကို *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 ဂဏန်းဟာ သုညမဟုတ်မချင်း `total` မှာ ပေါင်းထည့်ပြီး နောက်ဂဏန်းတစ်ခု ထည့်ပေးဖို့ input တောင်းမှာပါ။ Sentinel တန်ဖိုး သုည ထည့်လိုက်ရင် loop က ထွက်သွားမယ်။ ပြီးတဲ့အခါ `total` ကို ပြပေးမှာပါ။ စစချင်းပဲ သုည ထည့်လိုက်ရင် loop က တစ်ခါမှ မကျော့ဘဲ ထွက်သွားမယ်။ `total` လည်း သုညပဲ ထွက်ပါမယ်။ အခုပရိုဂရမ်ကို 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 ပြန်ကျော့တာ ဘယ်တော့မှ ရပ်မှာ မဟုတ်ဘူး။ ဒီလိုဖြစ်နေတာကို ဖြတ်ချပစ်ဖို့အတွက် `break` စတိတ်မန့်ကို သုံးထားပါတယ်။ `if val == SENTINEL:` (ထည့်ပေးတဲ့ တန်ဖိုးက သုညဆိုရင်) လက်ရှိ အလုပ်လုပ်နေတဲ့ loop ကို `break` လိုက်မှာဖြစ်တယ်။

အထက်ပါ sentinel loop ပုံစံနှစ်ခုကို ယှဉ်ကြည့်ရင် ပထမတစ်ခုမှာ input တန်ဖိုးထည့်ခိုင်းတာကို loop မစမီ တစ်ခါ ကြိုလုပ်ထားရတယ်။ နောက်တစ်ခုမှာတော့ အဲ့ဒီလို ကြိုလုပ်ထားစရာမလိုဘူး။ Loop ဘလောက်ထဲက စတိတ်မန့်တစ်ခုကို အပြင်မှာထုတ်ထားရတဲ့အတွက် တချို့က ပထမပုံစံထက် ဒုတိယပုံစံက ကုဒ်စတိုင်လ်အားဖြင့် ပိုပြီး သပ်ရပ်ကြော့ရှင်းတယ်လို့ ယူဆကြပါတယ်။

Result Controlled Loop

တစ်နှစ်ငါးရာခိုင်နှုန်း အတိုးနဲ့ ဘဏ်မှာ ဒေါ်လာတစ်ထောင် စုထားတယ်။ တစ်နှစ်ပြည့်တိုင်း အတိုးအရင်းပေါင်းပြီး နောက်နှစ်အတွက် အတိုးတွက်တယ်။ နှစ်ထပ်တိုး (yearly compound interest) လို့ခေါ်ပါတယ်။ ဒီနည်းလမ်းနဲ့ အနည်းဆုံး ဒေါ်လာတစ်သန်း စုမိဖို့ (မီလျံနာတစ်ယောက်ဖြစ်ဖို့) ဘယ်နှစ်နှစ်စောင့်ရမလဲ။ သုံးနှစ်ပြည့်ရင် စုမိမဲ့ ငွေပမာဏ တွက်ထားတာကို 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	$1102.5 \times 0.05 = 55.125$	$1102.5 + 55.125 = 1157.625$

တေဘဲလ် ၂.၂ သုံးနှစ်ဈ 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 တွက်ထားပြီး `yr` ကိုလည်း တစ်နှစ် တိုးပေးတယ်။ `yr`, `interest`, `balance` ထုတ်ပြပေးတယ် (မပြလည်း ပြဿနာမရှိပါ။ တစ်နှစ်စာ တွက်ထားတာ စစ်ကြည့်လို့ရအောင် ထုတ်ကြည့်တာပါ)။ တစ်သန်းမပြည့်မချင်း ပြန်ကျော့ပေးအောင် loop ကွန်ဒီရှင်ကို `balance < TARGET` နဲ့ စစ်ထားတယ်။ တစ်သန်းနဲ့ညီရင် (သို့) တစ်သန်းကျော်သွားတာနဲ့ ကွန်ဒီရှင် `False` ဖြစ်သွားပြီး ထပ်မကျော့တော့ဘူး။ Loop တစ်ခါကျော့တိုင်း တစ်နှစ်စာ အတိုး

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

ပရိုဂရမ် output ကို အခုလို ကော်လံတစ်ခုစီကို အကျယ် တစ်သမတ်တည်းဖြစ်၊ စာသားတွေ ညာဘက်ကပ်ပြီး ညီနေအောင် f-strings ကို format spec လို့ခေါ်တဲ့ ဖော့မတ်သတ်မှတ်တဲ့ နည်းစနစ်ကို 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 အကျယ်တဝင့် မဖော်ပြတော့ဘဲ အခုလို ဖော့မတ်ရအောင် ဘယ်လိုလုပ်ထားလဲကိုပဲ ရှင်းပြပါမယ်။

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

ကော်လံ ခေါင်းစည်း အတွက် f-string ပါ။ ဖော့မတ်လုပ်ရမဲ့ တန်ဖိုး/ဗေရီရေဘဲလ်/အိပ်စ်ပရက်ရှင်က `:` (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။ ကျန်တဲ့ 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 ပြပေးဖို့ပါ။ ကော်လံအကျယ် `4`, `10`, `10` သတ်မှတ်တယ်။ `yr` ကို ကိန်းပြည့်ဂဏန်းအနေနဲ့ ပြပေးအောင် `d` သုံးတယ်။ ကျန်တဲ့နှစ်ခုကို ဒဿမနဲ့ ပြဖို့ `f` သုံးတယ်။ `.2` ကတော့ ဒဿမနောက် ဂဏန်းနှစ်လုံး ပြပေးခိုင်းတာ။ ကိန်းဂဏန်းဆိုရင် သူ့နဂိုအတိုင်း ညာဘက်ကပ်ပေးတဲ့အတွက် `>` ထည့်ဖို့ မလိုဘူး။ ဘယ်ဘက်ကပ်ချင်ရင် `<` သုံးလို့ရတယ်။

၂.၅ လေ့ကျင့်ရန် ဥပမာများ

ကွန်ထရိုးလ် စထရက်ချာတွေ အသုံးချတတ်လာအောင် များများလေ့ကျင့်၊ များများစဉ်းစား၊ များများရေးကြည့် ရမှာပါ။ ဘယ်အတတ်ပညာမဆို များများလေ့ကျင့်မှ ကျွမ်းကျင်လာနိုင်မှာပါ။ ဒီသဘောအရ ပရိုဂရမ်းမင်း ပညာရပ်ဟာလည်း ခွင်းချက်မဖြစ်နိုင်ပါဘူး။

> အောက်ပါ ဥပမာတစ်ခုချင်းကို ပုစ္ဆာနားလည်အောင်ဖတ်ပြီး ကိုယ်တိုင်စဉ်းစား ရေးကြည့်ဖို့ လေးလေးနက်နက် အကြံပြုပါတယ်။ ရေးပြီးသွားရင် ပုစ္ဆာမှာ ဖော်ပြထားချက်နဲ့ အညီ အလုပ်လုပ်/မလုပ် ဟာကွက်မရှိအောင် ဘယ်လိုစစ်ဆေး စိစစ်မလဲ မိမိဘာသာ စဉ်းစားကြည့်ပါ။

Internet Delicatessen

Online ကနေ အစားအသောက်တွေ delivery ပို့ပေးဖို့ အမှာလက်ခံတဲ့ ဆိုင်လေးတစ်ဆိုင်အတွက် ပရိုဂရမ်ရေးပေးရပါမယ်။ အော်ဒါမှာတဲ့အခါ မှာမဲ့ အစားအသောက် နံမည်၊ ဈေးနှုန်းနဲ့ အိပ်စ်ပရက်စ် deliv-

ery ယူမယူ ပရိုဂရမ်မှာ ထည့်ပေးရပါမယ်။ ပရိုဂရမ်က အော်ဒါနဲ့ စုစုပေါင်းကျသင့်ငွေကို ထုတ်ပေးရပါမယ်။ တစ်သောင်းနဲ့အထက်မှာရင် delivery ဖိုး ပေးစရာမလိုပါ။ တစ်သောင်းအောက်ဆိုရင်တော့ နှစ်ရာပေးရပါမယ်။ 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_exp_deli = int(input('Express delivery (0==no, 1==yes): '))

tot_deli_fee = 0
if price >= 10_000 and is_exp_deli == 1:
    tot_deli_fee = 300
elif price >= 10_000 and is_exp_deli == 0:
    tot_deli_fee = 0
elif price < 10_000 and is_exp_deli == 1:
    tot_deli_fee = 200 + 300
elif price < 10_000 and is_exp_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) သုည ထည့်ပေးရမှာပါ။ တစ်/သုည မဟုတ်တဲ့ ဂဏန်းတစ်ခု မှားထည့်မိရင် `if...elif` ကွန်ဒီရှင်းတွေ တစ်ခုမှ `True` ဖြစ်မှာ မဟုတ်ပါဘူး။ မှားထည့်ထားတယ်လို့ သတိပေးဖို့ `else` အပိုင်းမှာ လုပ်ထားတယ်။ မှန်/မမှန် စိစစ်တဲ့အခါ အောက်ပါဇယားမှ ဖြစ်နိုင်ခြေအားလုံး ခြုံငုံမိအောင် စစ်သင့်တယ်။ အိပ်စ်ပရက်စ် delivery အတွက် input ဂဏန်း မှားထည့်ရင် သတိပေးတာကိုလည်း စစ်သင့်တယ်။ တစ်သောင်းဖိုး ဝယ်တာကို သုံးမျိုးစစ်ထားတာ လေ့လာကြည့်ပါ။ တစ်သောင်း မပြည့်တာနဲ့ တစ်သောင်းကျော်ဖိုး အတွက်လည်း အလားတူ စစ်ကြည့်လို့ အားလုံးမှန်တယ်ဆိုရင် ဒီပရိုဂရမ်မှာ bug ပါနိုင်ခြေ လုံးဝမရှိသလောက် ဖြစ်သွားပါပြီ။

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

တေဘဲလ် ၂.၃ သုံးနှစ်ဓာ 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
```

ပရိုဂရမ်တစ်ခုရဲ့ လိုအပ်ချက်ဟာ မကြာခဏ ပြောင်းလဲသွားလေ့ရှိတယ်။ အခုပရိုဂရမ်မှာ ဈေးနှုန်းသတ်မှတ်ချက်တွေ ပြောင်းလဲနိုင်တယ်။ အနည်းဆုံး တစ်သောင်းခွဲဝယ်မှ ပို့ခ free ရမှာဖြစ်ပြီး တစ်သောင်းခွဲအောက်ဆိုရင် ပို့ခ သုံးရာငါးဆယ် ပြောင်းလဲသတ်မှတ်လိုက်တဲ့အပြင် အိပ်စ်ပရက်စ် delivery ကလည်း တစ်ရာဈေးထပ်တက်သွားတယ် ဆိုပါစို့။ တကယ့်လက်တွေ့မှာလည်း ဒါမျိုးဖြစ်လေ့ရှိပါတယ်။ ဒီအတွက်ကို ပရိုဂရမ်ကို ပြင်ပေးရပါမယ်။ ဆိုင်ပိုင်ရှင်ကလည်း ဈေးနှုန်းမကြာခဏ ပြောင်းဖို့လိုနိုင်ကြောင်း ပြောလာပါတယ်။ နောင်လည်း ထပ်ပြင်ပေးဖို့လိုမဲ့ သဘောပါ။ လွယ်လွယ်ကူကူ ပြင်ပေးနိုင်ရင် အကောင်းဆုံးပါ။ ပြင်ဆင်တဲ့အခါ မှားနိုင်ခြေနည်းဖို့လည်း အရေးကြီးပါတယ်။ ခုရေးထားတဲ့အတိုင်းဆိုရင် ပြီးခဲ့တဲ့ပရိုဂရမ်မှာ ပြဿနာရှိနေပါတယ်။

ပြီးခဲ့တဲ့ ပရိုဂရမ်မှာ ပြင်မယ်ဆိုရင် `10_000` ကို လေးနေရာ၊ ပုံမှန်ပို့ခ `200` နဲ့ အိပ်စ်ပရက်စ်အပိုကြေး `300` စတာတွေကို နှစ်နေရာစီ လိုက်ပြင်ရပါမယ်။ အလုပ်ရှုပ်တဲ့အပြင် ပြင်ဖို့ကျန်ခဲ့တာလို့ မှားတာလည်း

ဖြစ်နိုင်တယ်။ “Find and Replace” လုပ်မှာပေါ့လို့ စောဒကတက်စရာ ရှိပါတယ်။ အခုလို ကုဒ်လိုင်းနည်းနည်းပဲရှိတဲ့ ပရိုဂရမ်အသေးလေးမှာ အဆင်ပြေနိုင်ပေမဲ့ ပို့ရှုပ်ထွေးပြီး ကုဒ်လိုင်းတွေများတဲ့ ပရိုဂရမ်မျိုးတွေမှာ “Find and Replace” လုပ်ရင် မပြင်သင့်တာတွေကိုပါ မရည်ရွယ်ပဲ ပြင်မိသွားတာဖြစ်တတ်ပါတယ်။ ပရိုဂရမ်ရေးတဲ့အခါ ကျင့်သုံးရမဲ့ အလေ့အထကောင်းတစ်ခုက ကုဒ်ထဲမှာ ဒီတိုင်းချရေးထားတဲ့ တန်ဖိုးတွေ (literal constants) တွေကို နာမည်ပေးထားတာပါ။ အပေါ်က ပရိုဂရမ်မှာ literal constants တွေချည်း သုံးထားတယ်။ နံမည်ပေးထားတဲ့ constants (named constants) တွေအဖြစ် 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))
```

ဒီပရိုဂရမ်ကို cascading `if` မသုံးဘဲ ရိုးရိုး `if` နဲ့ ရေးလို့လည်း 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_exp_deli == 1:
    tot_deli_fee += EXP_DELI_FEE

if not (is_exp_deli == 0 or is_exp_deli == 1):
    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))
```

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

ဒီပရိုဂရမ်နဲ့ ဒီမတိုင်ခင် သူ့ရှေ့က ပရိုဂရမ်က ဖြစ်နိုင်ခြေအားလုံးအတွက်တော့ ရလဒ် တူတူမထွက်ပါဘူး။ အခြေအနေ တစ်ခုကလွဲလို့ ကျန်တာတွေအတွက်တော့ ရလဒ်တူပါတယ်။ တစ်သောင်းငါးထောင် ထက်ငယ်တဲ့ တန်ဖိုးနဲ့ အိပ်စ်ပရက်စ် delivery အတွက် ဂဏန်း လွဲထည့်ကြည့်ပါ။ 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 ပရိုဂရမ်က အိပ်စ်ပရက်စ် delivery အတွက် မပေါင်းထည့်ပေမဲ့ ပုံမှန် delivery ခ သုံးရာ့ငါးဆယ်ကိုတော့ ထည့်ပေါင်းသွားတာ တွေ့ရမယ်။ မှားထည့်တာကိုပဲ သတိပေးစာသားပြပေးတာ၊ ထည့်မတွက်သွားတာက ပိုပြီး သဘာဝကျတယ်လို့ ယူဆရမှာပါ။

ပြင်ပကနေ ထည့်ပေးတဲ့အခါ မဖြစ်သင့်တဲ့ input တန်ဖိုးတွေ ဝင်မလာအောင် စိစစ်တာကို input validation လို့ ခေါ်တယ်။ တကယ့် လက်တွေ့ အသုံးချ ပရိုဂရမ်တွေမှာ input validation လုပ်ထားဖို့ အရေးကြီးပေမဲ့ ဘီဂင်နာအဆင့် လေ့လာတဲ့ ဥပမာတွေမှာတော့ လေ့လာရင်းကိစ္စကနေ လမ်းကြောင်းမချော်သွားအောင် ဆင်ခြင်ရမှာဖြစ်ပြီး သင့်တော်ရုံ ဆက်စပ်ရှင်းပြပါမယ်။

တာရာ ပရက်ရှာ

ကားဘီးလေထိုးတာဟာ ကားရဲ့ စွမ်းဆောင်ရည်ရော အန္တရာယ်ကင်းဖို့အတွက်ပါ ပမာနကျပါတယ်။ ကားတစ်စီးအတွက် အကြံပြုထားတဲ့ တာယာပရက်ရှာ (recommended pressure) အပေါ် မူတည်ပြီး လေဘယ်လောက်တင်းလို့ရလဲ၊ လျော့လို့ရလဲ ရှိပါတယ်။ ဥပမာ recommended pressure က 35 psi (pounds per square inch) ဖြစ်ရင် အလွန်ဆုံး 31.5 psi ထိ လေလျော့လို့ရပါတယ်။ လေပိုတင်းမယ်ဆိုရင်လည်း အလွန်အလွန်ဆုံး 44 psi အထိ ရပါနိုင်ပါတယ်။

ရှေ့တာယာနှစ်လုံး ပရက်ရှာအနီးစပ်ဆုံးတူသင့်ပြီး 3 psi အထိ ကွာဟလို့ရတယ်။ ကွာဟချက်က 3 psi ထက်တော့ မများသင့်ဘူး။ နောက်တာယာနှစ်လုံးလည်း ထိုနည်းတူစွာပဲ ဖြစ်တယ်။ ကားမော်ဒယ်အလိုက် recommended pressure ကွာခြားပေမဲ့ အိမ်စီးကားအများစုအတွက် 35 psi ဖြစ်တယ်လို့ ယူဆပြီး တာယာပရက်ရှာ အိုကေမကေ စစ်ပေးတဲ့ ပရိုဂရမ် ရေးပေးရပါမယ်။ Input အနေနဲ့ တာယာတစ်ခုချင်းအတွက် ပရက်ရှာ psi တန်ဖိုး ထည့်ပေးရမှာပါ။ ထည့်ပေးလိုက်တဲ့ တာယာပရက်ရှာ 31.5 psi ထက်နည်းနေရင် သို့မဟုတ် 44 psi ထက်များနေတာနဲ့ သတ်မှတ်ဘောင် မဝင် (out of range) ဖြစ်နေကြောင်း သတိပေးရပါမယ်။ တာယာအားလုံးရဲ့ ပရက်ရှာတွေ သတ်မှတ်ဘောင်အတွင်း ဝင်တယ်၊ ရှေ့တာယာနှစ်လုံး ကွာဟချက်၊ နောက်တာယာနှစ်လုံး ကွာဟချက်တွေ ခွင့်ပြုလို့ရတာထက် မပိုဘူးဆိုရင် 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 ရှင်းပြဖို့ လိုပါမယ်။ စစချင်း `False` တန်ဖိုး ထည့်ထားတာ တွေ့ရမှာပါ။ `if` စတိတ်မန့်တွေက တာယာတစ်ခုချင်းကို out of range ဖြစ်နေလား စစ်ထားတာတွေ့ရတယ်။ တာယာတစ်ခု out of range ဖြစ်တာနဲ့ `is_out_of_range` က `True` ဖြစ်သွားမှာပါ။

အောက်ပိုင်းမှာ `front_diff` နဲ့ `rear_diff` ရှာတဲ့အခါ `abs` ဖန်ရှင်နဲ့ ပကတိတန်ဖိုး ယူထားတာ သတိပြုပါ။ ပရက်ရှာ ခြားနားချက်ရှာတဲ့အခါ အနှုတ်တန်ဖိုး ထွက်နိုင်တဲ့အတွက်ကြောင့်ပါ။ ဘယ်ဘက် တာယာက ပရက်ရှာနည်းနေတဲ့အခါ (ဥပမာ $32 - 37 = -5$) အနှုတ်တန်ဖိုး ဖြစ်နေမယ်။ ဒါကြောင့် ပကတိတန်ဖိုးယူမှပဲ ကွာဟချက် 3 psi ကျော်မကျော်စစ်ပေးတဲ့အခါ အဖြေမှန်ရပါမယ်။

```
front_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
    and rear_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
    and not is_out_of_range
```

`and` နှစ်ခုနဲ့ ဆက်ထားရင် သုံးခုလုံးမှန်မှပဲ `True` ထွက်မှာပါ။ တစ်ခုမှားတာနဲ့ အိပ်စ်ပရက်ရှင်တစ်ခုလုံး ရလဒ် `False` ပဲ။ Out of range မဖြစ်ရဘူး ဆိုတာကို `not is_out_of_range` နဲ့ စစ်ထားတယ်။ `is_out_of_range` တန်ဖိုး `False` ဖြစ်မှ **`not`** `is_out_of_range` က `True` ဖြစ်မယ်။

အခန်း ၃

Inheritance and Polymorphism

Inheritance ဟာ ရှိထားပြီး ကလပ်စ်တစ်ခုရဲ့ attribute နဲ့ မက်သဒ်တွေကို အခြားကလပ်စ်ကနေ ဆက်ခံယူဖို့ ဖြစ်နိုင်စေတယ်။ ကလပ်စ်တစ်ခုမှာ ရေးထားတဲ့ ပရိုဂရမ် ကုဒ်တွေကို ထပ်ခါထပ်ခါ ရေးစရာ မလိုဘဲ ပြန်လည် အသုံးပြုလို့ရစေတဲ့ နည်းလို့လည်း ယူဆနိုင်တယ်။ ဆက်ခံရရှိတဲ့ attribute နဲ့ မက်သဒ်တွေအပြင် အသစ် ထပ်ထည့်တာ၊ နဂိုရှိရင်းကို ပြင်ဆင်တာလည်း ကလပ်စ်အသစ်က လုပ်ဆောင်လို့ရမှာ ဖြစ်တယ်။ Object-oriented programming မှာ inheritance ဟာ အရေးအပါဆုံး သဘောတရား တစ်ခု ဆိုရင်လည်း မမှားဘူး။ ဂိမ်းရေးတဲ့ လိုက်ဘရီတွေ၊ Graphical User Interface (GUI) အတွက် လိုက်ဘရီတွေ အသုံးပြုမယ်ဆိုရင် inheritance သဘောတရားက မသိမဖြစ်ပါပဲ။ ဒီအခန်းအတွက် inheritance အသုံးချ ဥပမာအနေနဲ့ 'Breakout' လို့ခေါ်တဲ့ ဂိမ်းလေးတစ်ခုကို Arcade လိုက်ဘရီနဲ့ ရေးထားတာကို နောက်ပိုင်းမှာ လေ့လာကြရမှာပါ။

ရှေ့အခန်းမှာ ဖော်ပြခဲ့တဲ့ has-a အပြင် object-oriented programming မှာ အရေးပါတဲ့ နောက် relationship တစ်မျိုးက *is-a* ဖြစ်ပါတယ်။ Is-a relationship ရှိတဲ့ အရာတွေကို inheritance နဲ့ ထင်ဟပ်ဖော်ပြနိုင်တယ်။ ဘတ်စ်ကားသည် ကားဖြစ်သည်။ ကားသည် ယာဉ်ဖြစ်သည်။ ယာဉ်၊ ကားနဲ့ ဘတ်စ်ကားတို့အကြား ဆက်စပ်မှုဟာ is-a relationship ဖြစ်တယ်။ `Vehicle`, `Car` နဲ့ `Bus` ကလပ်စ်တွေဟာ အစဉ်အလိုက် inherit လုပ်ယူနိုင်တာကို တွေ့ရမှာပါ။

၃.၀ Inheritance ဥပမာ '`Account` Class Hierarchy'

ဘဏ်အကောင့် အမျိုးအစား အမျိုးမျိုး ရှိပါတယ်။ အတိုးနှုန်း၊ တစ်နေ့တာ လုပ်ဆောင်နိုင်တဲ့ transaction အကြိမ်အရေအတွက်၊ အနည်းဆုံး ထားရှိရမဲ့ ငွေပမာဏ၊ လက်ရှိ လက်ကျန်ငွေထက် ပိုထုတ်လို့ ရ/မရ (overdraft) စတဲ့အချက်တွေ အကောင့်တစ်မျိုးနဲ့တစ်မျိုး မတူကြပါဘူး။

ဥပမာ savings account (ငွေစုအကောင့်) ဆိုရင် ဘဏ်တိုးရမယ်။ တစ်နေ့တာ transaction အကြိမ်အရေအတွက်ရော ထုတ်ယူနိုင်တဲ့ ငွေပမာဏပါ အကန့်အသတ်ရှိတယ်။ ဒါကြောင့် ဒီအကောင့် အမျိုးအစားက လုပ်ငန်းသုံးအတွက် အဆင်မပြေဘူး။ Current account ကတော့ စီးပွားရေး လုပ်ငန်းတွေအတွက် အဓိက ရည်ရွယ်တယ်။ Transaction အကန့်အသတ်မရှိဘူး။ လိုသလောက် ထုတ်ယူလို့ရတယ်။ Overdraft လို့ခေါ်တဲ့ ကိုယ့်မှာရှိတာထက် (အကန့်အသတ်တော့ရှိတာပေါ့) ပိုထုတ်လို့ရတယ်။ Current account ကို ဘဏ်က အတိုးပေးလေ့မရှိဘူး။

`SavingAccount` နဲ့ `CurrentAccount` ကလပ်စ်ကို အောက်ပါအတိုင်း သတ်မှတ်နိုင်ပါတယ်။ ကလပ်စ်နှစ်ခုကို နှိုင်းယှဉ်ကြည့်ရင် တူညီတဲ့အပိုင်းတွေ ပါဝင်နေတာ တွေ့ရမှာပါ။ နှစ်ခုလုံးမှာ `deposit` မက်သဒ်က တူတူပါပဲ။ `_holder`, `_balance`, `_acc_number` ဗေရီရေဘဲလ်တွေ ပါဝင်တာချင်းလည်း တူ

တယ်။ ကွာခြားချက်တချို့ကိုလည်း တွေ့ရပါတယ်။ withdraw မက်သဒ် နှစ်ခု မတူကြဘူး။ _txn_cnt_tot, _txn_amt_tot နဲ့ provide_interest တို့ကို SavingAccount မှာပဲ တွေ့ရတယ်။ တစ်ဖက်မှာလည်း _overdraft_amt နဲ့ is_overdrafted တို့ကိုတော့ CurrentAccount မှာ တွေ့မှာဖြစ်ပြီး 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)
```

ဒီကလပ်စ်နှစ်ခုမှာ ထပ်နေတဲ့ တူညီတဲ့ကုဒ်တွေကို တစ်ခါပဲ ရေးဖို့လိုမယ်ဆိုရင် ပိုပြီးအဆင်ပြေမှာပါ။ ဒီလိုအခြေအနေမျိုးမှာ inheritance က ဘယ်လို အထောက်အကူပြုလဲ ကြည့်ရအောင်။ ထပ်နေတဲ့ တူညီတဲ့ကုဒ်တွေကို ဘုံထုတ်လိုက်ပြီး ကလပ်စ်တစ်ခု 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 ကလပ်စ်မှာ ခွဲထုတ်ထားတာ သတိပြုကြည့်ပါ။ 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 က Account ကို inherit လုပ်မှာဖြစ်တာကြောင့် ကလပ်စ်သတ်မှတ်တဲ့အခါ အခုလို ရေးရပါမယ်

```
class CurrentAccount(Account):
```

Account ကို *superclass* လို့ခေါ်ပြီး သူ့ဆီကနေ ဆက်ခံယူမဲ့ CurrentAccount ကို *subclass* လို့ ခေါ်ပါတယ်။ ပေးမဲ့ကလပ်စ်က superclass ၊ ယူမဲ့ ကလပ်စ်က subclass ပေါ့။ *parent* နဲ့ *child* လို့ လည်းခေါ်တယ်။

Subclass initialize လုပ်တဲ့အခါ superclass အပိုင်းကိုလည်း initialize လုပ်ဖို့လိုတယ်။ ဒီအတွက် superclass `__init__` ကို အခုလို ခေါ်ရပါတယ်

```
# call __init__ method of the superclass
super().__init__(holder, acc_number, balance)
```

`_holder`, `_acc_number`, `_balance` instance variable တွေက ဆက်ခံရရှိထားတာပါ။ ၎င်းတို့ကို initialize လုပ်ဖို့အတွက် superclass initializer ကို ခေါ်ရပါမယ်။ Subclass က တိုက်ရိုက် မလုပ်ရပါဘူး။ 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 တွေ့ရမှာပါ။ ဆက်ခံရရှိတဲ့ instance variable တွေကို subclass မက်သဒ်ထဲမှာ `self` နဲ့ ရည်ညွှန်းအသုံးပြုလို့ ရတယ်။

`SavingAccount` ကိုအောက်ပါအတိုင်း သတ်မှတ်ပါတယ်။ တစ်နေ့တာ transaction အကြိမ်အရေအတွက်နဲ့ ထုတ်ယူတဲ့ ပမာဏအတွက် `_txn_cnt_tot` နဲ့ `_txn_amt_tot` instance variable တွေ ထပ်ဖြည့်ထားတယ်။ (`_txn_cnt_tot` နဲ့ `_txn_amt_tot` ကို တစ်နေ့တာကုန်ဆုံးတိုင်း သုညဖြစ်အောင် ပြန်လုပ်ထားဖို့ လိုပါလိမ့်မယ်။ အခုဥပမာမှာ 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 အပြင် သတ်မှတ်ကာလအတွင်း မထုတ်ဘဲထားရတဲ့ fixed deposit ၊ အသေးစား လုပ်ငန်းတွေအတွက် SME စတဲ့ အခြားအကောင့် အမျိုးအစားတွေလည်း ရှိတယ်။ ဒီအကောင့်တွေနဲ့ သက်ဆိုင်တဲ့ ကလပ်စ်တွေကလည်း `Account` ကို inherit လုပ်နိုင်ပါတယ်။

၃.၂ Overriding

Superclass ကနေ ဆက်ခံရရှိထားတဲ့ မက်သဒ်ကို subclass မှာ ပြင်ဆင်သတ်မှတ်တာကို *method overriding* လို့ ခေါ်ပါတယ်။ Deposit လုပ်တဲ့အခါ current account က အနည်းဆုံး ဆယ်သိန်း ဖြစ်မှ လက်ခံတယ်ဆိုပါစို့။ ဆက်ခံ ရထားတဲ့ မူလ `deposit` မက်သဒ်က ဒီလိုအပ်ချက်နဲ့ မကိုက်ညီတော့ဘူး။ အခုလို အခြေအနေမျိုးမှာ ဆက်ခံယူမဲ့ `CurrentAccount` ဟာ `deposit` မက်သဒ်ကို လက်ရှိ လိုအပ်ချက်နဲ့ 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
```

မက်သဒ်တစ်ခုကို ဆက်ခံရရှိတဲ့အတိုင်း အသုံးပြုလို့ အဆင်မပြေတဲ့အခါ subclass က override လုပ်နိုင်ပါတယ်။ Subclass အများစုက ဆက်ခံရရှိတဲ့အတိုင်း အသုံးပြုလို့ရပြီး တချို့ အနည်းစုကပဲ override ရတဲ့အခါ ကုဒ်တွေထပ်နေတာ သိသိသာသာ လျော့သွားမှာပါ။

ဆက်ခံရရှိတဲ့အတိုင်း အသုံးပြုလို့ရနိုင်ပေမဲ့ optimization အတွက် override လုပ်ရတာလည်း ရှိတယ်။ Polygon ပါတ်လည်နားရှာတဲ့ နည်းလမ်းကို rectangle အတွက်လည်း သုံးလို့ရပေမဲ့ rectangle အတွက် သီးသန့်ဖော်မြူလာ $2 \times (width + height)$ က ပိုရှင်းလင်းပြီး တွက်ချက်ရပိုမြန်မှာ အမှန်ပါပဲ။ ဒါကြောင့် စွမ်းဆောင်ရည်အတွက် `Rectangle` ကလပ်စ်မှာ `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
```

```python
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 ပရိုဂရမ်တစ်ခုရဲ့ စထရက်ချာကို ထိန်းကွပ်ပေးဖို့ အသုံးပြုတာကိုလည်း တချို့လိုက်ဘရီတွေမှာ တွေ့ရတယ်။ ဥပမာ Arcade လိုက်ဘရီနဲ့ ဂိမ်းရေးတဲ့အခါ သူမှာပါတဲ့ ကလပ်စ်တွေကို inherit လုပ်ပြီး ကိုယ်လိုချင်တဲ့ ပုံစံနဲ့ အလုပ်လုပ်အောင် မက်သဒ်တွေကို လိုအပ်သလို override လုပ်ပေးရတယ်။ ဒီအခန်း နောက်ပိုင်းမှာ Arcade လိုက်ဘရီနဲ့ ဂိမ်းလေးတစ်ခု တည်ဆောက်တဲ့အခါ တွေ့ရမှာပါ။

၃.၃ Multilevel Inheritance

ကလပ်စ် A ကို B က ဆက်ခံထားမယ်။ တစ်ခါ B ကို C က ထပ်ဆင့် ဆက်ခံယူမယ်။ ဒီလို အဆင့်ဆင့် inherit လုပ်တာကို *multilevel inheritance* လို့ခေါ်တယ်။ ကလပ်စ် C ဟာ B နဲ့ A နှစ်ခုလုံးရဲ့ attribute နဲ့ မက်သဒ်တွေကို ရရှိမှာပါ (A မှာ ရှိတာတွေကို B ကနေတစ်ဆင့် ရတာ)။ ရှေ့က ဥပမာမှာ inheritance level နှစ်ခုတွေ့ရတယ်။ အပေါ်အဆင့်မှာ `Account`, အောက်တစ်ဆင့်မှာ သူ့ကိုဆက်ခံယူထားတဲ့ `SavingAccount` နဲ့ `CurrentAccount` ရှိတယ်။ `SavingAccount` နဲ့ `CurrentAccount` ကို ဆက်ခံထားတဲ့ ကလပ်စ်တွေ နောက်တစ်ဆင့်ရှိလာရင် သုံးဆင့်ဖြစ်သွားမှာပါ။

```python
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
```

```python
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
```

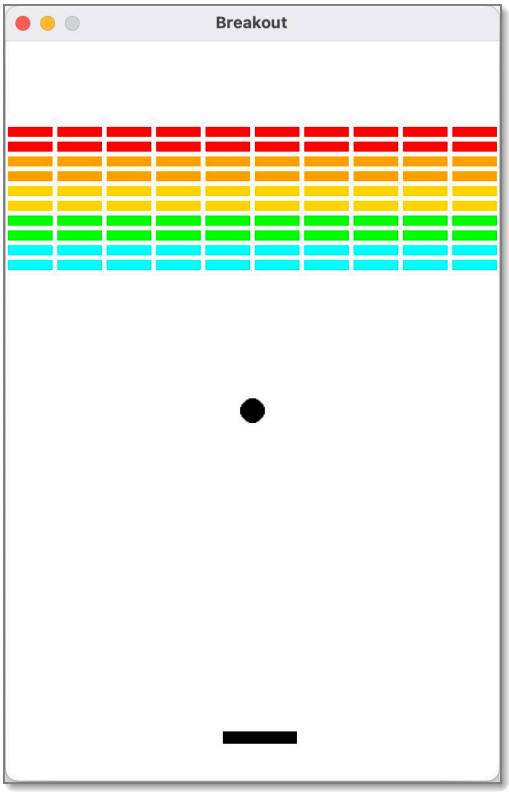
၃.၄ Class Hierarchy and UML

၃.၅ Is-A Relationship and Inheritance

၃.၆ အသုံးချ ဥပမာ (၁) Breakout Game

ဒီအခန်းအတွက် အသုံးချဥပမာက Breakout လို့ခေါ်တဲ့ ရိုးရှင်းတဲ့ ဂိမ်းလေးတစ်ခု တည်ဆောက်ပါမယ်။ Breakout ဂိမ်းဟာ ကမ္ဘာကျော် Apple ကုမ္ပဏီ ပူးတွဲတည်ထောင်သူ Steve Wozniak ဒီဇိုင်းလုပ်ခဲ့တဲ့ ဂန္ထဝင်ဂိမ်းတစ်ခု ဖြစ်ပါတယ်။ ဗားရှင်းတွေအမျိုးမျိုး ရှိပေမဲ့ အဓိကအနှစ်သာရကတော့ ရွေ့နေတဲ့ ဘောလုံးကို paddle ပြားနဲ့ လိုက်ဖမ်းပြီး အစီအရီရှိနေတဲ့ အုတ်ခဲတွေ အားလုံးကို ကုန်တဲ့အထိ ဖျက်ရတာပါ။ ဂိမ်းစစချင်း အနေအထားကို ပုံ (၃.၁) မှာ တွေ့ရပါမယ်။ အပေါ်ပိုင်းမှာက အုတ်နံရံပေါ့။ ဘောလုံးက အောက်ကိုကျလာမယ်။ paddle ပြားနဲ့ လိုက်ဖမ်းရမယ်။ ဘောလုံးက paddle, အုတ်ခဲ၊ ဘယ်/ညာ/အပေါ် ဘောင်တွေနဲ့ ဝင်တိုက်ရင် ပြန်ကန်ထွက်တယ်။ အုတ်ခဲတွေက ဘောလုံးနဲ့ တိုက်မိရင် ပျက်သွားမယ်။ ဘောလုံးကို မဖမ်းလိုက်နိုင်လို့ paddle ပြားအောက်ဘက် ကျသွားရင် ရှုံးမယ် (အောက်ဘက်ကျသွားရင် ပြန်ကန်မထွက်ဘူး)။ အုတ်ခဲတွေ ကုန်တဲ့ထိ ဘောလုံးမကျသွားအောင် ထိန်းထားနိုင်ရင် အနိုင်ရမှာဖြစ်တယ်။

Arcade လိုက်ဘရီနဲ့ ရေးမှာပါ။ Arcade လိုက်ဘရီဟာ inheritance ကို အခြေခံထားတယ်။ `Window`, `View`, `Sprite` စတဲ့ ကလပ်စ်တွေကို လိုက်ဘရီကပေးထားတယ်။ ဂိမ်းတစ်ခုရေးတဲ့အခါ ဒီကလပ်စ်တွေကို inherit လုပ်ပြီး လိုအပ်တဲ့ မက်သဒ်တွေကို override လုပ်ပေးရုံပဲ။ လိုက်ဘရီက ပရိဂရမ်ကို အကြမ်းထည် စထရက်ချာ ချထားပေးတယ်။ အဲ့ဒီ စထရက်ချာထဲမှာ ကိုယ်တိုင်စိတ်ကြိုက် ဖန်တီးချင်တဲ့နေရာတွေကို ပြင်ဆင်ဖြည့်စွက်လို့ရအောင် လုပ်ပေးထားတာလို့ ယူဆနိုင်တယ်။ ကြိုတင်သိထားဖို့



ပုံ ၃.၁

လိုအပ်တာတွေ တစ်ဆင့်ချင်း အရင်ကြည့်ရအောင်။

`arcade.Window`

`Window` ကလပ်စ်ဟာ ဂရပ်ဖစ် window တစ်ခုကို ကိုယ်စားပြုတယ်။ ဂရပ်ဖစ်ရုပ်ပုံတွေ ဆွဲဖို့နဲ့ အန်နီမေးရှင်း လုပ်ဖို့ လိုအပ်တာတွေ ဒီကလပ်စ်မှာ ထောက်ပံ့ပေးထားတယ်။ ဒါ့အပြင် မောက်စ်၊ ကီးဘုဒ် စတဲ့ input device တွေနဲ့ ဂိမ်းကို ကွန်ထရိုးလ်လုပ်လို့ရစေမဲ့ နည်းလမ်းတွေလည်း ပါတယ်။

Arcade ပရိုဂရမ်တစ်ခုကို run တဲ့အခါ `Window` မှာ ပါတဲ့ `on_draw` မက်သဒ်ကို တစ်စက္ကန့်အကြိမ် ခြောက်ဆယ် ခေါ်ပေးတယ်။ သူ့နဂိုအတိုင်း `on_draw` မက်သဒ်က ဘာမှ လုပ်ဆောင်မပေးဘူး။ ကိုယ့်လိုချင်တဲ့ ဂရပ်ဖစ်ပုံ ဖော်ဖို့ `Window` ကို inherit လုပ်ပြီး override လုပ်ပေးရမယ်။ အပြာရောင် ဘောလုံးတစ်ခု အလယ်ဗဟိုမှာ ဆွဲမယ်ဆိုပါစို့ ...

```
# 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။ ဒီပရိုဂရမ် run တဲ့အခါ on_draw ကို တစ်စက္ကန့်တိုင်း တစ်စက္ကန့်တိုင်းမှာ အကြိမ်ခြောက်ဆယ် (60 frames per second) တောက်လျှောက် ခေါ်နေမှာပါ။ ပရိုဂရမ် window မပိတ်မချင်းပေါ့။ ဒီလိုဖြစ်အောင် Arcade လိုက်ဘရီက 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။ အန်နီမေးရှင်းအတွက် ဒီမက်သဒ်က အရေးကြီးတယ်ဆိုတာ တွေ့ရပါမယ်။

မောက်စ် (သို့) ကီးဘုဒ်နဲ့ ဘောလုံးကို ရွှေ့လို့ရအောင်လည်း လုပ်လို့ရတယ်။ ခက်ခဲတဲ့ ကိစ္စတွေကို Window က အဆင်သင့် လုပ်ထားပေးတာပါ။ ဒါဟာ အထူးအဆန်း ဖြစ်နေနိုင်ပါတယ်။ ဘယ်လိုများ လုပ်ထားလဲ အကြမ်းဖျဉ်း သဘောတရား နားလည်ချင်တယ်ဆိုရင် အောက်ပါ ဥပမာကို 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 ကလွဲလို့ ကျန်မက်သဒ်တွေအားလုံး အပြည့်အစုံရေးထားတာ တွေ့ရမှာပါ။ do_my_task က my_task ကိုခေါ်ထားတယ်။ MyConsolePrinter က ထုတ်ပေးချင်တဲ့ စာသားအတွက် my_task ကို override လုပ်ထားတာကို ဂရုပြုပါ။ ဒီသဘောတရားအတိုင်း Arcade လိုက် ဘရီက Window, View, Sprite စတဲ့ ကလပ်စ်တွေကလည်း ဒီသဘောတရားပဲ။ ဂိမ်းတစ်ခုအတွက် လိုအပ်ချက်တွေကို ပြင်ဆင်ပေးထားတယ်။ ပုံစံချပေးထားတယ်။ အသုံးပြုသူက ဒီကလပ်စ်တွေကို inherit လုပ်ပြီး သူ့ဂိမ်း လိုအပ်ချက်အတိုင်း ဖြစ်အောင် မက်သဒ်တွေကို override လုပ်ပြီး စိတ်ကြိုက် ဖန်တီးယူရတာ။

အန်နီမေးရှင်း

စောစောက ပရိုဂရမ်မှာ စက်ဝိုင်းကို ညာဘက်ဘက်အပေါ်ကို ရွေ့သွားအောင် အန်နီမေးရှင်း လုပ်မယ်ဆိုပါစို့။ 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(arcade.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

ဂိမ်းတစ်ခုမှာ ပါဝင်တဲ့ ဇာတ်ကောင်တွေ၊ အရာဝတ္ထုပစ္စည်းတွေကို Sprite ကလပ်စ်ကို inherit လုပ်ပြီး ကိုယ်စားပြုနိုင်ပါတယ်။ တစ်ခုနဲ့တစ်ခု ဝင်တိုက်တာ (collision) ၊ အပေါ်/အောက် (သို့) ဘယ်/ညာလှည့်တာ၊ အရာဝတ္ထုတွေကို အုပ်စုလိုက် ရွှေ့လျားစေတာ စတာတွေကို အလွယ်တကူ လုပ်ဆောင်လို့ရအောင် Sprite ကလပ်စ်က ထောက်ပံ့ပေးထားတယ်။ Window ဘောင် တစ်ဖက်ဖက်နဲ့ ဝင်တိုက်ရင် ဘောလုံးက ပြန်ကန်ထွက်တဲ့ အန်နီမေးရှင်းဥပမာကို ကြည့်ရအောင် ...

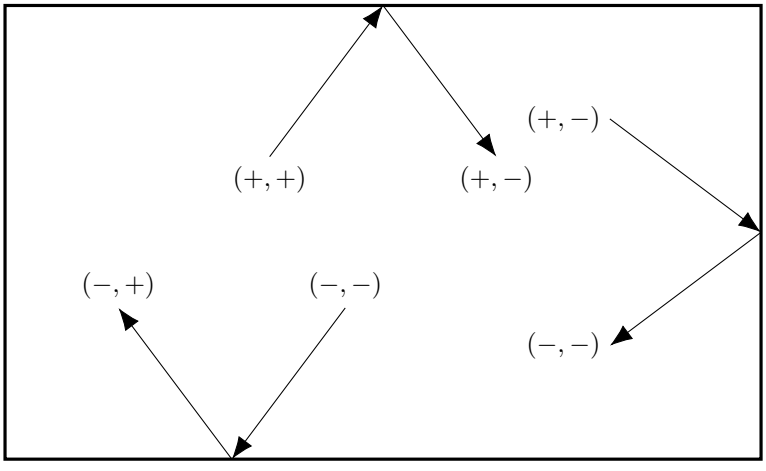
```
# File: arcade_bouncing_ball.py
import arcade
SCREEN_WIDTH = 680
SCREEN_HEIGHT = 480


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 က Sprite ဗဟိုမှတ် တည်နေရာ။ change_x, change_y က အန်နီမေးရှင်းလုပ်ရင် နည်းနည်းချင်း ရွှေ့ပေးရမဲ့ x နဲ့ y ပမာဏ။ Sprite အမြန်နှုန်းနဲ့ ဦးတည်ရာကို ဒီနှစ်ခု

ရဲ့ တန်ဖိုးနဲ့ ထိန်းညှိပေးနိုင်တယ်။ `left`, `right`, `top`, `bottom` attribute တွေကတော့ `Sprite` ရဲ့ ဘယ်/ညာ x တန်ဖိုးနဲ့ အထက်/အောက် y တန်ဖိုးတွေကို ဖော်ပြတယ်။ ဒီ ဗေရီရေဘဲလ်တွေ အားလုံးကို superclass `Sprite` ကနေ ဆက်ခံရရှိထားတာ။

ဘယ် (သို့) ညာဘက် ဘောင်နဲ့ ထိရင် `change_x` ကို အပေါင်းအနှုတ် ပြောင်းပြန် လုပ်ပေးရပါမယ်။ အထက် (သို့) အောက် ဘောင်နဲ့ ထိရင် `change_y` ကို ပြောင်းပြန် လုပ်ပေးရပါမယ်။ ပုံ (၃.၂) မှာ ကြည့်ပါ။

`Ball` ကလပ်စ်မှာ `__init__` မက်သဒ် မပါဘူး။ (Subclass မှာ `__init__` မပါရင် အော့ဂျက် ဖန်တီးတဲ့အခါ ဆက်ခံရရှိထားတဲ့ superclass `__init__` ကိုပဲခေါ်ပါတယ်)။



ပုံ ၃.၂ ဘောင်နဲ့တိုက်တဲ့အခါ velocity လက္ခဏာ ပြောင်းလဲပုံ

`MyGame` ကလပ်စ်မှာ ဒီ ဘောလုံးကို ဘယ်လို အသုံးပြုထားလဲ ဆက်စပ်ကြည့်ပါ။

```
# 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
        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` အော့ဘ်ဂျက် ဖန်တီးတဲ့အခါ image file နံမည်နဲ့ scale factor ထည့်ပေးရပါတယ်။ ဘောလုံးကို `ball.png` ဖိုင်၊ scale factor 0.2 နဲ့

```
self._ball = Ball('ball.png', 0.2)
```

ဖန်တီးတယ်။ Image file က ကုဒ်ဖိုင်နဲ့ ဖိုဒါတစ်ခုထဲမှာ ရှိရပါမယ်။ ဒါက အလွယ်နည်းကို ပြောတာပါ။ အခြားဖိုဒါမှာ ထားလို့ရပေမဲ့ နည်းနည်း ပို ရှုပ်လို့။

`Sprite` ရွှေ့လျားတာနဲ့ collision ဖြစ်တဲ့ ကိစ္စတွေအတွက် အဓိက ဂိမ်းလော့ဂျစ်ကို `on_update` မက်သဒ်မှာ ရေးပေးရမယ်။ ဒါလည်းပဲ တကယ်က ဆက်ခံထားတဲ့ `on_update` ကို override လုပ်တာပါ။ ဒီမက်သဒ်ကိုလည်း တစ်စက္ကန့် အကြိမ်ခြောက်ဆယ် အလိုအလျောက် ခေါ်ပေးပါတယ်။ ဒါပေမဲ့ `on_draw` ရည်ရွယ်ချက်ချင်း မတူပါဘူး။ `on_draw` က စခရင်ကိုပဲ refresh လုပ်ပေးတာ။ တစ်နည်းအားဖြင့် window ပေါ်မှာ ဂရပ်ဖစ်ပုံပဲ ဖော်ပေးတာ။ ဂိမ်းလော့ဂျစ်နဲ့ သက်ဆိုင်တာတွေ မလုပ်ဘူး။ `on_update` ကျတော့ ပုံဆွဲတဲ့ကိစ္စကို မလုပ်ဘူး။ ဂိမ်းလော့ဂျစ်သီးသန့် လုပ်ဆောင်တယ်။ ဒီမက်သဒ်နှစ်ခု လုပ်ဆောင်ပေးတဲ့ တာဝန်ကို ခွဲခြားထားရပါမယ်။ ဂိမ်းမှာပါဝင်တဲ့ `Sprite` အားလုံးရဲ့ state ကို update လုပ်ပေးခြင်းဟာ `on_update` ရဲ့ အဓိကတာဝန်တွေထဲက တစ်ခုအပါအဝင် ဖြစ်တယ်။ အခုဥပမာမှာ ဘောလုံးရဲ့ state ကို

```
self._ball.update()
```

ခေါ်ပြီး update လုပ်ပေးထားတယ်။

`on_update` မက်သဒ် `delta_time` ပါရာမီတာကို ရှင်းပြဖို့ ကျန်ပါသေးတယ်။ `on_update` ကို ခေါ်တဲ့အခါ နောက်ဆုံးခေါ်ခဲ့တဲ့ အချိန်နဲ့ အခုခေါ်တဲ့ အချိန်ကြား ကွာဟချက်ကို `delta_time` အနေနဲ့ ထည့်ပေးတယ်။ Arcade လိုက်ဘရီ နောက်ကွယ်က အလုပ်လုပ်ပုံ သဘောတရားအရ ထည့်ထားတဲ့ ပါရာမီတာဖြစ်ပြီး လိုက်ဘရီ သုံးတဲ့သူအနေနဲ့ အသေးစိတ်သိဖို့ မလိုအပ်ပါဘူး။ `on_update` ကို override လုပ်တဲ့အခါ `delta_time` ပါရာမီတာ မကျန်ခဲ့ရင် ရပါပြီ။

Event Handling

မောက်စ်၊ ကီးဘုဒ်၊ joystick စတဲ့ input device တွေနဲ့ ဂိမ်းကို ထိန်းချုပ်ဖို့အတွက်လည်း `Window` က လပ်စ်က လွယ်အောင်လုပ်ထားတယ်။ `on_mouse_motion`, `on_mouse_press`, `on_key_press` စတဲ့ မက်သဒ်တွေကို override လုပ်ရုံပါပဲ။ အဖြစ်အပျက် တစ်စုံတစ်ခု (event) ကို ပရိုဂရမ်က တုံ့ပြန်လုပ်ဆောင်ပေးတာကို *event handling* လို့ ခေါ်တယ်။ ကီးဘုဒ် ကီးနှိပ်/လွှတ် လိုက်တာ၊ မောက်စ် ကလစ်နှိပ်/လွှတ် လိုက်တာ၊ မောက်စ်ရွှေ့တာ စတာတွေဟာ event ဥပမာတချို့ ဖြစ်တယ်။ ဒါတွေကို ပရိုဂရမ်က တုံ့ပြန်လုပ်ဆောင်ချင်တဲ့အခါ event handling ကို သုံးရပါတယ်။

ဒါက ဘောလုံးကို မောက်စ်နဲ့ ရွှေ့တဲ့ နမူနာပါ။ မောက်စ်ကို နှိပ်ထားပြီး ရွှေ့ရတာ (dragging) မဟုတ်ဘူး။ ဒီတိုင်း ပို့ငိုတာရွှေ့တဲ့ နောက်ကို ဘောလုံးက 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 ရွှေ့ရင် မောက်စ်ပွိုင့်တာရွေ့နေသ၍ on_mouse_motion ကို တစ်ခါပြီးတစ်ခါ အဆက်မပြတ် ခေါ်နေမှာပါ။ မောက်စ် ရပ်လိုက်ရင် ဒီမက်သဒ်ခေါ်တာလည်း ရပ်သွားမယ်။ ဒီလိုဖြစ်အောင် Window က လုပ်ပေးထားတာ။ x နဲ့ y က လက်ရှိ မောက်စ်ပွိုင့်တာရဲ့ တည်နေရာ။ dx က ဒီမက်သဒ်ကို နောက်ဆုံး ခေါ်ခဲ့တဲ့ အချိန်နဲ့ အခုခေါ်တဲ့ အချိန်အတွင်း ပြောင်းလဲသွားတဲ့ x ကွာဟချက်။ dy က ဒီမက်သဒ်ကို နောက်ဆုံးခေါ်ခဲ့တဲ့ အချိန်နဲ့ အခုခေါ်တဲ့ အချိန်အတွင်း ပြောင်းလဲသွားတဲ့ y ကွာဟချက်။ နောက်ဆုံးခေါ်ခဲ့တုံးက မောက်စ်က (x_1, y_1) မှာ ရှိခဲ့တယ်၊ အခုခေါ်တဲ့အချိန် (x_2, y_2) မှာ ဆိုပါစို့။ $dx = x_2 - x_1$, $dy = y_2 - y_1$ ဖြစ်တယ်။

မောက်စ် ကလစ်နှိပ်တဲ့အခါ တစ်ခုခု လုပ်မယ်ဆိုရင် on_mouse_press ကို override လုပ်ရပါမယ်။ ဒီမက်သဒ်ကိုတော့ ကလစ်နှိပ်တော့မှပဲ ခေါ်မှာပါ။ မောက်စ် ဘယ်ဘက် ကလစ်နှိပ်လိုက်တဲ့နေရာကို ဘောလုံး (ချက်ချင်း)ရောက်စေချင်ရင် အခုလို ...

```
# 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
```

ကီးဘုဒ်နဲ့ ထိန်းချုပ်ရင် on_key_press ကို override လုပ်ရပါမယ်။ arcade_k_press.py မှာ လေ့လာကြည့်ပါ။ မောက်စ် နှိပ်ထားပြီး ဘောလုံးကိုရွှေ့ (dragging) တာကို arcade_m_drag.py ဖိုင်မှာ ကြည့်နိုင်ပါတယ်။

arcade.View

arcade.View ဟာ Window နဲ့ သဘောတရား အတော်လေးဆင်တူပါတယ်။ Window လိုပဲ စခရင်မှာ ဂရပ်ဖစ်ပုံဖော်ဖို့ View ကို သုံးလို့ရတယ်။ Event handling အတွက် override လုပ်ရတာတွေ ကလည်း Window နဲ့ တူတူပါပဲ။ ဒါပေမဲ့ View က သူ့ချည်း မရပ်တည်နိုင်ပါဘူး။ View ကို ပြပေးဖို့ အတွက် Window တစ်ခုလိုပါတယ်။ Window တစ်ခုတည်းဟာ View အမျိုးမျိုးကို အလှည့်ကျ လဲပြီးပြ လို့ရတယ်။ ဂိမ်းတစ်ခုမှာ welcome screen, game over screen, pause screen စသည်ဖြင့် ပြပေး ဖို့လိုပါတယ်။ ဒီလို လိုအပ်ချက်မျိုးအတွက်ဆိုရင် Arcade မှာ View ကို အသုံးပြုရမှာပါ။ နမူနာကြည့်

ရင် ပို့ရှင်းသွားပါလိမ့်မယ်။

အောက်ပါဥပမာက ပထမ စစချင်း Click to Start! လို့ ပြနေမှာပါ။ ကလစ်နှိပ်လိုက်ရင် ဘောလုံးက စရွေ့ပါမယ်။ ကလစ်ထပ်နှိပ်လိုက်ရင် ဘောလုံးရွေ့နေတာ ပျောက်သွားပြီး The End စာသားပြပါတယ်။ `MsgView` က စာသားပြပေးဖို့အတွက်။ `MainView` က ဘောလုံး အန်နီမေးရှင်းအတွက်။ နှစ်ခုလုံး `arcade.Window` အစား `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()
```

စောစောကပြောခဲ့သလို View ကို ပြဖို့ Window ရှိရပါမယ်။ Arcade ပရိုဂရမ်တစ်ခု run တဲ့အခါ Window တစ်ခုကတော့ ရှိကိုရှိရမှာပါ။ အဲဒီ Window ကို View က self.window နဲ့ ရည်ညွှန်းအသုံးပြုနိုင်တယ် (View ကို ဆက်ခံထားတဲ့ subclass မှာလည်း သုံးလို့ရတယ်)။ Window မှာ ပြချင်တဲ့ View ကို window.set_view မက်သဒ်နဲ့ သတ်မှတ်ရတယ်။ MsgView နဲ့ MainView မှာ on_mouse_press ကို ဒီလို override လုပ်ထားတယ်

```
def on_mouse_press(self, x, y, button, modifiers):
    if self._next_view:
        self.window.show_view(self._next_view)
```

ကလစ်နှိပ်ရင် self._next_view ကို ပြောင်းပေးမှာပါ။

နောက်တစ်ခုက View အကျယ်နဲ့ အမြင့်ဟာ ၎င်းကိုပြပေးတဲ့ Window အပေါ် မူတည်တယ်။ ဒါကြောင့် View တစ်ခုချင်းအတွက် အကျယ်၊ အမြင့် မသတ်မှတ်ဘူး။ Window အကျယ်နဲ့ အမြင့် သတ်မှတ်ရင် ရပြီ။

main မက်သဒ်မှာ Window တစ်ခု နဲ့ View သုံးခု ဖန်တီးထားတယ်။ View တစ်ခုကနေ တစ်ခုပြောင်းလဲဖို့ အခုလို ချိတ်ဆက်ပေးထားတာ

```
start_view._next_view = main_view
main_view._next_view = end_view
```

တွေ့ရမှာပါ။ start_view ပေါ်မှာ ကလစ်နှိပ်ရင် main_view, main_view ပေါ်မှာ နှိပ်ရင် end_view ကို ပြပေးမှာပါ (on_key_press မက်သဒ်နဲ့ ဆက်စပ်ကြည့်ပြီး နားလည်အောင်လုပ်ပါ)။

၃.၇ Breakout တည်ဆောက်ခြင်း

ဂိမ်း မတည်ဆောက်ခင် ကြိုတင်နားလည်ထားရမဲ့ Arcade သဘောတရား အတော်များများ ရှင်းပြခဲ့ပြီး ပြီ။ တကယ် လက်တွေ့ ဂိမ်းရေးဖို့ပဲ ကျန်ပါတော့တယ်။ ပရိုဂရမ် အပြည့်အစုံကို စာမျက်နှာ (၆၃) မှာ ကြည့်ပါ။ အခု တစ်ပိုင်းချင်း ခွဲထုတ် ရှင်းပြပါမယ်။ ဂိမ်းအတွက် လိုအပ်တဲ့ constant တွေကို ပထမဆုံး သတ်မှတ်ထားတယ်။ အများစုက 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 နံရံအောက်ခြေ အကွာအဝေး
BRICK_Y_OFFSET = 414
# Window ဗဟိုမှတ်
CENTER_X = WIN_WIDTH // 2
CENTEr_Y = WIN_HEIGHT // 2
```

အုတ်ခဲစီခြင်း

setup_bricks က အုတ်ခဲတွေ နေရာတကျ စီပေးတဲ့ ဖန်ရှင်။ အုတ်ခဲကို SpriteSolidColor နဲ့ ဆွဲ လို့ရတယ်။ ဒီကလပ်စ်က အရောင်အပြည့် ထောင့်မှန်စတုဂံပုံ Sprite ပါပဲ။ 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 × 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
            # အုတ်ခဲတစ်ခဲချင်း SpriteList ထဲထည့်
            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 ခဏနေ တွေ့ရမှာပါ။ [[တည်နေရာ အတွက်အချက် အသေးစိတ်နားလည်ချင်ရင် စာမျက်နှာ (၂၈) မှ ကျားကွက်ခုံ ဥပမာကို ကြည့်ပါ]]။

ဘောလုံးနှင့် တာထွက် velocity

ဘောလုံးအတွက် Ball ကလပ်စ်ကို အခုလို သတ်မှတ်ပါမယ်။ ဘောင်တွေကို တိုက်တဲ့အခါ ပြန်ကန်ထွက်အောင် လုပ်တဲ့နည်းလမ်းက ရှေ့မှာတွေ့ခဲ့တဲ့ ဥပမာကလိုပါပဲ။ `self.left < 0` ဖြစ်ရင် ဘယ်ဘက်ဘောင်နဲ့ တိုက်တာ၊ `self.right < WIN_WIDTH` ဆိုရင် 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 စက်ဝိုင်းပုံ Sprite အတွက်။ image ဖိုင် မလိုဘူး။

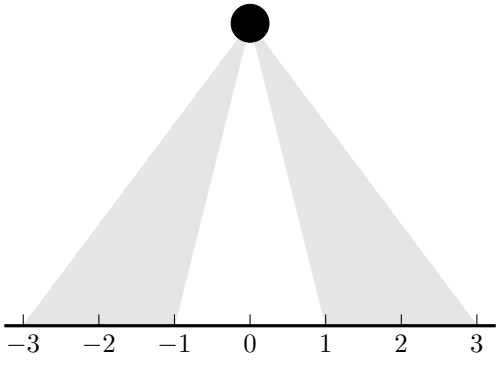
`setup_ball` ကတော့ ဘောလုံးရဲ့ ကနဦးတည်နေရာနဲ့ တာထွက် velocity ကို အဓိက တွက်ချက်သတ်မှတ်ပေးတာပါ။ ဖန်တီးထားတဲ့ ဘောလုံးကိုလည်း return ပြန်ပေးတယ်။

```
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
```

အစမှာ ဘောလုံးအောက်ကို ကျလာတဲ့အခါ ဦးတည်ရာက အောက်တည့်တည့်ကိုပဲ အမြဲတစ်သမှတ်တည်း ဖြစ်နေရင် သိပ်မကောင်းဘူး။ ငြီးငွေ့စရာဖြစ်နေမယ်။ ဘယ်ဘက်ကို ဦးတည် ဆင်းလာမှာလဲ ခန့်မှန်းလို့မရရင် ပိုမိုက်တယ်။ စိတ်လှုပ်ရှားဖို့ ပိုကောင်းတယ်။ ဒီအတွက် `change_x` ကို 1.0 ကနေ 3.0 အတွင်း နဲ့ -1.0 ကနေ -3.0 အတွင်း random ထုတ်ထားတယ်။ `random.uniform` ဖန်ရှင် သုံးပါတယ်။ `change_x` အနှုတ်တန်ဖိုးဆိုရင် ဘယ်ဘက်၊ အပေါင်းဆိုရင် ညာဘက်ကို ဦးတည်တယ်။ ဘယ်ဘက်လား၊ 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 အနှုတ်တန်ဖိုး ထွက်အောင်လုပ်ထားတယ်။ ဒါကြောင့် စစချင်းမှာ ဘောလုံးကျလာတဲ့ လားရာက ပုံ (၃.၃) မှာ ပြထားတဲ့ နယ်ပယ်အတွင်းမှာပဲ ရှိပါမယ်။



ပုံ ၃.၃ ဘောလုံးကျလာနိုင်တဲ့နေရာ

Breakout Class

`Breakout` ကလပ်စ်ကို ဆက်ကြည့်ရအောင်။ ဂိမ်းရဲ့ အဓိက `View` ဖြစ်ပြီး အရုံးအနိုင်ဆုံးဖြတ်တာ၊ ဘောလုံးနဲ့ အုတ်ခဲ၊ ဘောလုံးနဲ့ paddle တိုက်မိတာ စတာတွေကို ဒီကလပ်စ်မှာ ရေးထားတာပါ။ paddle ကို မောက်စ်နဲ့ ရွှေ့လို့ရအောင် `on_mouse_motion` ကလည်း ဒီထဲမှာပဲ။ [[Breakout ကလပ်စ် အပြည့်အစုံကို စာမျက်နှာ (၆၄) မှာ ကြည့်ပါ]]။

`__init__` မက်သဒ်က သိပ်ရှုပ်ရှုပ်ထွေးထွေး မရှိဘူး။ မောက်စ်ပွိုင်တာ 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 အော့ဘ်ဂျက် ဖန်တီးတဲ့အခါမှာပဲ အလိုအလျောက်ခေါ်ပေးတာ။ တိုက်ရိုက်ခေါ်ရတဲ့ မက်သဒ်မဟုတ်ဘူး။ ရှုံးသွားလို့ (သို့) နိုင်သွားလို့ နောက်တစ်ခါ ပြန်ဆော့ချင်ရင် အစအနေအထား ဖြစ်အောင် `setup` မက်သဒ်ကို ခေါ်လို့ရမယ်။ (ဒီတော့ Breakout အော့ဘ်ဂျက် အသစ်တစ်ခု ဖန်တီးတာ၊ `__init__` ကို တိုက်ရိုက်လည်း မခေါ်ဘဲ ပြန်စချင်ရင် `setup` ကို ခေါ်လို့ရမယ်)။ ရှေ့မှာတွေ့ခဲ့တဲ့ `setup_ball`, `setup_paddle`, `setup_bricks` တို့ကို တစ်ဆင့် ပြန်ခေါ်ထားတာ တွေ့ရမယ်။ `_brick_remains` က လက်ကျန်အုတ်ခဲ အရေအတွက် မှတ်ထားဖို့။ အုတ်ခဲ တစ်လုံး ပျက်သွားတိုင်း 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` မက်သဒ်က ဂိမ်းမှာပါတဲ့ အုတ်နံရံ၊ ဘောလုံးနဲ့ paddle ပြားတို့ကို ဆွဲတယ်။ ဒီမက်သဒ်က Breakout ကလပ်စ်မှာ ထပ်ဖြည့်ထားတာ။ View ကလပ်စ်ရဲ့ တစ်စက္ကန့် အကြိမ်ခြောက်ဆယ် ခေါ်နေမဲ့ `on_draw` ကို override လုပ်တာ မဟုတ်ဘူး။ `draw` ကို ခွဲထုတ်ထားရတာက လိုတဲ့အချိန်မှာ ကိုယ်တိုင် ခေါ်လို့ရအောင် ရည်ရွယ်တာပါ။

```
def on_draw(self):
    self.draw()

def draw(self):
    self.clear()
    self._paddle.draw()
```

```
    self._brick_lst.draw()
    self._ball.draw()
```

ဂိမ်းရဲ့ အဓိကလော့ဂျစ်ကို on_update မက်သဒ်မှာ တွေ့ရမှာပါ။ View ကလပ်စ်ရဲ့ 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 တစ်စက္ကန့် အကြိမ်ခြောက်ဆယ် အဆက်မပြတ် အလိုအလျောက် ခေါ်ပေးတယ်လို့ ပြောခဲ့တာ ပြန်အမှတ်ရမယ် ထင်ပါတယ်။ ဂိမ်းရဲ့ state ကို ဒီမက်သဒ် တစ်ကြိမ်ခေါ်တိုင်း update လုပ်ပေးရပါမယ်။ ဒါဟာ ဂိမ်းတစ်ခုရဲ့ အချိန်နဲ့အမျှ ပြောင်းလဲနေတဲ့ အရာအားလုံးအတွက် အဓိကသော့ချက်ပဲ။ ပထမဆုံး Breakout မှာပါတဲ့ ဘောလုံး၊ paddle နဲ့ အုတ်ခဲအားလုံးရဲ့ လက်ရှိအခြေအနေကို update လုပ်ရမယ်။ ဒီအတွက် ဂိမ်းမှာပါဝင်တဲ့ သက်ဆိုင်ရာ Sprite (သို့) SpriteList အားလုံးရဲ့ update မက်သဒ်ကို ခေါ်ပေးရမှာပါ။

```
self._ball.update()
self._paddle.update()
self._brick_lst.update()
```

check_for_collision_with_list က ဝင်တိုက်မိတဲ့ Sprite တွေကို စစ်ထုတ်ပေးတဲ့ မက်သဒ်။ ဘောလုံးနဲ့ တိုက်မိတဲ့ အုတ်ခဲတွေကို လိုချင်တာ။ ဒီတော့ အခုလို ခေါ်ရမယ်

```
bricks_hit = arcade.check_for_collision_with_list(self._ball,
                                                  self._brick_lst)
```

အုတ်ခဲ (တွေ) နဲ့ ဝင်တိုက်ရင် ဘောလုံးကို အထက် (သို့) အောက် ပြန်ကန်ထွက်အောင် change_y ကို လက္ခဏာ ဆန့်ကျင်ဘက် ပြောင်းပေးပါတယ်။

```
if len(bricks_hit) > 0:
    self._ball.change_y *= -1
```

ဝင်တိုက်တဲ့ အုတ်ခဲတွေကို ဖျက်ပစ်ရပါမယ်။ ဖျက်လိုက်ရင် လက်ကျန်အုတ်ခဲလည်း လျော့သွားရမယ်။ ဒီအတွက် အခုလို

```
for brick in bricks_hit:
    brick.remove_from_sprite_lists()
    self._brick_remains -= 1
```

တိုက်မိတဲ့ အုတ်ခဲတစ်ခဲချင်း ဖယ်ထုတ်ပါတယ်။

Start, Main and End **Views**

ပထမ စစချင်းမှာ ပုံ (၃.၁) မှာ တွေ့ရတဲ့ အနေအထားအတိုင်း ရှိနေမယ်။ ဒါပေမဲ့ စတာနဲ့ ဘောလုံးက ချက်ချင်းထွက်ရင် ဆော့ရတာ သိပ်အဆင်မပြေဘူး။ ကလစ်နှိပ်လိုက်မှ ဘောလုံးစထွက်မယ်ဆိုရင် ပိုကောင်းမယ်။ 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):
        # ကလစ်နှိပ်ရင် Breakout View ကို ပြောင်းပေးတာ
        if self._next_view:
            self._next_view.setup()
            self.window.show_view(self._next_view)
```

စောစောကရှင်းပြခဲ့တဲ့ Breakout ဟာ ဂိမ်းရဲ့ အဓိက View ။ ဒါပေမဲ့ ဒီ View က စတာနဲ့ ဘောလုံးက ရွေ့မှာ on_draw နဲ့ on_update က ရပ်ထားလို့မရဘူး။ View ကို Window မှာ ပြပြီဆိုတာနဲ့ တောက်လျှောက် ခေါ်နေမှာ။ StartView က ရုပ်ပုံကို အငြိမ်ပဲ ပြပေးရမယ်။ ဖြစ်နိုင်တဲ့ နည်းလမ်းတစ်ခုက Breakout ရဲ့ setup နဲ့ draw ကို StartView ရဲ့ on_draw ကနေ ခေါ်လို့ရပါတယ်။ StartView နဲ့ Breakout ကို main မက်သဒ်ထဲမှာ အခုလို ချိတ်ပေးထားပါမယ်။

```
start_view = StartView()
game_view = Breakout()
# ...
start_view._next_view = game_view
```

```
window.show_view(start_view)
# ...
```

အနှုံးအနိုင် You Win!/You Lose! ပြဖို့ EndView ရဲ့ တာဝန်။ နှုံး (သို့) နိုင်ရင် မက်ဆေ့ချ် ပြပေးပြီး တစ်ခါထပ်ဆော့ချင်ရင် ကလစ်နှိပ်ရပါမယ်။

```
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 မက်သဒ်ထဲမှာ အခုလို ချိတ်ပေးထားပါတယ်

```
end_view = EndView()
# ...
end_view._next_view = start_view
```

နိဂုံး

ဒီအခန်းမှာ Breakout ဂိမ်းကို အသုံးချ ဥပမာအနေနဲ့ ထည့်ပေးထားတဲ့ ရည်ရွယ်ချက်က Arcade လိုက်ဘရီနဲ့ ဂိမ်းတွေ ဖန်တီးနိုင်ဖို့ အဓိက မဟုတ်ပါဘူး။ Inheritance ကို လိုက်ဘရီတွေမှာ အသုံးချလေ့ရှိတဲ့ ပုံစံတချို့ကို နားလည်သဘောပေါက်အောင်၊ သတိပြုမိအောင်၊ ဆက်စပ်မိအောင် အဓိက ရည်ရွယ်တာပါ။ စလေ့လာသူတွေအတွက် လွယ်လွယ်နဲ့ နားလည်နိုင်မယ်လို့တော့ မမျှော်လင့်နိုင်ဘူး။ စိတ်ရှည်ရှည်ထားအချိန်ပေးပြီး နားလည်သဘောပေါက်အောင် လေ့လာဖို့ လိုပါလိမ့်မယ်။ ပရိုဂရမ်အပြည့်အစုံကို အောက်မှာ ဖော်ပြပေးထားပါတယ်။

တည်နေရာ အတွက်အချက်တွေ နားမလည်လို့လည်း သန်ျာကြောက်သူတွေ စိတ်ဓါတ်ကျစရာ မလိုပါဘူး။ အတွက်အချက်တွေ အကြမ်းဖျဉ်းလောက် နားလည်အောင် ကြည့်၊ ကျန်တဲ့ ပရိုဂရမ်မင်းနဲ့ဆိုင်တဲ့ သဘောတရားတွေ အဓိကထား ကြည့်မယ်ဆိုရင်လည်း 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):
            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():
    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()
```