

Begin Modern Programming

with

C
O
D
I
N
G
A

Pyi Soe

အခန်း ၁

စက်ရုပ်ကားရဲလိုဖြင့် ပရီဂရမ်းမင်းမိတ်ဆက်

ကွန်ပျိုတာတွေဟာ သက်မဲ့ စက်ပစ္စည်းတွေပါပဲ။ ကားတို့ လေယာဉ်တို့နဲ့ မတူတာက ကွန်ပျိုတာတွေဟာ စက်ချဉ်းသက်သက် ဘာအစွမ်းမှ မယ်မယ်ရရ မရှိဘူး။ ဒါပေမဲ့ ဆောင်ရွက်လိုတဲ့ ကိစ္စအဝေဝအတွက် ပ ရိုကရမ်အမျိုးမျိုး ထည့်ပေးလိုက်တဲ့ အခါမာ သူ့အစွမ်းက အတိုင်းအဆမဲ့ပဲ။ နေရာမျိုးစံ၊ နယ်ပါယ်မျိုးစံ မှာ အကူးအညီပေးနိုင်တဲ့ စွယ်စုံသုံး ပစ္စည်းတစ်ခုဖြစ်သွားတယ်။ ဂိတ်သံစဉ်တွေကို ဖွင့်ပေးနိုင်သလို အသံ လည်းသွင်းပေးနိုင်တယ်။ ရုပ်ရှင်တည်းဖြတ် လုပ်ချင်တာလား။ ပြဿနာမရှိဘူး၊ ကူညီပေးနိုင်တယ်။ နျှော လီးယား ပါတ်ပေါင်းဖို့တွေကို စီမံခိုင်သလို မောင်းသူမဲ့ အုံပုံတွေကိုလည်း ပဲထိန်းပေးနိုင်တယ်။

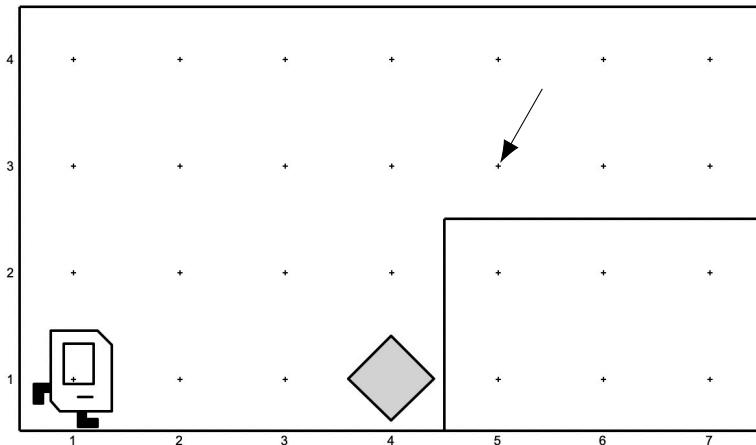
ကျွန်တော်တို့တွေ နိစ္စဓာဝ် အသုံးပြုနေကြတဲ့ ကား၊ စမတ်နှင့်း၊ လက်ပါတ်နာရီ၊ မိုက်ခရှိဝေဖွံ့ဖို့ အဝတ်လျှော့စက် စတဲ့ စက်ပစ္စည်း အမျိုးမျိုးဟာလည်း ကွန်ပျိုတာတွေနဲ့ မကင်းပြန်ပါဘူး။ “ကွန်ပျိုတာ နည်းပညာ အကူးအညီမပါတဲ့ အတိုင်းဆက်သံစုံတွေမှာ မရှိဘူး” လို့ ဆိုနိုင်ပါတယ်။

တစ်ချက်တစ်ချက် ရိုက်ခတ်လိုက်တဲ့ ကွန်ပျိုတာနည်းပညာ လိုင်းလုံးကြီးတွေဟာ ကမ္မာတစ်စုံမှုးလုံး ပုံစံပြောင်းသွားလောက်အောင် အဟုန်ပြင်းထန်လာတယ်။ ဘီလီယံချွဲ့တဲ့ လူတွေ ဆီရှယ်မီဒီယာတွေပေါ်က နေ ရုပ်သံတွေနဲ့ ချိတ်ဆက်ပြောဆိုသွေးတွေ ရစေတာဟာလည်း ကွန်ပျိုတာစနစ်တွေပါပဲ။ Artificial Intelligence (AI) နည်းပညာကြောင့် သက်ရှိတွေမှာပဲတွေ့ရတဲ့ ညာတ်ရည်မျိုးကို ကွန်ပျိုတာတွေ မှာလည်း တွေ့လာရပါပြီ။ သံချွာပွဲတွေ ဖြေရှင်းခြင်း၊ စစ်တုံးရှင်းထိုးခြင်း၊ စတဲ့ကိစ္စမျိုးတွေအပြင် ပန်းချီ ဆွဲခြင်း၊ ကဗျာရေးစပ်ခြင်း၊ သီချင်းရေးဖွဲ့ခြင်း ကဲ့သို့ အနုပညာဖန်တီးမှုတွေကိုပါ AI က လုပ်ဆောင်ပေး နိုင်ပါတယ်။ နှစ်ဆယ်တစ်ရာစုံ အထူးမြှားဆုံး AI နည်းပညာလိုင်းဟာ အရှိန်အဟုန်ပြင်းပြင်း ရိုတ်ခတ် ဖို့ အားယူစ ပြုနေပါပြီ။

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

၁.၁ စက်ရုပ် ကားရဲလ်

ပရီဂရမ်းမင်းဆိုတာ ဘယ်လိုမျိုးလဲ သဘောပေါက်အောင် စာတွေတစ်သီဥပါးရေး ရှင်းပြတာထက် ပရီဂရမ် လေးတွေ လက်တွေ့ ရေးကြည့်လိုက်တာ ပိုပြီးထိရောက်ပါတယ်။ ဒါကြောင့် စက်ရုပ်ကားရဲလ်ကို ပရီဂရမ် လေးတွေရေးပြီး အလုပ်တွေခုင်းကြည့်ကြမယ်။ ပဲ (??) မှာ တွေ့ရတာက ကားရဲလ် ရောက်ရှိနေတဲ့ နဗုံနာ ကမ္မာတစ်ခုပါ။ မီးခါးရောင် မှုန်ကူကုံးပုံလေးကို ဘိပါ (beeper) လို့ ခေါ်တယ်။ အဲဒီဘိပါကို မြားပြထားတဲ့ နေရာကို ရွှေခိုင်းချင်တယ်။ မျှေားမည်းအထူးတွေက နံရံတွေပါ။ ကားရဲလ်ကို ကိစ္စတစ်ခု ဆောင်ရွက်စေ



ပုံ ၁.၁ ခက်ရုပ်လေး ကားရဲ့

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

ကားရဲ့လိုက်မန်းများ

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

move ကွန်မန်းက ကားရဲ့လိုက် ရှေ့တစ်ကွန်နာကို ရွှေ့ခိုင်းတာ။ ကားရဲ့လိုက်မှာ တစ်ခုနဲ့တစ်ခု အကွာအဝေးတူ ခြားထားတဲ့ အတန်းလိုက် အတန်းလိုက် အစက်ကလေးတွေ့ဟာ ကွန်နာ (corner) တွေ ဖြစ်တယ်။ ကဲ့သို့ မျဉ်းမည်းအထူ နံရုံတွေ့နဲ့ ထောင့်မှုန်းစတုဂံပဲ ပါတ်လည် ဘောင်ခတ်ထားတယ်။ ကွန်နာတွေ့ကြားမှာလည်း နံရုံတွေ့ရှိနိုင်တယ်။ နုမ်နာကဲ့မှာ ဘေးတိုက် နံပါတ်စဉ် ၄ နဲ့ ၅ ကြား ထောင်လိုက် နံရုံတစ်ခု၊ အထက်အောက် နံပါတ်စဉ် ၂ နဲ့ ၃ ကြား အလျေားလိုက် နံရုံတစ်ခုကို တွေ့ရှုပါမယ်။ ကွန်နာရှုံးမှာ နံရုံကာနေရင် ကားရဲ့လိုက် ပေါ်ပေါ်ခဲ့ပါဘူး။

put_beeper က ကားရဲ့လိုက် လက်ရှိ ရှိနေတဲ့ ကွန်နာမှာ ‘ဘိပါတစ်ခုချု’ ထားခိုင်းတာ၊ pick_beeper က ပုံပေါ်နေတဲ့ ကွန်နာမှာ ‘ဘိပါတစ်ခုကောက်’ ခိုင်းတာပါ။ ကွန်နာမှာ ဘိပါရှိနေမှု ကောက်ခိုင်းလို့ရမှုပါ။ မရှိရင် ကောက်ခိုင်းလို့ မရှုံးမှုပါ။ ဘိပါချိန်းရင်လည်း ကားရဲ့လိုက် ဘိပါရှိမှ ချိန်းလို့ရတယ်။ ကားရဲ့လိုက် ဘိပါတွေ လို့သလောက် ဖြည့်ပေးထားတယ်လို့ ယူဆပါ။ turn_left က ‘ဘယ်လှည့်’ ခိုင်းတာ။

ဘိပါကို ဘယ်လိုရွှေ့ခိုင်းမလဲ

ပုံ (??) အနေအထားကနေ ရှေ့ကို သုံးနေရာရွှေ့ ဘိပါကောက်၊ ဘယ်ဘက်လှည့်၊ အပေါ် နှစ်နေရာရွှေ့ ညာဘက်လှည့်၊ ရှေ့တစ်နေရာထပ်ရွှေ့ပြီး ဘိပါချုထားခိုင်းလိုက်ရင် အလုပ်ပြီးသွားပါပြီ။

ကားရဲ့လိုက် ညာဘက်လှည့်ခိုင်းဖို့ turn_right ကွန်မန်း မရှိဘူး။ ဒါပေမဲ့ ဘယ်သုံးခါလှည့်ခိုင်းလို့ရတယ်။ ဒါကြောင့် ညာဘက်ချင်တဲ့အခါ ဘယ်သုံးခါလှည့်ခိုင်းလို့ရတယ်။

၁.J Meet Karel ပရိုဂရမ်

ပရိုဂရမ် ရေးတယ်ဆိုတာ ကွန်ပူးတာကို ကိစ္စတစ်ခုခဲ့ အောက်ရှုက်ပေးဖို့ ခိုင်းစေတဲ့ ညွှန်ကြားချက်တွေ ရေးတာပါပဲ။ ဒီလို ညွှန်ကြားချက်တွေကို ပရိုဂရမ်ကုဒ် (program code) လို့ ခေါ်တယ်။ ပရိုဂရမ်ကုဒ် တွေကို ကွန်ပူးတာနားလည်တဲ့ programming language တစ်ခုခဲနဲ့ ရေးရတယ်။ ဒီစာအပ်မှာ အသံးပြုမဲ့ programming language ကတော့ Python ပါ။ Programming language တစ်မျိုးပဲ ရှိတာ မဟုတ်ပါဘူး။ ရာနဲ့ချိပြီး ရှိတာပါ။ လူဘာသာစကားတွေ အမျိုးမျိုးရှိသလိုပဲပေါ့။ Python ဟာ ဒီလို ရာနဲ့ချိတဲ့ထဲက လက်ရှိအသံးအများဆုံး ထိပ်ဆုံးဆယ်ခု ထဲမှာ ပါဝင်တယ်။ Python နဲ့ ဘိပါရွှေခြင်းတဲ့ 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 အတိုချိုးပဲ ပြောတာများတယ်။ Python ‘စာ/စကား’ မတတ်ရင် ဒီ ‘Python ကုဒ်’ တွေကိုလည်း နားလည်မှာ မဟုတ်ဘူး။ ဒီတော့ Python ‘စာ/စကား’ အခြေခံက စပြီး လေ့လာစွဲလိုပါမယ်။

ကွန်းမာန် (Comment)

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

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

import စတိတ်မန်

```
from stanfordkarel import *
```

ကတေသာ အင်ပိုစတိတ်မန် ဖြစ်ပါတယ်။ “stanfordkarel လိုက်ဘရီမှ အာလုံးကို ထည့်သွင်းပေးပါ” လို တောင်းဆိုတဲ့ အဓိပ္ပာယ်။ * သက်တကို ‘အားလုံး’ လို ယူဆပါ။ stanfordkarel လိုက်ဘရီမှာ ကား ရဲလ်ပရီဂရမ်အတွက် လိုအပ်တာအားလုံး ပါဝင်တယ်။ ဒီလိုက်ဘရီကို အင်ပိုလုပ်ထားမှ ကားရဲလ်ကွန်းမန်း တွေ သုံးလို့ရမှုပါ။ သီးခြား ကားရဲလ်ပရီဂရမ် တစ်ခုစီတိုင်းအတွက် အင်ပိုလုပ်ရမှာ ဖြစ်တယ်။

လိုက်ဘရီ

လိုက်ဘရီ (*library*) ဆိုတာ ပညာရပ်နယ်ပယ် တစ်ခုအတွက် ရည်ရွယ်ရေးထားတဲ့ ပရီဂရမ်ကုဒ်တွေပါပဲ။ သချုပ်အတွက်အချက် လိုက်ဘရီ ဂိမ်းရေးဖို့ လိုက်ဘရီ 2D/3D ဂရပ်ဖစ်ဆဲဖို့ လိုက်ဘရီ အေအးစိုင်အတွက် လိုက်ဘရီ စသည်ဖြင့် နယ်ပယ်အသီးသီး ကိစ္စရပ်အဖို့ဖို့အတွက် သက်ဆိုင်ရာ ကျမ်းကျင်ပညာရှင်တွေ ထုတ်လုပ်ဖြန့်ချီပေးထားတဲ့ လိုက်ဘရီတွေရှိတယ်။ Matrix အော်ပရေးရှင်းတွေအတွက် numpy ၊ ဂရပ် ဖွဲ့မယ်ဆိုရင် matplotlib စတဲ့ လိုက်ဘရီတွေကို အင်ပိုလုပ် အသုံးပြနိုင်ပါတယ်။ မေ့ထရစ် A ကို B နဲ့ မြောက်ရင် ဒီလိုပါ

```
from numpy import *

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

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

result = matmul(A, B)

print(result)
```

ရလဒ် အခုလိုထွက်ပါမယ်။

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

ဒါကတေသာ ဘားချုပ် အတွက် numpy နဲ့ matplotlib လိုက်ဘရီ သုံးထားတာပါ။

```

from matplotlib.pyplot import *
from numpy import *

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

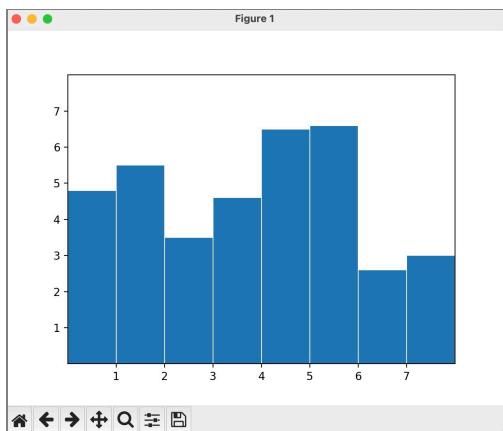
# plot
fig, ax = subplots()
bar(x, y, width=1, edgecolor="white", linewidth=0.7)

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

show()

```

ဘားချုတ်ကို ဒီလို ထုတ်ပေးပါတယ်။



ပုံ ၁၂

လိုက်ဘရီတွေဟာ ပရိုဂရမ်တွေ တည်ဆောက်ရာမှာ အင်မတန်မှ အရေးပါတယ်။ ဖော်ပြထားတဲ့ မူသရစ်နဲ့ ဘားချုတ် ကုဒ်တွေကို (အခုတော့) နားလည်မှာ မဟုတ်သေးဘူးပေါ့။ ဒါပေမဲ့ သက်ဆိုင်ရာ လိုက်ဘရီတွေနဲ့ ဒီလိုကိစ္စတွေကို သိပ်မခက်ခဲဘဲ လုပ်လို့ရနိုင်တယ်ဆိုတာ မြင်မယ် ထင်ပါတယ်။ လိုက်ဘရီတွေ သာမရှိရင် ပရိုဂရမ်တွေကို အခုထက် အဆပေါင်းများစွာ အချိန်ပေးပြီး ရှုပ်ရှုပ်တွေးတွေး ခက်ခက်ခဲ့တယ်ဆောက်ကြရမှာပါ။

တုံကင်၊ ဓာတ်မန်နှင့် ဆင်းတာကို

ဂျိန်ပန်စာ၊ ပြင်သစ်စာ စတဲ့ လူဗာသာစကားတွေဟာ စကားလုံးတွေ ဝါကျတွေနဲ့ ဖွဲ့စည်းထားသလို ပရိုဂရမ်ကုဒ်တွေဟာလည်း စကားလုံးတွေ၊ ဝါကျတွေနဲ့ ဖွဲ့စည်းထားတာပါပဲ။ Programming language တွေမှာ စကားလုံးတွေကို တုံကင် (token) လို့ခေါ်ပြီး ဝါကျတွေကိုတော့ စတိတ်မန် (statement) လို့ ခေါ်ပါတယ်။ ဝါကျတွေကို စကားလုံးတွေနဲ့ ဖွဲ့စည်းထားသလို စတိတ်မန်တွေကတော့ တုံကင်တွေနဲ့

ဖွဲ့စည်းထားတာပါ။ စတိတ်မန် ပုံစံတစ်မျိုးကို တွေ့ခဲ့ပြီးပါပြီ။ အဲဒါကတော့ ရှေ့စာမျက်နှာက အင်ပိုစတိတ်မန်ပဲ ဖြစ်ပါတယ်။

လူဘာသာစကားတွေမှာ ဖွဲ့စည်းတည်ဆောက်ပုံ စထရက်ချာရှိသလို programming language တွေမှာလည်း စထရက်ချာရှိဖို့ လိုအပ်တာပေါ့။ ဖွဲ့စည်းပုံ စထရက်ချာ မှန်/မမှန်ကို သွှေ့စည်းမျဉ်တွေနဲ့ ထိန်းကွပ်ထားတာပါ။ ပရိုဂရမ်ကုဒ် စထရက်ချာ မှန်/မမှန် ထိန်းကွပ်ပေးတဲ့ သွှေ့စည်းမျဉ်တွေကိုတော့ ဆင်းတက်စ် (syntax) လိုခေါ်တယ်။

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

Keywords

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

main ဖန်ရှင်

‘Meet Karel’ ပရိုဂရမ် အင်ပိုစတိတ်မန် အပြီးမှာ တွေ့ရတာကတော့ `main` ဖန်ရှင်သတ်မှတ်ချက်ပါ။ ကြည့်ရအဆင်ပြအောင် သူချည်းသီးသန်း 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 ဖွံ့ဖြည်းထားတဲ့ ပရီဂရမ်ကုဒ် အစုအဝေးတစ်ခုပါပဲ။ ဖန်ရှင်ကို အသုံးပြုတဲ့အခါ ငါးရဲ့ လုပ်ငန်းတာဝန်အတိုင်း ဖန်ရှင် က လုပ်ဆောင်ပေးမှာ ဖြစ်တယ်။ ဖန်ရှင်အသုံးပြုတာကို ‘ဖန်ရှင်ကောလ်’ (function call) လုပ်တာ လို့ ပြောတယ်။

main ဖန်ရှင်သတ်မှတ်ချက်ကို အပိုင်းနှစ်ပိုင်းခဲ့ ကြည့်နိုင်တယ်။ ပထမတစ်ပိုင်း

def main():

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

def myfun(x, y):

ပါရာမိတေမပါရှင်လည်း ပိုက်ကိုင်းအလွတ် တစ်စုံ ‘()’ တော့ပါရမယ်။ main ဖန်ရှင်မှာ ပါရာမိတေ မပါဘူး။ ပါရာမိတေတွေအကြောင်း နောက်ပိုင်းအခန်းတွေမှာ အသေးစိတ် လေ့လာရမှာပါ။ ကားခဲ့လ်ပရီဂရမ် တွေမှာ ပါရာမိတေအကြောင်း သိဖို့မလိုအသေးပါဘူး။ ပါရာမိတေ မလိုတဲ့ ဖန်ရှင်တွေပဲ တွေ့ရမှာပါ။

ဖန်ရှင်ခေါင်းစဉ်းအောက် ဒုတိယပိုင်းကတော့ main ဖန်ရှင် ကုဒ်ဘလောက် (code block) ပါ။ ကုဒ်ဘလောက် ဆိုတာ ကုဒ်တွေကို အပ်စုတစ်စုံ အဖြစ် ဖွံ့ဖြည်းထားတာကို ဆုံးလိုတာပါ။ ဘလောက်လို့ လည်း ခေါ်တယ်။ ဘလောက်တစ်ခုမှာ ပါဝင်တဲ့ ကုဒ်လိုင်းတွေကို ညာဘက် ခွာရေးရပါမယ်။ အင်ဒန်ထုတ် (indent) လုပ်တာလို့ ခေါ်တယ်။ တက်ဘ်ကိုးတစ်ချက် (သို့) စပေါ်လေးခုစာ ခွာလေ့ရှိတယ်။ ဘော်ဒီ ပထမတစ်ကြောင်း

||||“Karel code goes here!||||

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

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

turn_left()

pick_beeper()

စသည်ဖြင့် ဖြစ်တယ်။

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

ကားရဲလ်ပရိုဂရမ်တစ်ခုမှာ main ဟာ အထူးတာဝန်တစ်ခု လုပ်ဆောင်ပေးရတယ်။ အဲဒါကတော့ ပရိုဂရမ်ဝင်းသီးမှာ Run Program ခလုတ် (ပု ?? မှာကြည့်ပါ) နှိပ်လိုက်ရင် တဲ့ပြန် လုပ်ဆောင်ပေးရတာပါ။ ဒါကြောင့် ကွန်မန်းတွေဟာ အဲဒီခလုတ် နှိပ်တော့မှုပဲ စအလုပ်လုပ်တာ ဖြစ်တယ်။

Entry Point

'Meet Karel' ပရိုဂရမ်မှာ main ဖန်ရှင်နောက် အောက်ဆုံးအပိုင်းဟာ ပရိုဂရမ် run တဲ့အခါ ပထမဆုံး စတင်လုပ်ဆောင်ပေးရမဲ့ ဖန်ရှင်ကို ဖော်ပြပေးတာပါ။ အန်ထရီပိုင့် (entry point) လိုခေါ်တယ်။

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

run_karel_program ဖန်ရှင်ဟာ ကားရဲပရိုဂရမ် တစ်ခုအတွက် အန်ထရီပိုင့် ဖြစ်တယ်။ ပရိုဂရမ် တက်လာတဲ့ တစ်ပါတည်း ခေါ်တင်ချင်တဲ့ ကဗ္ဗာကို ဒီဖန်ရှင်မှာ ထည့်ပေးတယ်။ meet_karel.w ကဗ္ဗာကို စစ်ချင်းခေါ်တင်ထားချင်ရင် "meet_karel" ထည့်ပေးရမယ်။ ကဗ္ဗာဖိုင်မရှိရင် အယ်ရာတက်ပြီး ပရိုဂရမ်ပွင့်လာမှာ မဟုတ်ဘူး။ ကဗ္ဗာမထည့်ပေးထားဘဲ ဒီလို

```
run_karel_program()
```

ဆိုရင် 8×8 အရွယ် default ကဗ္ဗာကို တင်ပေးပါတယ်။

ကားရဲလ်ကဗ္ဗာတစ်ခုကို လိုချင်တဲ့ပဲ့စဲ ဒီဇိုင်းဆဲပြီး ဖိုင်နဲ့ သိမ်းထားရတာပါ။ ကဗ္ဗာ ပုံစံချကဲ့ ပရိုဂရမ်လည်း ရှိတယ်။ ကဗ္ဗာဖိုင်တွေက .w ဖိုင် အိပ်စာန်းရှင်းနဲ့ ဖြစ်တယ်။ ဒီစာအုပ်မှာပါတဲ့ ဥပမာတွေ လေ့ကျင့်ခန်းတော့ အားလုံးအတွက် လိုအပ်တဲ့ ကဗ္ဗာတွေကို အဆင့်သင့်ပေးထားမှာပါ။ ကိုယ့်ဟာကို လုပ်ဖို့ မလိုဘူး။ စိတ်ဝင်စားရင် စမ်းကြည့်လို့ရအောင် စာမျက်နှာ (??) နောက်ဆက်တဲ့ (??) မှာ အကျဉ်းဖော်ပြပေးထားပါတယ်။

၁.၃ ကားရဲလ် ပရိုဂရမ် run ခြင်း

လိုအပ်တဲ့ဆော်ဖဲတွေ ထည့်သွင်းနည်းကို စာမျက်နှာ (??) နောက်ဆက်တဲ့ (??) မှာ တစ်ဆင့်ချင်း ဖော်ပြပေးထားပါတယ်။ အခုက Python ပရိုဂရမ်တစ်ခုကို အရှိုးရှင်းဆုံး (လုပ်တယ်လို့ မဆိုလို) run လိုရတဲ့ နည်းကိုဖော်ပြပေးမှာပါ။ သဘောတရားပိုင်း နားလည်ဖို့ အထောက်အပံ့ဖြစ်မယ်။ အခုနည်းလမ်းကို အကြမ်းဖျော်း နားလည်အောင် ဖတ်ပြီးမှ နောက်ဆက်တဲ့ (??) ကို ဖတ်စေချင်ပါတယ်။

မိုက်ခရီးဆော် ဝင်းသီးမှာ Notepad ၊ အက်ပဲလ် မက်ခံအိအက်စဲမှာTextEdit စတဲ့ တက်စ်အယ်ဒီတာတစ်ခုခုနဲ့ ပရိုဂရမ်ကုဒ်ရေးလိုရတယ်။ ကုဒ်ဖိုင်ကို .py အိပ်စာန်းရှင်းနဲ့ သိမ်းရပါမယ်။ ပလိန်းတက်စ် (plain text) ဖိုင် ပါပဲ။ Python ကုဒ်ဖိုင်မို့လို့ .txt အစား .py နဲ့ သိမ်းတာပါ။ Python ဖိုင်နံမည်ကို စာလုံးအသေးနှပဲ ပေးတဲ့ ထုံးစံရှိတယ်။ စပေါ်နေရာမှာ _ (underscore) သုံးတဲ့ ထုံးစံရှိတယ်။ ဒါကြောင့် 'Meet Karel' ပရိုဂရမ်ကုဒ်ကို meet_karel.py ဖိုင်မှာ သိမ်းသင့်တယ်။

Python နဲ့ ရေးထားတဲ့ ပရိုဂရမ်ကို run မယဆိုရင် Python ဆော်ဖဲရှိရမှာပါ။ ဒီဆော်ဖဲ အင်စတောလ် လုပ်နည်းကို နောက်ဆက်တဲ့ (??) စာမျက်နှာ (??) မှာ ဖော်ပြပေးထားပါတယ်။ Python ကုဒ်တွေကို ကွန်ပျူးတာက တိုက်ရှိက် နားမလည်ပါဘူး။ Python ဆော်ပဲတွေ ကုဒ်တွေကို တိုက်ရှိက် နားလည်ပြီး ကွန်ပျူးတေပါ်မှာ run လိုရအောင် ကြားခံဆောင်ရွက်ပေးတဲ့ ဆော်ပဲလို့ ယေဘုယျအားဖြင့် ယူဆနိုင်တယ်။

Python ထည့်ပြီးရင် stanfordkarel လိုက်ဘရီကို အောက်ပါကွန်မန်းနဲ့ အင်စတောလ် လုပ်ရပါမယ်။ အင်တာန်က်ပေါ်ကနေ ဒေါင်းလုဒ် လုပ်ရတာမြဲလို့ ကွန်နှုက်ရှင်ရှိရမယ်။

```
pip install stanfordkarel
```

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

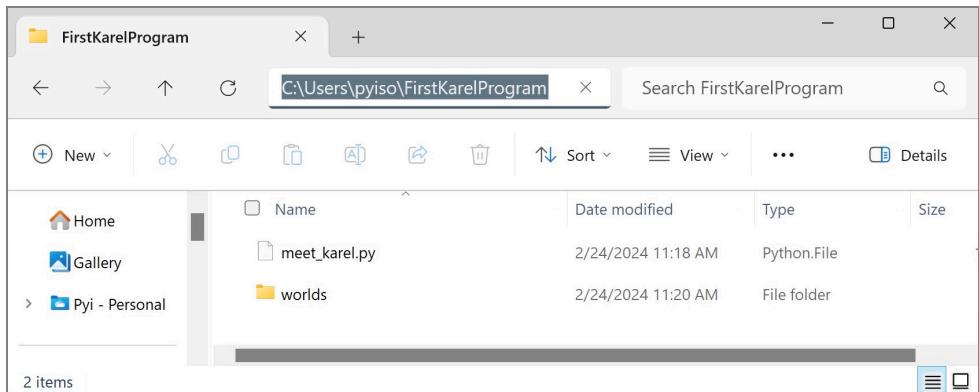
```
python meet_karel.py
```

ကုဒ်ရေးထားတာ ဆင်းတက်စံအမှား မရှိဘူးဆိုရင် ကားရဲပရိုကရမ် ပွင့်လာမှာပါ။

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

```
python your_karel_program.py
```

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



ပုံ ၁.၃

၁.၄ Python အင်စတောလ်လုပ်တဲ့အခါ ဘတွေပါလဲ

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

Python Interpreter

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

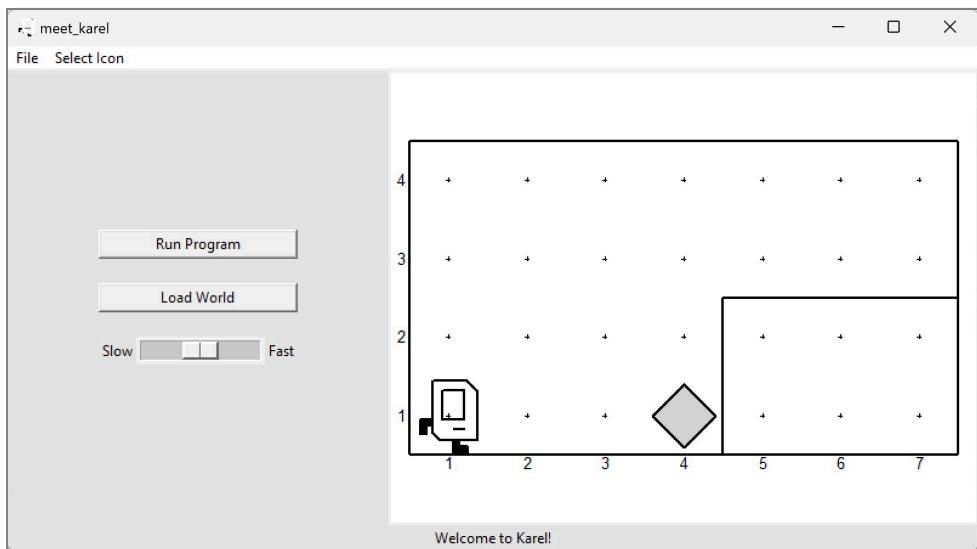
```

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

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

```

ပုံ ၁.၄



ပုံ ၁.၅

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

`python meet_karel.py`

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

Python Standard Library

Python အင်စတောလုပ်တဲ့အခါ standard library လည်း တစ်ပါတည်း ထည့်သွင်းပေးသွားတယ်။ Standard library မှာ ကဲ့ပြားခြားနားတဲ့ ရည်ရွယ်ချက်အမျိုးမျိုးအတွက် ကွန်ပိုးနှင့်တွေ့အမြာက်အများပါဝင်ပါတယ်။ Standard library ကွန်ပိုးနှင့်တွေ့ကို နောက်ပိုင်းမှာ အသုံးပြုကြရမှာပါ။ ငြင်းတိုကို အလျဉ်းသင့်သလို ဖော်ပြပေးသွားပါမယ်။

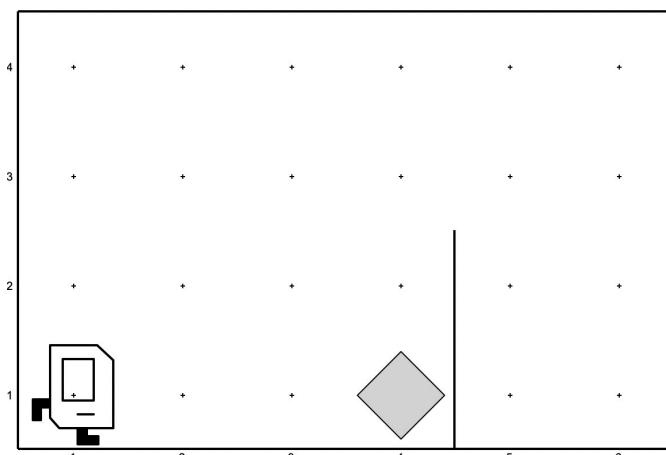
Standard အပြင် အခြားရရှိနိုင်တဲ့ လိုက်ဘရှိတွေလည်း အများအပြားရှိပါတယ်။ pip ဟာ လိုက်ဘရှိတွဲ အင်စတောလ်လုပ်ဖို့ အသုံးပြုတဲ့ ပရိုဂရမ်တစ်ခုပါပဲ။ Python documentation မှာတော့ pip ကို Python package တွေ အင်စတောလ် လုပ်ဖို့ အသုံးအများဆုံး tool (ထောက်ကူပြုပစ္စည်း) လို့ ဆိုတားပါတယ်။ Programming language တွေနဲ့ပါတ်သက်ပြီး မော်ချုံး (module)၊ ပက်ကော်ချုံး (package)၊ လိုက်ဘရှိ (library) စိတ် အသုံးအနှစ်နဲ့တော့ တွေ့ရကြားရပါတယ်။ ဒီစကားလုံးတော့ အခေါ်အဝေါ်တွေရဲ့ အဓိပ္ပာယ်သတ်မှတ်ချက်ကဗျာည်း သက်ဆိုက်ရာ programming language အလိုက် ကွာ ခြားလေ့ရှိတယ်။ ဘိုင်နာအနေနဲ့ ဒါနဲ့ပါတ်သက်ပြီး သိပ်ပြီးခေါင်းရှုပ်ခံစရာ မလိုသေးပါဘူး။ ပရိုဂရမ်ကုပ္ပါဒ် သိမ်းဆည်း ဖွဲ့စည်း ထားသို့ ဖြန့်ဖြူးတဲ့ ကိစ္စတွေနဲ့ သက်ဆိုင်တဲ့ အခေါ်အဝေါ်တွေလို့ သိတားရင် လုံးလောက်ပါပြီ။ နောက်ပိုင်းမှာ ဒီအခေါ်အဝေါ်တွေနဲ့ သက်ဆိုင်တဲ့ အဓိပ္ပာယ်သတ်မှတ်ချက်တွေကို အလျဉ်းသင့်သင့်လို့ ဖော်ပြပေးပါမယ်။

၁.၅ Move Beeper to Other Side

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

ပုံ (??) မှ ဘိပါကို နံရုံအခြားတစ်ဘက် အောက်ခြေကို ရွှေ့ပေးတဲ့ ပရိုဂရမ် ရေးကြည့်ပါ။ Python ထုံးစုံအရ move_beeper_to_other_side.py ဖိုင်နဲ့ သိမ်းသင့်ပါတယ်။ meet_karel.zip ဖိုင်ထဲမှုပါတဲ့ worlds ဖုဒါမှုပဲ အခု ကဗ္ဗာဖိုင် ထည့်ပေးထားပါတယ်။ move_beeper_to_other_side.w နံမည်နဲ့ပါ။ အန်ထရိုပိုင့်အတွက် အခုလိုရေးရပါမယ်။

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



ပုံ ၁.၆

•J

အခန်း J

ကားရဲလိန့် ကွန်ထရီးလ် စတိတ်မန္ဒြုဂျား

ကားရဲလိန့်ပရှိကရမ်တစ်ခု ဖွံ့ဖည်းတည်ဆောက်ပုံကို ရှေ့အခန်းမှာ လေ့လာခဲ့ကြပြီး ကွန်ထရီးလ် စတိတ်မန့် တွေ ဖြစ်ကြတဲ့ **for loop**, **while loop**, **if** နဲ့ **if...else** တိုကို အခုဆင်လက် လေ့လာကြပါမယ်။ ပရှိကရမ်တစ်ခုကို run တဲ့အခါ ပရှိကရမ်ကုဒ်ထဲက ညွှန်ကြားချက်တွေအတိုင်း ကွန်ပြုတာက လုပ်ဆောင် ပေးတာပါ။ ဒီလိုလုပ်ဆောင်ပေးတာကို အကွဲခံကျိုး (execute) လုပ်တယ်လို့ ခေါ်တယ်။ ကွန်ထရီးလ် စတိတ်မန့်တွေဟာ ပရှိကရမ်တစ်ခုရဲ့ execution flow ကို ထိန်းချုပ်ပေးတာ ဖြစ်တဲ့အတွက် control flow statements တွေလိုလည်း ခေါ်ပါတယ်။

J.၁ for loop

ကွန်ထရီးလ်စတိတ်မန့် တစ်မျိုးဖြစ်တဲ့ **for** loop ဟာ စတိတ်မန့် တစ်ခု (သို့) စတိတ်မန့် တစ်စုံကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ထပ်ခါထပ်ခါ ပြန်ကျေားပါတယ်။ move ကို နှစ် ဆယ့်ငါးကြိမ် ကျေားဖို့ for loop နဲ့ အခုလို

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

ရေးရပါတယ်။ put_beeper, move, turn_left စတိတ်မန့် သုံးကြောင်းတစ်စုံကို အကိုမ်တစ်ရာ ကျော်ချင်ရင် ဒီလိုပါ

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

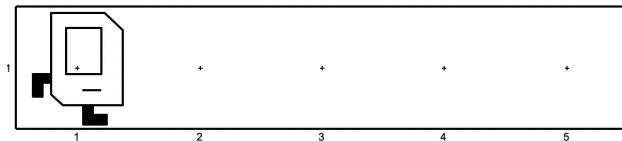
နှစ်ဆယ့်ငါးကြိမ်ကို range(25), အကြိမ်တစ်ရာကို range(100) စသည်ဖြင့် တွေ့ရတယ်။ ယောက်အားဖြင့် အကြိမ်အရေအတွက် N ကြိမ် ကျော်ချင်ရင်

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

ပုံစံနဲ့ သတ်မှတ်ရတာ။ N က အကြိမ်အရေအတွက်ကို ဖော်ပြတဲ့ ကိန်းပြည့်ကဏ္ဍးဖြစ်တယ်။ for loop ရေးတဲ့အခါ ကော်လံ : မကျွန်းခဲ့ဖို့ သတိပြုရပါမယ်။ x ဟာ ပေခိုရောဘဲလ်တစ်ခုဖြစ်ပြီး i , j , k စတဲ့ အကွားရာတစ်ခုနဲ့ ကိုယ်တားပြုလေ့ရှိတယ်။

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

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



ပုံ J.၁

ပုံ (??) ကားရဲလ်ကဲဗာမှာ ကွန်နာအားလုံးမှာ ဘိပါတစ်ခုစီ ချထားပေးရမယ်။ ကွန်နာဝါးခုရှိတာမိုလို စုစုပေါင်း ဘိပါဝါးခု ချထားရမှာပါ။ move ကတော့ လေးကြိမ်ပဲ လုပ်ရမယ် (ကားရဲလ်ရှေ့မှာ ကွန်နာလေး ခုပဲ ရှိတယ်)။ ဒီအတွက်ကို အခုလိုရေးဆိုင်ပါတယ်။

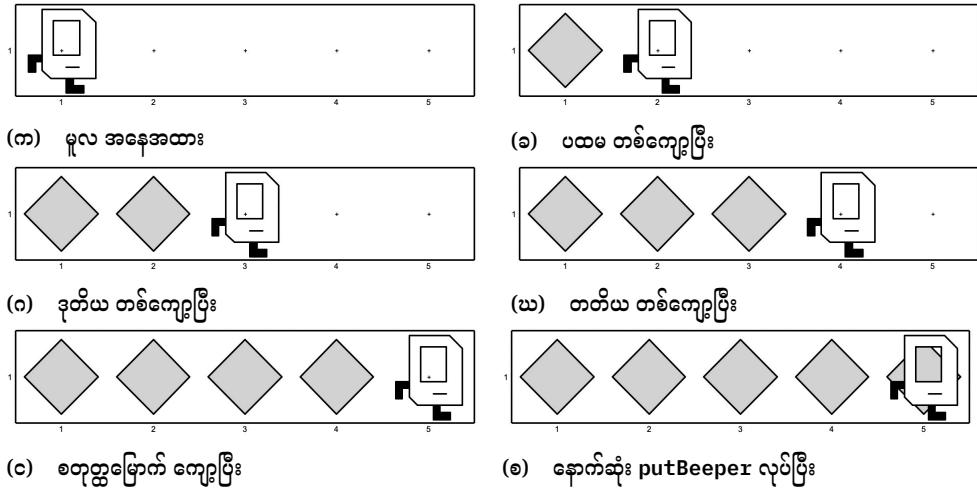
```
# File: make_row_of_5 beepers.py
from stanfordkarel import *

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

    put_beeper()

if __name__ == "__main__":
    run_karel_program("make_row_of_5 beepers")
```

for loop ဟာ put_beeper နဲ့ move ကို လေးကြိမ်ကျော်ပေးမှာပါ။ တစ်ကြိမ်ပြီးတိုင်း ရှိနေမဲ့ အနေအထား တစ်ခုချင်းစိတ်ပုံ (??) မှာ တွေ့နိုင်ပါတယ်။ လေးကြိမ်ပဲမှာက် နောက်ဆုံးတစ်ကျော်အပြီး ပုံ (??) (c) မှာ ဘိပါတစ်ခုလိုနေသေးတာ တွေ့ရမယ်။ for loop ပြီးတဲ့အခါ put_beeper တစ်ခါတပ် လုပ်ပေးတော့မှ တစ်တန်းလုံးအပြည့်ဖြစ်သွားမှာပါ။



ဦး J-J

Off-by-one bug

ပြီးခဲ့တဲ့ ဥပမာဏ ဒါ၏ loop အပြီး put_beeper မလုပ်မိရင် ပရိုကရမ်ဟာ လိုချင်တဲ့ အတိုင်း မဖြစ်ပါဘူး။ ဘိပါတစ်ခုလိုနေမယ်။ ဒီလိုပြဿနာမျိုးဟာ အဖြစ်များတဲ့ အမှားဖြစ်ပြီး off-by-one bug လိုခေါ်ပါတယ်။ Loop သုံးတဲ့ အခါ ကြိုရလေ့ရှိတဲ့ ဖြစ်တတ်တဲ့ အမှား (bug) ဖြစ်တယ်။ သတ်မှတ်ထားတဲ့ အတိုင်း၊ ဖြစ်သင့်တဲ့ အတိုင်း ပရိုကရမ်က အလုပ်မလုပ်ဘဲ ဖြစ်နေတဲ့ အမှားကို ကွန်ပျိုတာအသုံးအနောက်မှာ ပေးပို့လိုပေးတယ်။ Bug ကိုလိုက်ရှာပြီး မှန်အောင်ပြင်ပေးတာကိုတော့ debug လုပ်တယ်လိုခေါ်ပါတယ်။

ဘိပါတစ်ခုကျွန်ခဲ့တဲ့ အမှားမျိုးဟာ off-by-one bug ပေးပို့လိုပေးတာကတစ်ခုမျှသာ ဖြစ်တယ်။ ခုနှစ်နဲ့ ဆယ့်သုံးကေား ကောင်း ခုနှစ်လုံးရှိပါတယ် (ခုနှစ်နဲ့ ဆယ့်သုံးအပါဝင် ဆိုလျှင်)။ $13 - 7 = 6$ လိုတွက်မိရင် တစ်လုံး လိုနေပါလိမ့်မယ်။ သုံးပေါ်ဗျား တစ်တိုင် အလျားပေသုံးဆယ်ရှိ တပြောင့်တည်း ခြေစည်းရှိုးတစ်ခု အတွက် တိုင်စုစုပေါင်း တစ်ဆယ့်တစ်တိုင် လိုပါမယ်။ $30 \div 3 = 10$ လို တွက်ရင် မမှန်ပါဘူး။ တစ်ခု လိုတာ အပြင် တစ်ခုပုံ့နေတာလည်း ဖြစ်တတ်တယ်။ စောနက ခြေစည်းရှိုးမှာပဲ ထရံအချုပ်အရေအတွက် ကတော့ ကိုးချုပ်ဖြစ်ရမှာပါ။ $30 \div 3 = 10$ လို တွက်ရင် တစ်ခုပုံ့နေပါမယ်။ ခုနှစ်နဲ့ ဆယ့်သုံးကို ထည့်မစဉ်းစားရင် ကြားမှာ ကောင်းငါးလုံးရှိတာပါ။ $13 - 7 = 6$ လိုတွက်မိရင် တစ်လုံး ပို့နေပါလိမ့်မယ်။ ဒီ ဥပမာ အားလုံးဟာ အခြေခံသောတရားအားဖြင့် သိပ်မက္ခာဌားတဲ့ off-by-one bug ပုံစံကွဲအမျိုးမျိုး ဖြစ်တယ်။

'Make Row of Five Beepers' ခုတိယ ဗုံးရှင်း

ပရိုကရမ်ရေးတဲ့ အခါ ပရိုကရမ်မာတွေ တစ်ယောက်နဲ့ တစ်ယောက် စဉ်းစားပုံ စဉ်းစားနည်း၊ ပြောင်းနည်းထပ်တွက်လေ့ရှိပါဘူး။ ဘိပါဝါးခု အတော်းလိုက် ချေပေးတဲ့ ပရိုကရမ်ကိုပဲ နောက်ဟာရှင်းတစ်မျိုးနဲ့ ရေးနိုင်ပါတယ်။

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

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

J.J အင်ဒန်ထုတ်လုပ်ခြင်းနှင့် ကုဒ်စာရင်းချုပ်

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

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

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

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

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

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

J.2 while loop

အခြေအနေတစ်ရပ် မှုန်နေသူ၏ စတိတ်မန့်တွေကို တစ်ကြိမ်ပြီးတစ်ကြိမ် ပြန်ကျော့ လုပ်ဆောင်စေချင် ရင် while loop ကို အသုံးပြုပါတယ်။ for နဲ့ while loop နှစ်ခုလုံးက ပြန်ကျော့ပေးတာ ဖြစ်ပေါ့။ for ကို အကြိမ်အရေအတွက် အတိအကျသိတဲ့ကိစ္စမျိုးမှာ သုံးလေ့ရှုပြီး while ကိုတော့ ဘယ်နှစ်ကြိမ် လဲ ကိုတွေက်လို့မရတဲ့ အခြေအနေ အပေါ်မှုတည်ပြီး လုပ်ဆောင်ပေးရမဲ့ အကြိမ်အရေအတွက် ကွား နိုင်တဲ့ ကိစ္စမျိုးတွေမှာ သုံးလေ့ရှိတယ်။ ဥပမာတစ်ခုနဲ့ ကြည့်ရင် ပိုနားလည်ပါလိမ့်မယ်။

(၁) လမ်းပေါ်မှာ ဘိပါတစ်ခုရှိမယ်။ ဘိပါ ဘယ်ကိုနာများရှိမှုလဲ၊ လမ်းဘယ်လောက်အရှည်ဖြစ်မလဲ ကြိုတင်မသိဘူးလို့ ယူဆပါ။ နုတ္တနာကဗ္ဗာ တစ်ခုကို ပုံ (??) မှာ တွေ့နိုင်တယ်။ ဘိပါကို သွားပြီး ကောက်ခိုင်း ရပါမယ်။ အလားတူ မည်သည့်ကဗ္ဗာအတွက်မဆို ဘိပါကောက်ပေးနိုင်ရမှာ ဖြစ်တယ်။ ဘိပါရှိမဲ့ ကွန်နာကို



ဗိုလ်ချုပ်

ကြိုတင်မသိထားတဲ့အတွက် ဘယ်နှစ်ကြိုမ် move လုပ်ခိုင်းရမှာလဲ မသိဖြစ်နေတယ်။ အကြိုမ်အရေအတွက် မသိတဲ့အတွက် for loop နဲ့ အဆင်မပြတော့ပါဘူး။

ဘိပါမရှိသ၍ တစ်ကြိုမ်ပြီးတစ်ကြိုမ် move လုပ်ခိုင်းမယ်ဆိုရင် နောက်ဆုံးမှာ ဘိပါရှိတဲ့ကွန်နာကို ရောက်သွားမှာ သေချာပါတယ်။ အဲဒီလို လုပ်ခိုင်းဖို့ရာအတွက် while loop နဲ့ ရေးထားတာကို အခုလုံ တွေ့ရှုပါမယ်။

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

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

while loop အလုပ်လုပ်ပုံက ဒီလိုပါ။ no_beeper_present ကွန်ဒီရှင် စစ်ပါတယ်။ မှန်ရင် move လုပ်ပါတယ်။ ပြီးရင် ကွန်ဒီရှင်ပြန်စစ်တယ်။ မှန်ရင် move ကို နောက်ထပ်တစ်ကြိုမ် ထပ်ကျော်ပါတယ်။ ကွန်ဒီရှင်စစ်လိုက်၊ မှန်ရင် တစ်ခါထပ်ကျော်လိုက်၊ ကွန်ဒီရှင်ပြန်စစ်လိုက်၊ မှန်ရင် နောက်တစ်ခါ ထပ်ကျော်လိုက်၊ ... ဒီအတိုင်းဆက်ပြီး ကွန်ဒီရှင်မှန်နေသ၍ while loop က move ကို ပြန်ကျော်ပေး မှာပါ။ ကွန်ဒီရှင် စစ်လိုက်လို့ မှားသွားပြီဆုံးရင်တော့ ထပ်မကျော်တော့ဘဲ ရပ်သွားမှာဖြစ်တယ်။ while loop တစ်ကြိုမ်ကျော်ပြီးတိုင်း ရှိနေမဲ့ အနေအထား တစ်ခုချင်းကို ပုံ (??) မှာ လေ့လာကြည့်ပါ။ လေးကြိုမ်မြောက် ကျော်ပြီးနောက် ကွန်ဒီရှင်စစ်လိုက်တဲ့အခါ မှားနေပြီ။ ဒီအခါ while loop က ထပ်မကျော်တော့ဘူး။ ဒီလို loop က ပြန်ကျော်နေတာ ရပ်သွားရင် loop ကနေ ထွက်တယ် (loop exits) လို့ ဖြေ လေ့ရှိတယ်။

အပြည့်အစုံ ဖော်ပြပေးထားတဲ့ ပရိုဂရမ်ကို လေ့လာကြည့်ပါ။

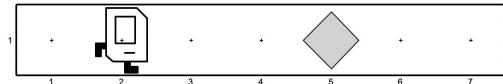
```
# File: go_pick_beeper.py
from stanfordkarel import *
```

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

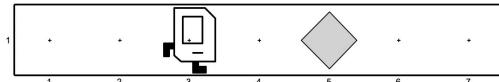
    pick_beeper()
```



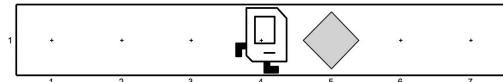
(က) မူလ အနေအထား



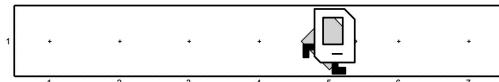
(ခ) ပထမ တစ်ကျွွှုပြီး



(ဂ) ဒုတိယ တစ်ကျွွှုပြီး



(ဃ) တတိယ တစ်ကျွွှုပြီး



(င) စတုတ္ထမြောက် ကျွွှုပြီး

ပုံ J-၄

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

ကားရဲလ် ကွန်ဒါရိုင်များ

beepers_present ကွန်ဒါရိုင် ကျတော့ လက်ရှိကွန်နာမှာ ဘိပါရိုရင် အမှန်၊ မရှိရင် အမှား ရလဒ်ထဲက် တယ်။ no beepers_present နဲ့ ဆန်ကျင်သာက်ပေါ့။ ကားရဲလ် ကွန်ဒါရိုင်တွေအားလုံးကို တေဘာလ် (၁.၄) မှာ ကြည့်ပါ။ ပုံမှန်စစ်တာနဲ့ အငြင်းပုံစစ်တာ နှစ်မျိုးကို ယုံ့တွဲပြထားပါတယ်။

ကွန်ဒါရိုင်	ဆန်ကျင်ဘက် ကွန်ဒါရိုင်	စစ်ပေးသည့် အခြေအနေ
front_is_clear	front_is_blocked	ရှေ့မှာ နံရံကပ်လျက် ရှိမရှိ
left_is_clear	left_is_blocked	ဘယ်ဘက်မှာ နံရံကပ်လျက် ရှိမရှိ
right_is_clear	right_is_blocked	ညာဘက်မှာ နံရံကပ်လျက် ရှိမရှိ
beepers_present	no beepers_present	လက်ရှိကွန်နာမှာ ဘိပါရိုမရှိ
beepers_in_bag	no beepers_in_bag	ကားရဲလ်၏ ဘိပါအိပ်ထဲ ဘိပါရိုမရှိ
facing_north	not_facing_north	အရှေ့ဘက် မျက်နှာမှုလျက် ရှိမရှိ
facing_east	not_facing_east	အနောက်ဘက် မျက်နှာမှုလျက် ရှိမရှိ
facing_west	not_facing_west	တောင်ဘက် မျက်နှာမှုလျက် ရှိမရှိ
facing_south	not_facing_south	မြောက်ဘက် မျက်နှာမှုလျက် ရှိမရှိ

တေဘာလ် J.၁ ကားရဲလ် စစ်ဆေးသည့် ကွန်ဒါရိုင်များ

while loop ဆင်းတက်ခဲ့

while loop ရေးတဲ့ ပုံစံက ယေဘုယျအားဖြင့် ဒီလိုပါ။

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

condition က ကားရဲလ်ဂွန်ဒါရှင် တစ်ခုဖြစ်တယ်။ ကော်လံ : မကျွန်ခဲ့အောင် ဂရိုစိုက်ပါ။ condition မှန်နေသေးသ၍ ပြန်ကော့စေချင်တဲ့ စတိတ်မန်တွေကို while အောက်မှာ အင်ဒန်ထုတ်လုပ်ထား ရပါမယ်။ ရှေ့မှာရှင်းနေသ၍ move လုပ်တဲ့ while loop ကို အခုလို

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

ရေးပါတယ်။

‘Make Beeper Row’ ဥပမာ

‘Make Row of Five Beepers’ ဥပမာမှာ လမ်းခွဲအလျေားဟာ မပြောင်းလဲဘူး ယူဆတာမိုလို for loop ကို အသုံးပြုတာ ဆိုလျော်ပါတယ်။ လမ်းခွဲအရှည်ကို ပြိုမသိထားဘူး၊ တစ်လမ်းလုံး ဘိပါတွေ ဖြန့်ထားပေးရမယ်ဆုံးရင် while နဲ့ ရေးရမှာပါ။

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

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

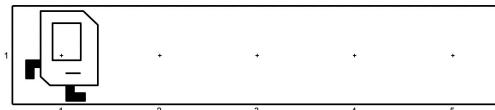
    put_beeper()

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

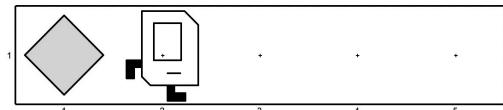
တစ်ကြိမ်ကော့ပြီးတိုင်း ရှိနေမဲ့ အနေအထားကို ပုံ (??) မှာ ကြည့်ပါ။ လေးကြိမ်မြောက်အပြီး front_is_clear စစ်တဲ့အခါ မှားနေပြီဖြစ်လို့ ထပ်မကော့တော့ဘဲ loop ကနေ ထွက်သွားမယ်။ ဒီအခါမှာ ဘိပါတစ်ခု လိုနေသေးတယ်။ put_beeper ထပ်လုပ်ရမယ်။ တစ်ခါပဲ လုပ်ရမှာပါ။ ဒါကြောင့် while loop ထဲမပါအောင် ပထမအဆင့် အင်ဒန်ထုတ်ပဲ ဖြစ်ရမယ်။

J.၄ if စတိတ်မန်

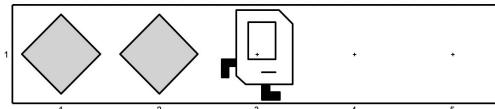
တစ်ခု (သို့) တစ်ခုထက်ပိုတဲ့ စတိတ်မန်တွေကို အခြေအနေတစ်ရပ် မှန်တော့မှာပဲ လုပ်ဆောင်စေချင်တဲ့ အခါ if ကို အသုံးပြုနိုင်တယ်။ ဥပမာ



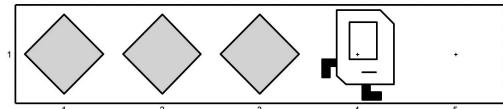
(က) မူလ အနေအထား



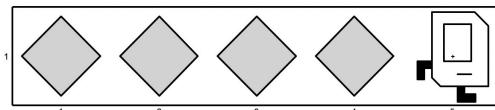
(ခ) ပထမ တစ်ကျော်ပြီး



(ဂ) ဒုတိယ တစ်ကျော်ပြီး



(ဃ) တတိယ တစ်ကျော်ပြီး



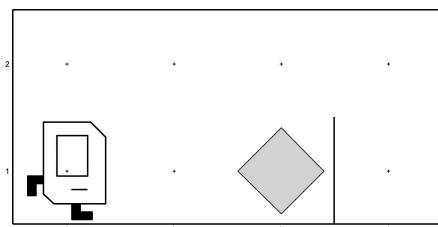
(ဃ) စတုထွေ့ပြောက် ကျော်ပြီး

ပုံ J.၅

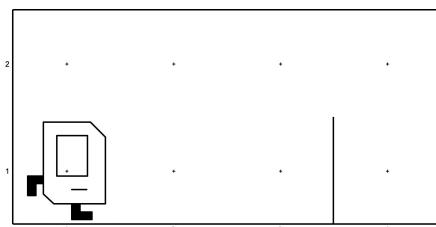
```
if beepers_present():
    pick_beeper()
```

if စတိတ်မန်ဟာ beepers_present ကွန်ဒါရှင် မှန်တော့မှုပဲ pick_beeper လုပ်မှာပါ။ မှားရင် မလုပ်ပါဘူး။

ပုံမှာတွေ့ရတဲ့ ကဗ္ဗာနှစ်ခုထဲက တစ်ခုမှာ ကားရဲလှုံးနေမယ် ဆိုပါစို့။ ဘိပါရှိတဲ့ ကဗ္ဗာဆိုရင် ဘိပါကို



(က)



(ခ)

ပုံ J.၆

နံ့ရံအခြားဘက် အောက်ခြေကို ရွှေ့ပေးရမယ်။ မရှိတဲ့ကဗ္ဗာဆိုရင် နံ့ရံ ဒီဘက် အောက်ခြေမှာပဲ ရပ်နေရမယ်။ ပရိုဂုရမ်က ကဗ္ဗာနှစ်ခုလုံးမှာ မှန်အောင် အလုပ် လုပ်နိုင်ရပါမယ်။ ဒီအတွက် if စတိတ်မန် သုံးထားတာ လေ့လာကြည့်ပါ။

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

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

J.๒

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

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

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

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

condition က front_is_clear, beepers_present စတဲ့ ကားရဲလ် ကွန်ဒါရှင် တစ်ခုခုဖြစ်မယ်။
တေဘာလ (၁.၄) မှာ ကားရဲလ်ကို စစ်ခိုင်းလိုရတဲ့ ကွန်ဒါရှင်အားလုံး ပြထားပါတယ်။

J.၃ if...else စတိတ်မန့်

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

```
from stanfordkarel import *
```

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

```

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

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

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

```

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

```

if condition :
    statement1a
    statement2a
    statement3a etc.

else:
    statement1b
    statement2b
    statement3b etc.

```

J.၆ Nested ကွန်ထရီးလ် စတိတ်မန်များ

ကွန်ထရီးလ် စတိတ်မန် ဘလောက်ထဲမှာ ကွန်ထရီးလ် စတိတ်မန် ရှိလိုက်တယ်။ Nested ကွန်ထရီးလ်
စတိတ်မန်လို့ ခေါ်ပါတယ်။

Nested for loop

ဘိပါသုံးချခြင်ကော် ရှေ့တိုးလိုက် လုပ်တာကို လေးကြိမ်ကျော်ချင်ရင် for loop နဲ့ အခုလို ရေးရမယ်ဆုံး တာ သိခဲ့ပြီးပါပြီ။

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

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

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

for loop နှစ်ခု ဆင့်ရေးထားလို nested for loop လိုခေါ်ပါတယ်။ Loop တစ်ခုချင်းကို သို့ခြား ရည်ညွှန်းချင်တဲ့အခါ အပြင် for loop နဲ့ အတွင်း for loop လို ပြောလေ့ရှိတယ်။ အင်ဒန့်ထဲ လုပ် ထားပုံအရ အပြင် for loop ဘလောက်ထဲမှာ အတွင်း for loop နဲ့ move ပါဝင်တယ်။ အတွင်း for loop ဘလောက်ထဲမှာတော့ put_beeper ပဲပါတယ် move မပါ ပါဘူး။ အပြင် for loop မှာ ပေရီ ရောကဲလ် i ဆိုရင် အတွင်းမှာ i မဖြစ်ရပါဘူး။ j, ဒါမှုမဟုတ် k မတူတာ တစ်ခုခုဖြစ်ရပါမယ်။

အတန်းလိုက် ကွန်နာ ငါးခုမှာ တစ်ကွန်နာ ဘိပါ (၂၅) ခုစီ ချထားပေးဖို့ nested loop နဲ့ ရေးထား တာကို လေ့လာကြည့်ပါ။

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



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

        for i in range(25):
            put_beeper()

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

Nested for and while

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

တောင်ဘက် နံရုံတလျောက် ဘိပါတွေချေသွားမယ် (နောက်ဆုံး ကွန်နာမှာ ဘိပါမချေဘဲ ချိန်ထားမယ်)။ ပြီးရင် အရှေ့ဘက် နံရုံအတွက် အဆင်သင့်ဖြစ်အောင် ဘယ်ဘက်လှည့်မယ်။ ပုံ (??) (က) နဲ့ (ခ) ကို ကြည့်ပါ။ အရှေ့၊ မြောက် နဲ့ အနောက်ဘက် နံရုံတွေအတွက်လည်း ဒီအတိုင်း လုပ်သွားရှုပါပဲ။ ပုံ (??) (ဂ)၊ (ယ)၊ (က) အသီးသီးကို ကြည့်ပါ။

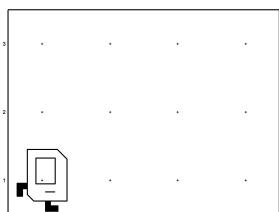
နံရုံတစ်ဘက် အတွက် ဆိုရင် အခုလို

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

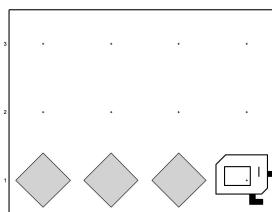
ဖြစ်မယ်။ (while ဘလောက်မှာ turn_left မပါတာ ဂရုပြုပါ)။ နံရုံလေးဘက်အတွက် လေးကြိမ် လုပ် ရမှာဆိုတော့ for နဲ့ အခုလို

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

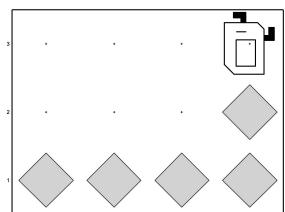
ရေးရပါမယ်။



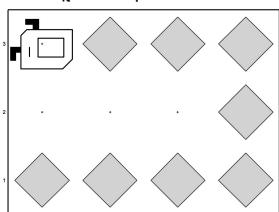
(က) မူလ အနေအထား



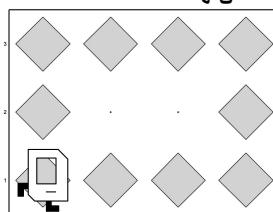
(ခ) ပထမ တစ်ကျော်ပြီး



(ဂ) ဒုတိယ တစ်ကျော်ပြီး



(ယ) တတိယ တစ်ကျော်ပြီး



(က) ဓမ္မတွေ့မြောက် ကျော်ပြီး

ပုံ J-7

ပရီဂရမ် တစ်ခုလုံး အတွက်ဆိုရင် အခုလိုပါ

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

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

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

Nested while and if

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

```
from stanfordkarel import *

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

        if beepers_present():
            pick_beeper()

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

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

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

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

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

    if beepers_present():
        pick_beeper()

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

if...else သုံးထားတယ်။ ကျန်တာအားလုံး လမ်းရှင်းတဲ့ ပရိုဂရမ်နဲ့ တူတူပဲ။

Nested if

အခြေအနေ တစ်ရပ် မှန်တော့မှုပဲ လုပ်ချင်ရင် if ကို သုံးတယ်။ အခြေအနေ ‘နှစ်ရပ်’ လုံးနဲ့ ကိုက်ညီမှ လုပ်ဆောင်စေချင်တဲ့အခါ nested if သုံးနိုင်ပါတယ်။ ညာဘက်လည်းရှင်း လက်ရှိကွန်နာမှာလည်း ဘိပါမရှိမှ ဘိပါတစ်ခု ချထားခိုင်းမယ်ဆိုပါစို့။ အခုလိုရေးနိုင်ပါတယ်

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

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

ပုံ (??) (က) မှာ ဒုတိယ လမ်းတစ်လျှောက် လမ်းပြင်တဲ့အလုပ်ကို ကားရဲလ်ကို တာဝန်ပေးတယ်လို့ ယူဆပါ။ ဘိပါရေး ညာဘက်နဲ့ရုံပါ မရှိတဲ့ ကွန်နာတွေက လမ်းအခြေအနေ တော်တော်ဆိုးနေတဲ့ နေရာတွေ။ ဒီလိုနေရာတွေမှာ ဘိပါတစ်ခု ဖြည့်ပြီး လမ်းပြင်ပေးရမှာပါ။

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

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

```

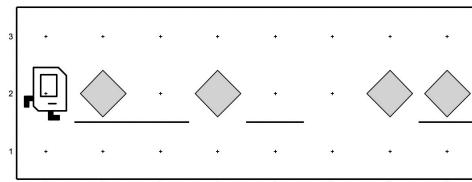
        put_beeper()
move()

if right_is_clear():
    if no beepers_present():
        put_beeper()

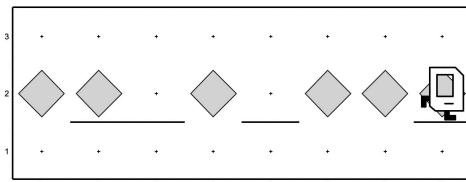
if __name__ == "__main__":
    run_karel_program("repair_street")

```

while ဘလောက်ထဲမှ nested if နဲ့ move ပါဝင်တယ်။ nested if က ညာဘက်လည်ရှင်း ဘိပါလည်းမရှိမှ လက်ရှုံးကွန်နာမှာ ဘိပါတစ်ခု ချခိုင်းထားတာပါ။ while ထဲမှာ if၊ အဲဒီ if ထဲမှာမှ နောက်ထပ် if တစ်ခု ဆင့်ထားတဲ့အတွက် သုံးဆင့် nest လုပ်ထားတာပါ။ အင်ဒန်ထုတွေ ဂရုံးကိုကြည့်ဖို့ လိုတယ်။ ဘယ်ဘလောက်ထဲမှာ ဘာရှိနေလဲဆိုတာ မြင်အောင် လုပ်ရမှာပါ။ ကိုယ်တိုင် ရေးရှင်လည်း အင်ဒန်ထု ဂရုံးကိုပြီး လုပ်ရပါမယ်။ လမ်းပြင်ပြီး အခြေအနေကို ညာဘက် ပုံ (??) (ခ) တွင် ကြည့်ပါ။



(က) မူလ အနေအထား



(ခ) လမ်းပြင်ပြီး အနေအထား

ပုံ J.7

အခန်း ၃

ဖန်ရှင်များ (Functions)

ဖန်ရှင် (function) တွေဟာ ပရိုကရမ်းမင်းမှာ အရေးကြီးဆုံး အခြေခံသဘောတရားတစ်ခု ဖြစ်တယ်။ ဖန်ရှင်ဆိုတာ ဘာလဲ၊ ဘာကြောင့် အရေးပါရတာလဲ၊ ဖန်ရှင်တွေကို ပရိုကရမ် ဒီဇိုင်းပြုလုပ် ရေးသားတဲ့အခါ ဘယ်လိုအသုံးချဖော်လဲ စတေတွေကို ဒီအခန်းမှာ လေ့လာကြပါမယ်။

၃.၁ ဖန်ရှင် သတ်မှတ်ခြင်း

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

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

ဒီလို သတ်မှတ်ထားပြီးရင် ညာဘက်လှည့်ချင်တဲ့အခါ

```
turn_right()
```

လို တိုက်ရှိက်ပြေလို ရသွားမှာ ဖြစ်ပါတယ်။ turn_left သုံးခါ ရေးဖို့ မလိုတော့ပါဘူး။

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

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

```
def turn_right():
```

ကို ဖန်ရှင် ဟက်ဒေါ (function header) လို ဆော်တယ် (မြန်မာလို ဆိုရင်တော့ ဖန်ရှင် ခေါင်းစဉ်းပေါ့)။ turn_right က ဖန်ရှင် အမည်။ ပါရာမီတာပါတဲ့ ဖန်ရှင်ဆိုရင် ဂိုက်ကွင်းထဲမှာ ပါရာမီတာတွေ

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

ဖန်ရှင် ဟက်ဒီအောက် အင်ဒန်ထုတ်လုပ်ထားတဲ့ လိုင်းအားလုံးဟာ ငှင်းဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ကုဒ်ဘလောက် ဖြစ်တယ်။ အခု turn_right ဖန်ရှင် ဘလောက်မှာ turn_left သုံးကြိုးမြှင့်ပါတယ်။

| turn_right()

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

ဖန်ရှင်သတ်မှတ်တာနဲ့ ဖန်ရှင်ကောလ လုပ်တဲ့ပုံစံကို အောက်ပါအတိုင်း ယော့ယျာအားဖြင့် တွေ့ရပါမယ်။ Python ထုံးစံအရ ဖန်ရှင်နံမည်မှာ စာလုံးအသေးကိုပဲ သုံးလေ့ရှိတယ်။ စကားလုံး နှစ်ခုနဲ့ အထက်ဆိုရင် ကြားမှ underscore (_) ခြားပေးလေ ရှိတယ်။

```
def name_of_function():
    statement1
    statement2
    statement3 etc.
```

| name_of_function()

ဖန်ရှင်နံမည် အဓိပ္ပာယ် အရေးကြီးပါတယ်

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

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

ဖန်ရှင်သတ်မှတ်ချက်နဲ့ ဖန်ရှင်ကောလ ဥပမာ တရာ့ကို လေ့လာကြည့်ပါ။ ဘိပါ နှစ်ဆယ့်ငါးခု ချုပ် တဲ့ put_25_beeper ဖန်ရှင်ပါ

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

put_25_beeper()

ဒါကတော့ ကွန်နာတစ်ခုမှာ ရှိတဲ့ ဘိပါအားလုံးကောက်ပေးတဲ့ ဖန်ရှင်ဖြစ်ပါတယ်

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

pick_all_beeper()
```

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

အခန်း (၁) မှာ လိုက်ဘရိဆိုတာ ဘာလဲ အကျဉ်း ဖော်ပြခဲ့တယ်။ မေ့ထရစ် $A \times B$ မြှောက်လဒ် $A \times B$ ကို numpy လိုက်ဘရိ ဖန်ရှင် matmul နဲ့ အဖြော်ရွေ့ပါတယ်။

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

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

ဖန်ရှင် အခြေခံအုတ်ချုပ်များ

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

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

    pick_all_beeper()
```

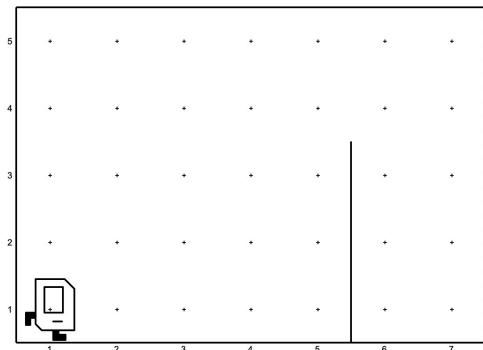
လမ်းတစ်လမ်းလုံး ရှင်းဖို့အတွက် ကွန်နာတစ်ခုက ဘိပါအားလုံး ကောက်ပေးတဲ့ pick_all_beeper ကို အခြေခံ အုတ်ချုပ်သွေ့ယူ အသုံးပြုထားတာပါ။

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

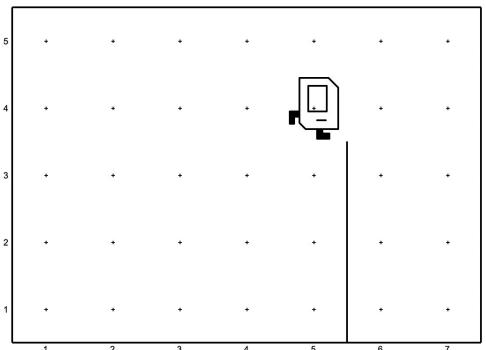
၃.၂ ပရီကွန်ဒီရှင်နှင့် ပို့ခိုကွန်ဒီရှင်

ပရီကွန်ဒီရှင် (precondition) နဲ့ (postcondition) ဟာ ဖန်ရှင်နဲ့ ပါတ်သက်ပြီး ကယ်နကာ နားလည်ဖို့ လိုအပ်တဲ့ အရေးကြီးတဲ့ သဘောတရားနှစ်ခုပါ။ ဖန်ရှင် မလုပ်ဆောင်မဲ့ ကြိုတင်ရှိနေရမဲ့ အခြေအနေကို ပရီကွန်ဒီရှင်လို ခေါ်ပြီး လုပ်ဆောင်ပြီး ရှိရမဲ့ အခြေအနေကို ပိုစ်ကွန်ဒီရှင်လို ခေါ်ပါတယ်။ သတ်မှတ်ထားတဲ့ ပရီကွန်ဒီရှင်နဲ့ ကိုက်ညီမှုသာ ဖန်ရှင်တစ်ခုဟာ သူလုပ်ဆောင်ပေးရမဲ့ ကိစ္စကို မှန်ကန်အောင် ဆောင်ရွက်ပေးနိုင်မှုပါ။ ဖန်ရှင် အသုံးပြုတဲ့ အခါမှုပါ ပရီကွန်ဒီရှင် ပိုစ်ကွန်ဒီရှင်တွေ အပေါ် အခြေခံပြီး တိတိကျကျဖြည့်တော်းဖို့ ပစာနက်ပါတယ်။

ပုံ ?? (??) မှ (??) အနေအထားသို့ ကားရဲလ်က တိုင်ထိပ်အရောက် တက်သွားရပါမယ်။ တိုင်အကွား အဝေး၊ အမြင့် အမျိုးမျိုးနဲ့ အလားတူ ကဲ့တွေမှာလည်း အလုပ်လုပ်ရပါမယ်။ (နံရုံကို တိုင်ဟု ယူဆပါ)။ တိုင်အောက်ခြေကိုသွားတာနဲ့ တိုင်ထိပ်တက်တာ အလုပ်နှစ်ခု ပါဝင်တယ်လို့ မြင်နိုင်တယ်။ ဒီအတွက် ဖန်



(က) မတိုင်မဲ့ အခြေအနေ



(ခ) ပြီးဆောက် အခြေအနေ

ပုံ ၃.၁ တိုင်ထိပ်သို့

ရှင်နှစ်ခု သတ်မှတ်ပေးပါမယ်။

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

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

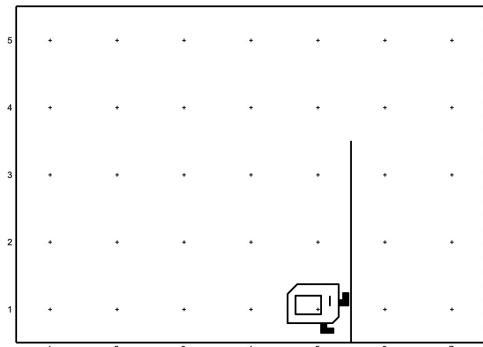
အထက်ပါ ဖန်ရှင်နှစ်ခုနဲ့ go_to_top ကို ဆက်လက် သတ်မှတ်ပါမယ်

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

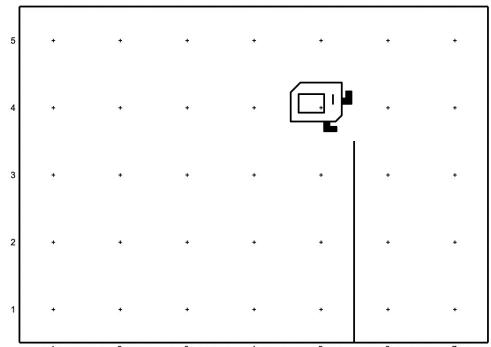
go_to_pole အပြီးမှာ ကားရဲလ်ဟာ တိုင်ခြေမှု အရော့ဘက်မျက်နှာမှုပြီး ရှိနေမှုပါ။ ascend_pole က တိုင်ခြေမှု ကားရဲလ် အပေါ်ဘက်ကို မျက်နှာမှုတဲ့ အနေအထားကနေ စရပါမယ်။ go_to_pole ပြီး ရင် ascend_pole အတွက် အသင့်အနေအထားဖြစ်အောင် turn_left လုပ်ပေးရပါမယ်။

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

ဖန်ရှင် လုပ်ဆောင်အပြီးမှာ ရှိနေရမဲ့ အခြေအနေကို ပိုစ်ကွန်ဒီရှင်လို့ ဖော်ပြုခဲ့တယ်။ ဖန်ရှင်စံခဲ့ဟာ ပရီကွန်ဒီရှင်နဲ့ ကိုက်ညီတဲ့ အနေအထားကနေ စတင်ရင် ပိုစ်ကွန်ဒီရှင်နဲ့ ကိုက်ညီအောင်လုပ်ဆောင်ပေးပြီး အဆုံးသတ်ရမှာပါ။ ပုံ ?? (?) မှာ `ascend_pole` ပိုစ်ကွန်ဒီရှင်ကို တွေ့နိုင်ပါတယ်။



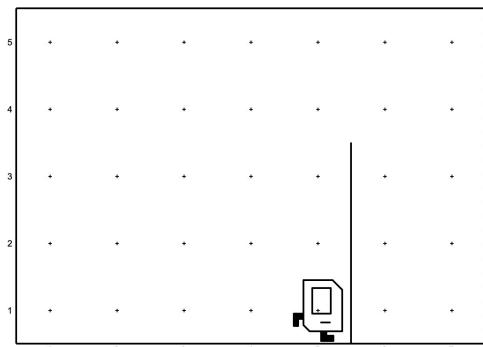
(က)



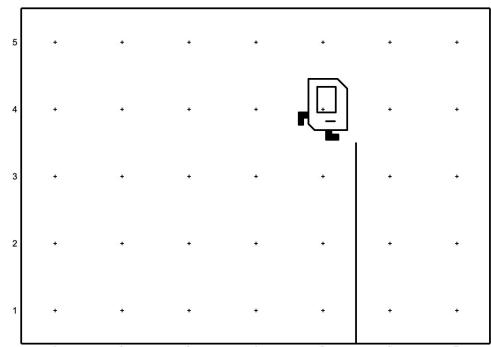
(ခ)

ပုံ ၃.၂ `ascend_pole` ပရီကွန်ဒီရှင်းနှင့် ပိုစ်ကွန်ဒီရှင်း

ဖန်ရှင် ပရီကွန်ဒီရှင် ပိုစ်ကွန်ဒီရှင်ကို သင့်တော်သလို သတ်မှတ်နိုင်ပါတယ်။ ပုံ ?? (?) တွင် `ascend_pole` အတွက် အခြား ရွေးချယ်နှင့် ပရီ နဲ့ ပိုစ် ကွန်ဒီရှင်ကို ကြည့်ပါ။



(က)



(ခ)

ပုံ ၃.၃ `ascend_pole` ပရီ နဲ့ ပိုစ် ကွန်ဒီရှင်

အထက်ပါ ပရီ နဲ့ ပိုစ် ကွန်ဒီရှင်အရ `ascend_pole` ကို အခုလို

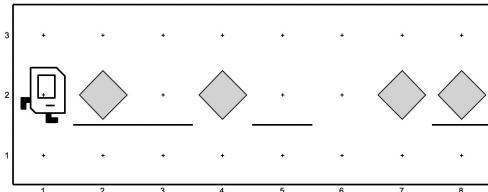
```
def ascend_pole():
    turn_left()
    while right_is_blocked():
        move()
        turn_right()
```

သတ်မှတ်ရမှာပါ။ go_to_top ကလည်း ဒီလို ဖြစ်သွားမယ်

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

၃.၃ ဖန်ရှင်များဖြင့် abstraction များ တည်ဆောက်ခြင်း

အခန်း (??) စာမျက်နှာ (??) မှ လမ်းပြင်တဲ့ ပရိုကရမ်မှာ ကွန်နာတစ်ခုဟာ ဘိပါလည်းမရှိ၊ အောက်ဘက် ကပ်လျက် နံရုံလည်းမရှိရင် ဘိပါတစ်ခု ချထားပေးရပါတယ်။ ဒီကိုစွဲ ဆောင်ရွက်ပေးဖို့အတွက် ဖန်ရှင်တစ်ခု သတ်မှတ်နိုင်ပါတယ်။



ပုံ ၃.၄

```
def repair_corner():
    if right_is_clear():
        if no_beeper_present():
            put_beep()
```

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

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

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

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

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

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

၃.၄ Bottom-up Programming

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

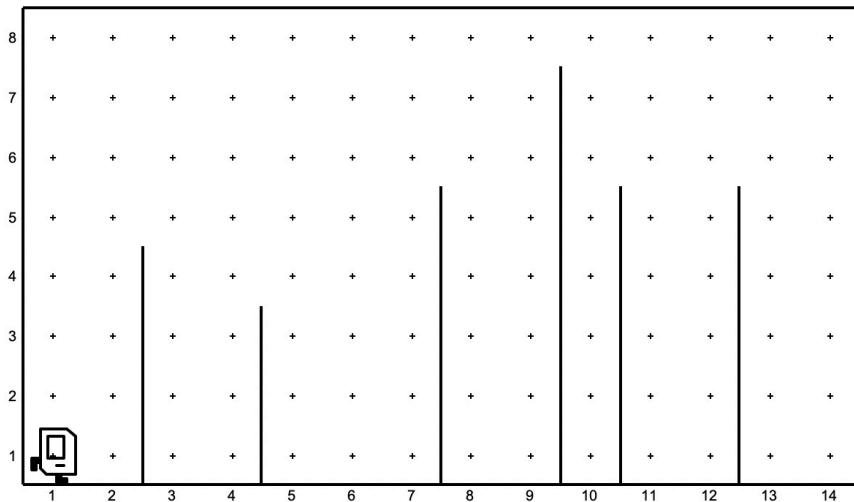
ခက်ခဲရှုပ်ထွေးတဲ့ ကိစ္စတစ်ခုကို ဖြေရှင်းတဲ့အခါ အပိုင်းတွေခဲ့ပြီး တစ်ပိုင်းချင်းကို သီး၏ဖြေရှင်းလေ့ရှိတယ်။ အပိုင်းတွေခဲ့လိုက်ခြင်းအားဖြင့် တစ်ပိုင်းစီဟာ မူလဖြေရှင်းရမဲ့ ကိစ္စလောက် မခက်ခဲတော့ ပါဘူး။ အရွယ်အစားအားဖြင့်လည်း နိဂုံထက် ငယ်သွားမယ်။ အပိုင်း တစ်ပိုင်းချင်းစီကို ဖြေရှင်းတဲ့အခါ မှာလည်း အလားတူပဲ လုပ်ဆောင်နိုင်တယ်။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

ပုံ (??) ကဗျာမှာ ကားရဲ့လှ တန်းကော်ပြီးပြိုင်ရပါမယ်။ ထောင်လိုက်နံရုံတွေကို တန်းတွေလို့ယူဆပါ။ တန်းတွေဟာ အနိမ့်အမြင့် အမျိုးမျိုးဖြစ်နိုင်ပါတယ်။ ကဗျာရဲ့ အပေါ်ဘက် နံရုံကို ထိတဲ့အထိတော့ မြင့်လို့မရပါဘူး။ ဒါမှ တန်းကိုကော်ပြီး အခြားဘက်ကို သွားလို့ရမှာပါ။ (1, 1) ကွန်နာကနေ တာစထွက်မှာဖြစ်ပြီး (14, 1) မှာ ပန်းဝင်ရမှာပါ။

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

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

တန်းကော်ပြီးပဲ့မှာ Problem Decomposition လုပ်ထားတာကို တစ်လွှာချင်း ခွဲကြည့်မယ်ဆို



ပို ၃၅

ရင် အခုလိုတွေရမှာ ဖြစ်ပါတယ်။

- တန်းကျော်ပြေးပိုင်ပွဲဝင်ခြင်း
- ▶ တန်းတစ်ခုကျော်ခြင်း
- ↳ အပေါ်တက်ခြင်း
- ↳ အောက်ဆင်းခြင်း

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

```
def ascend():
    """
    တန်းနဲ့လွှတ်တဲ့အထိ အပေါ်ဘို့ တက်သည်
    Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ
    Postcondition: တန်းထိပ်အပေါ် ဘယ်ဘက်ကွန်နာမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ
    """

    turn_left()
    while right_is_blocked():
        move()
        turn_right()

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

    turn_right()
    while front_is_clear():
```

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

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

```
def jump_over_hurdle():
```

```
    """
```

တန်းတစ်ခုကို ကျော်သည်

Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရော့ဘက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရင်းမှာ အရော့ဘက်မျက်နှာမူပြီး ရှိနေ

```
    """
```

```
ascend()
```

```
move()
```

```
descend()
```

ဒီဖန်ရှင်နဲ့ နောက်တစ်ဆင့်ကို ဆက်လက်တည်ဆောက်ရပါမယ်။ ရှေ့မှုရှင်းနေလျှင် ရှုံးတိုးမယ်၊ မရှင်းလျှင် တန်းကိုကျော်ရမယ်။ ပန်းဝင်ဖို့အတွက်ဆိုလျှင် ဒီအလုပ်ကို ဆယ့်သုံးခါလုပ်ပေးရုံပါပဲ။

```
def compete_hurdle_race():
```

```
    """
```

တန်းကျော်ပြီးပြုင့်ပဲ ပန်းဝင်သည်ထိ ယုံးပြိုင်သည်

Precondition: (၁,၁) ကွန်နာတွင် အရော့ဘက်မျက်နှာမူနေ

Postcondition: (၁၄,၁) ကွန်နာတွင် အရော့ဘက်မျက်နှာမူနေ

```
    """
```

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

```
    if front_is_clear():
```

```
        move()
```

```
    else:
```

```
        jump_over_hurdle()
```

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

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

```
from stanfordkarel import *
```

```
def main():
```

```
    """Hurdle Jumping Program"""
    compete_hurdle_race()
```

```
def ascend():
```

```
    """
```

တန်းနဲ့လွှတ်တဲ့အထိ အပေါ်သို့ တက်သည်

Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရောက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းထိပ်အပေါ် ဘယ်ဘက်ကွန်နာမှာ အရောက်မျက်နှာမူပြီး ရှိနေ

„ „ „

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

def descend():

„ „ „

တန်းအောက်သို့ ပြန်ဆင်းသည်

Precondition: တန်းထိပ်အပေါ် ညာဘက်ကွန်နာမှာ အရောက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရင်းမှာ အရောက်မျက်နှာမူပြီး ရှိနေ

„ „ „

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

def jump_over_hurdle():

„ „ „

တန်းတစ်ခုကို ကျော်သည်

Precondition: တန်းဘယ်ဘက် ခြေရင်းမှာ အရောက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရင်းမှာ အရောက်မျက်နှာမူပြီး ရှိနေ

„ „ „

```
ascend()
move()
descend()
```

def compete_hurdle_race():

„ „ „

တန်းကျော်ပြီးပြုင့်ပဲ ပန်းဝင်သည်ထိ ယဉ်ပြုင်သည်

Precondition: (၁,၁) ကွန်နာတွင် အရောက်မျက်နှာမူနေ

Postcondition: (၁၄,၁) ကွန်နာတွင် အရောက်မျက်နှာမူနေ

„ „ „

```
for i in range(13):
    if front_is_clear():
        move()
    else:
        jump_over_hurdle()
```

```

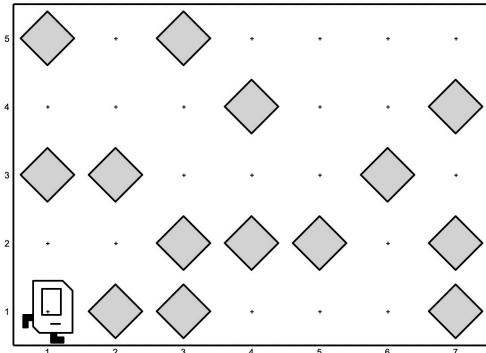
def turn_right():
    for i in range(3):
        turn_left()

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

```

အမိုက်သိမ်းတဲ့ ကားခဲလ်

အခုတစ်ခါမာ ကားခဲလ်က အမိုက်တွေရှင်းပေးရပါမယ်။ ပုံ (??) နူးနာ ကဗ္ဗာမှာ ကြိုရာကျပ်နံး (random) ပြန်ကျေနေစဲ ဘိပါတွေကို အမိုက်လို ယူဆပါ။ ကဗ္ဗာအရွယ်အစား အမျိုးမျိုးအတွက် အမိုက်ရှင်းပေးတဲ့ ပရိုဂရမ်တစ်ခု ရေးပေးရမှာပါ။

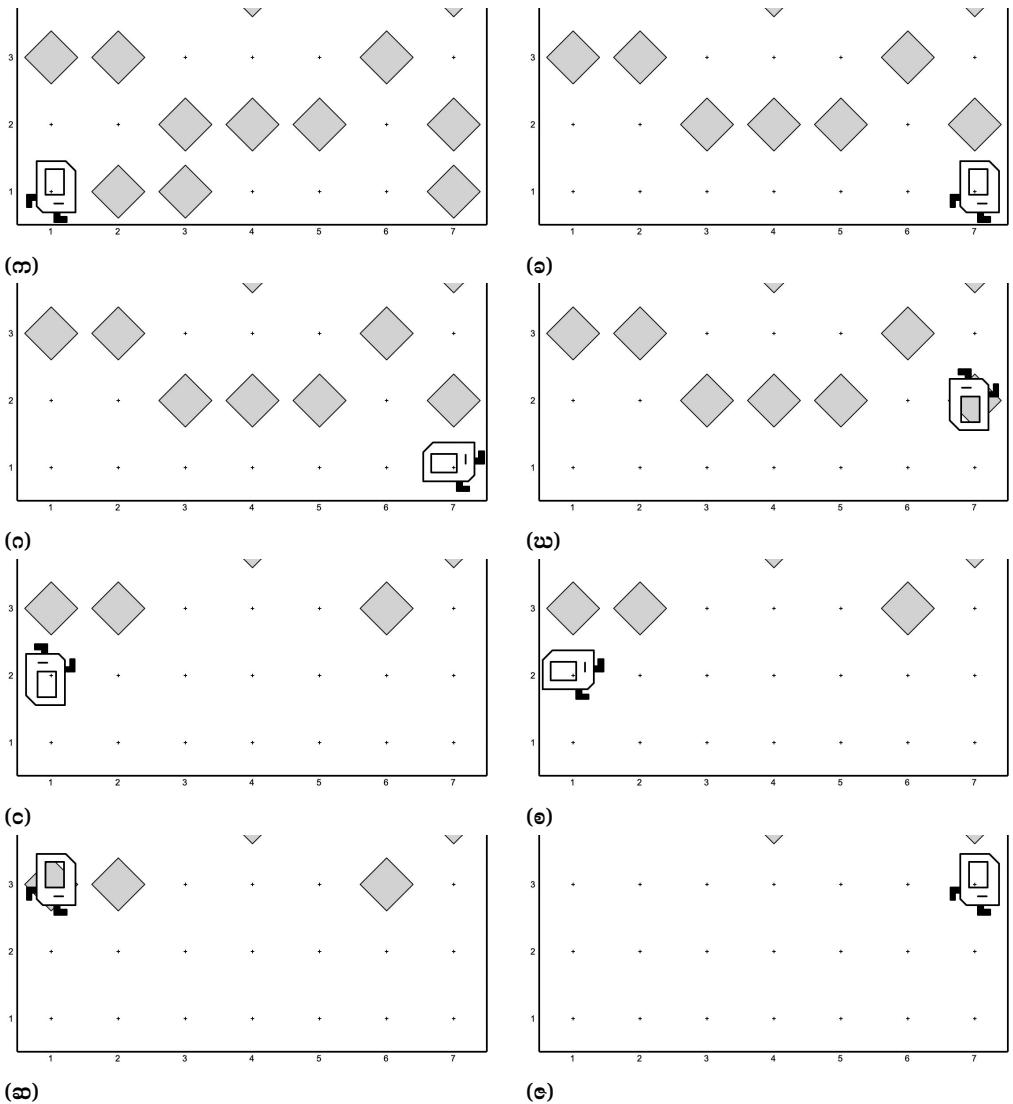


ပုံ ၃.၆

ဒီကိစ္စကို ဖြေရှင်းဖိုက နည်းလမ်းတစ်ခုတည်း ရှိတာ မဟုတ်ပါဘူး။ နည်းလမ်းတစ်ခုက ဒီလိုပါ။ (၁) လမ်းကနေ စပြီး ရှင်းတယ် 『စာမျက်နှာ ?? ပုံ (??) (က) နှင့် (ခ) တွင် ကြည့်ပါ။』 ပြီးရင် မြောက်ဘက် လှည့်ထားမယ် 『ပုံ (??) (ဂ)』။ ရှင်းနေသေးတယ်ဆိုရင် ဒုတိယလမ်းကို တက်၊ အနောက်ဘက် မျက်နှာမှူထားပါ ထားမယ် 『ပုံ (??) (ယ)』။ ဒုတိယလမ်းကို ဆက်ရှင်းပြီးတော့လည်း မြောက်ဘက်ကိုပဲ မျက်နှာမှူထားပါ မယ် 『ပုံ (??) (င)၊ (စ)』။ ရှင်းနေသေးရင် တတိယလမ်းကို ကူး၊ အရှေ့ဘက်မျက်နှာ မူထားမယ် 『ပုံ (??) (ဆ)』။ ဒီအတိုင်း တစ်လမ်းပြီးတစ်လမ်း ရှင်းသွားမှုဖြစ်တယ်။ လမ်းတစ်လမ်းရှင်းပြီး မြောက်ဘက် မျက်နှာမှူလို ပိတ်နေရင် နောက်ထပ် လမ်းမရှိတော့ဘူး။ လမ်းအားလုံး ရှင်းပြီးသွားပြီ။ စာမျက်နှာ ?? ပုံ (??) (က)၊ (ခ)၊ (ဂ) တို့ကို ကြည့်ပါ။ Problem decomposition လုပ်ကြည့်ရင် အောက်ပါအတိုင်း တော်းခြင်ပါတယ်။

- ကဗ္ဗာတစ်ခုလုံး အမိုက်တွေရှင်း (clean world)
 - ▶ လမ်းတစ်လမ်း အမိုက်ရှင်း (clean street)
 - ▶ ကွန်နာတစ်ခု အမိုက်ရှင်း (clean corner)
 - ▶ မြောက်ဘက်လှည့် (turn north)
 - ▶ လမ်းပြောင်း/နောက်တစ်လမ်းကူး (change street)

လမ်းတစ်လမ်း ရှင်းတဲ့ ကိစ္စကို ထပ်ခဲ့ကြည့်ရင် ကွန်နာတစ်ခုရှင်းတာကို တွေ့ရမှာပါ။ Bottom-up နည်းအရ အောက်ဆုံးအလွှာက ကွန်နာတစ်ခု အမိုက်ရှင်းတဲ့ ကိစ္စအတွက် ဖန်ရှင်းကို ပထမဆုံး သတ်မှတ်



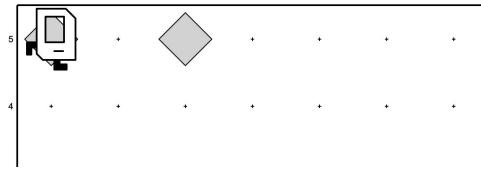
ပုံ ၃၇

ရပါမယ်။

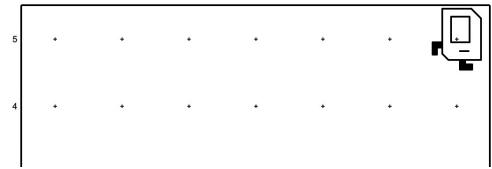
```
def clean_corner():
    """
    ကွန်နာတစ်ခုမှာ ဘိပါရှင်းပေးသည်
    Precondition: ကွန်နာတစ်ခုမှာ ကားခဲ့လဲ ရှိနေမယ်။ အများဆုံး ဘိပါတစ်ခု ရှိနိုင်တယ်။
    Postcondition: ဘိပါ မရှိတဲ့ ကွန်နာဖြစ်ရမယ်
    """

    if beepers_present():
        pick_beeper()
```

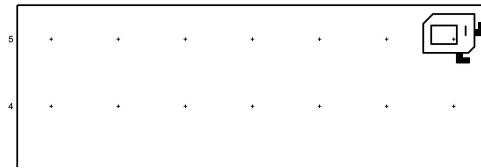
clean_corner နဲ့ လမ်းတစ်လမ်းရှင်းတဲ့ ဖန်ရှင် သတ်မှတ်ပါမယ်။



(a)



(b)



(c)

ပုံ ၃.၈

```
def clean_street():
```

```
    """

```

လမ်းတလျောက် ဘိပါအားလုံးရှင်းပေးသည်

Precondition: လမ်းအစွန်းတစ်ဖက်မှာ အခြားဘက်စွန်းကို မျက်နှာမှု၍ ရှိမယ်

Postcondition: လမ်းပေါ်ဘိပါအားလုံး ရှင်းပြီး အခြားဘက်စွန်း ရောက်နေမယ်

```
"""

```

```
    while front_is_clear():

```

```
        clean_corner()

```

```
        move()

```

```
        clean_corner()
```

မြောက်ဘက်လှည့်တဲ့ ဖန်ရှင်

```
def turn_north():

```

```
    while not_facing_north():

```

```
        turn_left()
```

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

လမ်းတစ်လမ်းရှင်းပြီး နောက်တစ်လမ်းကူးတဲ့ ဖန်ရှင်

```
def change_street():

```

```
    """

```

လမ်းတစ်လမ်း၏ အစွန်းတစ်ဖက်မှာ အပေါ်လမ်းသို့ ကူးသည်

Precondition: လမ်းအစွန်းတစ်ဖက်မှာ ကပ်လျက် နံရံကို မျက်နှာမှု၍ ရှိမယ်

Postcondition: အပေါ်တစ်လမ်းကူးပြီး အခြားဘက်စွန်းကို မျက်နှာမှု၍ ရှိမယ်

```
"""

```

```
        move()

```

```
        if right_is_blocked():

```

```
            turn_left()

```

```
        else:

```

```
            turn_right()
```

တစ်လမ်းချင်း အမိုက်အကုန် ရှင်းတဲ့ ဖန်ရှင်ကို သတ်မှတ်လို့ ရပါပြီ

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

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

# File: clean_world.py
from stanfordkarel import *

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

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

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

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

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

```

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

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

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

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

```

၃.၅ Top-down Programming

Top-down programming ဟာ အပေါ်ဆုံးအလွှာမှ စ၍ တည်ဆောက်တဲ့နည်း ဖြစ်ပါတယ်။ Bottom-up မှာလို အောက်မှ အထက် မထက်ဘဲ အထက်မှအောက် တစ်လွှာပြီးတစ်လွှာ အဆင့်ဆင့် တည်ဆောက် သွားမှာပါ။ နည်းလမ်းနှစ်ခုလုံးမှာ problem decomposition လုပ်ရမှာဖြစ်ပေမဲ့ လုပ်ငန်းစဉ် ကွားခြင်းရှိတယ်။ Top-down နည်းအတွက် problem decomposition လုပ်တဲ့အခါ bottom-up မှာ လို အောက်ဆုံးအလွှာထိ တစ်ခါတည်း ခွဲခြမ်းစိတ်ဖြာစရာမလိုဘူး။ တစ်ဆင့်ချင်းပဲ ခွဲရပါတယ်။ ပြီးခဲ့တဲ့ ဥပမာ အမိုက်သိမ်းတဲ့ ကိစ္စကို ခွဲခြမ်းစိတ်ဖြာမယ်ဆိုရင် အခါလို

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

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

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

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

ဒီနေရာမှ ပရိနဲ့ပိုစ် ကွန်ဒါရှင်တွေ တိတိကျကျ သတ်မှတ်ထားဖို့ အလွန်အရေးပါတယ်။ clean_street, turn_north, change_street ဖန်ရှင်တစ်ခုချင်း၊ ပရိနဲ့ပိုစ် ကွန်ဒါရှင်တွေ တိတိကျကျ ရှိထား မှပဲ clean_world ကို မှန်ကန်အောင် ရေးဖို့ ဖြစ်နိုင်မှာပါ။ ဥပမာအားဖြင့် clean_street ပိုစ်ကွန်ဒါရှင်ကသာ မြောက်ဘက်လှည့် အနေအထားနဲ့ အဆုံးသတ်မယ်ဆိုရင် clean_world ကို အခုလို ရေးပါ လိမ့်မယ်။

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

အပေါ်ဆုံးအလွှာ ဖြေရှင်းပြီးသွားရင် အောက်အလွှာကို တစ်ဆင့် ဆင်းပြီး ဖြေရှင်းပါတယ်။ အောက်ပါ ကိစ္စရပ်

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

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

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

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

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

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

```

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

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

```

ဒုတိယတစ်ဆင့် ဖြေရှင်းပြီးသွားပါပြီ။

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

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

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

```

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

```

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

```

def main():
    clean_world()

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

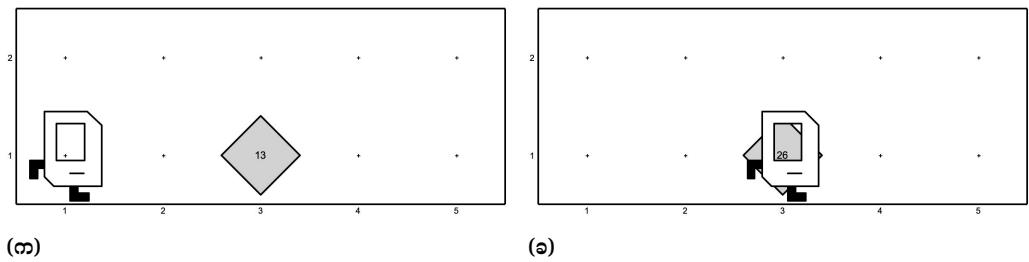
```

(turn_right ဖန်ရှင်ကို ထပ်မဖော်ပြတော့ပါ)

ခုတိယ top-down ဥပမာ (Double Beeper Pile)

အမှန်တကယ် မရှိသေးတဲ့ ဖန်ရှင်တွေကို ရှိထားပြီးဖြစ်တယ်လို ယူဆထားရတာဟာ top-down နည်းရဲ အခိုက် သော့ချက်ဖြစ်သလို ပရိုဂရမ်းမင်း စလေ့လာသူတွေအတွက် နားလည်ရ အခက်ဆုံး သဘောတရား တစ်ခုဆိုရင်လည်း မမှားဘူး။ အောက်တစ်ဆင့်အလွှာ ဖန်ရှင်တွေ ‘ဘာလုပ်ပေးလဲ’၊ ပရိုနဲ့ ပိုစ် ကွန်ဒီရှင် တွေ ဘယ်လိုဖြစ်သင့်လဲ တိတိကျကျ စိတ်ကူးပုံဖော်ထားပြီး လက်ရှုဖန်ရှင်ကို ဘယ်လိုရေးမလဲ စဉ်းစားရု တာ ဦးနှောက်ခြောက်စရာ ဖြစ်နေတာပါ။ ဥပမာ နောက်တစ်ခုလောက် ထပ်ကြည့်ရင် ပိုပြီး သဘောပေါက် သွားမယ် ထင်ပါတယ်။

အခုပ်ရိုဂရမ်ဗုံး ကားရဲ့လ်က ကွန်နာတစ်ခုမှာ စုပုံထားတဲ့ ဘိပါတွေကို နှစ်ဆဖြစ်အောင် လုပ်ပေး ရပါမယ်။ ပုံ (??) တွင်ကြည့်ပါ။ နိုင် ဆယ့်သုံးကနေ နှစ်ဆယ့်ခြောက်ခု ဖြစ်သွားပါတယ်။ ကားရဲ့လ်က ကိန်းကောင်းတွေအကြောင်း နားမလည်သလို ရော့က်တာလည်း မလုပ်တတ်ပါဘူး။ ဒါကြောင့် ဘိပါနှစ် ဆွားဖို့ အခြားနည်းလမ်းရှာရပါမယ်။



ပုံ ၃၉

ဘိပါပုံ (အစုအပ်ကို ဆိုလို) ထဲကနေ ဘိပါတစ်ခုကောက်လိုက်၊ ရှေ့ကွန်နာမှာ နှစ်ခုချထားလိုက် ဆက်တိုက် လုပ်မယ်ဆိုရင် နိုင်ဘိပါတွေ ကွန်သွားတဲ့အခါ နှစ်ဆရှိတဲ့ ဘိပါပုံတစ်ခု (ရှေ့ကွန်နာမှာ) ရ မှာပါ။ တစ်ခါတည်း ဘိပါအားလုံး ကောက်လို မရပါဘူး။ ကားရဲ့လ်က ဘယ်နှစ်ခု ကောက်ထားလဲ မမှတ်ပို တဲ့ အတွက်ကြောင့်ပါ။ တစ်ခုကောက်လိုက် ရှေ့ကွန်နာမှာ နှစ်ခုပြန်ချထားလိုက် လုပ်ရပါမယ်။

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

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

ဘိပါပုံဆိုကို သွားတာက လွှယ်တယ်

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

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

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

```
def double_beeper_pile_next():
    """
    ဘိပါပုံကို ရွှေ့ကွန်စာတွင် နှစ်ဆဖြစ်အောင် ရွှေ့ပေးသည်
    Precondition: ဘိပါပုံပေါ်စွင် အရောက် မျက်နှာမူ၍ ရှိနေမယ်
    Postcondition: နှစ်ဆဘိပါပုံပေါ်စွင် အရောက် မျက်နှာမူ၍
    """
    pass

def move_beeper_pile_next():
    """
    ဘိပါပုံကို ရွှေ့ကွန်စာသို့ ရွှေ့ပေးသည်
    Precondition: ဘိပါပုံပေါ်စွင် အနောက်ဘက်မျက်နှာမူ ရှိနေမယ်
    Postcondition: ရွှေ့ကို ရွှေ့ပြီးဘိပါပုံပေါ်စွင် အနောက်ဘက်မျက်နှာမူ ရှိနေမယ်
    """
    pass
```

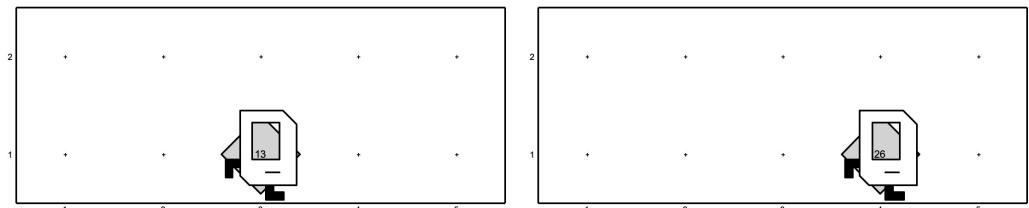
Python မှာ ဗလာဖန်ရှင်ဆိုပေမဲ့ စတိတ်မန့်တစ်ခုတော့ ပါရမယ်။ မဟုတ်ရင် ဆင်းတက်စ် အမှားဖြစ်မှပါ။ ဒါကြောင့် pass စတိတ်မန့်ကို ယာယီအနေနဲ့ သုံးလေ့ရှိတယ်။ ဒေါ်မိစတိတ်မန့် (*dummy statement*) လို့ ခေါ်ပါတယ်။ ဖန်ရှင်နှစ်ခု၊ ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေကို ပုံ (??) နဲ့ (??) မှာ တော်းပါတယ်။ နှစ်ခုလုံး မျက်နှာမူတဲ့ဘက် မပေါ်ပေါ်တာကို ကရပြုပါ။ ဘိပါပုံက မူလနေရာမဟုတ်ဘဲ ရွှေ့ကိုရောက်သွားတာကိုလည်း ကျော်ပါ။

မူလနေရာမှာပဲ ဘိပါနှစ်ဆဖြစ်အောင် လုပ်တဲ့ ဖန်ရှင်က ဒီလိုဖြစ်ပါမယ်

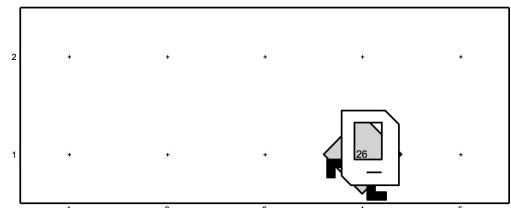
```
def double_beeper_pile():
    """
    ဘိပါပုံကို မူလနေရာတွင်ပင် နှစ်ဆတိုးပေးသည်
    Precondition: ဘိပါပုံပေါ်စွင် အရောက်မျက်နှာမူလျက် ရှိနေ
    Postcondition: နှစ်ဆတိုးပြီး ဘိပါပုံစွင် မူလအတိုင်း အရောက်မျက်နှာမူလျက် ရှိနေ
    """
    double_beeper_pile_next()
    turn_left()
    turn_left()
    move_beeper_pile_next()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

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

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

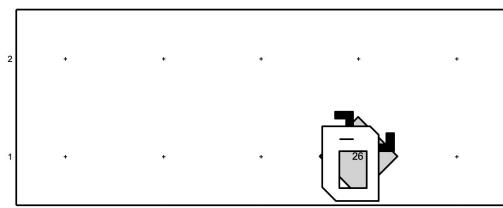


(က)

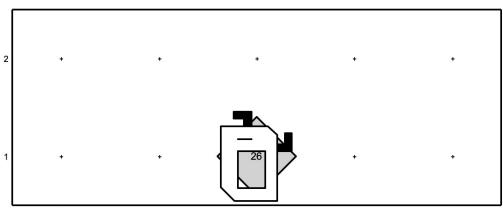


(ခ)

ပို ၃.၀၀



(က)



(ခ)

ပို ၃.၀၀

```
def double_beeper_pile():
    """
    ဘိပိုပိုကို မူလနေရာတွင်ပင် နှစ်ဆတိုးပေးသည်
    Precondition: ဘိပိုပေါ်တွင် အရှေ့ဘက်မျက်နှာမူလျက် ရှိစေ
    Postcondition: နှစ်ဆတိုးပြီး ဘိပိုပိုတွင် မူလအတိုင်း အရှေ့ဘက်မျက်နှာမူလျက် ရှိစေ
    """
    double_beeper_pile_next()
    turn_around()
    move_beeper_pile_next()
    turn_around()
```

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

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

```
def double_beeper_pile_next():
    """
    ဘိပိုပိုကို ရေးကွန်နာတွင် နှစ်ဆဖြစ်အောင် ရွှေ့ပေးသည်
    Precondition: ဘိပိုပေါ်တွင် အရှေ့ဘက် မျက်နှာမူလျက် ရှိနေမယ်
    Postcondition: နှစ်ဆဘိပိုပေါ်တွင် အရှေ့ဘက် မျက်နှာမူလျက် ရှိနေမယ်
```

```

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

    move()

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

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

    move()

# File: double_beeper_pile.py
from stanfordkarel import *

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

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

```

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

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

    move()

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

    move()

def turn_around():
    turn_left()
    turn_left()

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

အခန်း ၄

ကားရဲလိနှင့် ရီကားဆုံးဖြစ်ပါသည်။

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

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

၄.၁ ရီကားဆုံးဖြစ်ပါသည့် ဘယ်လို အလုပ်လုပ်လဲ

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

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

ဒီဖန်ရှင်ကို ခေါ်လိုက်ရင် ဘာဆက်ဖြစ်မလဲဆိုတာ စိတ်ဝင်စားစရာပါ။ ဖန်ရှင်မစတင်မဲ့ အနေအထား ကို ပဲ (??) မှာ ကြည့်ပါ။ ဖန်ရှင်ကို ကန်းမြို့ စခေါ်လိုက်တဲ့လို ဖြစ်ပါမယ်။

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



ပို ၄.၁

ဖန်ရှင် စတင် လုပ်ဆောင်ပါမယ်။ ရွှေမှာရှင်းနေတဲ့ အတွက် if ဘလောက်ကို လုပ်မှာပါ

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

ဘိပါချာ ရွှေဗိုး [ပု (??) (??)] ကြည့်ပါ။ ပြီးရင် သူကိုယ်သူ ပြန်ခေါ်ထားတယ်။ ဒါဟာ ပထမဆုံး တစ် ကြိမ်ပါ။ ဖန်ရှင်ခေါ်ရင် ဖြစ်မြေအတိုင်းပဲ ဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို ဆောက်ရွှေက်တာပေါ့။ ဒီတော့ make_beeper_row ဖန်ရှင်ဘလောက်ကိုပဲ တစ်ခါထပ်လုပ်မှာပါ။ ရွှေမှာ ရှင်းနေတဲ့အတွက် if အပိုင်း ကို လုပ်တယ်

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

ဘိပါချာ ရွှေဗိုး [ပု (??) (??)] ပြီး သူကိုယ်သူ ပြန်ခေါ်ထဲတဲ့ကိစ္စ တစ်ခါထပ်ဖြစ်ပြန်တယ်။ ဒါနဲ့ဆို နှစ်ကြိမ်။ ဖန်ရှင်ဘလောက် အလုပ် ပြန်လုပ်မယ်။ ရွှေမှာရှင်းတယ်၊ if ကိုပဲ ထပ်လုပ်

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

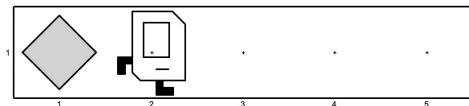
ပု (??) (??) နေရာရောက်ပြီး သူကိုယ်သူ ထပ်ခေါ်ထားပြန်တယ်။ သုံးကြိမ်ရှိပြီ။ ဒီတစ်ခါလည်း if အပိုင်းပဲ ထပ်လုပ်

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

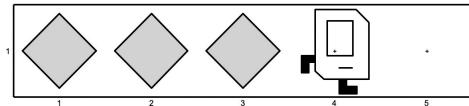
ရွှေဗိုးပြီးရင် နံရံပိတ်နေပြီ [ပု (??) (??)]။ သူကိုယ်သူ ခေါ်တယ်။ ရွှေမှာ ပိတ်နေတဲ့အတွက် else အပိုင်းကို လုပ်မှာပါ။

```
put_beeper()
```

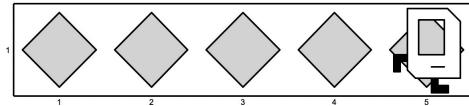
သူကိုယ်သူ ပြန်ခေါ်တဲ့ကိစ္စ ထပ်မဖြစ်တော့ဘူး။ ဒီမှာပဲ ပြီးဆုံးသားတယ်။ ရီကားဆစ်ဖုံး ဖန်ရှင် အလုပ် လုပ်ပဲ အခြေခံသဘောတရားက ဒါပါပဲ။ loop တွေ မသုံးဘဲ ပြန်ကျော်နေတဲ့ သဘောကို ရီကားဆစ်ဖုံး ဖန်ရှင်မှာ တွေ့ရပါတယ်။ ဖန်ရှင်က သူကိုယ်သူ (သို့ ကိုယ့်ကိုကိုယ်) ပြန်ခေါ်ဘက် recursive call လို



(က) ပထမ တစ်ကျိုးပြီး



(က) တတိယ တစ်ကျွဲ့ပြီး



(c) နောက်ဆုံး putBeeper လုပ်ပြီး



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

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

make_beeper_row()
```

ဝန်ဆောင်ရေးဝန်ကြီးခွဲ

ဦး ဖန်ရှင်ကောလ် စတင်တာဖြစ်ပေါ်တာကို ဖော်ပြတယ်။ ရီကားဆစ်ဖိုကောလ် မဟုတ်သေးဘူး။ မြား အပြာက ရီကားဆစ်ဖိုကောလ်ကြောင့် ဖန်ရှင်အစ ပြန်ရောက်သွားတာကို ပြတယ်။ ပထမနဲ့ ဒုတိယ ရီကားဆစ်ဖိုကောလ် နှစ်ခုအတွက်ပြထားတာပါ။ ရွှေ့က ဥပမာအတက် မြှားအပြာ လေးခု ရို့ရမှာပါ (ရီကားဆစ်ဖိုကောလ် လေးကြိမ်အတွက်)။ မြားအပြာ နောက်ထပ် နှစ်ခုရှိတယ် မှတ်ယူပါ (ပုံမှာထပ်ထည့်ရင် ကြပ်ညပါး ကြည့်ရရှိပဲလို မဆဲပြတာ)။ လေးကြိမ်မြောက်မှာ ရီကားဆစ်ဖိုကောလ် ထပ်မဖြစ်တော့ဘူး (if အပိုင်းကို မလုပ်တော့တဲ့အတွက်)။ ဘိပါချုပြုး ဖန်ရှင်ကောလ် ခဲ့တဲ့နေရာကို ပြန်ရောက် သွားမယ် (မြှားအနဲ့)။ ဘယ်လို ပြန်ရောက်သွားတာလ ဆက်ကြည့်ရအောင်။

ፈኞስና የሚገኘውን ስምዎች

နောက်ဆုံး ရိုကားဆစ်ဖြင့်ကောလ်ကနဲ မူလ ဖန်ရှင်ပေါ်ခဲ့တဲ့ နေရာကို ဘယ်လိုပြန်ရောက်သွားတာလဲ။ ဒီ ကိစ္စနားလည်ဖို့ ဖန်ရှင် return ပြန်ခြင်းအကြောင်း အရင်ကြည့်ရပါမယ်။ ဖန်ရှင်ကောလ် လုပ်ဆောင် တဲ့အခါ အဲဖော်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘလောက်ဆီကို ခုံနကျော် ရောက်ရှိသွားမှာပါ။ ဖန်ရှင်ဘလောက်ကို လုပ်ဆောင်ပြီး ပေါ်ခဲ့တဲ့ နေရာကို ပြန်လည်ရောက်ရှိသွားမှာ ဖြစ်တယ်။ ဒီဖြစ်စဉ်ကို ဖန်ရှင် return ပြန် တယ်လို့ ပြောပါတယ်။

```

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

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

```

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

```

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

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

```

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

```

def main():
    do_tricks()
    move()

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

def put_two():
    put_beeper()
    put_beeper()

```



`put_two` ပြီးသွားတဲ့အခါ `do_tricks` ထဲ ပြန်ရောက်သွားမယ်။ `turn_left`, `move` ဆက်လုပ် ပြီး `do_tricks` ခေါ်ခဲ့တဲ့နေရာ `main` ထဲ ပြန်ရောက်သွားမယ်။ နောက်ဆုံး `main` ထဲက `move` ကို ဆက်လုပ်ပါတယ်။

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

ရိုကားဆစ်စ် ဖန်ရှင်ကောလ်တွေ ဘယ်လို `return` ပြန်လဲ။ ရှေ့ကလို မြှားတွေနဲ့ ဆွဲပြလို့ ရပေမဲ့ ကြည့်ရတဲ့ ရှုပ်ရှက်ခတ်နေမှာပါ။ အခုလို မြင်ကြည့်ရင် ပိုရှင်းပါတယ်။

```
# initial call
make_beeper_row()
    # 1st recur
    make_beeper_row()
        # 2nd recur
        make_beeper_row()
            # 3rd recur
            make_beeper_row()
                # 4th recur
                make_beeper_row()
```

 မြှားအနက်တွေက ဖန်ရှင်ကောလ် တစ်ဆင့်ပြီးတစ်ဆင့် ဖြစ်တာကို ပြတာပါ။ အထက်မှအောက် အစီအစဉ်အတိုင်း ဖြစ်ပါတယ်။ လေးကြိမ်မြှာက်မှာ နောက်ထပ် ရိုကားဆစ်စ်ကောလ် ထပ်မဖြစ်တော့ဘဲ နောက်ဆုံး ရိုကားဆစ်စ်ကောလ်က အရင်ဆုံး `return` စပြန်ပါတယ် (အောက်ဆုံး မြှားအနီနဲ့ ပြထား)။ ဒီအခါ တတိယ ရိုကားဆစ်စ်ကောလ်ကို ပြန်ရောက်သွားမှာပါ။ ဒီအတိုင်း တစ်ဆင့်ပြီးတစ်ဆင့် အထက်ကို `return` ပြန်ပြီး နောက်ဆုံးမှာ ပထမဆုံး ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားမှာပါ။ (အခုပြထားတာကို တကယ့် Python ကုဒ် အနေနဲ့ မယူဆရပါ၊ ဖြစ်စဉ် နားလည်အောင် ပြခြင်းသာဖြစ်ပါတယ်)။ နဲ့ပြင်ထားတဲ့ `make_beeper_row` ဗားရှင်းမှာ `return` ပြန်တဲ့ ဖြစ်စဉ်ကို ကြည့်ရအောင်။

```
| make_beeper_row()
```

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

`if` အပိုင်း ရိုကားဆစ်စ်ကောလ်ပြီး `turn_left` နဲ့ `move` ထပ်ဖြည့်ထားတာ။ ရိုကားဆစ်စ်ကောလ် `return` ပြန်လာပြီးမှ ဒီနှစ်ခု ဆက်လုပ်မှာပါ။ နောက်ဆုံး ရိုကားဆစ်စ်ကောလ်က `return` စဖြစ်တယ်။ ဒီ

တော့မှ တတိယအောက် turn_left နဲ့ move ကို လုပ်ဆောင်မှာပါ။ ပြီးမှ တတိယကောလ် return ပြန်တယ်။ ဒီအတိုင်း အထက်ကို တက်သွားပြီး ကျွန်ုန်းနေသေးတဲ့ စတိတ်မနဲ့တွေကို လုပ်ဆောင်ပါတယ်။ ပထမဆုံးကောလ်အောက် ကျွန်ုန်းနေတာတွေ နောက်ဆုံးကျွမ်းပြီးမှာပါ။

```
# initial call
make_beeper_row()
    # 1st recur
    make_beeper_row()
    turn_left()
    move()
        # 2nd recur
        make_beeper_row()
        turn_left()
        move()
            # 3rd recur
            make_beeper_row()
            turn_left()
            move()
                # 4th recur
                make_beeper_row()
```



ရိုကားဆစ်ဖောလ် နှစ်ခါပဲ ဖြစ်မယ်ဆိုရင် အောက်ပါအတိုင်း မြင်ကြည့်လို့ ရပါတယ်။ မြှားအပြာနှစ်ခုက ရိုကားဆစ်ဖောလ်ဖြစ်တာ။ ဒုတိယကောလ်က ဘိပါချုပြီး (else အပိုင်း) အရင် return မယ်။ အထက်ကို ညွှန်တဲ့ မြှားအနီးကို ကြည့်ပါ။ ဘယ်လှည့်၊ ရွှေ့တိုးပြီး ပထမ ကောလ်က နောက်မှ initial call လုပ်ခဲ့တဲ့ဆို ပြန်ရောက်တာ။ အောက်ကိုညွှန်တဲ့ မြှားအနီးကို ကြည့်ပါ။

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

# initial call
make_beeper_row()
```



ရိုကားဆစ်ဖောလ်အကြောင်း လေ့လာတဲ့အခါ ဘိုင်နာအများစုံ ကြော်ကြည်လာက နားလည်ဖို့ အတွက် အခက်အခဲဆုံးတစ်ခုက return ပြန်တဲ့ သဘောတရားပါပဲ။ များများစဉ်းစား၊ များများလေ့ကျင့်ရင် ဒီအခက်အခဲ ကျော်ဖြတ်နိုင်မှာပါ။ if...else အပြီးမှာ put_beeper လေးတစ်ခဲ့ပဲ ထပ်ဖြည့်လိုက်ရင် ဘယ်လိုဖြစ်မလဲ။

```
# File: make_beeper_row2.py
def make_beeper_row():
```

```

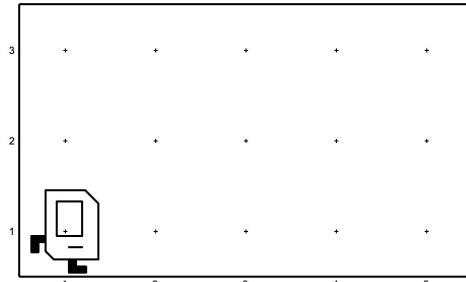
if front_is_clear():
    put_beeper()
    move()
    make_beeper_row()
    turn_left()
    move()

else:
    put_beeper()
    put_beeper()

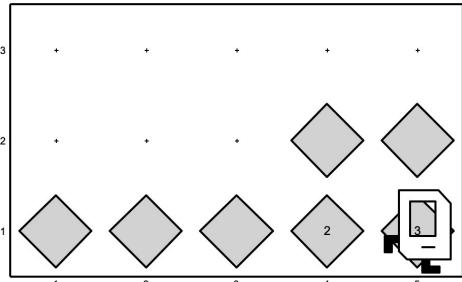
# initial call
make_beeper_row()

```

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



(က)



(ခ)

ပုံ ၄.၃

၄.၂ ရီကားဆစ်ဖုနည်းဖြင့် ပုံစံဖြေရှင်းခြင်း

ရှေ့စက်ရှင်မှာ လေ့လာခဲ့တာက ရီကားဆစ်ဖန်ရှင် ဘယ်လို အလုပ်လုပ်လဲပဲ ရိုပါသေးတယ်။ တစ်နည်းအားဖြင့် မဏေနစ်မဲ့ (mechanism) ကို လေ့လာတာပါ။ အခုက်ခုတ်ခါ ရီကားဆစ်ဖန်ရှင်တွေနဲ့ ပုံစံတွေ ဘယ်လိုဖြေရှင်းမလဲ ဆက်လက်လေ့လာပါမယ်။ ‘ရီကားဆစ်ဖန်ရှင်တွေ’၊ ‘ရီကားဆစ်စ် စဉ်းစားခြင်း’ (thinking recursively) သို့မဟုတ် ‘ရီကားဆစ်ဖန်ရှင်းဖြင့် ပုံစံဖြေရှင်းခြင်း’ (solving problems recursively) ကို လေ့လာမှာပါ။

ရီကားဆစ် စဉ်းစားခြင်း ဥပမာ (၁)

ကွန်္ဘာမှုရှိတဲ့ ဘိပါတွေအားလုံး ကောက်မယ်ဆိုပါစို့။ ဘိပါတစ်ခုနဲ့ အထက်ရှိရှိတယ်။ ဘိပါမရှိတာလည်း ဖြစ်နိုင်တယ် ယူဆပါ။ ဒီအတွက် ရီကားဆစ်ဖန်ရှင် သတ်မှတ်ပါမယ်။

```

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

```

ဖြေရှင်းမဲ့ကိစ္စတစ်ခုကို ဂင်းကိုယ်ဝိုင်နဲ့ ပုံပန်းသဏ္ဌာန်တူပြီး အရွယ်အစားအားဖြင့် တစ်ဆင့်ထက် တစ်ဆင့် သေးငယ်တဲ့ ကိစ္စတွေအဖြစ် ခဲ့ခြမ်းကြည့်ပါတယ်။ ဥပမာ ဘိပါဝါးခုရှိတဲ့ ကိစ္စကို ဘိပါလေးခဲ့သုံးခဲ့ နှစ်ခု နဲ့ တစ်ခု ရှိတဲ့ ကိစ္စတွေအဖြစ် ခဲ့ပြီး ပြင်ကြည့်ရမှာပါ။ ဒီကိစ္စမှာ ဘိပါအရေအတွက်ဟာ အရွယ်အစားပဲ။ လေးခုရှိတဲ့ ကိစ္စဟာ ငါးခုရှိတဲ့ကိစ္စထက် အရွယ်အားဖြင့် တစ်ဆင့်ငယ်တာပေါ့။ ဘိပါမရှိတာလည်း ဖြစ်နိုင်တော့ သူည်ဘိပါဟာ အငယ်ဆုံးဖြစ်တယ်လို့ ယူဆနိုင်တယ်။

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

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

လက်ရှိဖုန်ရှင်ဟာ $n - 1$ ဘိပါကို ကောက်နိုင်ပြီးသားလို့ ယူဆထားတယ်။ n ဘိပါရှိရင် ဘိပါတစ်ခု ကောက်လိုက်ရင် $n - 1$ ဘိပါဖြစ်သွားမယ်။ ကျွန်ုန်နေတဲ့ $n - 1$ ဘိပါကောက်ဖို့ လက်ရှိဖုန်ရှင်ကိုပဲ ပြန်ခေါ်လိုက်မှာပေါ့။

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

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

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

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

ရှိကားဆစ်ဖန်ရှင် မှန်မမှန် စ်ဆေးခြင်း

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

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

if ဘလောက် မလုပ်ဆောင်ဘဲ return ဖြစ်သွားမှာပါ (ရီကားဆစ်စံကောလ် မဖြစ်လိုက်ဘူး)။ သူည့်ဟို ပါအတွက် ဖြစ်သင့်တဲ့အတိုင်း ဖြစ်ပါတယ်။ ဘိပါတစ်ခုပဲ ရှိရင်ရော ဘယ်လိုဖြစ်မလဲ။ ဘိပါကောက်တယ် သူညာဘိပါဖြစ်သွားပြီး ရီကားဆစ်စံကောလ် ဖြစ်မယ်။ တစ်ကြိမ်ပဲ ဖြစ်မယ်။ if ဘလောက် အလုပ်မလုပ် ဘဲ return ပြန်မယ်။

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

~~မျှော်လုပ်~~ ရီကားဆစ်စံကောလ် နှစ်ခါဖြစ်ပြီးမှ return ပြန်မှာပါ။

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

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

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

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

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

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

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

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

ရိကားဆုံး စဉ်းစားခြင်း ဥပမာ (၂)

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

ဖြေရှင်းမဲ့ ကိစ္စတွေကို ရှင်းကိုယ်တိုင်းနဲ့ သဏ္ဌာန်တူပြီး အရွယ်အစားအားဖြင့် တစ်ဆင့်ထက်တစ်ဆင့် သေး ယော်တဲ့ ကိစ္စတွေအဖြစ် ခွဲခြမ်းကြည့်ရမှာပါ။

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

`clean_street` ဖန်ရှင်ဟာ လက်ရှိဖြေရှင်းမဲ့ အရွယ်အစားထက် တစ်ဆင့်ယော်တဲ့ ကိစ္စကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူရပါမယ်။

အခါကိစ္စအတွက် ကွန်နာငါးခဲ့ ရှင်းမယ်ဆိုရင် `clean_street` ဖန်ရှင်ဟာ ကွန်နာလေးခုနဲ့ လမ်းကို ရှင်းနိုင်တယ်လို့ ယူဆရမယ်။ n ကွန်နာရှိတဲ့ လမ်းအတွက် $n - 1$ ကွန်နာကို ရှင်းနိုင်ပြီးသားလို့ ယူဆရမှာ ဖြစ်ပါတယ်။

လက်ရှိအရွယ်အစားကို သုထက်တစ်ဆင့်ယော်တဲ့ ကိစ္စဖြစ်သွားအောင် ဘာလုပ်ရမလဲ စဉ်းစားရတယ်။ ကွန်နာငါးခဲ့ လမ်းကိုရှင်းတဲ့ အခါကိစ္စတဲ့ ကွန်နာ လေးခဲ့ ရှင်းစရာကျွန်အောင် ဘာလုပ်မလဲ။ ယော်သူယျာပြာရင် n ကွန်နာဆိုရင် $n - 1$ ကွန်နာ ရှင်းဖို့လို့တော့အောင် ဘာလုပ်မလဲ။

လမ်းတစ်လမ်းရှင်းတဲ့ အခါ လမ်းအစ သို့မဟုတ် လမ်းအခုံးမှာ အခြားဘက်စွန်းကို မျက်နှာမူထားတယ်။ ရှုံးတစ်ကွန်နာ ရွှေလိုက်ရင် ရှင်းစရာ ကွန်နာတစ်ခု လျော့သွားမှာပါ။

တစ်ဆင့်ယော်တဲ့ ကိစ္စ $n - 1$ ကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူပြီး လက်ရှိကစ္စ n ကို ဘယ်လို့ ဖြေရှင်းမလဲ စဉ်းစားရပါမယ်။

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

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

```

အသေးငယ်ဆုံး ကိစ္စကို ခွင့်ချက်အနေနဲ့ စဉ်းစားရမယ်။ ကွန်နာတစ်ခုဟာ အသေးငယ်ဆုံးကိစ္စ ဖြစ်တယ်။ ဒီကိစ္စကို ဘယ်လို့ ဖြေရှင်းမလဲ။

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

$n = 1$ ဖြစ်နေပြီ။

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

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

ဒီအတိုင်းဆက်စစ်သွားရင် တစ်ခုထက်ပိုတဲ့ ကွန်နာတွေအတွက်လည်း မှန်အောင် အလုပ်လုပ်နေမယ်ဆိုတာ သက်သေပြနိုင်ပါတယ်။ စာရေးသူ အတော့အကြံအရ အသေးဆုံးနဲ့ သူထက်ကြီးတာ နှစ်ခုဆုံးခဲ့လောက်ထိ မှန်တယ်ဆိုရင် နောက်ဟာတွေအတွက် မှားစရာ အကြောင်းမရှိတော့ဘူး။

ရိုကားဆစ်စောင်းနဲ့ လမ်းတစ်လမ်း ရှင်းလို့ရပါပြီ။ ကားရဲလ်ကမ္ဘာတစ်ခုလုံး ရှင်းဖို့ ရိုကားဆစ်စောင်းပြီး စဉ်းစားပါမယ်။

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

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

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

- အသေးငယ်ဆုံး ကိစ္စကို ချွင်းချက်အနေနဲ့ စဉ်းစားရမယ်။ လမ်းတစ်လမ်းပဲရှိတာက အသေးငယ်ဆုံး။ လမ်းတစ်လမ်းရှင်းပြီး မြောက်ဘက်လုညွှေအပြီး ပိတ်နေပြီးဆိုရင်တော့ နောက်ထပ် ဆက်ပြီး ရှင်းစရာ လမ်းမရှိတော့ဘူး။ အပေါ်ဆင့်က လမ်းကူးတာနဲ့ ကျိုန်တဲ့လမ်းတွေကို ရှင်းတဲ့ကိစ္စကို မြောက်ဘက်လုညွှေပြီးတဲ့အခါ ရှေ့မှာရှင်းနေမှု လုပ်ရမှာပါ။ ဒီတော့ အခုလို

```

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

```

ဖြစ်ရမယ်။

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

```

# File: clean_world_recur1.py
from stanfordkarel import *

def main():
    clean_world()

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

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

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

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

```

```

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

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

# File: checkerboard_recur.py
from stanfordkarel import *

def main():
    mk_checkerboard()

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

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

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

def switch_row():

```

```
move()
if right_is_blocked():
    turn_left()
else:
    turn_right()

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

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

if __name__ == "__main__":
    run_karel_program("4x5")
```

အခန်း ၅

ဒေတာများနှင့် ဖန်ရှင်များ

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

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

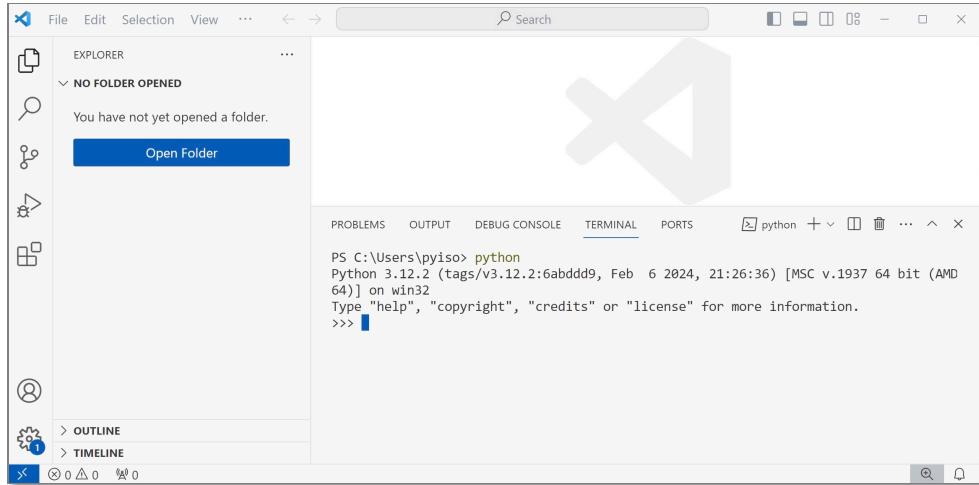
ဒေတာ အမျိုးအစား အများအပြားရှိပေမဲ့ အခြေခံအကျခုံးက ကိုန်းဂဏန်းတွေပါ။ ကွန်ပျိုးတာတွေ ကို စတင်တိထွင်ဖို့ ကြိုးစားလာကြတဲ့ အဓိက အကြောင်းအရင်းကလည်း ဂဏန်းသချာ အတွက်အချက် တွေကို လုပ်ဆောင်ရာမှာ လူတွေကို အကျအညီ ပေးဖို့အတွက်ပဲလို့ ဆိုရိုင်ပါတယ်။ ဒါအပြင် စာသား၊ ရုပ်သံ စတဲ့ အခြားဒေတာ အမျိုးအစားတွေကို ကွန်ပျိုးတာထဲမှာ ဖော်ပြသိမ်းဆည်းထားဖို့အတွက် ကိုန်းဂဏန်းတွေကိုပဲ အသုံးပြုထားတယ်ဆိုတာ နောက်ပိုင်းမှာ နားလည်သိမြင် လာပါလိမ့်မယ်။ ဒါကြောင့်လည်း ယနေ့ခေတ် ကွန်ပျိုးတာတွေကို အစိုးရှိခံတယ်။ ကိုန်းဂဏန်းကို အခြေခံပြီး အလုပ်လုပ်တဲ့ ကွန်ပျိုးတာတွေပေါ့။

၅.၁ ကိန်းဂဏန်းများ

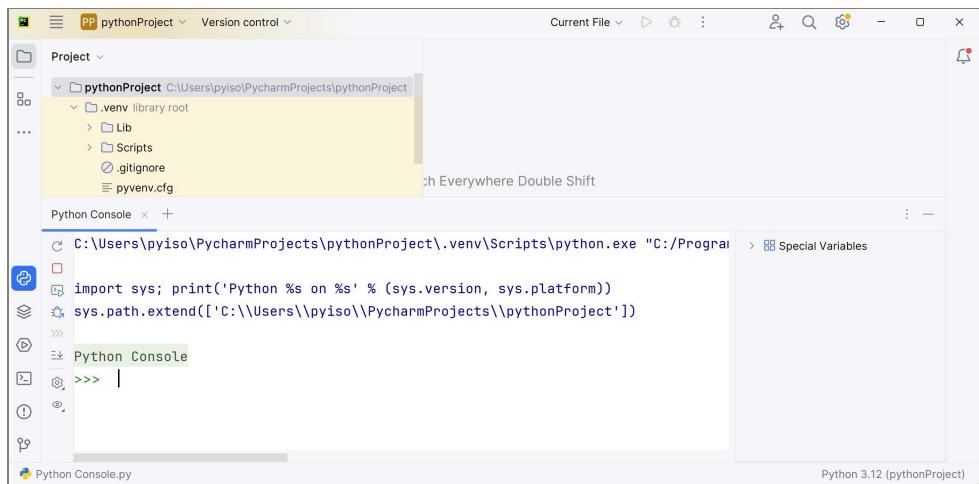
Python အင်တာပရက်တာနဲ့ Python ကုဒ်တွေကို run လိုရတဲ့နည်း နှစ်ခုရှိပါတယ်။ တစ်နည်းက ကုဒ်တွေကို .py ဖိုင်နဲ့သိမ်းပြီး python ကွန်မန်းနဲ့ run တာပါ။ ဒါက ကားရဲလ် ပရီဂရမ်တွေမှာ သုံးခဲ့တဲ့နည်း၊ နောက်တစ်နည်းကတော့ ကုဒ်တစ်ကြောင်းချင်း ရိုက်ထည့်ပြီး run တဲ့နည်းပါ။ ဒီနည်းလမ်းက interactive mode နဲ့ အင်တာပရက်တာကို အသုံးပြုတာပါ။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

ကြည့်ရတာ လွယ်တဲ့အတွက်ကြောင့်ပါ။

ဝင်ဒီး Command Prompt သိမဟတ် မက်ခံအိအက်စ Terminal မှ python ကွန်မန်း run ပြီး interactive mode ကို ဝင်နိုင်ပါတယ်။ VS Code မှာပဲ သုံးချင်လည်း ရတယ်။ View မိန္ဒာမှု Terminal ကိုဖွံ့ဖြိုး (Ctrl + ` ရှေ့ကတ်ကီးနဲ့ ဖွံ့ဖြိုးလိုလည်းရတယ်) ပြီး python ကွန်းမန်း run ရုံး။ PyCharm မှာဆိုရင် Python Console အိုင်ကွန်နှုပ်ပြီး ဝင်ရပါမယ်။ ပုံ (၁.၁)၊ (၁.၂) တွင်ကြည့်ပါ။



ပုံ ၉.၁ VS Code Python Console



ပုံ ၉.၂ PyCharm Python Console

`2 + 5` ထည့်ပြီး Enter ကီးနှုပ်ပါ။ အင်တာပရက်တာက ရလဒ် 7 နဲ့ တုံ့ပြန် လုပ်ဆောင်ပေးပါလိမ့်မယ်။ ဒီလို အင်တာပရက်တာက လုပ်ဆောင်ပေးတာကို `evaluate` လုပ်တယ်လို့ ပြောတယ်။

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

အောက်ပါအတိုင်း တစ်ခုပြီးတစ်ခု ဆက်လက်စမ်းသပ်ကြည့်ပါ။

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

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

Values and Types

တန်ဖိုးတိုင်းဟာ တိုက်ပ် (type) တန်မျိုးမျိုးမှာ ပါဝင်ပါတယ်။ -3, 0, 2 စတဲ့ တန်ဖိုးတွေဟာ int (integer ရဲ့ အတိုက်) တိုက်ပ်ဖြစ်ပြီး -3.0, 0.1, 3.3333 စတာတွေက float တိုက်ပ် ဖြစ်ပါတယ်။ ဒဿ်မကိန်းတွေကို ဂွန်ပျူးတာနဲ့ ဖော်ပြန့် floating point လိုပေါ်တဲ့ နည်းစနစ်ကို အသုံးပြုတယ်။ ဒဿ်မကိန်း အတွက်အချက်တွေကိုလည်း ဒီနည်းစနစ်ကို အခြေခံပြီး ဂွန်ပျူးတာက လုပ်ဆောင်တာပါ။ ဒါကြောင့် floating point ဟာ ဒဿ်မကိန်းတွေကို ဖော်ပြန့်နဲ့ ဒဿ်မကိန်း အတွက်အချက်တွေ လုပ်ဆောင်ဖို့ တိတွင်ထားတဲ့ နည်းစနစ်တစ်ခုလို ဆိုရိုင်ပါတယ်။ ဒီစနစ်ကို အခြေခံထားတဲ့ ဒဿ်မကိန်း တွေကို programming language တွေမှာ float တိုက်ပ်လို ခေါ်တာပါ။

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

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

ကွားချက်က မဆိုစလောက် သေးငယ်တယ် ဆိုပေမဲ့ ဒီအချက်ကို ပရှုပြုမိဖို့ လိုပါတယ်။ Floating point စနစ်ဟာ လုံးဝကြီးတိုကျဖို့ မလိုအပ်တဲ့ (တနည်းအားဖြင့် မဆိုစလောက် သေးငယ်တဲ့ ကွားချက်ကို လက်ခံနိုင်တဲ့) ကိန်းကဏ္ဍးအတွက်အချက် ကိစ္စတွေအတွက် ရည်ရွယ်တာပါ။ သိပ္ပါနဲ့ နည်းပညာဆိုင်ရာ တိုင်းတာ တွေက်ချက်မှုတွေအတွက် အသုံးပြုလေ့ရှိတယ်။ ဒဿ်မကိန်းတွေ လုံးဝအတိအကျ ဖြစ်ဖို့ လိုအပ်တဲ့ ကိစ္စမျိုးတွေ (ဥပမာအားဖြင့် ငွေကြေးကိစ္စ အတွက်အချက်) မှာ အသုံးမပြုသင့်ပါဘူး။ ဆယ်ပြားကို 0.1 နဲ့ဖော်ပြရင် ဆယ်ပြားစေ ဆယ်စွေားတော့ တစ်ကျပ် ဖြစ်ကို ဖြစ်သင့်ပြီး 0.9999999999999999 မဖြစ်သင့်ဘူး။ ဒီလိုကိစ္စမျိုးတွေအတွက် Python မှာ Decimal ကို အသုံးပြုနိုင်ပါတယ်။ လောလောဆယ် ကိန်းကဏ္ဍးတွေနဲ့ ပါတ်သက်ပြီး စကတည်းက သိထားသင့်တာတချို့ကို ကြိုးကြော် အကြောင်း မကြေခင် လေ့လာမှုပါ။

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

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

မှာတော့ အထူးပြုတဲ့ ကုန်ပျိုတာစနစ်ပေါ် မူတည်ပြီး ကန့်သတ်ချက်ရှိမှာပါ။ ဒီစာရေးနေတဲ့ ကုန်ပျိုတာ မှာ အောက်ပါကိန်းပြည့်တွေကို အသာလေး တွက်ချက်ပေးနိုင်ပါတယ်။ ဒို့ထက်အများကြီးကြီးတဲ့ဟာတွေ ကိုလည်း ကိုင်တယ်တွက်ချက်ရှိနိုင်အေးမှာပါ။ တော်ရုံကိစ္စတွေအတွက် ကုန်ပျိုတာစနစ်တစ်ခုခဲ့ လက်တွေ ကန့်သတ်ချက်ကို ကျော်လိုက်သူးဖို့ မလုပ်ပါဘူး။

```
>>> 90000000000000000000000000000000 + 5000000000  
9000000000000000000000000000000050000000000  
>>> -90000000000000000000000000000000 + 1  
-899999999999999999999999999999999999999
```

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

အသုတေသနများကို e အမှတ်အသားနဲ့ရေးထားတာပါ။ အကွဲရာ e နောက် လိုက်တဲ့ ကိန်းပြည့်ကဏ္ဍးကို 10 ထပ်ကိန်းလို့ မှတ်ယူရပါတယ်။ e3 က 1×10^3 , e-3 က 1×10^{-3} ပါ။ အကွဲရာ E အကြီးနဲ့လည်း ရေး နိုင်တယ်။ ထပ်ကိန်း ကြီးလွန်းတဲ့အတွက် ကန့်သတ်ချက် ကျော်လွန်ရင် inf (အနှစ်ကိန်းဆိုရင် -inf) ရပါမယ်။ Infinity ကို ဆိုလိုတာပါ။ (၁) မပြည့်တဲ့ ပဟနာ သေးငယ်လွန်းတဲ့ ကဏ္ဍးတွေကိုလည်း အနီး စပ်ဆုံး သူညာအနေနဲ့ ယူပါတယ်။ အနီးစပ်ဆုံးယူတဲ့အခါ အပေါင်း/အနှစ်ကိုတော့ ခွဲ့မြားပေးပါတယ်။ e အမှတ်အသားနဲ့ရေးထားတဲ့ နောက်ထပ် ဥပမာဏေး နှစ်ခုပါ

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

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

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

တိက်ပိမတ္ထသည့် ကိန်းဂဏေနှင့် အိပ်ခိုပရက်ရင်များ

အိပ်စံပရက်ရှင်တွေမှာ ပါဝင်တဲ့ တိုက်ပ် တစ်မျိုးတည်း ဖြစ်ရမယ် မရှိပါ။ တိုက်ပ်မတူတဲ့ ကိန်းဂဏန်းတွေ အိပ်စံပရက်ရှင်တစ်ခုမှာ ရော်ပီး ပါဝင်နှင့်ပါတယ်။

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

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

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

အင်တိဂျာ ဒီပီးရှင်း၊ မော်ဒူးလို နှင့် ထပ်ကိန်းတင်ခြင်း

အကယ်၍ သောမကိန်းမထွက်ဘဲ ကိန်းပြည့် လိုချင်ရင် // ကို သုံးရပါမယ်။ ဒီအခါ အကြွင်းကိုဖယ်ပြီး စားလဒ်ကိုပဲ ကိန်းပြည့်အနေနဲ့ ရမှာပါ။

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

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

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

အင်တိဂျာ ဒီပီးရှင်းနဲ့ မော်ဒူးလိုကို အနှုတ်ကိန်းတွေနဲ့ သုံးမယ်ဆိုရင် သတိပြုပါ။ စားလဒ် အနှုတ် ကိန်း ဖြစ်ရင် // က ပိုင်ယ်တဲ့ အနှုတ်ကိန်းကို အနီးစပ်ဆုံး ယူမှာပါ။ တစ်နည်းအားဖြင့် round down လုပ်တာ ဖြစ်တယ်။

```

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

```

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

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

```

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

```

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

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

ဒါဇာုံး $B = 10, A = -17$ ဖြစ်လျှင်

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

အကယ်၍ $B = -10, A = 17$ ဖြစ်လျှင်

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

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

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

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

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

သချာဖန်ရှင်များ

ကိန်းဂဏ်းတွေအကြောင်း လေ့လာလက်စနဲ့ math လိုက်ဘရဲ့ သချာဖန်ရှင်တရီးကိုလည်း တစ်ခါတည်း ဆက်ကြည့်လိုက်ရအောင်။ အဓိကက သချာဖန်ရှင်ဆိုတာထက် ဖန်ရှင် အခြော့အသုံးပြုပုံကို စပီးလေ့လာ မှုပါ။ math လိုက်ဘရဲ့က Python မှာ တစ်ခါတည်း ထည့်ထားပေးပြီးသား (built-in) လိုက်ဘရဲ့ပါ။ အင်စတော်လုပ်စရာ မလိုဘဲ အင်ပို့လုပ်ပြီး သုံးလို့ရတယ်။

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

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

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

သချာဖန်ရှင်အားလုံးဟာ input တန်ဖိုးတစ်ခု ထို့မဟုတ် တစ်ခုထက်ပို၍ လက်ခံပြီး output တန်ဖိုး တစ်ခု ပြန်ထုတ်ပေးပါတယ်။ 27 နဲ့ 81 ဟာ input ဖွံ့ဖြိုး 3.0 နဲ့ 9.0 က output ဖြစ်တယ်။

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

အကြိုးဆုံးဘုံးဆွဲကိန်းကို gcd ဖန်ရှင်နဲ့ ရှာတာပါ။ အင်တိဂျာ input တစ်ခုနဲ့အထက် လက်ခံတဲ့ ဖန်ရှင်ဖြစ်တယ်။ input ကတန်းအားလုံးကို စားလို့ပြတ်တဲ့ အကြိုးဆုံးကိန်းကို ရှာပေးတယ်။ ကိန်းပြည့်မဟုတ်တာ ထည့်ရင် အယ်ရာဖြစ်ပါတယ်။

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

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

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

၅.၂ ‘တိုက်ပ်’ ဆိုတာ ဘာလဲ

Programming language အားလုံးမှာ တိုက်ပ် သို့မဟုတ် ဒေတာတိုက်ပ် သဘောတရား ပါရှိပါတယ်။ int နဲ့ float တိုက်ပ်မှာ ပါဝင်တဲ့ ကိန်းပြည့်တွေနဲ့ ဒသမကိန်းတွေကို မိတ်ဆက်ပြီးတဲ့အခါ ‘တိုက်ပ်’ ဆိုတာဘာလဲ တိုတိကျကျ ရှင်းပြလို့ရပါပြီ။ တိုက်ပ်တစ်ခုဟာ

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

$$\{ \dots, -3, -2, -1, 0, 1, 2, 3, \dots \}$$

$$\{ +, -, *, /, //, \%, **, \dots \}$$

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

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

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

```
1.4142135623730951
```

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

```
5
```

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

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

၅.၃ ဖော်ရောဲလ်များ

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

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

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

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

```
>>> age
```

```
12
```

```
>>> weight
```

```
35.5
```

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

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

```
13
```

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

```
17.75
```

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

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

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

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

အဆိုင်းမန် ခတိတ်မန်

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

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

၆.၄ စာသားများ

စာသား (text) ဟာ အသုံးအများဆုံး ဆက်သွယ်ဆောင်ရွက်ရေး ကြားခံနယ်တစ်ခုပါ။ ဝက်ဘ်ဆိုက် စာမျက်နှာ၊ အီးမေးလုံ၊ အီးဘွတ်ခုနှင့် အီးလက်ထွက်နှင့် စာရွက်စာတမ်း (e-documents) စတာတွေမှာ ရုပ်သံတွေ အသုံးပြုလာကြပေမဲ့ စာသား အဓိကဖြစ်နေဆဲပါပဲ။ ဆိုရှယ်မီဒီယာ၊ ဂိမ်းနှင့် အခြားအပ်ပဲတွေ ဟာလည်း စာသားနှင့် မကင်းနိုင်ကြပါဘူး။ ဒါကြောင့် ပရိုက်များမင်းအတွက် စာသားဟာ ဘယ်လောက်ထိ အရေးပါကြောင်း အများကြီးပြောစရာ လိုမယ်မထင်ပါဘူး။

ပရိုက်များမင်းမှာ စာသားကို *string* လိုအော်ပြီး ကာရိုက်တာ (character) တွေနဲ့ စီတန်းစွဲစည်းထားတယ်။ ကာရိုက်တာဆိုတာ အခြေခံ သတင်းအချက်အလက် ယူနစ်တစ်ခုပါပဲ။ အကွဲရာ၊ ဂဏန်း (digit)၊ သက်တာ သို့မဟုတ် ကွန်ထရှုံးလုံကုဒ် တစ်ခုခဲ့ဖြစ်နိုင်ပါတယ်။ ဥပမာ A, B, C, \$, @, #, 1, 3, _ စသည်ဖြင့် Double quotes ("") တစ်စုံကြား ညှပ်ရေးထားတဲ့ ကာရိုက်တာတွေ အသီအတန်းလိုက်ကို စာသားအနေနဲ့ ယူဆတယ်။ Python မှာ စာသားရဲ့ တိုက်ပဲ ၁၂၀၀ ဖြစ်တယ်။ string ကို အတိုကောက် ယူထားတာပါ။

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

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

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

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

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

```

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

```

str တိုက်ပဲခဲ့အခြေခံကျတဲ့ အော်ပရေးရှင်းတစ်ခုက စာသားတစ်ခုနဲ့ တစ်ခု ဆက်တာပါ။ + အော်ပရိတ်တာနဲ့ စာသားတွေကို ဆက်နိုင်ပါတယ်။

```

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

```

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

```

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

```

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

```

>>> str(pi)
'3.141592653589793'

```

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

```

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

```

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

```

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

```

ကဏ္ဍးပြောင်းလိုမရတဲ့ စာသားဖြစ်နေရင် အယ်ရှုဖြစ်မှာပါ။

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

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

```
>>> 'hello' * 3
'hellohellohello'
```

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

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

စာသားနဲ့ ဂဏန်း ဖလှယ်လို့ရပါတယ်

```
>>> 3 * 'Hello'
'HelloHelloHello'
```

string ပေါ်ရောဲလ်များ

ပေါ်ရောဲလ်တွေကို string တန်ဖိုးတော့အတွက်လည်း အသုံးပြုနိုင်တယ်။ အောက်ပါ အိပ်ပရက်ရှင်တွေ ကို နားလည်နိုင်မလား ကြိုးစားကြည့်ပါ။ ပေါ်ရောဲလ် တစ်ခုချင်းကို သူ့ရဲ့တန်ဖိုးနဲ့ အစားထိုးပြီး +, *, * အော်ပရိတ်တာတွေ အလုပ်လုပ်ပုံနဲ့ ဆက်စပ်စဉ်းစားရင် ဘာကြောင့် အခုလို့ အဖြေထွက်လဲ ခန့်မှန်းနိုင်မှာ ပါ။ သိပ်ခက်ခက်ခဲ့ခဲ့ မဟုတ်ပါဘူး။

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

Escape Character and Escape Sequence

String တစ်ခု ရေးတဲ့အခါ ပုံမှန်အားဖြင့် လိုချင်တဲ့ စာသားအတိုင်း ကီးဘုဒ်ကနေ ကာရက်တာ တစ်လုံး ချင်း ရိုက်ရုံပါပဲ။ စပယ်ရှုယ် ကာရက်တာ တဲ့ ကီးဘုဒ်ကနေ တိုက်ရိုက် ရိုက်ထည့်လို့ မရဘဲ သိ

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

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

Escape sequence တော့ကို bold ဖော်နဲ့ ပြထားပါတယ်။ Python ကွန်ဆိုးလုံး \t နဲ့ \n ကို အရှိ အတိုင်း ပြနေပါတယ်။ ဒါပေမဲ့ အခုလို စမ်းကြည့်ရင် သိသာပါလိမ့်မယ်။

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

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

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

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

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

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

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

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

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

တေဘတ် ၅.၁ Python Escape Sequences

၅.၅ အိပ်စ်ပရက်ရှင်များ

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

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

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

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

ပထမအဓိပ္ပါယ်အရ အိပ်စ်ပရက်ရှင်း စတိတ်မန့် မတူဘူးလို့ ယူဆပါတယ်။ ဒီရှုထောင့်က ကြည့်ရင် စတိတ်မန့်ဟလည်း အော်ပရေးရှင်း အတွဲအဆက်ဖြစ်ပေမဲ့ တန်ဖိုးပြန်မပေးဘူး။ အိပ်စ်ပရက်ရှင်ကတော့ တန်ဖိုးပြန်ပေးရမှာပါ။ $3 * 2$ ကို လုပ်ဆောင်တဲ့အခါ 6 ရပါတယ်။ ဒါကြောင့် $3 * 2$ ဟာ အိပ်စ်ပရက်ရှင်ဖြစ်တယ်။ $\text{result} = 3 * 2$ က စတိတ်မန့်ဖြစ်တယ်။ အဆိုင်းမန့်ဟာ ဖေရှိရောဘဲလိုကို တန်ဖိုးတစ်ခုနဲ့ တွဲဖက်ပေးတာ။ တန်ဖိုးပြန်မပေးဘူး။

ဒုတိယအဓိပ္ပါယ်အရ အိပ်စ်ပရက်ရှင်သည်လည်း စတိတ်မန့်ပဲ။ စတိတ်မန့်တွေကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်နဲ့ ပြန်မပေးတဲ့ စတိတ်မန့် အုပ်စုနှစ်စု ခွဲခြားတယ်။ တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်တွေကို အိပ်စ်ပရက်ရှင် သို့မဟုတ် အိပ်စ်ပရက်ရှင်စတိတ်မန့်လို့ ဒုတိယအဓိပ္ပါယ် သတ်မှတ်ချက်အရ ခေါ်ဘာပါ။

တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်ကောလ်တွေကိုလည်း အိပ်စ်ပရက်ရှင်လို့ ယူဆပါတယ်။ ဥပမာ $\sqrt{9}$ က 3.0 ရပါတယ်။

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

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

```
>>> print(3)
3
```

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

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

၅.၆ ဘူလီယန် တိုက်ပန့် ဘူလီယန် အိပ်စ်ပရက်ရှင်

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

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

တန်ဖိုးရှာတဲ့အခါ ဘူလီယန်တိုက်ပ် ရလာမဲ့ အိပ်စ်ပရက်ရှင်တွေကို ဘူလီယန် အိပ်စ်ပရက်ရှင်လို့ ခေါ်တယ်။ အောက်ပါတို့ကို ကြည့်ပါ

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

<, >, <=, >=, ==, != စတဲ့ သကောက်တတွေဟာ comparison operator တွေပါ။ Relational operator တွေလိုလည်း ခေါ်ကြတယ်။ အလယ်တန်းသချ်မှာ သင်ခဲ့ရတဲ့ <, >, ≤, ≥, =, ≠ စာတွေနဲ့ အဓိပ္ပာယ်တူပါတယ်။ ညီ/မညီ စစ်ချင်ရင် ညီမျှခြင်းသကောက်တန်စဲ့ == နဲ့ စစ်ရပါမယ်။ Comparison operator တောက် အောက်တိုက်ပ် အမျိုးမျိုးနဲ့ တွဲဖက် အသုံးပြုခိုင်တောက် တွေရမှာပါ။ တန်ဖိုးတစ်ခုနဲ့ တစ်ခု နှိမ်းယူဉ်နိုင်ခြင်းဟာ အရေးပါတဲ့ ကိစ္စဖြစ်ပါတယ်။

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

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

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

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

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

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

ကိန်းကဏ္ဍးတစ်ခုက တစ်နဲ့ တစ်ဆယ်ကြား ရှိ/မရှိ အခုလိုစစ်နိုင်ပါတယ်

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

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

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

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

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

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

ကိန်းတစ်ခုဟာ တစ်နဲ့ တစ်ဆယ်ရဲ့ ပြင်ပမှာရှိလား အခုလို စစ်နိုင်တယ်

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

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

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

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

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

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

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

not အော်ပရိတ်တာကတော့ True/False တန်ဖိုးကို ပြောင်းပြန် ဖြစ်စေတယ်။

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

တစ်နဲ့ တစ်ဆယ် ပြင်ပ (ငှုံးတို့မပါ) မှာ ရှိ/မရှိကို အခုလိုလည်း စစ်လိုရတယ်

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

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

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

not အော်ပရီတဲ့ သုံးခြင်းအားဖြင့် အကြောင်းအရာတစ်ခုကိုပဲ အဓိပါယ်အားဖြင့် ညီမျှတဲ့ အိပ်စံပရ်ရှင် အမျိုးမျိုးနဲ့ ဖော်ပြနိုင်ရလာတာကို တွေ့နိုင်တယ်။

အခန်း ၆

အော်ဂျက်များ

“အော်ဂျက် (object) ဆိတာဘာလ” ရှိယောင့် အမျိုးမျိုးကနေ ရှင်းပြနိုင်ပါတယ်။ အပြည့်စုံဆုံး၊ အမှန်ကန်ဆုံး ဥပမာ သို့မဟုတ် အဓိပ္ပာယ်ဖွင့်ဆိုချက် ဆိတာ မရှိပါဘူး။ သူနည်း သူ့ဟန်နဲ့ မှန်ကန်ကြတေပါပဲ။ ဒီအခြားမှာတော့ အော်ဂျက်ဆိတာ ဘာလဲ၊ ဘယ်လိမ့်မျိုးလဲ ခံစားလို့ရအောင်နဲ့ အခြေခံ အသုံးချကတ်ရှု လောက်ပဲ အဓိကထား လေ့လာကြမှာပါ။ လက်ရှိအခြေအနေနဲ့ သင့်တော်မဲ့ အဓိပ္ပာယ်ဖွင့်ဆိုချက် တချို့ကို လည်း ဖော်ပြပေးသွားမှာပါ။ သိထားသင့်တဲ့ ဘာသာရပ်ဆိုင်ရာ စကားလုံး အသုံးအနှစ်းတွေကိုလည်း မိတ်ဆက်ပါမယ်။

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

အော်ဂျက်မှာလည်း တိုက်ပ်သဘောတရား ရှိတယ်။ တိုက်ပ်တူတဲ့ အော်ဂျက်အားလုံး အော်ဂျက်အားလုံး စည်းထားပဲ တူတယ်။ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေလည်း တူပါတယ်။ ဘဏ်အကောင့် အော်ဂျက်အားလုံးဟာ လက်ကျော်ငွေနဲ့ အကောင့်နံပါတ် ပါရှိပြီး ငွေသွေး၊ ငွေထုတ်၊ ငွေလွှဲ အော်ပရေးရှင်းတွေ လုပ်ဆောင်လို့ ပါပါမယ်။

အော်ဂျက်တွဲရဲ တိုက်ပဲနဲ့ နီးနီးစပ်စပ် ဆက်နှုတ်နေတာကတော့ ကလပ်စ် (class) သဘောတရားပါ။ ကလပ်စ်ကို တိုက်ပ်တူအော်ဂျက်တွေ ဖန်တီးဖို့အတွက် သတ်မှတ်ထားတဲ့ ပရှိရမ်ကုဒ် အစုအဝေးလို့ ယေဘုယျ ပြောနိုင်ပါတယ်။ Account ကလပ်စ်၊ date ကလပ်စ်၊ Fraction ကလပ်စ် စသည်ဖြင့် အော်ဂျက် တိုက်ပ် တစ်မျိုးအတွက် ကလပ်စ်စ်ခု ရှိမှာပါ။ တိုက်ပ်တူ အော်ဂျက်တွေ အားလုံးမှာ ပါဝင်မဲ့ အချက်အလက်တွေ၊ အော်ပရေးရှင်းတွေနဲ့ အော်ဂျက် ဖန်တီးယူတဲ့ ဖန်ရှင်တွေကို ကလပ်စ်တစ်ခုနဲ့ သတ်မှတ်ရတာပါ။ ကလပ်စ်ကနေ အော်ဂျက်တွေ (တိုက်ပ် တူပါမယ်) ထုတ်ယူရတာ ဖြစ်တဲ့ အတွက် ကလပ်စ်ကို အော်ဂျက် စက်ရှုလိုလည်း ဆိုနိုင်ပါတယ်။ ပုံစံတူ အော်ဂျက်တွေ ထုတ်ပေးတာ မို့လို့ ကလပ်စ်ဆိုတာ အော်ဂျက်တည်ဆောက်တဲ့ blueprint သို့မဟုတ် template ပဲလှ့ ယူဆတာ ဟာလည်း သဘာဝကျတယ် ဆုံးရမှာပါ။

အော်ဂျက်တွေကို အက်ဘ်စရက်ရှင်း (abstraction) အနေနဲ့လည်း ရှိမြင်နိုင်တယ်။ ဘယ်လို ဖန်တီး တည်ဆောက်ထားလဲ မသိဘဲ အသုံးပြုလိုရတဲ့ အရာအားလုံးကို အက်ဘ်စရက်ရှင်းလို့ ဆိုနိုင်တယ်။ အပြင်မှာသုံးကြတဲ့ ကား၊ တို့စွာ ကွန်ပူးတာ စတာတွေဟာ အက်ဘ်စရက်ရှင်းတွေ ဖြစ်တယ်။ ဖန်ရှင် တွေဟာလည်း အက်ဘ်စရက်ရှင်းတွေပါပဲ။ အော်ဂျက်တွေကတော့ အော်ပရေးရှင်း တွဲဖောက်ပေါင်းစပ်ထားတဲ့ အက်ဘ်စရက်ရှင်းတွေပါ။ အော်ဂျက်တစ်ခုနဲ့ တွဲဆက်ထားတဲ့ အော်ပရေးရှင်းတွေဟာ

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

ဖန်ရှင် အသင့်ရှိပြီးသား ဆိုရင် အသင့်ရှိပြီးသား ဆိုရင် အော်ဂျက် တွေ ဖန်တီးအသုံးပြုနိုင်ပါတယ်။ ဖန်ရှင်ရေးရတာ ခက်ခဲနိုင်ပါတယ်။ ရှိပြီးသား ဖန်ရှင်သုံးတာကတော့ မ ခက်ပါဘူး။ ဒီသေားပါပဲ။ ကလပ်စံသတ်မှတ်ရတာ၊ ဒီဇိုင်းလုပ်ရတာ ရှုပ်ထေး ခက်ခဲနိုင်ပါတယ်။ ရှိပြီးသား ကလပ်စံကနေ အော်ဂျက် ဖန်တီးအသုံးပြုရတာ မခက်ပါဘူး။ အသုံးပြုသူ လွှယ်ကူးအဆင်ပြေစွဲနဲ့ တည်ဆောက်သူက အဓိကစွဲးစား ဖြေရှင်းရတာပါ။ သုံးစွဲသူအဆင့်ကနေ စတင်ပြီး တည်ဆောက်သူ ပရိုဂရမ့်မာ ဖြစ်လာအောင် တစ်ဆင့်ချင်း တက်လှမ်းစွဲဟာ အဓိကပန်းတိုင်ပါ။ အော်ဂျက်မိတ်ဆက် ခက်ရပ်၊ အခုပဲ လက်တွေစမ်းသပ် ကြည့်လိုက်ရအောင် ...

၆.၁ date, time and datetime

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

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

ဒုတိယလိုင်းက အော်ဂျက် ဖန်တီးတာပါ။ ဖန်ရှင်ခေါ်တာနဲ့ ပုံစံတူတာ တွေ့ရတယ်။ အော်ဂျက်ဖန်တီး ဖို့ စပယ်ရှယ် ဖန်ရှင်တစ်ခု ခေါ်ထားတယ်လို့ ယူဆနိုင်ပါတယ် (နောက်ပိုင်း ကလပ်စံအခန်းမှာ အသေးစိတ် လေ့လာရမှုပါ။)။ ကလပ်စံကနေ အော်ဂျက် ဖန်တီးယူတာကို instantiation လိုခေါ်ပြီး ရရှိလာတဲ့ အော်ဂျက်ကို အဲဒီကလပ်စံရဲ့ instance လိုလည်း ခေါ်ပါတယ်။

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

အော်ဂျက်ကို ဗေရီရေဘဲလ်နဲ့ အဆိုင်းမန်လုပ်တာပါ။ အော်ဂျက်ကို mmid ဗေရီရေဘဲလ်နဲ့ ရည်ညွှန်းအသုံးပြုလို့ ရှုံးဖြစ်တယ်။

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

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

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

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

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

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

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

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

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

ဖန်ရှင်တစ်ခုတည်းကို မတူညီတဲ့ အော်ဂျက်နှစ်ခုအပေါ်မှာ အသုံးချတာ ဖြစ်တယ်။ ဒေါ်ထိကိုပဲ သုံးတယ်။ ပထမတစ်ခုက *usid* အော်ဂျက်၊ နောက်တစ်ခုက *mmid* အော်ဂျက်အပေါ်မှာ သုံးထားတာပါ။ အမေရိကန် လွတ်လပ်ရေးရဲ့တာ ကြောသာပတေးနေ့၊ မြန်မာကတော့ တန်းနေ့တွေ (တန်လဲ့က တစ်၊ တန်းနေ့က ခုနှစ်ပါ)။ *isoweekday* ဖန်ရှင်ကို အော်ဂျက်တစ်ခုအပေါ် အသုံးပြုတဲ့ အခါ ငင်းအော် ဂျက်နဲ့ သက်ဆိုင်တဲ့ ဒေတာနဲ့ ဖန်ရှင်က အလုပ်လုပ်သွားတာပါ။ ဒါကြောင့်လည်း *attribute* မတူတဲ့ အော်ဂျက်တွေအပေါ်မှာ အသုံးချတဲ့ အခါ မတူညီတဲ့ ရလဒ်တွေ ထွက်လာရတာပေါ့။ ‘ဒေတာနဲ့ အော်ပရေးရှင်း တွဲဖက်ထားတယ်’ ဆိုတာ ဒီသဘောတရားကို ဆိုလိုတာပါ။ အော်ဂျက်ဒေတာနဲ့ တွဲဖက်အလုပ်လုပ်တဲ့ ဖန်ရှင်တွေကို မက်သွဲ (method) လို့ ခေါ်တယ်။ နောက် မက်သွဲတစ်ခုက *isoformat* ပါ။ နေ့ရက်ကို စားသားအဖြစ် 'yyyy-mm-dd' ဖော်မတဲ့ ပြန်ပေးတယ်။

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

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

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

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

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

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

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

```
>>> usid
```

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

```
>>> usid100
```

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

ခုနှစ်၊ လ၊ ရက် သုံးခုလုံး အစားထိုးချင်ရင်

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

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

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

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

```
>>> dt1
```

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

```
>>> dt2
```

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

```
>>> dt3
```

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

ခုနှစ်၊ လ၊ ရက် တခုခု ချုပ်ထားကြည့်ပါ

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

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

ချုပ်ထားခဲ့တော့ နိုင်အတိုင်းရှိပါမယ်

```
>>> dt4
```

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

```
>>> dt5
```

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

ရက်တစ်ခုတည်း အစားထိုးမယ်ဆိုရင် နံမည်နဲ့တဲ့ နည်းကပဲ အဆင်ပြေပါမယ်

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

```
>>> dt6
```

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

နံမည်မပါရင် ခုနှစ်၊ လ၊ ရက် အစဉ်အတိုင်းဖြစ်တော့ကြောင့် ခုနှစ်ကို အစားထိုးမှုပါ

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

time and datetime

နေ့ရက်နဲ့ အချင် တွဲရက်ကို datetime, ရက်စွဲမလိုဘဲ အချင်ပဆိုရင် time သုံးပါတယ်

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

timedelta

အချင်ကာလနဲ့ ပါတ်သက်ပြီး မဖြစ်မနေ သိထားသင့်တဲ့ နောက်ထပ်ကလပ်စ် တစ်ခုကတော့ ကြောချင် (duration) ကို ဖော်ပြတဲ့ timedelta ကလည်ပါ။

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

(... က အော်တို့ ထည့်ပေးသွားတာ။ ကိုယ်တိုင် ရှိက်ထည့်စရာမလိုဘူး။ တစ်လိုင်းချင်း Enter ခေါက်သွားရဲ့ပဲ။ အပိတ်ပိုက်ကွဲ့းမှာ စတိတ်မနဲ့ ဆုံးတယ်ဆုံးတာ အင်တာပရက်တာက နားလည်တယ်။)

အော့သ်ဂျက် အသုံးပြုသူအနေနဲ့ အချင်ကာလ ကြောမြင့်ချင်ကို days, weeks, hours ... စတေတွေနဲ့ သတ်မှတ်လိုက်ရတယ်။ ငြင်းတို့ကို ရက်၊ စက်နှုန်း၊ မိုက်ခရီးစက္ကန် ဖွဲ့ပြီး အော့သ်ဂျက် အတွင်းပိုင်း days, seconds, microseconds attributes ဒေတာအနေနဲ့ သိမ်းမှာပါ။

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

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

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

ဒီလိုစစ်ကြည့်ပါ

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

၆.J list

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

```
>>> odds = list()
```

ဒဲ ၁ၮ၂၁ ပါတွေပါလဲ

```
>>> odds
[]
```

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

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

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

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

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

list ဟာ mutable ဖြစ်တယ်။ အိုက်တမ် နောက်တစ်ခု ထပ်ထည့်ကြည့်ပါ

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

နှင့် အော့ဘ်ဂျက်မှာ အိုက်တမ်တစ်ခု ထပ်တိုးသွားတာ။ အော့ဘ်ဂျက်ရဲ့ စတိတ် ပြောင်းသွားတယ်။ အိုက်တမ်တစ်ခုကို ဖယ်ထုတ်ချင်ရင်

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

list အိုက်တမ်တွေရဲ့ တည်နေရာကို index လို့ခေါ်တယ်။ သူညျှနဲ့ စတယ်။ ဒုတိယက တစ်၊ တတိယက နှစ် စသည်ဖြင့် ဖြစ်မယ်။ အခုံ odds မှာ အိုက်တမ် လေးခုရှိနေတယ်။ 7 ဖြူတ်ချင်ရင် index နံပါတ် 2 ကို ပေါ်လုပ်ရမှာ

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

ဖယ်လိုက်တဲ့ အိုက်တမ်တွေ နှင့်နေရာမှာ ပြန်ထည့်ချင်တယ်ဆိုပါစွဲ။ insert ရှိပါတယ်

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

အိုက်တမ် အစားထိုးတာ၊ index နဲ့ နေရာတစ်ခုက အိုက်တမ်ကို ပြန်ထုတ်ကြည့်တာကို လေးထောင့်ကွင်းနဲ့ ရေးနည်းလည်းရှိတယ်

```

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

```

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

```

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

```

`replace` နဲ့ သဘောတရားတူတာကတော့

```

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

```

နောက်ဆုံး နေရာ (index နံပါတ် 5) ကို 14 လဲထည့်လိုက်တာ။

ပေရီရောဘဲလ် နှစ်ခုက အော့ဘ်ဂျက်တစ်ခုတည်းကို ရည်ညွှန်းနေရင် သတိပြုသင့်တဲ့ ထူးခြားချက် တွေ ရှိလာပါတယ်။

```

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

```

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

```

>>> fruits.append('orange')

```

`myfav` ပါ လိုက်ပြောင်းတာကို တွေ့ရမှာပါ

```

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

```

Mutable အော့ဘ်ဂျက်တွေမှ ဒီအချက်ကို သတိပြုဖို့ လိုပါတယ်။ ပေရီရောဘဲလ် တစ်ခုကနေ အော့ဘ်ဂျက် စတိတ်ကို ပြောင်းလဲဝဲအခါ အဲဒီအော့ဘ်ဂျက်ကို ရည်ညွှန်းတဲ့ အခြား ပေရီရောဘဲလ် အားလုံးက လည်း အပြောင်းအလဲကို မြင်မှာ ဖြစ်တယ်။ Immutable ဆိုရင်တော့ စတိတ်မပြောင်းနိုင်တော့ပြောင့် ဒီလိုကိစ္စမျိုး စဉ်းစားစရာ မလိုဘူး။

အခန်း ၃

ကွန်ထရီးလ် စတိတ်မန္ဒများ

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

၃.၁ if စတိတ်မန္ဒ

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

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

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

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

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

Python ဘားရှင်း 3.6 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 ဖော်ရောဘဲလ် (သို့) အပိုစ်ပရက်ရှင် တွေကို တွန်ကွင်းထဲမှာ ထည့်ရေးလို့ရတယ်။ ငှါးတိုကို f-string က တန်ဖိုးရှာပြီး အစားထိုးပေးမှာပါ။ ဒါကြောင့် {2 + 3} က 5 ဖြစ်သွားတာပါ။ ရိုးရိုးstring နဲ့ဆို အခုလို့

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

ရေးနေရမယ်။ F-string နဲ့ဆို ပိုအဆင်ပြေတယ်။ နမူနာတရှို့ကို လေ့လာကြည့်ပါ

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

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

```
from math import *

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

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

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

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

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

`test` ဟာ ဘူလီယန် အိပ်စံပရက်ရှင်း ဖြစ်ရပါမယ်။ (ကားရဲပုံ ကုန်ဒါရှင်တွေဟာ ဘူလီယန်တန်ဖိုး ပြန်ပေးတဲ့ predicate မက်သဒ်တွေပါ။ predicate မက်သဒ်တွေကို ဘူလီယန် အိပ်စံပရက်ရှင်းလို့ ယူဆနိုင်တယ်။)

အခုခက်ကြည့်ကြမဲ့ if...elif...else ပုံစံကတော့ ရော်ငြိုင်းမှာ မတေားဖူးသေးဘူး။ “Cascading if statement” လိုပေါ်တယ်။ အောက်ပါ ပေါ်မော်တော်များကို ကနေ grading ထုတ်ပေးမယ် ဆိုပါစို့။

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

တေဘတ် ဂုံး Score and Grading

ကျောင်းသူ/သား နံမည်နဲ့ ရမှတ်ကို ထည့်ပေးရင် ပရိုက်ရမ်းက အခုလို ပြပေးရပါမယ်။

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

ဒီပရိုက်များ အတွက် cascading if သုံးထားတာ ကြည့်ပါ

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

```

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

```

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

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

အခု grading ပရိုကရမှာ ရမှုတ်ဟာ သူညနဲ့ တစ်ရာကြား ဖြစ်သင့်တယ်။ အကယ်၍ ထည့်ပေးတာမှားရင် မှားတယ်လို့ ပြပေးချင်တယ်။ ဥပမာ

Student name: Sandy

Score: 110

You entered 110. Score must be between 0 and 100.

သူညနဲ့ တစ်ရာအတွင်း မဟုတ်ရင် အပေါ်မှာ စစ်ထားတဲ့ ဘူလီယန် အိပ်စ်ပရက်ရှင်တွေ တစ်ခုမှ မမှန်နိုင်ဘူး။ ဒါကြောင့် else အပိုင်းနဲ့ မှားထည့်ထားတယ်လို့ ပြပေးလိုက်တယ်။

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

Student name: Sandy

Score: 110

You entered 110. Score must be between 0 and 100.

Sandy get grade F.

အမှုတ်ထည့်ပေးတာ မှားနေရင် grade ကို မပြပေးသင့်ပါဘူး။ ဒီလို ပြင်ရေးလိုက်မယ် ဆိုရင်

```

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

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

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

```

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

Cascading if မှ ဘူလိယနဲ့ အပိုစ်ပရောဂျင် တစ်ခုချင်းကို အထက်အောက် အစဉ်အတိုင်း တာနိုးရှာတယ်၊ ‘ပထမဆုံး True ဖြစ်တဲ့ ဘလောက် တစ်ခုကိုပဲ လုပ်ဆောင်ပေးတယ်’ ဆိုတဲ့အချက်ကို နားလည်ဖို့ အရေးကြီးတယ်။ အောက်ကို ရောက်လာတာဟာ အပေါ်မှာ မှားခဲ့လိုပဲ။ တစ်ခုမှန်ပြီဆိုတာနဲ့ သက်ဆိုင်တဲ့ ဘလောက်ကို လုပ်ဆောင်ပြီး cascading if တစ်တဲ့လုံး ပြီးခံးသွားမှာ ဖြစ်တယ်။ အောက်ပိုင်းက elif (သို့) else တွေကို မရောက်လာတော့ဘူး။ ဒီအကြောင်းကြောင့် grading အတွက် အခုလိုလည်းရေးလိုရတယ်

```

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

if 0 <= score <= 100:
    grade = 'F'
    if score >= 90:
        grade = 'A'
    elif score >= 80:
        grade = 'B'
    elif score >= 70:
        grade = 'C'
    elif score >= 60:
        grade = 'D'
    else:
        grade = 'F'
    print(f'{stu_name} get grade {grade}.')
else:

```

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

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

Cascading if မသုံးဘဲ if...else တွေနဲ့လည်း ရေးလိုတော့ ရပါတယ်။ Nesting လုပ်တာ တွေ 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.')
```

၃.J for Loop

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

for loop နဲ့ list ထဲက အိုက်တမ်တစ်ခုချင်း ထုတ်ယူအသုံးပြနိုင်ပါတယ် ...

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

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

၆၆

Output:

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

Loop တစ်ခေါက်ပြန်ကျော့တိုင်း item မေရီရောငဲလဲမှာ list အိုက်တမ်တစ်ခုချင်း အစဉ်အတိုင်း ထည့်
ပေးမှာပါ။ အိုက်တမ်တွေကို နံပါတ်စဉ်နဲ့ တွဲချင်ရင် enumerate လုပ်ပြီး အခုလို ထုတ်လိုခြာယ်

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

Output:

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

နံပါတ်စဉ်ကို idx၊ အိုက်တမ်ကို item 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 ဝါ (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 ဘဲ list, dictionary, string စတဲ့ စထရက်ချာအမျိုးမျိုးနဲ့ အလုပ်လုပ်နိုင်တာ တွေ့ရမှာပါ။ တစ်ခါကျော်တိုင်း အိုက်တမ်တစ်ခုကို loop ဖေရံရော့လဲထဲမှာ ထည့်ပေးထားတယ်။ အိုက်တမ်တွေအားလုံး ပြီးတဲ့အခါ for loop ရပ်သွားမှာ ဖြစ်တယ်။ ပြန်ကျော်တဲ့ အကြိမ်အရေအတွက်ဟာ အိုက်တမ်အရေအတွက်ပဲ ဖြစ်တယ်။

range ဖန်ရှင်နှင့် for loop

သူညကနေ တစ်ဆယ်ထိ အစဉ်အတိုင်း ရေတွက်ချင်ရင် နည်းလမ်းတစ်ခုက

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

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

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

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

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

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

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

```

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

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

```

မှတ်ချက်။ ။ ကွားချက် တစ်မဟုတ်ရင် အစ အဆုံး ကွားချက် သုံးခုလုံး လိပါမယ်။

for loop အသုံးချုပ်များ

စတိတ်မန်တစ်စုံကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ပြန်ကျော့ဖို့ for loop ကို အသုံးပြုနိုင်ပါတယ်။ ကီးဘုံးကနေ ထည့်ပေးတဲ့ ကဏ္နား ဆယ်လုံးကို ပေါင်းမယ်ဆိုပါစို့။

```

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

print(f"Total: {tot}")

```

ပရိုဂရမ် run တဲ့ အချိန်ကျတော့မှ အကြိမ်အရေအတွက် သတ်မှတ်လိုလည်း ရတယ်။ ဥပမာ

```

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

print(f"Total: {tot}")

```

Loop တစ်ကျော်ပြီး တစ်ကျော် ဖေရီရောဘဲလ် တန်ဖိုးတွေ ဘယ်လောက်ဖြစ်နေလဲ အခုလို လိုက် ကြည့်ရှုနိုင်ပါတယ်။

```

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

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

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

3rd iter:
i = 2

```

```

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

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

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

```

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

```

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

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

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

```

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

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

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

```

list နဲ့ ပုံမာ ပြထားတာပါ။

# ဒဲ list ဆဲ ကေန္တော် ပေါင်းမှာ။
rows = [[1, 3, 5],
         [2, 4, 6]]


grand_tot = 0

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

# အပွဲ့ဗုံး 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 တစ်ကျော်ပြီး row_tot စုပ် အဆုံး

# အပျိုး for loop
2nd iter:
row = [2, 4, 6]
row_tot = 0 # ပေါ်ရောက်လဲ တစ်ခါယပ်ကြိုးတယ်

# အပွဲ့ဗုံး for loop
1st iter:
val = 2
row_tot += val           # 2

2nd iter:
val = 4
row_tot += val           # 6

3rd iter:
val = 6

```

```

row_tot += val           # 12

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

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

```

ခရစ်စမတ် သစ်ပင်လေး ဆွဲကြည့်ရအောင်။ တစ်တန်းမှာ စပေါ်ဘယ်နှစ်ခုပါလဲ ကြည့်ရလွယ်အောင်
မ လေးတွေနဲ့ ပြထားတယ်။

```

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

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

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

```

```

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

```

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

ထားတယ်။ ပင်စည် အကျယ် (TRUNK_SZ) ကိုလည်း * သုံးခု ထားတယ်။ အပေါ်ဆုံးကို row နံပါတ် သူညာ ဒုတိယကို တစ် စသည်ဖြင့် ယူဆပါမယ်။ row နံပါတ်စဉ်ကို ၁ မေရီရောင်းက ဖော်ပြတယ်။ တိုင်းပုံမှာ စပေါ်အရေအတွက်နဲ့ row နံပါတ်စဉ်ဟာ (LEAF_ROWS - 1 - r) ဖော်မြှုလာနဲ့ ဆက်စပ် နေတယ်။ * အရေအတွက်ကတော့ နှစ်ခုစီတိုးသွားပြီး (r * 2 + 1) ဖြစ်တယ်။ ပင်စည်ပိုင်း စပေါ် အရေအတွက်ကတော့ တစ်တွင်း ခြောက်ခဲ့ပါ (LEAF_ROWS - 2) ။

၃.၃ Python Arcade Library

Python Arcade (ဝက်ဘ်ဆိုက် <https://api.arcade.academy>) ဟာ အရည်အသွေးကောင်းတဲ့ ထိပ်ဆုံး Python ဂိမ်းလိုက်ဘရဲ့တွေထက တစ်ခုဖြစ်တယ်။ Arcade အပြင် အသုံးများတဲ့ အခြားတစ်ခုက pygame (ဝက်ဘ်ဆိုက် <https://www.pygame.org>) ပါ။ ဒီစာအပ်မှာ Arcade ကို သုံးပါမယ်။ Arcade ပိုကောင်းတယ်လို့ မဆိုလိုပါဘူး။ ဘိုင်နာတွေအတွက် ပို့သင့်တော်မယ် ယူဆတဲ့အတွက် အသုံးပြုတယ်။ အောက်ပါအတိုင်း အင်စတောလ်လုပ်နိုင်ပါတယ်

```
| pip install arcade
```

Arcade နဲ့ ပုံစွဲဖြို့အတွက် ဂရပ်ဖစ် ဝင်းဒီးတစ်ခုကို အောက်ပါအတိုင်း ယူရပါတယ်။ Run လိုက်ရင် ပုံ (၃.၁) မှာ တွေ့ရတဲ့ နောက်ခံရောင်နဲ့ ဝင်းဒီးအလွတ် တစ်ခု တက်လာမှာပါ။

```
# File: arcade_starter.py
import arcade

arcade.open_window(300, 200, "Arcade Starter")
arcade.set_viewport(left=0, right=300, top=0, bottom=200)

# 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 လိုက်ဘရဲ့ အင်ပိုလုပ်ထားတာပါ။ ရော်ပိုင်းမှာသုံးတဲ့ from lib import * ပုံစွဲ နဲ့ ကွာခြားတာက အခုနည်းနဲ့ အင်ပိုလုပ်ထားရင် လိုက်ဘရဲ့မှာ ပါတဲ့ အစိတ်အပိုင်းတွေကို ဒေါ်ထိ အမှတ်အသားနဲ့ အသုံးပြုရပါမယ်။ ဥပမာ open_window ဖန်ရှင်ကို

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

လိုက်ဘရဲ့ နံမည်နောက်မှာ (.) အမှတ်အသား ခံပြီး ခေါ်ရှုမှာပါ။ ဒီဖန်ရှင်မှာ လိုချင်တဲ့ဝင်းဒီး အကျယ် အမြင်၊ တိုက်တယ်လ်စာသား ထည့်ပေးထားတယ်။



ပုံ ၃.၁

`set_viewport` ဖန်ရှင်က ဝင်းဒီးနေရာယူထားတဲ့ စခရင်ချိယာမှာ ကိုထြုဒိန်တစ်စကမ် သတ်မှတ်ပေးတာပါ။ Origin အမှတ်နဲ့ x, y ဒါရိုက်ရှင် သတ်မှတ်ပေးတာပါ။

```
arcade.set_viewport(left=0, right=300, top=0, bottom=200)
```

ဝင်းဒီး ဘယ်ဘက်စွန်းကို $x = 0$, ညာဘက်စွန်းကို $x = 300$ သတ်မှတ်ထားတယ်။ အပေါ်ဘက်စွန်း (တိုက်တယ်လားမပါ) ကို $y = 0$, အောက်ဘက်စွန်းကို $y = 200$ သတ်မှတ်ထားပါတယ်။ ဝင်းဒီးရဲ့ ဘယ်ဘက်အပေါ်ထောင့်စွန်းက origin အမှတ် $(0, 0)$ ဖြစ်တယ်။ ဘယ်ဘက်ကို သွားရင် x တန်ဖိုးတိုးသွွှဲပြီး အောက်ဘက်ကို ဆင်းရင် y တန်ဖိုးတိုးသွားမှုဖြစ်တယ်။ အခုလို ကိုယ်တိုင် မသတ်မှတ်ပေးဘဲ သူ့နှင့်အရှုံအတိုင်းဆိုရင် ‘အောက်ခြေ’ ဘယ်ဘက်စွန်းက origin ဖြစ်နေမှာပါ။ အောက်ခြေကနေ အပေါ်ကိုတက်သွားရင် y တန်ဖိုးများလာမှာပါ။ သူ့နှင့်အတိုင်းက `set_viewport` ကို အခုလိုခေါ်ထားတာဖြစ်မယ်။ $top=200$, $bottom=0$ ပါ။

```
arcade.set_viewport(left=0, right=300, top=200, bottom=0)
```

အသုံးများတဲ့ ဂိမ်းလိုက်ဘရီတွေမှာ ဒီစနစ်ကို သုံးကြတာ မတွေ့ရဘူး။ ဒီစနစ်နဲ့ အကျင့်ဖြစ်သွားရင် အေားလုံးလိုက်ဘရီတွေ လေ့လာတဲ့အခါ အခက်အခဲရှိနိုင်တယ်။ အခုလေ့လာကြမဲ့ ဥပမာတွေ အတွက်လည်း အပေါ်မှာပြောတဲ့နည်းက ပိုအဆင်ပြေတယ်။ (Arcade ကိုပဲ တစိုက်မတ်မတ်သုံးမယ်၊ လေ့လာမယ်ဆိုရင်တွေ သူ့နှင့်အတိုင်းသုံးတာ အကောင်းဆုံးဖြစ်မှာပါ)။

အောက်ပါ ဖန်ရှင်ကတွေ့ ဝင်းဒီးရဲ့ နောက်ခံရောင် သတ်မှတ်တာပါ။ လိုက်ဘရီရဲ့ `color` မော်ချုံး (module) မှာ အရောင်တန်ဖိုးတွေ အဆင်သင့် သတ်မှတ်ပေးထားတယ်။ (မော်ချုံးဆိုတာ လိုက်ဘရီရဲ့ အစိတ်အပိုင်းတစ်ခုလို့ အကြမ်းဖျဉ်း ယူဆနိုင်တယ်)။

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

အခုလို အင်ပိုလုပ်ထားရင် 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။ ပုံဆွဲတဲ့ကိုစွဲကို ငှုံးတိုက်ခုကြားမှာ လုပ်ရမှာပါ။

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

arcade.run()
```

ဝင်းဒီကို မပိတ်မချင်း ပေါ်နေအောင် `run` ဖန်ရှင် ခေါ်ပေးရတာပါ။ မခေါ်ထားဘဲ ပရီဂရမ်ကို `run` ရင် ဝင်းဒီပွင့်လာပြီး ဖျက်ခနဲ့ပြန်ပိတ်သွားမှာပါ။ မကျွန်ခဲ့ဖို့ သတိပြုရပါမယ်။

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

```
import arcade
from arcade.color import *

arcade.open_window(300, 200, "Drawing Example")
arcade.set_viewport(0,300, 200, 0)

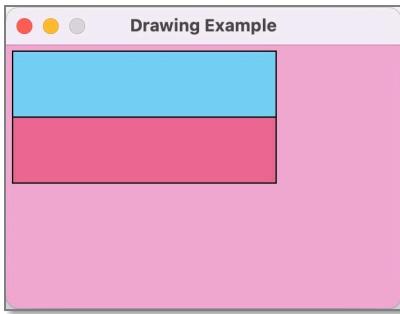
arcade.set_background_color(PINK_PEARL)
arcade.start_render()
# start drawing
arcade.draw_xywh_rectangle_filled(5,5,200, 50,BABY_BLUE)
arcade.draw_xywh_rectangle_outline(5,5,200, 50,BLACK)

arcade.draw_xywh_rectangle_filled(5,55,200, 50,PALE_VIOLET_RED)
arcade.draw_xywh_rectangle_outline(5,55,200, 50,BLACK)
#finish drawing
arcade.finish_render()
arcade.run()
```

အပေါ် ထောင့်မှန်စတုဂံပုံကို ဒီနှစ်ခုနဲ့

```
arcade.draw_xywh_rectangle_filled(5,5,200, 50,BABY_BLUE)
arcade.draw_xywh_rectangle_outline(5,5,200, 50,BLACK)
```

ဆွဲထားတာပါ 〔ပုံ (၃.၂)〕။ ပါရာမီတာတွေက $x, y, width, height, color$ အစဉ်အတိုင်းပဲ။ ဘယ်ဘက် ဘောင်ကနေ $x = 5$ ယူနစ် ဆွဲထားတယ်။ အပေါ်ဘောင်ကနေလည်း $y = 5$ ယူနစ်ဆွဲထားတယ်။ သုညထားပြီး စမ်းကြည့်ပါ။ အခြားတန်ဖိုးတွေ ထည့်ပြီး စမ်းကြည့်ပါ။ ပိုပြီးသဘောပေါက် လာပါလိမ့်မယ်။ ဒုတိယ ထောင့်မှန်စတုဂံက ဒီနှစ်ခုနဲ့



နဲ့ တဲ့ J

```
arcade.draw_xywh_rectangle_filled(5, 55, 200, 50, PALE_VIOLET_RED)
arcade.draw_xywh_rectangle_outline(5, 55, 200, 50, BLACK)
```

ဆွဲထားတာပါ။ ဘယ်ဘက်ကို ဆွဲထားတာက အပေါ်ပုံနဲ့ တူတူပဲ ($x = 5$)။ အပေါ်ပုံရဲ့ အောက်ခြေအနားနဲ့ ကပ်နေအောင် $y = 5 + 50(\text{height}) = 55$ ထားရပါမယ်။

သူနှင့်အတိုင်းဆိုရင် Arcade ကိုထိခိုက်စနစ်မှာ origin က အောက်ခြေဘယ်ဘက်စွန်းလို့ ပြောခဲ့ပါတယ်။ `set_viewport` နဲ့ အပေါ်ဘယ်ဘက်စွန်းကို origin အဖြစ်ပြောင်းလဲ သတ်မှတ်ထားတယ်။ ဒီအတွက်ပြောင့် အထက်အောက် ပြောင်းပြန်ဖြစ်သွားပါတယ်။ ဖန်ရှင် documentation မှာကြည့်ရင် အခုလိုတွေရမှာပါ (VS Code/PyCharm မှာ မောက်စိပိုင်တာကို ဖန်ရှင်နံမည် ပေါ်မှာထားရင် documentation ပြေားပါလိမ့်မယ်)

```
def draw_xywh_rectangle_filled(bottom_left_x: float,
                               bottom_left_y: float,
                               width: float,
                               height: float,
                               color: tuple[int, int, int]
                               | list[int]
                               | tuple[int, int, int, int]) -> None
```

Documentation မှာ bottom ဆိုရင် ကိုယ့်အတွက် top လို့ ပြောင်းပြန် စဉ်းစားရမယ်။ သူနှင့် စနစ်ကို ပြောင်းလိုက်လို့ ဒီအခက်အခဲ ပြောရတာပါ။ ဒါကြောင့် Arcade ကိုပဲ အဓိကသုံးမယ်ဆိုရင် သူ အရှုံးအတိုင်းသုံးတာ အကောင်းဆုံးပဲ။ အထက်အောက် ပြောင်းပြန် စဉ်းစားစရာ မလိုတော့ဘူး။

```
import arcade
from arcade.color import *

WIN_WIDTH = 600
BOARD_SIZE = 400
WIN_HEIGHT = 420
arcade.open_window(WIN_WIDTH, WIN_HEIGHT, "Arcade Checkerboard")
arcade.set_viewport(left=0,
                    right=WIN_WIDTH,
                    top=0,
```

```

        bottom=WIN_HEIGHT)
arcade.set_background_color(WHITE_SMOKE)
arcade.start_render()

COLS = 8
ROWS = 8
SQ_SIZE = BOARD_SIZE / ROWS
X_LFT = (WIN_WIDTH - BOARD_SIZE) / 2
Y_TOP = (WIN_HEIGHT - BOARD_SIZE) / 2 + 1

for i in range(ROWS):
    for j in range(COLS):
        x = X_LFT + SQ_SIZE * i
        y = Y_TOP + SQ_SIZE * j
        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)
    arcade.draw_xywh_rectangle_outline(x,
                                       y,
                                       SQ_SIZE,
                                       SQ_SIZE,
                                       BLACK)

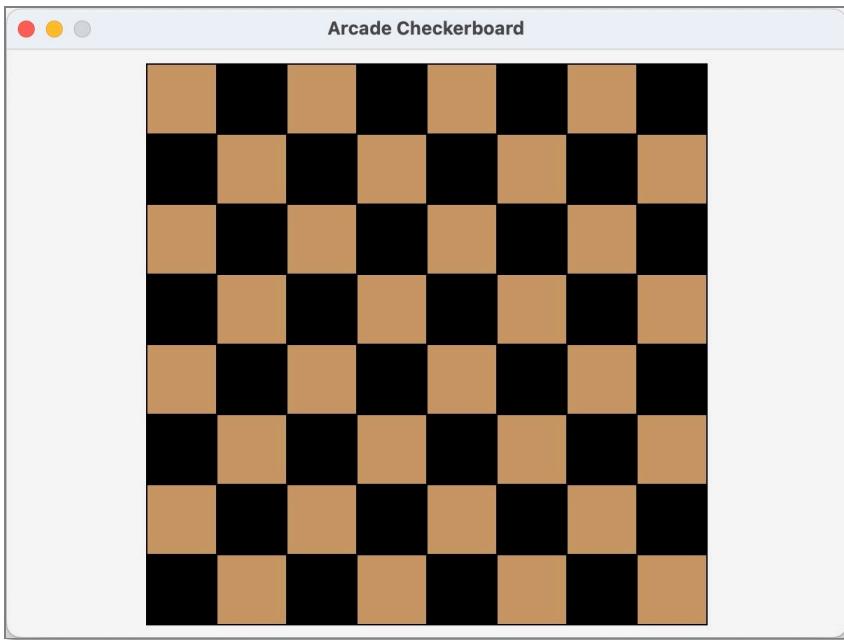
arcade.finish_render()
arcade.run()

```

၇.၄ while loop

while loop ဟာ *indefinite loop* ဖြစ်ပါတယ်။ Loop ကို စတင် လုပ်ဆောင်တဲ့အချိန်မှာ ဘယ်နှစ် ကြိမ် ပြန်ကျော့မလဲ အတိအကျ ကြို 'မသဲ' ရင် indefinite loop လို့ သတ်မှတ်တယ်။ တစ်နဲ့ တစ်ဆယ် ကြား (အပါအဝင်) ကိန်းပြည့်တစ်လုံးကို ကျေပန်း (random) ထုတ်ထားမယ်။ မမှန်မချင်း အဲဒီကဏ္ဍးကို ခန့်မှန်းပေးရမယ်။ ခန့်မှန်းတာ မှန်သွားရင် ဘယ်နှစ်ခါ ခန့်မှန်းရလဲနဲ့ မှန်တဲ့ကဏ္ဍးကို ပြပေးတဲ့ ပရိုဂရမ် မှာ while loop အသုံးပြု ရေးထားတာ တွေ့ရပါမယ်။

```
from random import *
```



ပုံ ၃၉

```
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` (ဘူတိယန် အိပ်စ်ပရက်ရှင်) မမှားမချင်း loop က ပြန်ကော့ပေးနေမှာပါ။ မှားတဲ့ အခါ ထပ်မကျော့တော့ဘဲ ရပ်သွားမှာဖြစ်တယ်။ ကျပ်န်းက ခုနှစ်ကျေတယ် ဆိုပါမီ။ တစ်၊ ကိုး၊ သံး၊ တစ်ဆယ်နဲ့ ခုံးစွဲ တို့ကို အစဉ်အတိုင်း ခန့်မှန်း ထည့်ပေးမယ်ဆိုရင် တစ်ကော့ချင်း လိုက်ကြည့်တဲ့အခါ အခုလို တွေ့ရမှာပါ။

```
num = randint(1, 10)                      # 7 ကျေတယ် ယူဆပါ
guess = int(input('? '))                   # 1 ဝေညွှေပေးတယ်
times = 1

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

```

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

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

guess != num:                      # 7 != 7 ၡ False ဖြစ်သွားခြား
    # ထပ်မကျော်တော့ဘူး

# loop အောက်က စတိတ်မနဲ့တွေ ဆက်လုပ်ပါတယ်
print(f'You get correctly after {times} guesses.')
print(f'The number is {num}.')

```

Loop က ထပ်မကျော်တော့ဘဲ ရပ်သွားတာကို *loop exits* ဖြစ်သွားတယ်လို့ ပြောတယ်။ မြန်မာ လိုတော့ loop ကနေ ထွက်သွားတယ်ပေါ့။ အကယ်၍ အောက် ဥပမာမှာ ပထမဆုံးတစ်ခါမှာပဲ ခုနှစ် ထည့်လိုက်ရင်တော့ ဘလောက်ကို တစ်ခါမှ မလုပ်ဆောင်ဘဲ ၁၀၁။ က ချက်ချင်းထွက်သွားမှာပါ။

Sentinel Controlled Loop

ကဏ္နားတွေ ဘယ်နှစ်လုံး ပေါင်းမှာလဲ ကြိုသိရင် `for` loop နဲ့ အလုပ်ဖြစ်တယ်။ ကဏ္နား ဘယ်နှစ်လုံးရှိ လဲ ကြိုမသိထားဘဲ ရှိသလောက် တစ်လုံးချင်းရှိက်ထည့်ပြီး အားလုံးပေါင်းချင်တာဆိုရင်တော့ `for` loop နဲ့ အဆင်မပြောဘူး (မရနိုင်ဘူးလို့ မဆိုလိုပါ။ မရမက လုပ်တဲ့နည်းတွေလည်း ရှိပါတယ်)။ ဒီလိုအခြေအနေ မျိုးမှာ *sentinel controlled loop* ကို သုံးပါတယ်။

Sentinel controlled loop မှာ loop ကနေ ထွက်ဖို့အတွက် အသုံးပြုတဲ့ သီးသန့်တန်ဖိုးတစ်ခု ရှိ ပေါ်မယ်။ ဒီတန်ဖိုးကို *sentinel value* လို့ ခေါ်တယ်။ ကဏ္နားတွေပေါင်းတဲ့ ကိစ္စအတွက် *sentinel value* ရွေးချယ်သတ်မှတ်တဲ့အခါ ပေါင်းရမှုကဏ္နား မဖြစ်နိုင်တဲ့ တစ်ခုကို ရွေးရပါမယ်။ သူညာ အပေါင်းထပ်တူ ရ ဂုဏ်သွေးရှိရှိတဲ့အတွက် ငှင်းကို *sentinel value* အဖြစ် ထားနိုင်တယ်။

```

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

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

print(f'Total is {total}')

```

ထည့်ပေးတဲ့ ကဏ္နားဟာ သူညာမဟုတ်မချင်း `total` မှာ ပေါင်းထည့်ပြီး နောက်ကဏ္နားတစ်ခု ထည့်ပေး ဖို့ `input` အောင်းမှာပါ။ Sentinel တန်ဖိုး သူညာ ထည့်လိုက်ရင် loop က ထွက်သွားမယ်။ ပြီးတဲ့အခါ

total ကို ပြပေးမှာပါ။ စစ်ချင်းပဲ သူည့် ထည့်လိုက်ရင် loop က တစ်ခါမှ မကျော့ဘဲ ထွက်သွားမယ်။ total လည်း သူည့်ပဲ ထွက်ပါမယ်။ အချုပ်ရိုက်ရမ်းကို 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။ အမြှုန်နောက်တဲ့အတွက် ပြန်ကျော့တဲ့ ဘယ်တော့မှ ရပ်မှာ မဟုတ်ဘူး။ ဒီလိုဖြစ်နေတာကို ဖြတ်ချပစ်ဖို့အတွက် break စတိတ်မန်ကို သုံးထားပါတယ်။ if val == SENTINEL: (ထည့်ပေးတဲ့ တန်ဖိုးက သူည့်ဆိုရင်) လက်ရှိ အလုပ်လုပ်နေတဲ့ loop ကို break လိုက်မှာဖြစ်တယ်။

အထက်ပါ sentinel loop ပုံစံနှစ်ခုကို ယူဉ်ကြည့်ရင် ပထမတစ်ခုမှာ input တန်ဖိုးထည့်ခိုင်းတာကို loop မစမိ တစ်ခါ ကြိုလုပ်ထားရတယ်။ နောက်တစ်ခုမှာတော့ အဲဒီလို ကြိုလုပ်ထားစရာမလိုဘူး။ Loop ဘလောက်ထဲက စတိတ်မန်တဲ့အတွက် အပြင်မှာထုတ်ထားရတဲ့အတွက် တချို့က ပထမပုံစံထက် ဒုတိယပုံစံက ကုၢ်စတိုင်လုံအားဖြင့် ပိုပြီး သပ်ရပ်ကြော့ရှင်းတယ်လို့ ယူဆကြပါတယ်။

Result Controlled Loop

တစ်နှစ်ငါးရာခိုင်နှစ်း အတိုးနဲ့ ဘဏ်မှာ ဒေါ်လာတစ်ထောင် စုထားတယ်။ တစ်နှစ်ပြည့်တိုင်း အတိုးအရင်းပေါင်းပြီး နောက်နှစ်အတွက် အတိုးတွေကိုတယ်။ နှစ်ထပ်တိုး (yearly compound interest) လိုခေါ်ပါတယ်။ ဒီနည်းလမ်းနဲ့ အနည်းဆုံး ဒေါ်လာတစ်သိန်း စုမံဖို့ (မိုလျုံနာတစ်ယောက်ဖြစ်ဖို့) ဘယ်နှစ်နှစ်တောင့်ရ မလဲ။ သုံးနှစ်ပြည့်ရင် စုမံမဲ့ ငွေပောက် တွေကို 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	$1102.5 \times 0.05 = 55.125$	$1102.5 + 55.125 = 1157.625$

တေတဲ့ ဂျော်လုံး သုံးနှစ်စာ နှစ်ထပ်တိုး

ဒေါ်လာတစ်သိန်း အနည်းဆုံးရဖို့ နှစ်ဘယ်လောက်ကြာမလဲ အခါလို 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 ပြသေနာမရှိပါ၊ တစ်နှစ် စာ တွက်ထားတာ စစ်ကြည့်လို့ရအောင် ထုတ်ကြည့်တာပါ)။ တစ်သန်းမပြည့်မချင်း ပြန်ကျော်ပေးအောင် loop ကွန်ဒီဂျင်ကို balance < TARGET နဲ့ စစ်ထားတယ်။ တစ်သန်းနဲ့ညီရင် (သို့) တစ်သန်းကျော်သွားတာနဲ့ ကွန်ဒီဂျင် False ဖြစ်သွားပြီး ထပ်မကျော်တော့ဘူး။ Loop တစ်ခါကျော်တိုင်း တစ်နှစ်စာ အတိုးအရင်းပေါင်း balance ကို တွက်ချက်ပြီး loop ကနေ ဘယ်အခိုင် ထွက်မလဲကလည်း အဲဒီရလဒ်အပေါ် မူတည်နေတာ တွေ့ရပါမယ်။ ဒီလို သဘောတရားရှိတဲ့ loop မျိုးကို result controlled loop လို့ ခေါ်ပါတယ်။ ကျော်မဲ့ အကြမ်အရေအတွက်က loop ထဲမှာ တွေ့က်ချက်ထားတဲ့ ရလဒ်ပေါ် မူတည်တယ်။

ပရိုဂရမ် output ကို အခုလို ကော်လံတစ်ခုစီကို အကျယ် တစ်သမတ်တည်းဖြစ်၊ စသားတော်ညာဘက်ကပ်ပြီး ညီနေအောင် f-strings ကို format spec လိုခေါ်တဲ့ 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 အကျယ်တဝါး မဖော်ပြတော့ဘဲ အခုလို ဖော့မတ်ရအောင် ဘယ်လိုလုပ်ထားလဲကိုပဲ ရှင်းပြပါမယ်။

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

ကော်လံ ခေါင်းစည်း အတွက် f-string ပါ။ ဖော့မတ်လုပ်ရမဲ့ တန်ဖိုး/ဗေရီရေတဲ့/အိပ်စ်ပရက်ရှုင်က : (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 ပြပေးပါလို ဆိုလိုတယ်။ ကျိုးနဲ့ ကော်လံနှစ်ခု အတွက် format spec ကို **>10s** သတ်မှတ်တယ်။ ကာရက်တာ ဆယ်လုံး ကော်လံအကျယ် သတ်မှတ်တယ်။

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

ဒါကတော့ yr, interest, balance တန်ဖိုးတွေကို ပြပေးဖို့ပါ။ ကော်လံအကျယ် 4, 10, 10 သတ်မှတ်တယ်။ yr ကို ကိန်းပြည့်ကဏ္ဍးအနေနဲ့ ပြပေးအောင် d သုံးတယ်။ ကျိုးနဲ့နှစ်ခုကို အသေမနဲ့ ပြဖို့ f သုံးတယ်။ .2 ကတော့ အသေမနောက် ကဏ္ဍးနှစ်လုံး ပြပေးခိုင်းတာ။ ကိန်းကဏ္ဍးဆိုရင် သူနှင့်အတိုင်း ညာဘက်ကပ်ပေးတဲ့အတွက် > ထည့်ဖို့ မလိုဘူး။ ဘယ်ဘက်ကပ်ချင်ရင် < သုံးလို့ရတယ်။

၃.၅ လေ့ကျင့်ရန် ဥပမာဏား

ကိုယ်ထရှိလဲ စထရက်ချေတွေ အသုံးချေတ်လာအောင် များများလေ့ကျင့်၊ များများစဉ်းစား၊ များများရေးကြည့် ရမှာပါ။ ဘယ်အတတ်ပညာမဆို များများလေ့ကျင့်မှ ကျွမ်းကျင်လာနိုင်မှာပါ။ ဒီသဘောအရ ပရိုဂုဏ်းမင်း ပညာရပ်ဟာလည်း ခွင့်းချက်မဖြစ်နိုင်ပါဘူး။

အောက်ပါ ဥပမာတစ်ချင်းကို ပုဂ္ဂိုလ်အောင်ဖတ်ပြီး ကိုယ်တိုင်စဉ်းစား ရေးကြည့်ဖို့ လေးလေးနက်နက် အကြံပြုပါတယ်။ ရေးပြီးသွေးရင် ပုဂ္ဂိုလ် ဖော်ပြထားချက်နဲ့ အညီ အလုပ်လုပ်/မလုပ် ဟာကွက်မရှိအောင် ဘယ်လိုစစ်ဆေး စစ်ဆေးလဲ မိမိဘာသာ စဉ်းစားကြည့်ပါ။

Internet Delicatessen

Online ကနေ အစားအသောက်တွေ delivery ပိုပေးဖို့ အမှာလက်ခံတဲ့ ဆိုင်လေးတစ်ဆိုင်အတွက် ပရီ ဂရမ်ရေးပေးရပါမယ်။ အော်ဒါမှာတဲ့အခါ မှာမဲ့ အစားအသောက် နံမည်၊ ဈေးနှုန်းနဲ့ အိပ်စံပရက်စံ delivery ယူမယူ ပရိုကရမှာ ထည့်ပေးရပါမယ်။ ပရိုကရမ်က အော်ဒါနဲ့ စုစုပေါင်းကျသင့်ငွေကို ထုတ်ပေးရပါမယ်။ တစ်သောင်းနဲ့အထက်မှာရင် delivery ဖို့ ပေးစရာမလိုပါ။ တစ်သောင်းအောက်ဆိုရင်တော့ နှစ်ရာ ပေးရပါမယ်။ အိပ်စံပရက်စံ delivery ယူမယ်ဆိုရင် သုံးရာအပို့ ထပ်ပေးရပါမယ်။

Enter the item: Tuna Salad

Enter the price: 4500

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

Invoice:

Tuna Salad	4500
delivery	500
total	5000

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

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

tot_cost = price + tot_deli_fee

print("Invoice: ")
```

```

print(f'{"item_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) သူည့် ထည့်ပေးရမှာပါ။ တစ်/သူည့် မဟုတ်တဲ့ ကဏန်းတစ်ခု မှားထည့်မိရင် if...elif ကွန်ဒီရှင်တွေ တစ်ခုမှ True ဖြစ်မှာ မဟုတ်ပါဘူး။ မှားထည့်ထားတယ်လို့ သတိပေးဖို့ else အပိုင်းမှာ လုပ်ထားတယ်။ မှန်/မမှန် စိစစ်တဲ့အခါ အောက်ပါဇေားမှ ဖြစ်နိုင်ခြေ အားလုံး ခြိုင်မိအောင် စစ်သင့်တယ်။ အိပ်စ်ပရက်စ် delivery အတွက် input ကဏန်း မှားထည့်ရင် သတိပေးတာကိုလည်း စစ်သင့်တယ်။ တစ်သောင်းဖိုး ဝယ်တာကို သုံးမျိုးစစ်ထားတာ လေ့လာကြည့်ပါ။ တစ်သောင်း မပြည့်တာနဲ့ တစ်သောင်းကော်ဖိုး အတွက်လည်း အလားတူ စစ်ကြည့်လို့ အားလုံးမှန်တယ် ဆိုရင် ဒီပရိုဂရမ်မှာ bug ပါနိုင်ခြေ လုံးဝမရှိသလာက် ဖြစ်သွားပါပြီ။

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

တေဘလ် ဂုရ် သုံးနှစ်စာ နှစ်ထပ်တိုး

Test Output:

```

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

```

```

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

```

```

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

```

delivery	0.00
total	10000.00

ပရိုဂရမ်တစ်ခုရဲ လိုအပ်ချက်ဟာ မကြာခဏ ပြောင်းလဲသွားလေ့ရှိတယ်။ အခါးရှိချက်မှာ ရွှေးနှုန်းသတ်မှတ်ချက်တွေ ပြောင်းလဲနိုင်တယ်။ အနည်းဆုံး တစ်သောင်းခွဲဝါယ်မှ ပို့ခဲ့ free ရမှာဖြစ်ပြီး တစ်သောင်းခွဲအောက်ဆိုရင် ပို့ခဲ့သုံးရှုံးစေယ် ပြောင်းလဲသတ်မှတ်လိုက်တဲ့အပြင် အိပ်စ်ပရက်စ် delivery ကလည်း တစ်ရာရွေးထပ်တက်သွားတယ် ဆုံးပါစို့။ တကဗုံးလက်တွေမှာလည်း ဒါမျိုးဖြစ်လေ့ရှိပါတယ်။ ဒီအတွက်ကို ပရိုဂရမ်ကို ပြင်ပေးရပါမယ်။ ဆိုင်ပိုင်ရှုံးကလည်း ရွှေးနှုန်းမကြာခဏ ပြောင်းဖို့ပို့နိုင်ကြောင်း ပြောလာပါတယ်။ နောင်လည်း ထပ်ပြင်ပေးဖို့လို့မဲ့ သဘောပါ။ လွယ်လွယ်ကူကူ ပြင်ပေးနိုင်ရင် အကောင်းဆုံးပါ။ ပြင်ဆင်တဲ့အခါ မှားနိုင်ခြေနည်းဖို့လည်း အရေးကြီးပါတယ်။ ခုရေးထားတဲ့အတိုင်းဆိုရင် ပြီးခဲ့တဲ့ပရိုဂရမ်မှာ ပြဿနာရှိနေပါတယ်။

ပြီးခဲ့တဲ့ ပရိုဂရမ်မှာ ပြင်မယ်ဆိုရင် 10_000 ကို လေးနေရာ၊ ပုံမှန်ပို့ခ 200 နဲ့ အိပ်စ်ပရက်စ်အပို့ကြေး 300 စတာတွေကို နှစ်နေရာစီ လိုက်ပြင်ရပါမယ်။ အလုပ်ရှုပ်တဲ့အပြင် ပြင်ဖို့ကျေနဲ့တာလို့ မှားတာလည်း ဖြစ်နိုင်တယ်။ “Find and Replace” လုပ်မှာပေါ့လို့ စောဒကတက်စရာ ရှိပါတယ်။ အခုလို့ ကုၤလိုင်းနည်းနည်းပရိုတဲ့ ပရိုဂရမ်အသေးလေးမှာ အဆင်ပြေားလိုက်ပေးပို့ပြုလိုက်တွေများတဲ့ ပရိုဂရမ်မျိုးတွေမှာ “Find and Replace” လုပ်ရင် မပြင်သင့်တာတွေကိုပါ မရည်ရွယ်ပဲ ပြင်မိသွားတာဖြစ် တတ်ပါတယ်။ ပရိုဂရမ်ရေးတဲ့အခါ ကျင့်သုံးရမဲ့ အလေ့အထကောင်းတစ်ခုက ကုၤထဲမှာ ဒီတိုင်းချရေးထားတဲ့ တန်ဖိုးတွေ (literal constants) တွေကို နာမည်ပေးထားတာပါ။ အပေါ်က ပရိုဂရမ်မှာ literal constants တွေချည်းလုံးထားတယ်။ 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))

```

202

```
print(f'%-10s %.2f' % ('delivery', tot_deli_fee))
print(f'%-10s %.2f' % ('total', tot_cost))

ဒီပရိုဂရမ်ကို cascading if မသံဃာတဲ့ ရှိုးရိုး if နဲ့ ရေးလိုလည်း ရတယ်။

FREE_DELI_AMT = 15_000
DELI_FEE = 350
EXP_DELI_FEE = 400

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

tot_deli_fee = 0

if price < FREE_DELI_AMT:
    tot_deli_fee += DELI_FEE

if is_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 %.2f' % (itm_name, price))
print(f'%-10s %.2f' % ('delivery', tot_deli_fee))
print(f'%-10s %.2f' % ('total', tot_cost))
```

ပထမ if က delivery အ ပေးပို့ လိုတယ်ဆိုရင် tot_deli_fee မှာ DELI_FEE ပေါင်းထည့်ပေးတယ်။ ဒုတိယ if က အိပ်စံပရဂ်စ် delivery ယူမယ်ဆိုရင် tot_deli_fee မှာ EXP_DELI_FEE ထပ်ပေါင်းထည့်တယ်။ အောက်ခုံး if ကတော့ အိပ်စံပရဂ်စ် delivery အတွက် တစ်နှုန်း သုည မဟုတ်ဘဲ ထည့်မြော် သတိပေးစာသား ပြပေးတယ်။

ဒီပရိုက်မျိန္ဒာ ဒီပထိုင်ခင် သူရောက ပရိုက်မှာ ဖြစ်စိန်ခြေအားလုံးအတွက်တော့ ရလဒ် တူတူမထွက်ပါဘူး။ အခြေအနေ တစ်ခုကလွှဲလို့ ကျွန်ုတာတွေအတွက်တော့ ရလဒ်တူပါတယ်။ တစ်သောင်းငါးထောင်ထက်ယောတဲ့ တန်ဖိုးနဲ့ အိပ်စိပ်ရက်စ် delivery အတွက် ဂဏန်း လဲထည့်ကြရှုပါ။ ဥမာ

```
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 အပိုပဲရက်စွဲ delivery အတွက် မပေါင်းထည့်ပေမဲ့ ပုံမှန် delivery ခ သုံးရှာင်းဆယ်ကိုတွေ့ ထည့်ပေါင်းသွားတာ တွေ့ရမှတ်။ မူးထည့်တာကိုပဲ သတိပေးစာသားပြုပေးတာ၊ ထည့်မတွက်သွားတာက ပိုပြီး သဘာဝကျတယ်လို့ ယူဆရမှာပါ။

ပြင်ပကနေ ထည့်ပေးတဲ့အခါ မဖြစ်သင့်တဲ့ input တန်ဖိုးတွေ ဝင်မလာအောင် စိစစ်တာကို input validation လို့ ခေါ်တယ်။ တကယ့် လက်တွေ့ အသုံးချ ပရိုဂရမ်တွေမှာ input validation လုပ်ထားဖို့ အရေးကြီးပေမဲ့ ဘိုင်နာအဆင့် လေ့လာတဲ့ ဥပမာတွေမှာတွေ့ လေ့လာရင်းကိစ္စကနေ လမ်းကြောင်းမချုပ်သွားအောင် ဆင်ခြင်ရမှာဖြစ်ပြီး သင့်တော်ရဲ့ ဆက်စပ်ရှင်းပြပါမယ်။

တာရာ ပရိုဂရမ်ရာ

ကားသီးလေထိုးတာဟာ ကားရဲ့ စွမ်းဆောင်ရည်ရော အနှစ်ရှာယ်ကင်းဖို့အတွက်ပါ ပဓနကျပါတယ်။ ကားတစ်စီးအတွက် အကြိုပြုထားတဲ့ တာယာပရက်ရာ (recommended pressure) အပေါ် မူးထည့်ပြီး လောက်လောက်တင်းလို့ရလဲ၊ လျော့လို့ရလဲ ရှိပါတယ်။ ဥပမာ 35 psi (pounds per square inch) ဖြစ်ရင် အလွန်ဆုံး 31.5 psi ထိ လေလျော့လို့ရပါတယ်။ လေပို့တင်းမယ်ဆုံ့ရင်လည်း အလွန်အလွန်ဆုံး 44 psi အထိ ရှုံးနိုင်ပါတယ်။

ရှေ့တာယာနှစ်လုံး ပရက်ရှာအနီးစပ်ဆုံးတူသင့်ပြီး 3 psi အထိ ကွာဟာလို့ရတယ်။ ကွာဟာချက်က 3 psi ထက်တော့ မများသင့်ဘူး။ နောက်တာယာနှစ်လုံးလည်း ထိနည်းတူစွာပဲ ဖြစ်တယ်။ ကားမော်ဒယ်အလိုက် recommended pressure ကွားမြားပေမဲ့ အိမ်စီးကားအများစုအတွက် 35 psi ဖြစ်တယ်လို့ ယူဆပြီး တာယာပရက်ရာ အိုကောမကေ စစ်ပေးတဲ့ ပရိုဂရမ် ရေးပေးရပါမယ်။ Input အနေနဲ့ တာယာတစ်ခုချင်းအတွက် ပရက်ရာ 3 psi တန်ဖိုး ထည့်ပေးလိုက်တဲ့ တာယာပရက်ရာ 31.5 psi ထက်နည်းနေရင် သို့မဟုတ် 44 psi ထက်များနေတာနဲ့ သတ်မှတ်ဘောင် မဝင် (out of range) ဖြစ်နေကြောင်း သတိပေးရပါမယ်။ တာယာအားလုံးရဲ့ ပရက်ရှာတွေ သတ်မှတ်ဘောင်အတွင်း ဝင်တယ်။ ရှေ့တာယာနှစ်လုံး ကွာဟာချက်၊ နောက်တာယာနှစ်လုံး ကွာဟာချက်တွေ ခွင့်ပြုလို့ရတာထက် မပိုဘူးဆုံ့ရင် 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 ရှင်းပြို လိုပါမယ်။ စစချင်း False တန်ဖိုး ထည့်ထားတာ တွေ့ရမှာပါ။ if စတိတ်မန်တွေက တာယာတစ်ခုချင်းကို out of range ဖြစ်နေလား စစ်ထားတာတွေရတယ်။ တာယာတစ်ခု out of range ဖြစ်တာနဲ့ is_out_of_range က True ဖြစ်သွား မှာပါ။

အောက်ပိုင်းမှ front_diff နဲ့ rear_diff ရှာတဲ့အခါ abs ဖန်ရှင်နဲ့ ပကတိတန်ဖိုး ယူထားတာ သတိပြုပါ။ ပရ်ဂ်ရှာ ခြားနားချက်ရှာတဲ့အခါ အနှံတ်တန်ဖိုး ထွက်နိုင်တဲ့အတွက်ကြောင့်ပါ။ ဘယ်ဘက် တာယာက ပရ်ရှာနည်းနေတဲ့အခါ (ဥပမာ 32 – 37 = -5) အနှံတ်တန်ဖိုး ဖြစ်နေမယ်။ ဒါကြောင့် ပကတိတန်ဖိုးယူမှုပဲ ကွာဟချက် 3 psi ကျော်မကျော်စစ်ပေးတဲ့အခါ အဖြော်ရပါမယ်။

```

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

```

and နှစ်ခုနဲ့ ဆက်ထားရင် သုံးခုလုံးမှန်မှပဲ True ထွက်မှာပါ။ တစ်ခုမှားတာနဲ့ အိပ်စ်ပရ်ဂ်ရှင်တစ်ခုလုံး ရလဒ် False ပဲ။ Out of range မဖြစ်ရဘူး ဆိုတာကို not is_out_of_range နဲ့ စစ်ထားတယ်။ is_out_of_range တန်ဖိုး False ဖြစ်မှ မှတ်ဆောင်မယ်။

•Jo

အခန်း ၈

ဖန်ရှင်များ

အခန်း (၃) မှာ ကိုယ်ပိုင် ကားရဲလ်ဖန်ရှင်တွေကို စတင် မိတ်ဆက်ခဲ့ပြီး အခန်း (၅) မှာတော့ ပါရာမီ return အကြောင်းကို မိတ်ဆက်ပေးခဲ့တယ်။ ဒီအခန်းမှာတော့ ဖန်ရှင်

ခြင့်နားလည်အောင် ပြောရရှင် မက်သဒ်ဆိုတာ ကိစ္စတစ်ခုခု လုပ်ဆောင်ပေးဖို့အတွက် နံမည်ပေးထားတဲ့ စတိတ်မန့်တွေပါပဲ။ နံမည်တစ်ခု (အဓိပ္ပာယ်ပေါ်လွှင်တဲ့) ဟာ အရေးကြီးပါတယ်။ မှန်မှန်ကန်ကန် ရွေးချယ်ထားတဲ့ နံမည်တစ်ခုဟာ မက်သဒ်ရဲ့ လုပ်ဆောင်ချက်ကို ပေါ်လှင်စေပြီး နားလည်ရလွယ်ကူးစေတယ်။ cleanStreet, cleanCorner, turnNorth စတဲ့ ပရိုဂရမ်ရဲ့ ဇာတ်လမ်းနဲ့ ကိုက်ညီမှုရှိတဲ့ နံမည်တွေဟာ ပရိုဂရမ်ကုဒ်ကို ဖတ်ရင် နားလည်ရလွယ်ကူးစေတယ်။ တကယ့်လက်တွေ့အသုံးချုပ်ပေါ်တယ်။ ကုမ္ပဏီတစ်ခုအသုံးပြုတဲ့ ပရိုဂရမ်တစ်ခုမှာ အခုလိုမက သဒ်တွေ ပါကောင်းပါနိုင်ပါတယ်။

၈.၁ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်များ

အခန်း (၅) မှာ ဖော်ပြခဲ့တဲ့ နှစ်ထပ်ကိန်းရှာတဲ့ square ဖန်ရှင်ကိုပဲ အသေးစိတ် တစ်ခါထပ်ကြည့်ရအောင်။ ဒီလောက် ရှင်းရှင်းလေးကို အကျယ်ချွဲနေတယ်လို့ ထင်ကောင်း ထင်ပါလိမ့်မယ်။ နည်းနည်းတော့ စိတ်ရှည်သည်းခံ ပေးရပါမယ်။ အခြေခံကျတဲ့ သဘောတရားတွေ ကျေည်ထားမှ ရှေ့ဆက်တဲ့ အခါ လွှာကူမှာ မို့လိုပါ။

```
>>> def square(x):
...     return x ** 2
...
>>>
```

ပိုက်ကွင်းထဲက ပေရီရောဘဲလ် x က ဖန်ရှင် ပါရာမီတာ (parameter) ဖြစ်ပြီး ဖန်ရှင်ခေါ်တဲ့အခါ ထည့်ပေးမဲ့ အားဂုံး (argument) တန်ဖိုးကို ကိုယ်စားပြုတယ်။ return စတိတ်မန့်က ဖန်ရှင်ခေါ်တဲ့နေရာ ကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန်ပါ။

ဖန်ရှင်အသုံးပြုတာကို function call လုပ်တယ်လို့ သိထားပြီးပါပြီ။ မြန်မာလိုတော့ ‘ဖန်ရှင်ခေါ်တယ်’ သို့မဟုတ် ‘ဖန်ရှင်ကောလ်တယ်’ လို့ အပြောများတယ်။ ဖန်ရှင်ခေါ်တဲ့ ပုံစံက ဒီလိုပါ

```
>>> square(2.5)
6.25
```

အခု ဖန်ရှင်ကောလ် အတွက် ပါရာမီတာ x ရဲ့ တန်ဖိုးက 2.5 ဖြစ်မှာပါ။ (ဖန်ရှင်ခေါ်တဲ့အခါ ပါရာမီတာ ပော်ရောဘဲလ် x ကို အူဂုံမန်နဲ့ အဆိုင်းမန်လုပ်ပေးတယ်လို့ ယူဆနိုင်တယ်။ ဒီကိစ္စအတွက် အူဂုံမန်က 2.5 ဖြစ်တယ်)။ အေးလုံးသီပြီး ဖြစ်တဲ့အတိုင်း ဖန်ရှင်ခေါ်ရင် ဖန်ရှင်ဘေးလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဖန်ရှင်ဘေးလောက်ထဲက return စတိတ်မန် လုပ်ဆောင်တဲ့အခါ အိပ်စ်ပရက်ရှင် x ** 2 ကို တန်ဖိုးအ ရင်ရှာတယ်။ 6.25 ရတယ်။ ဒီတန်ဖိုးကို ဖန်ရှင်ခေါ်ထားတဲ့ နေရာကို return က ပြန်ပို့ပေးလိုက်တာပါ။ အောက်ပါ ဖန်ရှင်ကောလ်မှာလည်း ဒီဖြစ်စဉ် သဘောအတိုင်း တစ်ခါထပ်ဖြစ်မှာ ဖြစ်တယ်။

```
>>> a = 1024
>>> result = square(a)
>>> result
1048576
```

အခုတစ်ခါ ပါရာမီတာ x ဟာ အူဂုံမန် a ရဲ့ တန်ဖိုး ဖြစ်တယ် ($x = a$ အဆိုင်းမန် လုပ်တဲ့သဘောပါ)။ x ** 2 က ရလာတဲ့ 1048576 ကို ဖန်ရှင်ခေါ်တဲ့ နေရာက ပြန်ရတယ်။ နောက်ဆုံးတော့ ဒီတန်ဖိုးကို result မှာ အဆိုင်းမန်လုပ်တယ်။ ဖြစ်စဉ်အရ ရှိုးရှင်းပါတယ်။

```
>>> x = 10
>>> square(x)
```

ဒီလိုလိုရင်ရော ဘယ်လို့ ဖြစ်မလဲ။ နည်းနည်းထူးခြားတာက အူဂုံမန်နဲ့ ပါရာမီတာ နံမည်တူ့နေတာ။ ပါရာမီတာရဲ့ စက်ပုံဟာ ဖန်ရှင်သတ်မှတ်ချက် အတွင်းမှာပဲ ရှိတယ်လို့ ယူဆရမှာပါ။ ဒါကြောင့် အူဂုံမန် x နဲ့ ပါရာမီတာ x နဲ့က သီးခြား ပော်ရောဘဲလ်တွေ။

```
>>> u = 15
>>> t = 5
>>> square(u + 2*t)
```

အူဂုံမန်က အိပ်စ်ပရက်ရှင် ဖြစ်နေရာရင် တန်ဖိုးအရင်ရှာပြီး ရလဒ်ကို ပါရာမီတာနဲ့ အဆိုင်းမန် လုပ်ပါတယ် ($x = u + 2*t$)။

```
>>> z = square(2.0) + 5
>>> square(z)
81.0
>>> square(square(2.0) + 5)
81.0
```

ဒုတိယ ဖန်ရှင်ခေါ်တဲ့နေရာမှာ အိပ်စ်ပရက်ရှင်ကို z နဲ့ အဆိုင်းမန် မလုပ်တော့ဘဲ တစ်ခါတည်း အူဂုံမန် အနေနဲ့ ထည့်လိုက်တာပါ။ သဘောတရား တူတူပါပဲ။

ဖန်ရှင် return လုပ်တဲ့ သဘောကို နားလည်ထားဖို့လည်း အရေးကြီးတယ်။ return စတိတ်မန် ဟာ ဖန်ရှင်ကနေ တန်ဖိုးတစ်ခုကို ဖန်ရှင်ခေါ်တဲ့ဆိုကို ပြန်ပေးတယ်လို့ သိထားပြီးပါပြီ။ ဖန်ရှင်ထဲကနေ return လုပ်လိုက်တာနဲ့ ခေါ်ထားတဲ့ နေရာကို ချက်ချင်း ပြန်ရောက်သွားတာ။

```
>>> def get_sign(r):
...     if r > 0:
...         return 'positive'
...     elif r < 0:
...         return 'negative'
```



```

...
    else:
...
        return 'zero/nosign'
...
>>>

```

```

>>> '10 is ' + get_sign(10)
'10 is positive'

```

အခုအိပ်ပရက်ရှင်ရဲ့ တန်ဖိုးရှာဖို့ `get_sign(10)` ခေါ်လိုက်တဲ့အခါ လက်ရှိနေရာကနေ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ဆီ ပြောင်းရွှေ့ ရောက်ရှိသွားပါမယ်။ ဖန်ရှင်ထဲက စတိတ်မန်တွေ အစဉ်အတိုင်း စတင် လုပ်ဆောင်တယ်။ ဖန်ရှင်က `return` လုပ်တဲ့အခါ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ထဲကနေ ခေါ်ခဲ့တဲ့ နေရာကို တဖန်ပြန်၍ ပြောင်းရွှေ့သွားတယ်။ `return` ပြန်လိုက်တဲ့ တန်ဖိုးကို ဖန်ရှင်ခေါ်တဲ့ နေရာမှာ ရရှိ ပြီး လုပ်လက်စ အိပ်ပရက်ရှင်ကို ဆက်လည်ပါတယ်။ ဒီလိုမြင်ကြည့်ပါ ...

```

def get_sign(r):
    if r > 0:
        return 'positive'
    elif r < 0:
        return 'negative'
    else:
        return 'zero/nosign'

'10 is ' + get_sign(10)

```

မြှားအနက်က ဖန်ရှင်ခေါ်လိုက်တဲ့အခါ လုပ်ဆောင်မှု ပြောင်းရွှေ့သွားတာကို ပြတယ်။ မြှားအနီက `return` ပြန်တဲ့အခါ ခေါ်ခဲ့တဲ့ နေရာ ပြန်ရောက်သွားတာကို ပြတာပါ။

ဆက်လက်ပြီး ပါရာမီတာ တစ်ခုထက်ပိုတဲ့ ဖန်ရှင်တချို့ကို ကြည့်ပါမယ်။ ပါရာမီတာဆိုတာ ဖန်ရှင် အတွက် လိုအပ်တဲ့ `input` ကို လက်ခံတဲ့ မေရီရောဘဲလ်ပါပဲ။ ထောင့်မှန်စတုဂံရဲ့ အလျားနဲ့ အနံကနေ ဓရိယူရှာပေးတဲ့ ဖန်ရှင်က ဒီလိုပါ

```

def rect_area(wid, len):
    return wid * len

```

ဖန်ရှင်တစ်ခုကို အခြေခံ အုပ်ချပ်သဖွယ် အသုံးပြု၍ အမြားဖန်ရှင်တွေ တည်ဆောက်ယူနိုင်တယ်။ `rect_area` ကို `box_vol` မှာ သုံးထားတာပါ

```

def box_vol(w, l, h):
    return rect_area(w, l) * h

```

ဒီဖန်ရှင်ကို ခေါ်ရင် ဘယ်လိုဖြစ်မလဲ ကြည့်တတ်သင့်တယ်။ အခုလို ခေါ်မယ် ဆိုပါစို့

```

>>> box_vol(10, 5, 3)

```

`w=10, l=5, h=3` ဖြစ်တယ်။ ဖန်ရှင် ဘလောက်ထဲကို ရောက်သွားမယ်။ `return` ပြန်ပေးဖို့ အိပ်စ်ပရက် ရှင်ကို တန်ဖိုးရှာပါတယ်

```
| rect_area(w, 1) * h
```

rect_area ဖန်ရှင်ခေါ်တယ်။ wid=w, len=1 ဖြစ်မယ်။ အခုက္ခာအတွက် ပါရေမီတာနှစ်ခုရဲ့ တန်ဖိုး ကဲ 10 နဲ့ 5 အသီးသီး ဖြစ်မှုပါ။ 50 ရပါမယ်။ 50 * h ကို တန်ဖိုးဆက်ရှုပြီး ရလာတဲ့ 150 ကို box_vol ခေါ်ထားတဲ့နေရာကို return ပြန်ပေးမှာ ဖြစ်တယ်။ အခြေခံသဘောတရားတွေ သိပြီးတဲ့အခါ အတန်အသင့်ရှုပ်ထွေးတဲ့ ဖန်ရှင်တချို့ကို ကြည့်ပါမယ်။

ဖန်ရှင်များနှင့် အက်ဘ်ခုတ်ရှင်းလုပ်ခြင်း

မွေးသက္ကရာဇ် (date of birth) ကနေ အသက် တွေက်ပေးတဲ့ ဖန်ရှင်ကို လေ့လာကြည့်ပါ။ အသက်တွေက် တဲ့ လေ့ဂျစ်ကို မရှင်းပြတော့ဘူး။ လေ့ကျင့်ခန်းအနေနဲ့ မိမိအသာ နားလည်အောင်ကြည့်ပါ။

```
# File: age_today.py
from datetime import *

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

print(age_today(date(1990, 4, 2)))
```

ဖန်ရှင်အတွင်းပိုင်း လေ့ဂျစ်တွေ ဘယ်လိုပဲ ရှုပ်ထွေးပါစေ၊ အသုံးပြုရတာကတော့ မခက်ပါဘူး။ ဖန်ရှင် ခေါ်တဲ့အခါ ဘယ်လို တည်ဆောက်ထားလဲ အတွင်းပိုင်း အယ်လုပ်ရှစ်သမဲတွေ၊ လေ့ဂျစ်တွေ သိစရာမ လိုဘဲ သုံးရတာပါ။ ဖန်ရှင်က ငင်းရဲ့ အတွင်းပိုင်း ကုဒ်တွေကို အက်ဘ်စရက်ရှင်း (abstraction) လုပ်ပေးလိုက်တာ ဖြစ်တယ်။ ဒါဟာ ဖန်ရှင်ရဲ့ အရေးပါဆုံး ဂုဏ်သိုလို ဆိုရင်လည်း မမှားဘူး။

age_today ဖန်ရှင်ဟာ ပို့ကြီးတဲ့ ပရှိကရမ်တစ်ခုရဲ့ တစ်စိတ်တစ်ပိုင်း ဖြစ်လာနိုင်ပါတယ်။ ပရှိကရမ် အသေးစားလေးတစ်ခုမှာ အသုံးပြုထားတာကို လေ့လာကြည့်ပါ။ နိုင်ငံအများစုမှာ (၁၈) နှစ် မပြည့်သေး တဲ့သူကို ဆေးလိပ်ရောင်းခွင့် မရှိဘူး။ ဥပဒေရှိပါတယ်။ စားသုံးသူရဲ့ မွေးသက္ကရာဇ် ထည့်ပေးလိုက်တာနဲ့ ရောင်းလို့ ရ/မရ ပြပေးတဲ့ ပရှိကရမ်လေးပါ။

```
# File: sell_cigarette.py
from datetime import *

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

def can_by_cig(dob):
```

```

age = age_today(dob)
return True if age >= 18 else False

def main():
    """
    Given date of birth, this program tells whether the customer
    is eligible to buy cigarette or not.

    Enter 'exit' to quit the program.
    """
    print("Please enter 'quit' to exit this program.")
    while True:
        dobstr = input('Enter date of birth (yyyy-mm-dd): ')
        if dobstr == 'quit': break
        dob = date.fromisoformat(dobstr)
        print(dob)
        if can_by_cig(dob):
            print("Okay!")
        else:
            print('Too young to sell cigarette!')
    print('Program exited...')

if __name__ == "__main__":
    main()

```

၈.၂ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်များ

ဖန်ရှင်အားလုံးတော့ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်တွေ မဟုတ်ကြပါဘူး။ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေလည်း ရှိတယ်။ ဥပမာ output ထုတ်တဲ့ print ဖန်ရှင်ဟာ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်မျိုးပါ။ အောက်ပါ print_sign ဖန်ရှင်ဟာ get_sign နဲ့ ဆင်တူပေမဲ့ တန်ဖိုး return ပြန်မပေးပါဘူး။

```

def print_sign(r):
    if r > 0:
        print('positive')
    elif r < 0:
        print('negative')
    else:
        print('zero/nosign')

```

ဒီဖန်ရှင်မှာ return မပါတာ တွေ့ရပါမယ်။ ကားခဲ့လ်ဖန်ရှင်တွေမှာလည်း return မသုံးခဲ့တာ ပြန်အမှတ်ရမှာပါ။ append_n_times ကို လေ့လာကြည့်ပါ

```

def append_n_times(lst, item, n):
    for i in range(n):
        lst.append(item)

```

```
lst = []
append_n_times(lst, 'hello', 10)
print(lst)
```

အိုက်တမ်တစ်ခုကို သတ်မှတ်ထားတဲ့ အချေအတွက်ပြည့်အောင် list တစ်ခုနောက်ကနေ ဆက်ပေးတယ်။ နှင့် list မှာ အိုက်တမ်တွေ တိုးသွားပြီး စတိတ်အပြောင်းအလဲ ဖြစ်စေတယ်။

Output ထုတ်တဲ့ ဖန်ရှင်တွေဟာ တန်ဖိုးပြန်ပေးလေ့မရှိဘူး။ စခရင်မှာ စာသား (သို့) ရုပ်ပဲ ပြပေး တာဟာ output ဖြစ်တယ်။ ဖိုင်တစ်ခုမှာ ရေးတာလည်း output ပဲ (ဥပမာ Python ကုဒ်ဖိုင်ကို ပြင် ပြီး save လုပ်တာ)။ အော့သ်ဂျက် စတိတ်ကို ပြောင်းလဲစေတဲ့ ဖန်ရှင်တွေဟာလည်း တန်ဖိုးပြန်မပေး တဲ့ ဖန်ရှင်တွေ ဖြစ်လေ့ရှိတယ် (ဥပမာ list ရဲ့ append နဲ့ insert ဖန်ရှင်)။ စတိတ်အပြောင်းအလဲ ဖြစ်စေတဲ့ ဖန်ရှင်အားလုံး တန်ဖိုးပြန်မပေးတာတော့ မဟုတ်ဘူး။ ဥပမာ pop ဟာ တန်ဖိုးပြန်ပေးပါတယ်။ စတိတ်အပြောင်းအလဲလည်း ဖြစ်စေတယ်။

တန်ဖိုးပြန်တဲ့ ဖန်ရှင်ပဲ return ပြန်လိုက်တာ မဟုတ်ပါဘူး။ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေမှာလည်း return ပါနိုင်ပါတယ်။ print_sign ကို ဒီလိုရေးလိုလည်း ရပါတယ်

```
def print_sign2(r):
    if r > 0:
        print('positive')
        return
    elif r < 0:
        print('negative')
        return
    else:
        print('zero/nosign')
        return
```

တန်ဖိုးပြန်မပေးတဲ့အတွက် return ပဲဖြစ်ရပါမယ်။ တန်ဖိုး/အိပ်စံပရက်ရှင် တွဲပြီး ပါလိုမရပါဘူး။ ဖန်ရှင်ဘလောက် ပြီးတဲ့အခါ ခေါ်တဲ့နေရာကို ပြန်ရောက်သွားရမှာ ဖြစ်တဲ့အတွက် return မပါတဲ့ ဖန်ရှင်တွေရဲ့ ဘလောက်အဆုံးမှာ return ရှိတယ်လို့ ယူဆနိုင်တယ်။ ဥပမာ return မပါတဲ့ print_sign ကို အခုလို ယူဆနိုင်တယ်

```
def print_sign(r):
    if r > 0:
        print('positive')
    elif r < 0:
        print('negative')
    else:
        print('zero/nosign')
    return
```

၈.၃ ခွဲခြမ်းခိုက်ဖြာခြင်းနှင့် ပရီဂရမ် ဒီဇိုင်း

Insurance Premium

နှစ် နှစ်ဆယ် သက်တမ်းကာလ \$500,000 အသက်အာမခံထားရင် ကျန်းမာတဲ့ အသက် ၃၀ အမျိုးသမီး တစ်ယောက်အတွက် နှစ်စဉ်ပျမ်းမျှ အာမခံကြေး (premium) \$229 ယူအက်စ် ဒေါ်လာ ကုန်ကျကုတ်။ ရွယ်တဲ့ အမျိုးသမီးတစ်ယောက် ဆိုရင်တော့ ဒီအာမခံအတွက်ကိုပဲ တစ်နှစ် ပျမ်းမျှ \$373 ဒေါ်လာ ကုန်ကျပါမယ်။ ဒါကယောက်ပျမ်းမျှ သဘောကိုပြောတာ။ တကယ်တမ်း အသက်အာမခံထားရင် သက်တမ်းကာလ၊ အကျိုးခံစားနိုင်မည့် ငွေပမာဏ (coverage)၊ ကျန်းမာရေး၊ အသက် စတဲ့ အချက်တွေပေါ် မူတည်ပြီး တွက်ချက်တာပါ။ လူတစ်ယောက်နဲ့ တစ်ယောက် ပရီမီယံ မတူဘူး။

Coverage Amount	Age 30	Age 40	Age 50	Age 60
\$250,000	\$142	\$193	\$392	\$989
\$500,000	\$205	\$307	\$685	\$1,781
\$1 million	\$325	\$526	\$1,227	\$3,375
\$2 million	\$593	\$984	\$2,388	\$6,758

တေဘာလ ၈.၁ နှစ်နှစ်ဆယ် အသက်အာမခံကြေး (မ)

Coverage Amount	Age 30	Age 40	Age 50	Age 60
\$250,000	\$162	\$224	\$499	\$1,375
\$500,000	\$251	\$360	\$891	\$2,567
\$1 million	\$408	\$628	\$1,681	\$4,952
\$2 million	\$749	\$1,190	\$3,267	\$9,660

တေဘာလ ၈.၂ နှစ်နှစ်ဆယ် အသက်အာမခံကြေး (ကျား)

မွေးသက္ကရာဇ်၊ အကျိုးခံစားလိုသည့် ပမာဏ (coverage amount)၊ ကျား/မ အလိုက် တစ်နှစ် ပျမ်းမျှ ပရီမီယံကြေး ကုန်ကျင့် ပြပေးတဲ့ ပရီကရမ်အတွက် အာမခံ ကုမ္ပဏီတစ်ခုက သင့်ထံ ပရောဂျက် လာအပ်တယ် ဆိုပါစို့။ ဒီအတွက် ပရီကရမ် တစ်ခုကို ဒီဇိုင်းလုပ် ရေးသားပုံအဆင့်ဆင့်ကို ဆက်လက်ဖော်ပြုမယ်။

အာမခံသက်တမ်း နှစ်ဆယ်နှစ်အတွက်ပဲ စဉ်းစားပါမယ်။ အသက် ၃၀ အောက် အနီးစပ်ဆုံး ယူပါမယ်။ ၃၁ မှ ၄၀၊ ၄၁ မှ ၅၀၊ ၅၁ မှ ၆၀၊ ၆၁ မှ ၇၀၊ ၇၁ မှ ၈၀၊ ၈၁ မှ ၉၀၊ ၉၁ မှ ၁၀၀ စသည်ဖြင့် အနီးစပ်ဆုံးယူပါမယ်။ အခြေခံ အဆင့် လေ့လာသူအတွက် အလွန်အမင်း မရှုပ်ထွေးစေဘဲ သဘောတရားပိုင်ရာကို အမိက မြင်သာ အောင် ပရီကရမ်ရဲ့ လိုအပ်ချက် (requirements) ကို ရှုံးရှင်းအောင် လုပ်ထားတာပါ။

Input သုံးခုထည်ပေးရမယ်။ မွေးသက္ကရာဇ် (date of birth), coverage amount နဲ့ ကျား/မ (gender) တို့ဖြစ်တယ်။ အသုံးပြုသူအနေနဲ့ အဆင်ပြဆုံး၊ အလွယ်ကူဆုံးဖြစ်အောင်၊ အမှားအယွင်းနည်းနိုင်သမျှနည်းအောင် စဉ်းစားသင့်တယ်။

မွေးသက္ကရာဇ်ကို ကြည့်ရအောင်၊ ရက်စွဲကို နိုင်ငံနဲ့ နေရာအောင် ပေါ်မူတည်ပြီး ဖော့မတ်အမျိုးမျိုး နဲ့ ရေးကြောက်။ ၀၄/၀၅/၂၀၂၀, ၀၄-၀၅-၂၀၂၀, Apr-4-2020 စသည်ဖြင့်။ ခုနှစ်ကို ကဗျာန်း နှစ်လုံးပဲ ရေးတာလည်း ရှိတယ်။ ရက်နဲ့လက် ရောမှ သုည်မပါဘဲလည်း ရေးတယ်။ ဥပမာ ၄/၅/၂၀၂၀, ၄/၅/၂၀၂၀, Apr-4-2020 ။ ဖော့မတ်တစ်နှစ်မျိုးကိုပဲ သတ်သတ်မှတ်မှတ် သုံးသင့်တာ ဖြစ်ပေမဲ့ ဆော်ဖဲ့အပ်သူက ဖော့မတ် သုံးမျိုးနဲ့ ထည့်လို့ရအောင် လုပ်ပေးဖို့ တောင်းဆိုထားတယ်။

01-01-2024

01/01/2024

Jan-1-2024

မွေးသက္ကရာဇ်ကို ဒီဖော့မတ်သံးမျိုးနဲ့ ပရီဂရမ်က လက်ခံရပါမယ်။ input ဖန်ရှင်က ကိုဘုဒ်ကထည့်ပေးတော်း string အနေနဲ့ ပြန်ပေးပါတယ်။

မွေးသက္ကရာဇ်ကနေ အသက်ကို တွက်ယူရမှာပါ။ ဒီအတွက် ဖန်ရှင် (ဥပမာ calc_age) သတ်မှတ်နိုင်တယ်။ နေ့ရက်၊ အချိန်နဲ့ သက်ဆိုင်တဲ့ အတွက်အချက်တွေ အတွက် စာသားကို အသံးမပြုသင့်ဘူး။ date, datetime စတာတွေ သုံးသင့်တယ်။ ဒါကြောင့် စာသားကနေ မွေးသက္ကရာဇ်ကို ဖော်ပြတဲ့ date (သို့) datetime အော့သုတေသနပျက် ပြောင်းစိုးလိုပါမယ်။ '01-01-2024' ကနေ 1, 1, 2024 ကေန်းတွေ ရအောင် စာသားကို တစ်ဖြတ်ချင်းဖြတ်ပြီး ပြောင်းမယ်။

```
>>> egstr.split('abc')
['123', '456', '789']
>>> dt1 = '01-01-2024'
>>> parts1 = dt1.split('-')
>>> parts1
['01', '01', '2024']
>>> dt2 = '01/01/2024'
>>> parts2 = dt2.split('/')
>>> parts2
['01', '01', '2024']
>>> dt3 = 'Feb-02-2024'
>>> parts3 = dt3.split('-')
>>> parts3
['Feb', '02', '2024']
```

split ဖန်ရှင်က string တစ်ခုကို အပိုင်းတွေ ပိုင်းပေးတာပါ။ အပိုင်းတွေ အားလုံးကို list အနေနဲ့ ပြန်ပေးတယ်။ ဘယ် ကာရ်ကာ (သို့) string နဲ့ စားထားရင် ပိုင်းဖြတ်ချင်လဲ ထည့်ပေးလိုပါတယ်။ အပေါ်မှာ 'abc', '-', '/' အသီးသီးနဲ့ ပိုင်းဖြတ်ထားတယ်။

```
>>> dt1 = '01-01-2024'
>>> parts1 = dt1.split('-')
>>> date(int(parts1[2]), int(parts1[1]), int(parts1[0]))
datetime.date(2024, 1, 1)
```

'Feb-02-2024' ဖော့မတ်က နည်းနည်း ပို့ချုပ်မယ်။ လက 'Jan', 'Feb' စသည်ဖြင့် စာသားဖြစ်နေတယ်။ အခုလို dictionary တစ်ခု ရှိရင် လန်မည်ကနေ သက်ဆိုင်တဲ့ ကေန်းကို အလွယ်တကူ ရနိုင်ပါတယ်

```
>>> mths = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
...           'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
...           'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
>>> mths['Jan']
1
>>> mths['Dec']
12
```

ဒီလိုခိုရင် date အော်ဂျက် ရဖိုအတွက်လည်း သိပ်မခက်တော့ဘူး။ ဥပမာ

```
>>> dt4 = 'Dec-26-2024'
>>> parts4 = dt4.split('-')
>>> date(int(parts4[2]), mths[parts4[0]], int(parts4[1]))
datetime.date(2024, 12, 26)
```

အခုဖော်ပြခဲ့တဲ့ နည်းအပြင် အဗြားနည်းလမ်းတွေလည်း ရှိပါတယ်။ datetime ကလပ်စုံမှာ နေ့ရက် ကို စာသားကနေ datetime ပြောင်းပေးတဲ့ strftime ဖန်ရှင် ရှိပါတယ်။

```
>>> dtstr1 = 'Jan-01-2024'
>>> datetime.strptime(dtstr1, "%b-%d-%Y")
datetime.datetime(2024, 1, 1, 0, 0)
>>> dtstr2 = '30-12-2024'
>>> datetime.strptime(dtstr2, "%d-%m-%Y")
datetime.datetime(2024, 12, 30, 0, 0)
>>> dtstr3 = '30/Dec/2024'
>>> datetime.strptime(dtstr3, "%d/%b/%Y")
datetime.datetime(2024, 12, 30, 0, 0)
```

ဒီဖန်ရှင်က စာသားနဲ့ ဖော်ပြထားတဲ့ အခိုင်နေ့ရက်ကို datetime အော်ဂျက် ပြောင်းပေးဖို့ ဖော့မတ် ကုဒ် (format code) လိုအပ်တဲ့ % နဲ့ စတဲ့ ကာရွက်တာတွေကို လက်ခံတယ်။ %d က ရက်၊ %m က လ ကို ကဏ္ဍား တစ်လုံး (သို့) နှစ်လုံးနဲ့ ဆိုတဲ့ အဓိပါယ် (ဥပမာ ကိုရက်နေ့ကို 09 သို့မဟုတ် ၅၊ ငါးလပိုင်းကို 05 သို့မဟုတ် 5)။ %Y က ခုနှစ်ကို ကဏ္ဍားလေးလုံးနဲ့ ရေးတယ်လို့ ဆိုလိုတာပါ။ %b ကတော့ လရဲ့ နံမည် ကို Jan, Feb, Mar စသည်ဖြင့် အတိကောက်ရေးတယ်လို့ ဆိုလိုတယ်။ 'Jan-01-2024' ရဲ့ ဖော့မတ် က '%b-%d-%Y'

နောက်ဆက်တဲ့ က

လိုအပ်သည့် ဆောဖံ့ဌများ ထည့်သွင်းခြင်း

အခြား အင်ဂျင်နီယာ/သိပ္ပံဌ ပညာရပ်တွေလိုပဲ ပရိုကရမ်မင်းလေ့လာတဲ့အခါ လက်တွေ့လုပ်ကြည့်ဖို့ အင်မတန်အရေးကြိုးပါတယ်။ လက်တွေ့ရေးမကြည့်ဘဲ စမ်းသပ်မကြည့်ဘဲ သဘောတရားပိုင်းဆိုင်ရာတွေကို အမှန်တကယ်နားလည်ဗုံး မဟုတ်ပါဘူး။ အခြားပညာရပ်တွေထက် ကွန်ပျိုးတာ ပရိုကရမ်မင်းရဲ့ အားသာချက်တစ်ခုကတော့ လက်တွေ့စမ်းသပ်ခန်းကြိုးတွေ ရှိစရာမလိုတာပါပဲ။ စမ်းသပ်ပစ္စည်းတွေ လည်း များများစားစား မလိုအပ်ဘူး။ ကွန်ပျိုးတာ တစ်လုံးနဲ့ လိုအပ်တဲ့ ဆောဖံ့ဌပဲတဲ့ ရှိရင်ရပြီ။ ဆောဖံ့ဌတွေကလည်း ပိုက်ဆံကုန်စရာမလိုဘူး။ မပေးဘဲ သုံးလို့ရတာ။

ပရိုကရမ်လက်တွေရေး လေ့ကျင့်ဖို့အတွက် လိုအပ်တဲ့ ဆောဖံ့ဌတွေ ထည့်ထားရပါမယ်။ Python programming language အတွက် Python ဆောဖံ့ဌ ထည့်ရပါမယ်။ Python ဆောဖံ့ဌမရှိဘဲ Python ကုဒ်တွေ၊ Python ပရိုကရမ်တွေ run လို မရပါဘူး။ Python အပြင် ပရိုကရမ်ကုဒ် ရေးဖို့ အတွက် အထောက်အကူဗြာ ဆောဖံ့ဌပဲတစ်ခုလည်း လိုအပ်တယ်။

ပရိုကရမ် ကုဒ်ရေးဖို့အတွက် PyCharm သိမဟုတ် Visual Studio Code (VS Code) ကို အသုံးပြုနိုင်ပါတယ်။ အက်ဆေးတစ်ပုဒ်ရေးတဲ့အခါ မိုက်ခရိုဆောဖံ့ဌ Word နဲ့ရေးလိုရသလို ရိုးရိုးရှင်းရှင်း Notepad လောက်နဲ့ ရေးလိုလည်း ရတာပါပဲ။ အက်ဆေးရဲ့ အဓိပ္ပာယ်ကသာ အစိုကပါ။ ပရိုကရမ်ကုဒ် ရေးတဲ့အခါမှာလည်း ခီးသောပါပဲ။ ဆောဖံ့ဌပဲရေးကြိုးတွေ တည်ဆောက်တဲ့အခါ လိုအပ်မဲ့ ဖီချာတွေ အားလုံး စုံစုံလင်လင်ပါပြီးသား PyCharm လို Integrated Development Environment(IDE) ဆောဖံ့ဌပဲများနဲ့ ကုဒ်တွေရေးလိုရသလို ပါ့ပေါ့ပါ့ပါ့နဲ့ လိုအပ်မှုပဲ လိုတဲ့ဖီချာအတွက် extensions (plugin လိုလည်းခေါ်ယ်) ထည့်သွင်းရတဲ့ Visual Studio Code (VS Code) လို ကုဒ်အသိဒ်ဘာ (Code Editor) ဆောဖံ့ဌပဲ့ သုံးပြီး ရေးရင်လည်း ရတာပါပဲ။ နောက်ဆုံး ကုဒ်နည်နည်း၊ ဖိုင်နည်းနည်း ရိုးရှင်းတဲ့ ပရိုကရမ်လေးတွေဆုံးရင် Notepad နဲ့ ရေးလိုတောင် ရပါတယ်။ ကုဒ်တွေများမယ် ဖိုင်တွေ များမယ်၊ အသင်းအဖွဲ့လိုက် ပူးပေါင်းရေးရတဲ့ ပရိုကရမ်မျိုးတွေ ဆိုရင်တော့လည်း Notepad လောက်နဲ့ အဆင်မပြနိုင်တော့ဘူးပေါ့။

PyCharm ရော VS Code ထည့်သွင်းနည်းပါ ဖော်ပြပေးပါမယ်။ မိမိ နှစ်သက်ရာ အဆင်ပြောရာ သို့မဟုတ် နီးစပ်ရာ ပရိုကရမ်ဘာ အသိမိတ်ဆွေ အကြုပြုတဲ့ တစ်ခုကို ရွေးချယ်သုံးပါ။ စလေ့လာသူအနေနဲ့ PyCharm ကို အသုံးပြုတာ ပိုအဆင်ပြောမယ်လို ထင်တယ်။ PyCharm သုံးကြည့်လို မိမိကွန်ပျိုးတာ မှာ နေးလွန်းတယ်ဆိုရင် VS Code ကိုစမ်းကြည့်ပါ။ နှစ်ခုလုံး စက်အရမ်းကောင်း/မြင့် ဖို့ မလိုပါဘူး။ တော်ရုံး အတန်အသင့်ကောင်းတဲ့ စက်လောက်နဲ့ အဆင်ပြောပါတယ်။

မိမိကိုယ်တိုင်က ကွန်ပျိုးတာ အသုံးပြုပဲ အခြေခံ အားနည်းပြီး ဖော်ပြပေးထားတဲ့ အတိုင်း တစ်ဆင့် ချင်း အင်စတောလ် လုပ်ဘာလည်း အဆင်မပြောစွာရင် ဒီဘာအုပ်ရဲ့ အောက်ပါ ဖွေစွဲဘွဲ့တ်ခဲ့နဲ့ ယူကျ။

ချုပ်နယ်တွေမှာ ကြည့်ရှုမေးမြန်း အကူအညီ တောင်းဆိုင်ပါတယ်။ ဒါမှာမဟုတ် အတွေးအကြိုးစိတ် နဲ့ စီးပွားရေးမှုပါ။ အကူအညီသူများ အင်စတောလ်လုပ်ပါ။

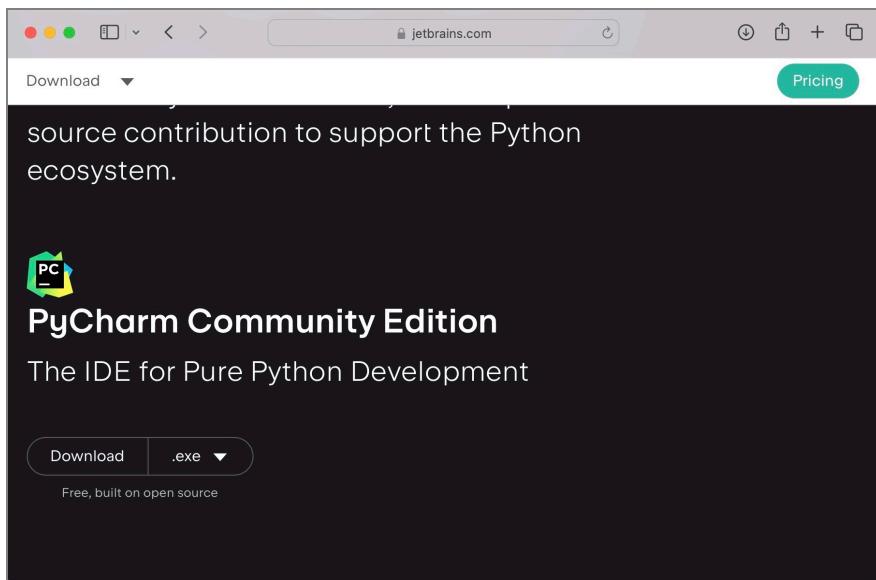
<https://www.facebook.com/bpwp>

<https://www.youtube.com/bpwp>

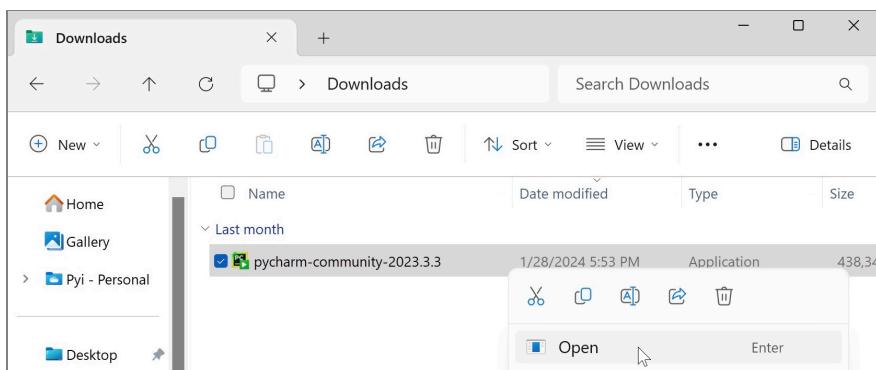
PyCharm အတွက် အရင်ဖော်ပြပေးပါမယ်။ VS Code အတွက် စာမျက်နှာ ?? မှာ ကြည့်ပါ။

Python နှင့် PyCharm IDE ထည့်သွင်းခြင်း

<https://www.jetbrains.com/pycharm/download/> လင့်ကိုဖွင့်ပါ။ ဝဘ်စာမျက်နှာ အောက်ဘက် နည်းနည်း ဆွဲချုပ်ကိုရင် PyCharm Community Edition ဒေါင်းလုပ်ခလုတ်ကို တွေ့ရပါမယ်။ ပုံ (??) ကိုကြည့်ပါ။



ပုံ ၈/၁၀



ပုံ ၉/၂၂

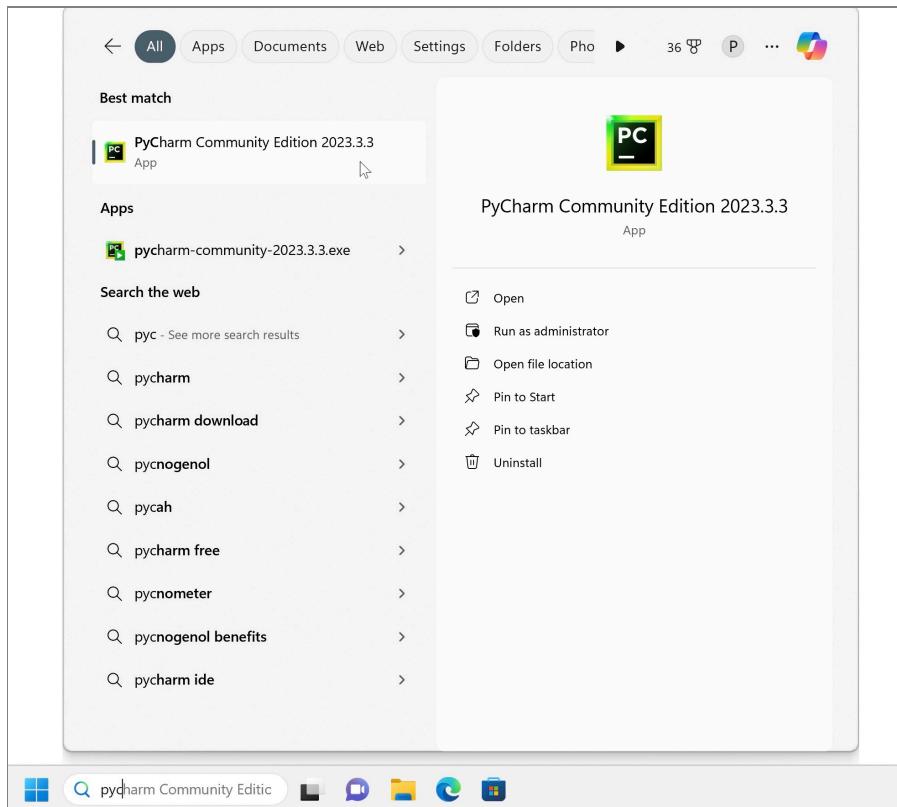
PyCharm Community Edition ကို အောင်လုပ်လုပ်ပါ။ (ဝဘ်စာမျက်နှာ အပေါ်ပိုင်းက ၀၂ သုံးရတဲ့ PyCharm Professional ကို အောင်လုပ် မှားမလုပ်မစိုး သတိပြုပါ)။ အင်စတော်လာဖိုင်ကို ညာကလစ်နှုပ်ပြီး Open လုပ်ပါ (ပုံ ?? ကိုကြည့်ပါ)။ Yes/No မေးတဲ့အခါ Yes နှုပ်ပါ။ Next > ကို နှုပ်၍ ရှုံးကြုံဆက်သွားပြီး နောက်ဆုံးမှာ Install နှုပ်ပြီး ကွန်ဖန်လုပ်ပါ။ အင်စတော်ပြီးသွားရင် Finish နှုပ်ပါ။

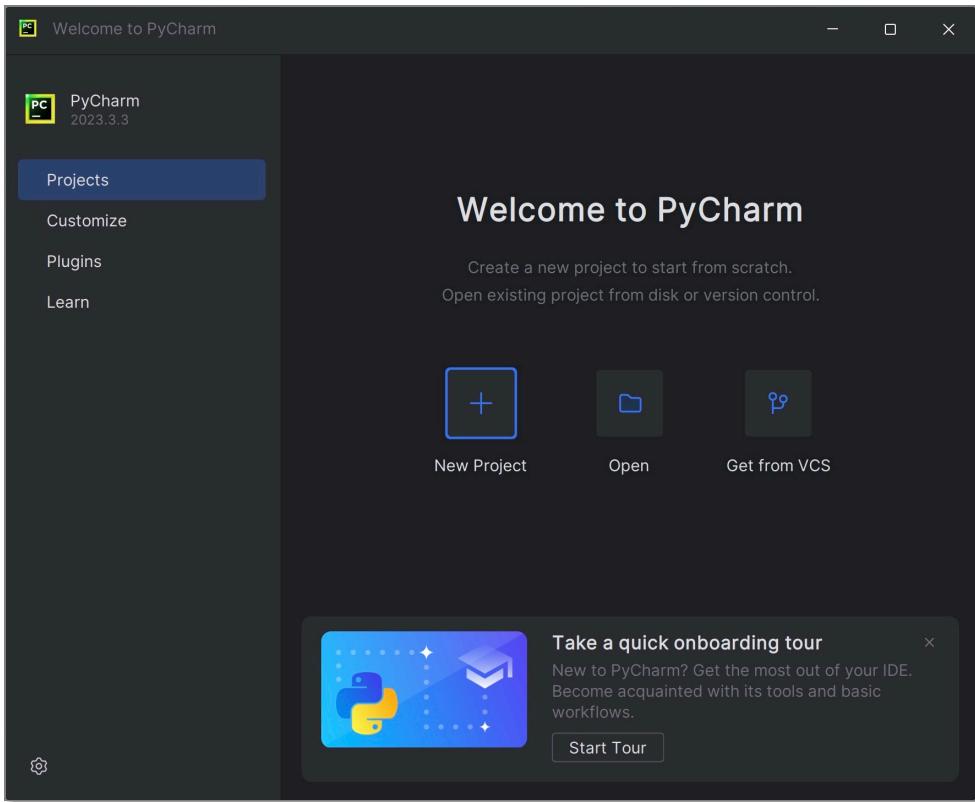
ဝင်းဒီး Taskbar Search ကနေ PyCharm ကိုရှာပြီးဖွင့်ပါ (ပုံ ?? ကို ကြည့်ပါ)။ သဘောတူကြောင်း ကွန်ဖန်လုပ်ခိုင်းရင် ချက်ချက်ဘောက်စ် ချက်ချက်လုပ်ပြီး Continue နှုပ်ပါ။ ဒေတာပို့ချင်လား ထပ်မေးပါလိမ့်မယ်။ Don't Send နှုပ်ပါ။ Welcome စခရင်ကိုပေါ်လာမယ်။ ပုံ (??) မှာ ကြည့်ပါ။

ဒီစာရေးနေဂျာနှင့် လက်ရှိ PyCharm ဟာရှင်းက ၂၀၂၃ ပါ။ သိပ်မကြာခင် ၂၀၂၄ ထွက်ပါတော့မယ်။ အကယ်၍ လက်ရှိမားရှင်းထက် နိမ့်တဲ့ဟာရှင်းတွေကို အောင်လုပ် လုပ်ချင်ရင် အောက်ပါ လင့်ကို သွားပါ။

<https://www.jetbrains.com/pycharm/download/other.html>

ဟာရှင်း ၂၀၂၄/၂၅ ထွက်ပြီးတဲ့ အချိန်မှ ၂၀၂၃ ဟာရှင်းကို လိုချင်ရင် ဝဘ်စာမျက်နှာမှ ရှာပြီး အောင်လုပ် လုပ်ပါ။ ၂၀၂၃ မှာလည်း ဟာရှင်းအခွဲတွေ ရှိပါသေးတယ်။ လက်ရှိအမြင်ဆုံး ဟာရှင်းအခွဲ (ဥပမာ ၂၀၂၃.၃.၃) ကို သုံးတို့ရပါတယ်။ ၂၀၂၄/၂၅ သုံးမယ်ဆုံးရင်လည်း ပြဿနာတော့ မရှိပါဘူး။ Update ဟာရှင်းဖြစ်တဲ့အတွက် ပုံတွေမှာ ပြထားတာနဲ့တော့ ကွာ့ခြားချက်တရုံး၊ ရှိကောင်းရှိနိုင်ပါတယ်။

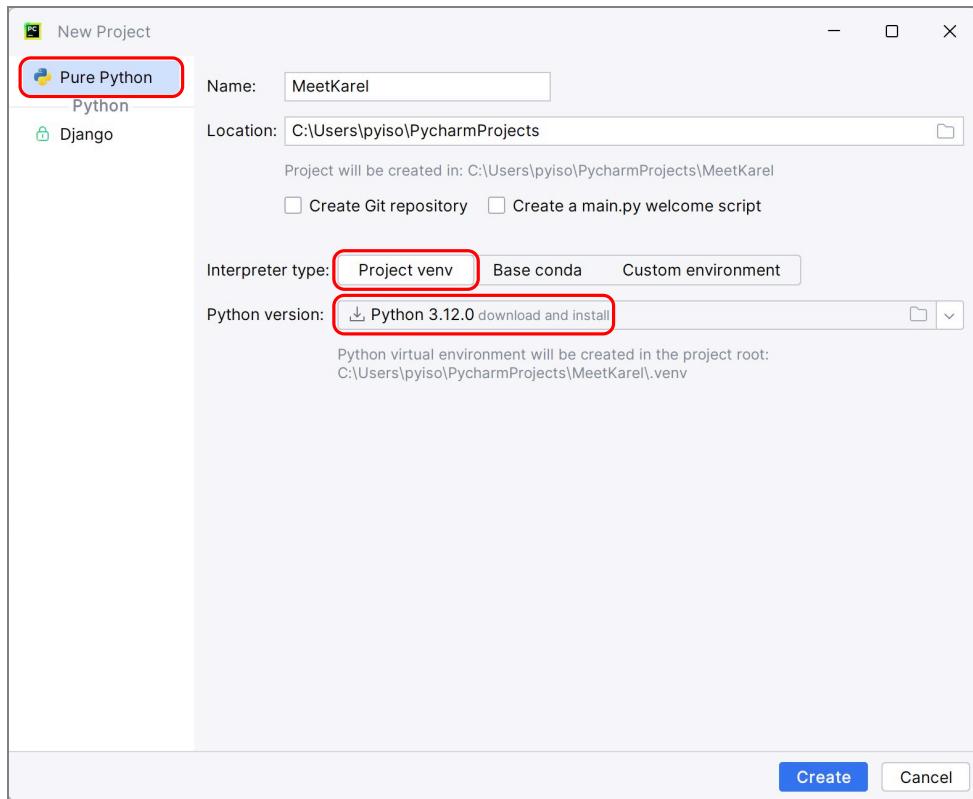




ပုံ ၂/၄

PyCharm IDE ဆိတာဘာလ

PyCharm ဟဲ Python နဲ့ ဆောင်ဝဲ ရေးဖိုအတွက် အထောက်အကူဗြှု Integrated Development Environment(IDE) ဆောင်ဝဲဖြစ်ပါတယ်။ စာစီအရှိက်လုပ်တဲ့အခါ Microsoft Word ကို အသုံးပြု ကြသလိုပဲ Python ကုဒ်ရေးဖို PyCharm ကိုထံးတဲ့ သဘောပေါ့။ PyCharm IDE ၏ Python ပရောဂျက်တွဲ အတွက် အခိုက်ရည်ရွယ်တယ်။ အထောက်အဦးတစ်ခု၊ တံတားတစ်ခု ဆောက်လုပ်တာ ကို ပရောဂျက်လို့ ပြောလေ့ရှိသလို ပရိုကရမ်/ဆောင်ဝဲတစ်ခု တည်ဆောက်တာကိုလည်း ပရောဂျက်လိုပဲ သုံးနှုန်းပါတယ်။ တစ်ဦးတစ်ယောက်တည်း ရေးတဲ့ ပရိုကရမ်အသေးလေးတွဲ အတွက် PyCharm ကို အသုံးပြုနိုင်သလို ပရိုကရမ်မာတွေ အဖွဲ့လိုက်နဲ့ တည်ဆောက်ရတဲ့ ပရောဂျက်ကြီးတွဲ အတွက်လည်း သုံးပါတယ်။ ဒီစာအုပ်မှာတော့ PyCharm ရဲ့ အဆင့်မြင့်ဖိုချာတွေကို အသုံးပြုမှာ မဟုတ်ပါဘူး။ ပရိုကရမ်းမင်း စလေ့လာသူတွေကို လွှာယ်ကူးအဆင်ပြေစေတဲ့ အခြေခံ ဖိုချာတွေလောက်ပဲ အသုံးပြုမှုပါ။



ပုံ ၂/၅ ပရောဂျက် အသစ်ယူခြင်း

PyCharm ပရောဂျက်ဆောက်ခြင်း

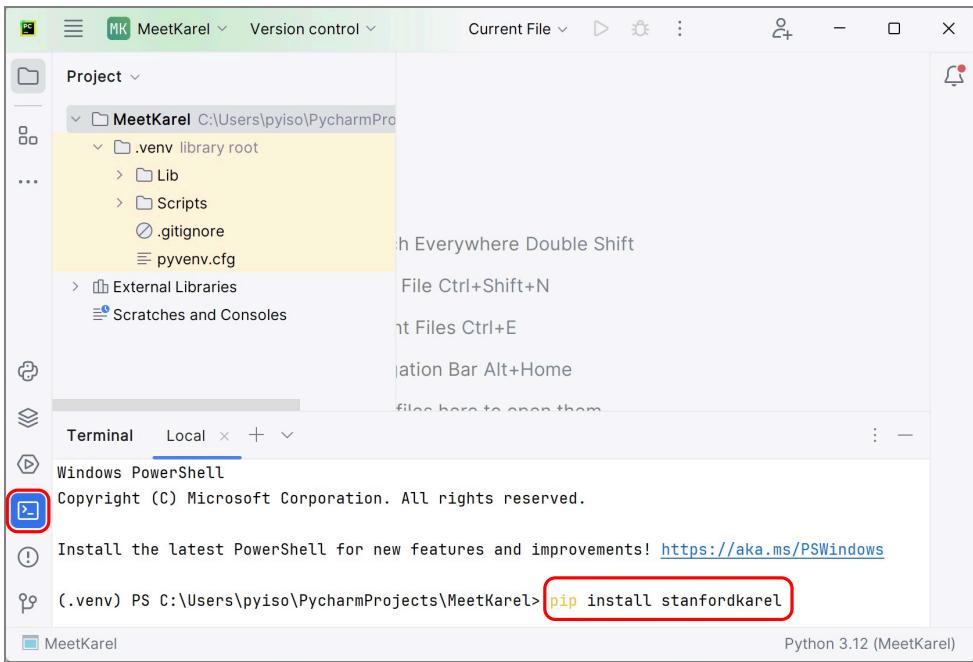
အင်စတောလုပ်ပြီးရင် PyCharm IDE ကိဖွဲ့ပြီး Welcome စခရင်မှ သိမဟုတ် File မိန္ဒီးမှ New Project နှိပ်ပြီး ပရောဂျက် အသစ်ယူပါ (ပြီးခတဲ့ ပုံ ?? ကို ပြန်ကြည့်ပါ)။ နံမည်ကို MeetKarel ပေးပါ။ ပုံ (??) မှာ တွေ့ရတဲ့အတိုင်း Pure Python, Proj venv နဲ့ Python ဘားရှင်း 3.12.xx ကို ရွေးပါ။ xx က အမြားကဏ္ဍး ဖြစ်နေနိုင်တယ်။ အမိက ဘားရှင်း 3.12 သာ ဖြစ်ပါခေါ် အတွက် အမြားကဏ္ဍး ဖြစ်ပါခေါ် Create ခလုတ်နှိပ်ပါ။ ပရောဂျက်အတွက် Python ကို အင်စတောလုပ်ပါလိမ့်မယ်။

မိမိလေ့လေနေတဲ့ အခန်းတစ်ခန်းချင်းစီအတွက် ပရောဂျက်တစ်ခု ဆောက်နိုင်ပါတယ်။ အခန်း (၂) အတွက် Chapter02၊ အခန်း (၃) အတွက် Chapter03 စသည်ဖြင့်။ သက်ဆိုင်ရာအခန်းအလိုက် ကုဒ်ဖိုင်တွေကို ပရောဂျက်တစ်ခုစီမှာ ထားတဲ့အတွက် ဖိုင်တွေများပြီး ရှုပ်ထွေဖောင်းပွဲနေတဲ့ ပြဿနာ မရှိတော့ဘူး။ ကုဒ်ဖိုင်တွေကို ပရောဂျက်တစ်ခုလဲမှာပဲ ဖို့အလိုက်ခဲ့ထားလို့ရပေမဲ့ စလေ့လာသူအနေနဲ့ ပရောဂျက်တစ်ခုချင်း ခဲ့ထားတာလောက် မလွယ်ကူဘူး။

ပရောဂျက် အသစ်ယူတဲ့အခါ တည်နေရာ Location: ကို သူ့ရှိအတိုင်း ထားနိုင်သလို မိမိထားချင်တဲ့နေရာကို ညာဘာက်စွန်း ဖို့အိုင်ကွန်လေးနှိပ်ပြီး ပြောင်းလို့ရတယ်။

stanfordkarel လိုက်ဘရီ အင်စတောလုပ်ခြင်း

ပုံ (??) မှာ အနီရောင် ဝိုင်းပြထားတဲ့ အိုင်ကွန်ကို နှိပ်ပြီး Terminal ကိဖွဲ့ပါ။ Terminal မှာ အောက်ပါ ကွန်မန်းဖြင့်



ပုံ ၂/၆ Karel လိုက်ဘရီ အင်စတောလုပ်ခြင်း

```
pip install stanfordkarel
```

ကားရဲလိုက်ဘရီကို အင်စတောလုပ်ပါ။ ပုံ (??) မှာ အနိရောင် ဝိုင်းပြထားပါတယ်။ ခက္ကာတဲ့အခါ အခုလို မက်ဆွဲချုံတွေ ကျလာပါလိမ့်မယ်။

```
Collecting stanfordkarel
```

```
  Downloading stanfordkarel-0.2.7-py3-none-any.whl (51 kB)
          ━━━━━━━━━━━━━━━━ 51.9/51.9 kB 443.1 kB/s
    eta 0:00:00
```

```
Installing collected packages: stanfordkarel
```

```
Successfully installed stanfordkarel-0.2.7
```

```
[notice] A new release of pip is available: 23.2.1 -> 24.0
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
(.venv) PS C:\Users\pyiso\PycharmProjects\MeetKarel>
```

ဟိုက်လိုက်ပြထားတဲ့ မက်ဆွဲချုံ တွေ့ရရင် အင်စတောလုပ်တာ အောင်မြင်လိုပါ။

မှတ်ချက်။ ပေါ်မြတ်မယ်။ ပရောဂျက် အသစ်ဆောက်တိုင်း လိုအပ်တဲ့ လိုက်ဘရီကို တစ်ခါတပ်ပြီး အင်စတောလုပ်ရပါမယ်။ ကားရဲလိုအခန်း ပရောဂျက်တစ်ခုစီအတွက် stanfordkarel လိုက်ဘရီကို အထက်ဖော်ပြပါအတိုင်း အင်စတောလုပ်စိုးလိုပါ။

နမူနာ ကားရဲလ် ကဗ္ဗာနှင့် ပရိဂရမ်ကုဒ် ဖိုင်များထည့်ခြင်း

meet_karel.zip ဖိုင်ကို ဒီလင့် <http://tinyurl.com/3mmmm9c7j> ကနေ ဒေါင်းလုပ်လုပ်ပါ။ ငှါး zip ဖိုင်ကို extract လုပ်ပါ။ meet_karel နံမည်နဲ့ ဖိုဒါတစ်ခု ရလာပါမယ်။ ငှါးဖိုဒါထဲမှ အောက်ပါ worlds ဖိုဒါနှင့် .py ဖိုင်အားလုံးကို ကော်ပီလုပ်ပါ။

- worlds
 - ▶ meet_karel.w
 - ▶ move_beeper_to_other_side.w
- meet_karel.py
- move_beeper_to_other_side.py
- world_editor.py

MeetKarel ပရောဂျက်ထဲတွင် ကူးထည့်ပါ။ ပင်မ ပရောဂျက် MeetKarel (ပု ?? မှာ မြှားပြထား) ပေါ်မှာ ညာကလစ်နှင့်ပြီး Paste လုပ်ရမှုပါ။ ကော်ပီကူးထည့်လိုက်တဲ့ ဖိုင်တွေက ပုံမှာ တွေ့ရတဲ့အတိုင်း MeetKarel ဖိုဒါအောက်မှ ရှိသင့်ပါတယ်။

အခန်းအလိုက် နမူနာ ကုဒ်ဖိုင်တွေ ထည့်ပေးထားတဲ့ .zip ဖိုင်တွေကိုလည်း အထက်ပါအတိုင်း အလားတူ လုပ်ရပါမယ်။ ပရောဂျက်အသစ်ဆောက်၊ .zip ဖိုင်ကို ဖြည့်၊ ရလာတဲ့ ဖိုဒါထဲက ဖိုင်တွေကို ပင်မ ပရောဂျက် ဖိုဒါထဲ ကော်ပီကူးထည့် ရုံပါပဲ။

meet_karel.py ဖိုင်ကို ကလစ်နှစ်ချက်နှင့် ဖွေ့စွဲပါ။ ပု (??) မှာ အနိဂုံးထားတဲ့ ကုဒ်အယ်ဒီတာ (code editor) ပွင့်လာပါမယ်။ အဲဒီ မူရင်းဖိုင်မှာ အောက်ပါအတိုင်း ဆက်လက် ပြင်ဆင်ဖြည့်စွက်ပါ။

```
from stanfordkarel import *

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

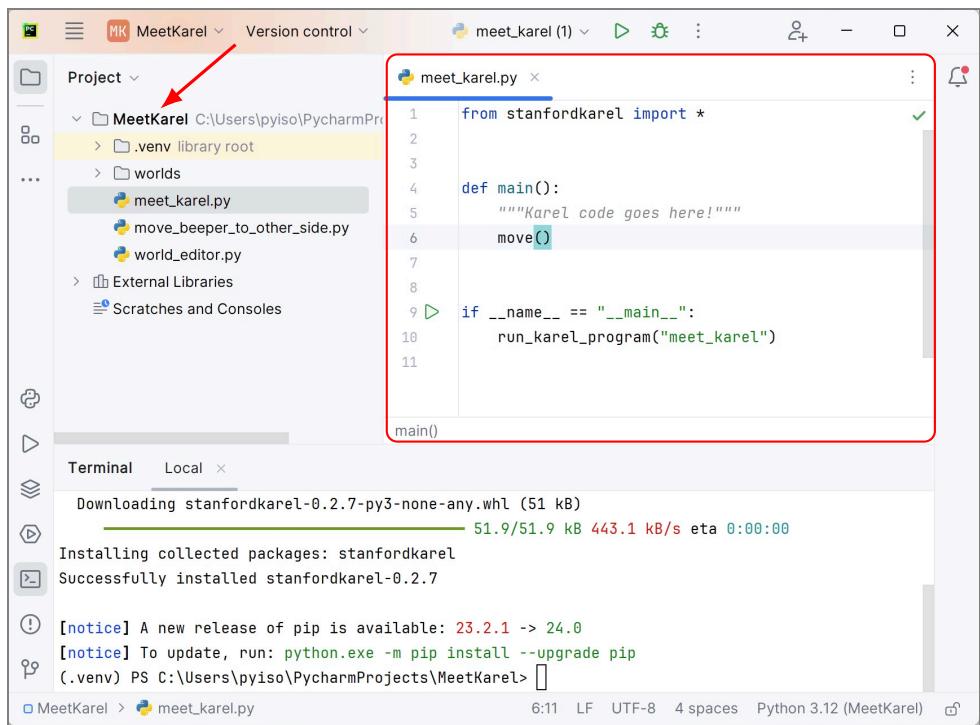
    turn_left()
    turn_left()
    turn_left()

    move()
    put_beeper()

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

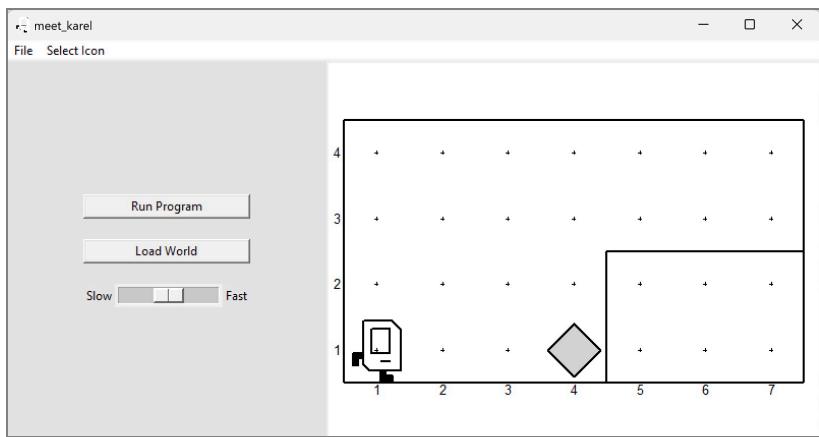
run ထားတဲ့ ပရိုကရမ်တစ်ခုကို တစ်ခါထပ် run ရင် Cancel (သို့) Stop and Rerun လုပ်မှုလား မေးတယ်။ Stop and Rerun လုပ်ရပါတယ်။

meet_karel.py ဖိုင်ကို ညာကလစ်နှီးမြို့ Run 'meet_karel' လုပ်ပါ (အယ်ဒီတာ ဧရိယာမှာ ညာကလစ်နှီးမြို့ Run 'meet_karel' လုပ်လို့လည်း ရပါတယ်)။ ပုံ (??) က ကားရဲ့လ် ပရိုကရမ် တက်လာရင် Run Program ခလုတ်ကိုနှိပ်ပါ။ ဘိပါကို ကားရဲ့လ်က ရွှေ့ပေးပါလိမ့်မယ်။



ပုံ ၂/၄

၁၃၉



ပုံ က/ဂ

ဆင်းတက်စံအမှားများ

အကယ်၍ ပရိုဂရမ် run လို့မရရင် ကုသရေးတာမှားနေလို့ ဖြစ်နိုင်တယ်။ မိမိရေးထားတာကို စာမျက်နှာ (??) က ပရိုဂရမ်ကုဒ်နဲ့ မိမိသူ့ စစ်ဆေးကြည့်ပါ။ PyCharm အယ်ဒီတာမှာ အနီလိုင့်တွန်လေးတွေ (ပုံ ??) ပြတဲ့နေရင် အဲဒီနေရာတွေမှာ ဆင်းတက်စံမှားနေလို့ (သို့) လိုက်ဘရီမထည့်ရသေးလို့ပဲ။

လိုက်ကွင်းကျို့နေတာက အဖြစ်များတဲ့ အမှားပါ။ ကျို့ခဲ့လို့ မရပါဘူး။ အင်ဒန်တေးရှင်း (indentation) လုပ်ရမဲ့နေရာမှာ မလုပ်ထားရင်လည်း ပြဿနာဖြစ်တယ်။ move, turn_left တွေကို ဘေးမျဉ်း ညာဘက်ခွဲပြီး အင်ဒန်လုပ်ပေးရမယ်။ အဲဒီတွေ ကရုမစိုက်ပိုရင် ဆင်းတက်စံအမှားဖြစ်ပြီး ပရိုဂရမ် run လို့ မရနိုင်ဘူး။

Terminal မှာ ထုတ်ပေးတဲ့ မက်ဆွဲချုပ်တွေကို ကြည့်ပြီးတော့လည်း ဘာပြဿနာဖြစ်နေလဲ မှန်းဆ လို့ရနိုင်တယ်။ ဘာကြောင့်ဖြစ်နိုင်လဲ ဆက်စပ်စဉ်းတော့လို့ ရတယ်။ ဥပမာ ဖြစ်တဲ့ပြဿနာအလိုက် အခုလို တွေ့ရပါမယ်။

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 6
```

```
    move(
```

```
    ^
```

```
SyntaxError: '(' was never closed
```

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 7
```

```
    move()
```

```
IndentationError: unexpected indent
```

Traceback (most recent call last):

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 1,
      in <module>
```

```
File "c:\Users\pyiso\PycharmProjects\MeetKarel\meet_karel.py", line 6
    move()
IndentationError: unexpected indent
```

Process finished with exit code 1

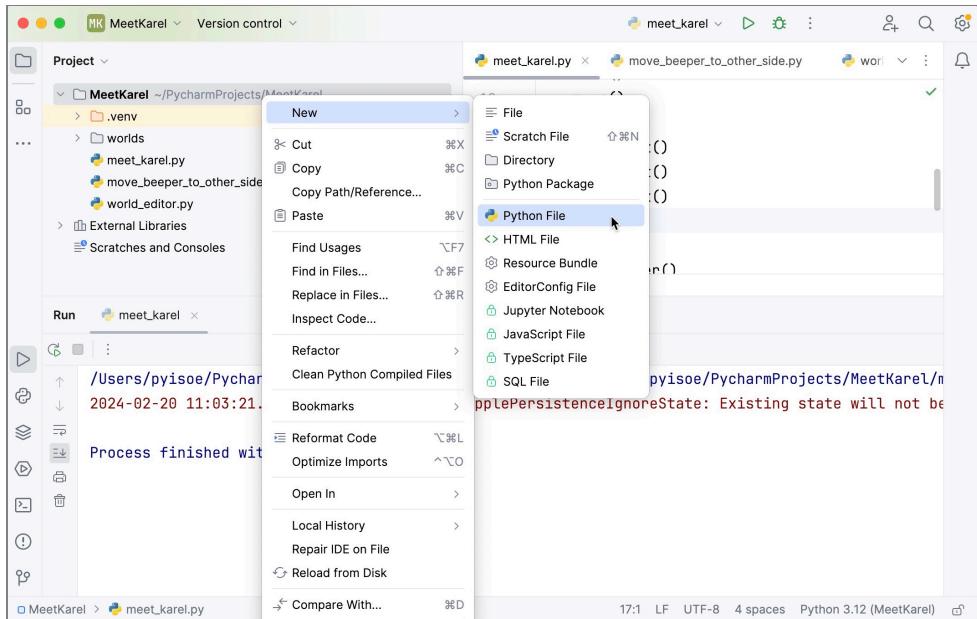
cc6

```
from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
```

Python ဖိုင် အသစ်ယူခြင်း

MeetKarel ပင်မ ပရောဂျက်ဖိုဒ်အပေါ်မှာ ညာကလစ်နှုပ်ပြီး Python ဖိုင် အသစ်ယူနိုင်ပါတယ်။ Python ဖိုင်တွေက .py အိုင်စာန်းရှင်းနှုပါ။ ကားရဲလ်ပရိုဂရမ်တစ်ခုကို Python ဖိုင်တစ်ခု ထားပါမယ်။ ပင်မ ပရောဂျက်ဖိုဒ်အောက်မှာပဲ တိုက်ရိုက်ရှိရပါမယ်။

နောက်ပိုင်း အဆင့်မြင့်လာရင် ပရိုဂရမ်တစ်ခုအတွက် ပရောဂျက်တစ်ခု ထားနိုင်တယ်။ ကုဒ်ဖိုင်တွေ အပြင် ပရိုဂရမ်အတွက် လိုအပ်တဲ့ ရုပ်ပုံတော့၊ အခြားဖိုင်တွေ (config ဖိုင်၊ setting ဖိုင် စသည်ဖြင့်) လည်း ပါနိုင်တယ်။ ပင်မပရောဂျက် အောက်မှာပဲ ဖိုင်တွေက တိုက်ရိုက်ရှိဖို့လည်း မလိုတော့ဘူး။ ဆက်စပ်ရာ ဖိုင်တွေကို အမျိုးအစားအလိုက် ဖန်ရှင်အလိုက် ဖို့ဒါတွေခဲ့ပြီး စနစ်ကျ စီစဉ်ဖွဲ့စည်း ထားရမှာပါ။ ပရောဂျက်တစ်ခုမှာ ဖိုင်တွေကို စနစ်တကျ စုဖွဲ့ထားဖို့ အရေးကြီးပါတယ်။



ပုံ က/၁၀

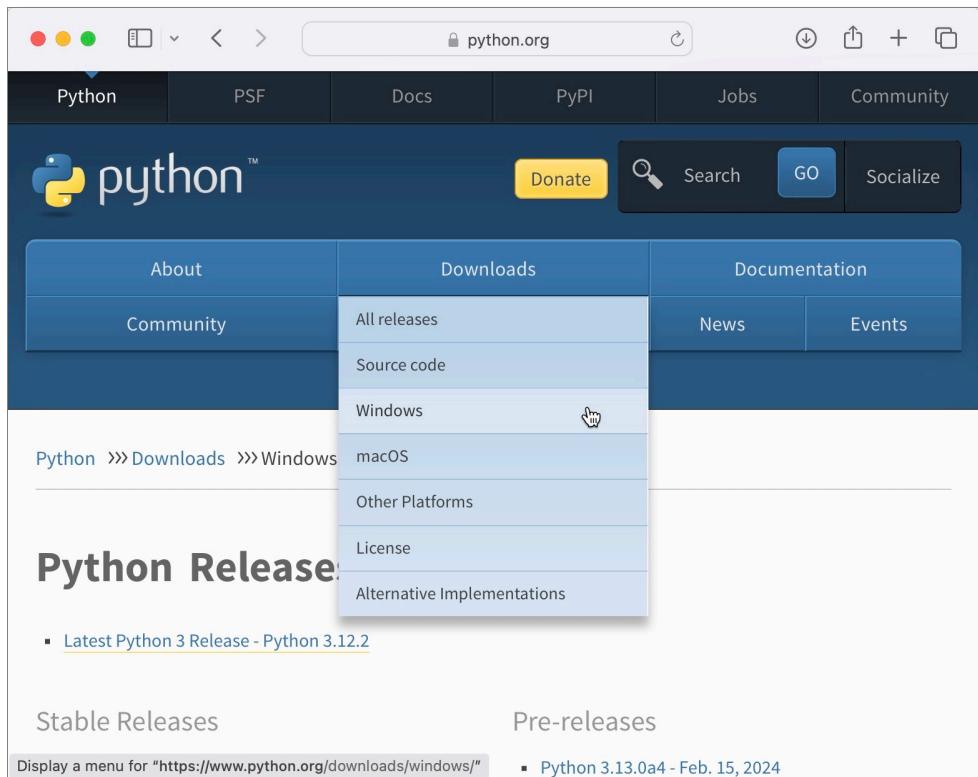
Visual Studio Code နှင့် Python အင်စတေလုပ်ခြင်း

Visual Studio Code(VS Code) ဟာ ပရိုဂရမ်မာအများစုံ ကြိုက်နှစ်သက်တဲ့ မောဒန် ကုဒ်အယ်ဒီတာ တစ်ခုပါ။ Python, JavaScript, C++ စတဲ့ programming language အမျိုးမျိုးအတွက် အသုံးပြု နိုင်ပါတယ်။

PyCharm မှာတော့ ပရောဂျက်အသစ်ဆောက်ရင် Python ပါ တစ်ခါတည်း ဒေါင်းလုဒ်လုပ်ပြီး အင်စတေလုပ်လို့ရတယ်။ VS Code နဲ့ Python ရေးမယ်ဆိုရင် Python Programming Language ကို သီး၌ ဒေါင်းလုဒ်လုပ်ပြီး အင်စတေလုပ်ရပါမယ်။

Python အင်စတေလုပ်လုပ်ခြင်း

ဒီလုပ် <https://www.python.org/> ကိုဖွင့်ပါ။ Download မီးနဲ့ Windows နှိပ်ပါ (ပုံ ?? ကို ကြည့်ပါ)။ Apple ကွန်ပျူးတာအတွက် ဆိုရင် macOS ရွေးပါမယ်။ လူများစုံတဲ့ မိုက်ခရိုဆော့ဖို့ ဝင်းဒိုးအတွက် အင်စတေလုပ်နည်းကို အခိုက်ပြေားမှာပါ။ ပုံ ?? မှာလို့ တွေ့ရပါလိမ့်မယ်။ အင်စတော်လာ ဗားရှင်း 3.12 ထဲက လက်ရှိအမြင့်ဆုံး ကိုရွေးပါ။ ဒီစာရေးချိန်မှာ 3.12.2 ဟာ အမြင့်ဆုံးဗားရှင်းပါ။ **Windows Installer (64-bit)** ကိုနိုင်ပြီး ဒေါင်းလုဒ်လုပ်ပါ။



ပုံ ၂/၀၀

ဒေါင်းလုဒ်ပြီးရင် အင်စတော်လာဖိုင်ကို ညာကလစ်နိုင်ပြီး Run as administrator လုပ်ပါ။ ပုံ (??) ကိုကြည့်ပါ။ ပုံ (??) မှာလို့ ဒိုင်ယာလော် ဆောက်တဲ့ ပွင့်လာပါမယ်။ အနိဂုင်းထားတဲ့ ချက်ချွဲဆောက်နှစ်ခုကို ချက်ချွဲလုပ်ပြီး Install Now နှိပ်ပါ။ အင်စတေလုပ်ပြီးလို့ Setup was successful ပေါ်လာရင်

Note that Python 3.12.2 cannot be used on Windows 7 or earlier.

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- **Download Windows installer (64-bit)**
- Download Windows installer (ARM64)

▪ Python 3.12.1 - Dec. 8, 2023

Note that Python 3.12.1 cannot be used on Windows 7 or earlier.

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)

▪ Python 3.13.0a2 - Nov. 21, 2023

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)
- Download Windows installer (ARM64)

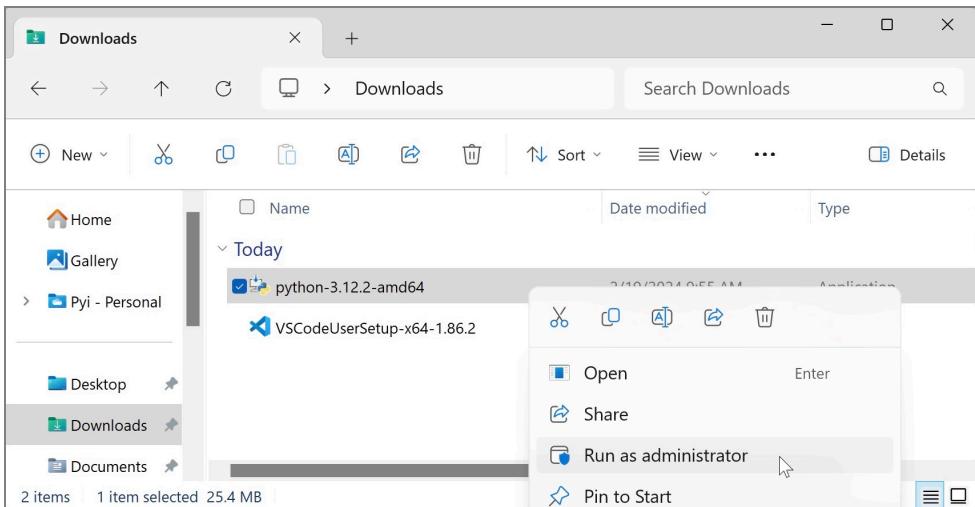
▪ Python 3.13.0a1 - Oct. 13, 2023

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)
- Download Windows installer (ARM64)

ပုံ ၂/၀၂

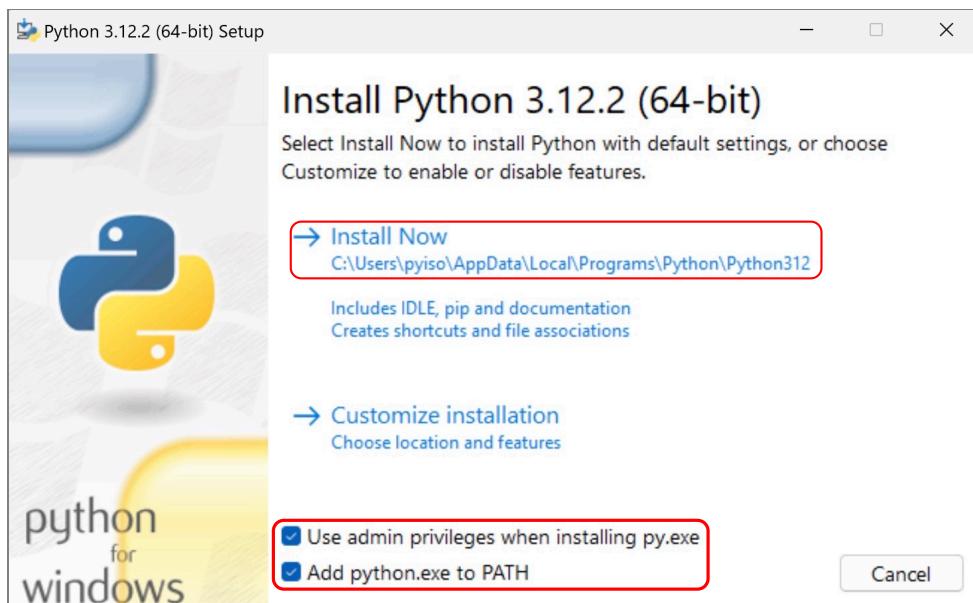
Close ခလုတ်နိပ်ပြီး ပိတ်ပါ။

ဝင်းခွဲး command prompt (cmd) မှာ `python --version` run ရင် ပုံ (??) မှာလို အင်စေတော်လုပ်ထားတဲ့ ဟာရှင်းကို ပြပေးသင့်ပါတယ်။



ပုံ ၃/၀၃

፩፭



፩ ተ/ዕና

The screenshot shows a Microsoft Windows Command Prompt window. The title bar says "Command Prompt". The window displays the following text:
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.
C:\Users\pyiso>python --version
Python 3.12.2
C:\Users\pyiso>

፩ ተ/ዕና

VS Code အင်စတောလ်လုပ်ခြင်း

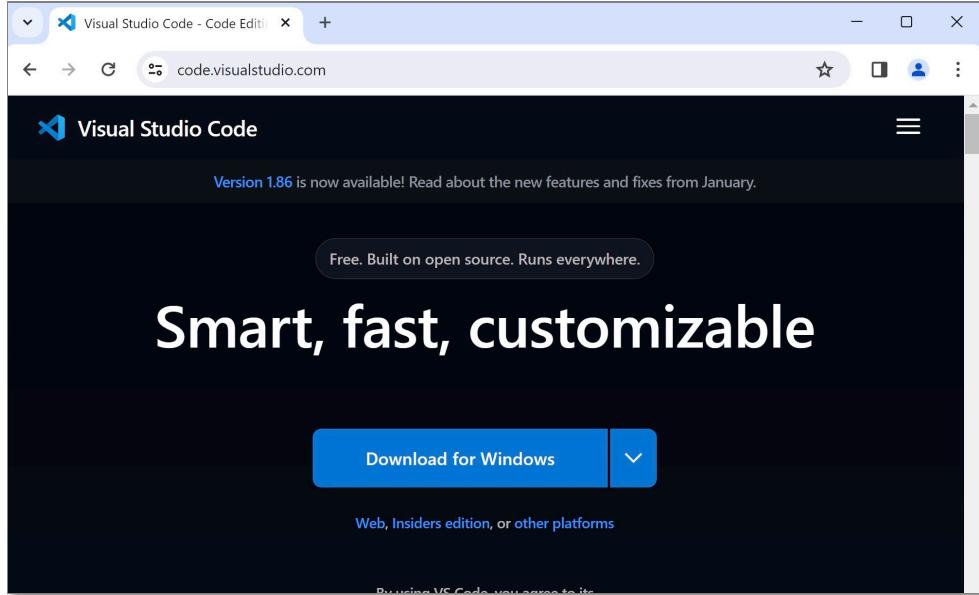
အခုနိရင် Python programming language အောင်မြင်စွာ ထည့်ပြီးသွားပါပြီ။ VS Code အင်စတောလ် ဆက်လုပ်ပါမယ်။ အင်စတောလ်လာ ဒေါင်းလုပ်လုပ်ရန် ဝော်စာမျက်နှာကို အောက်ပါလုပ်ခြင်းဖြင့် အောင်မြင်စွာ ထည့်ပြီးသွားပါပြီ။

<https://code.visualstudio.com>

မှတစ်ဆင့် သွားပါ။ ပုံ (??) ဝော်စာမျက်နှာကို တွေ့ရပါမယ်။ [Download for Windows](#) နှင့် ဒေါင်းလုပ်လုပ်ပါ။

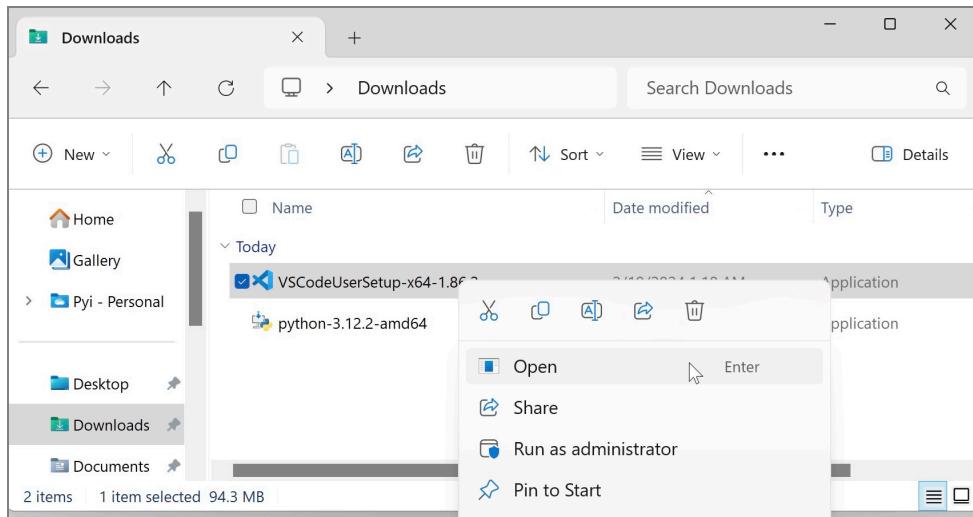
ပြီးတဲ့အခါ အင်စတောလ်လုပ်ကို ညာကလစ်နှင့်ပြီး ဖွင့်ပါ (ပုံ ?? မှာ ပြထားပါတယ်)။ အင်စတောလ် ဒိုင်ယာလော်ကောက်စ် ပွင့်လာရင် [I accept the agreement](#) ကို ချက်ချွဲပါ၍ Next > တစ်ခုပြီး တစ်ခု ဆက်နှုပ်သွားပြီး နောက်ဆုံးမှာ Install နှင့်ပါ။ အင်စတောလ်လုပ်နေတာကို ခကာတောင့်ပြီး၊ ပြီးသွားရင် Finish နှင့်ပါ။

Welcome စခရင်ကို ပုံ (??) လို တွေ့ရပါမယ်။ [Dark Modern](#) (သို့) [Light Modern](#) နှစ်သက်ရာ သီးမှတ် ရွှေးပါ။ ဒါဆိုရင် VS Code လည်း အင်စတောလ် လုပ်ပြီးသွားပါပြီ။ ကားရဲလ်ပရိုဂရမ် ရေးဖို့ ဆက်လုပ်ရပါမယ်။

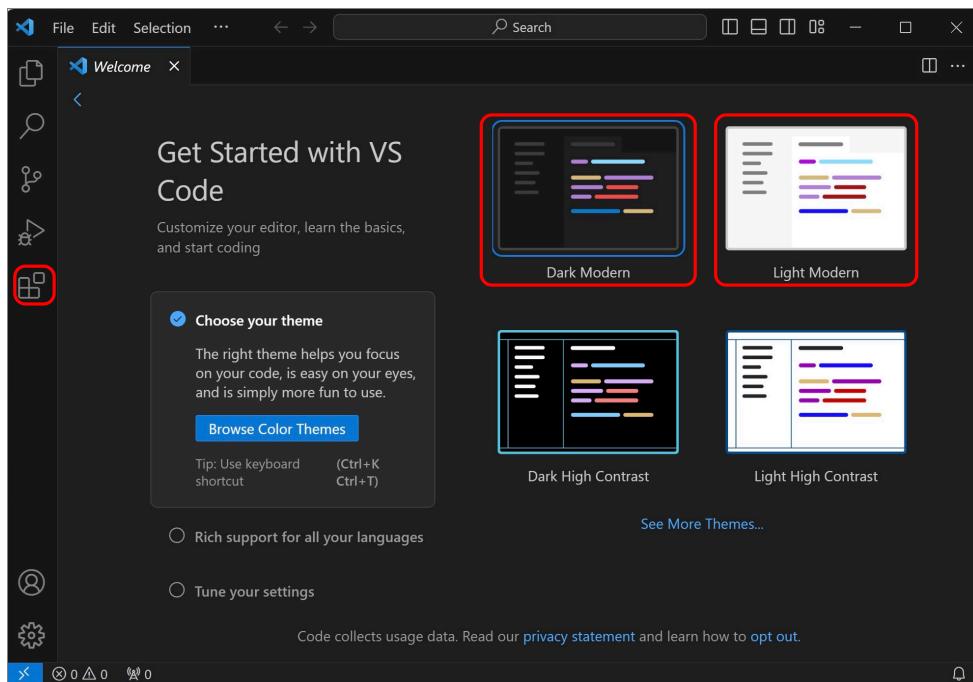


ပုံ က/၁၆

၃၆



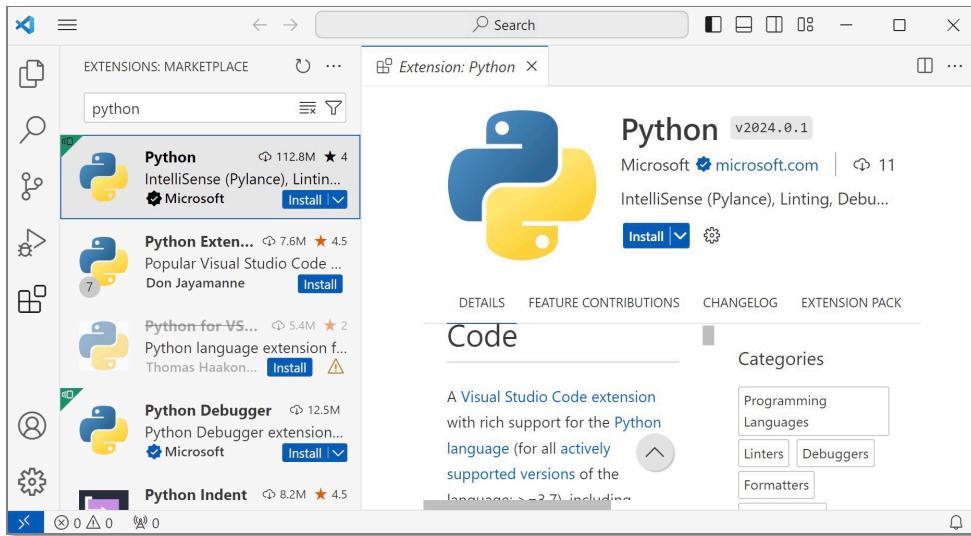
ပုံ ၂/၀၇



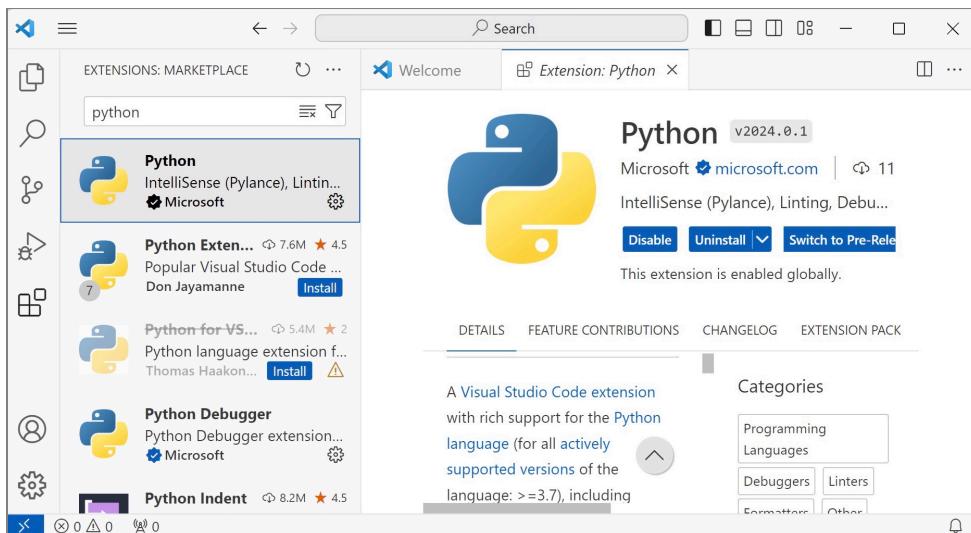
ပုံ ၃/၀၈

VS Code Python Extension ဆွဲခြင်း

Python ပရိဂုဂ္ဂများဖို့အတွက် VS Code က ဒီအတိုင်းဆိုရင် သိပ်အဆင်မပြေသေးပါဘူး။ Python အတွက် extension အင်စတောလ လုပ်ပေးရပါအံ့ဖယ်။ ပုံ (??) ဘယ်ဘက်တောင်နားမှာ အနိဂုံးထားတဲ့ အိုင်ကွန်လေးကို နှင့်ပါ။ Python extension ကိုရှာပါ။ ပုံ (??) မှာ ဘယ်လိုရှာရမလဲ ပြထားပါတယ်။ ပုံမှာတော်ရတဲ့ Microsoft က ထုတ်ပုံတဲ့ extension ကို အင်စတောလလုပ်ပါ။ အောင်မြင်ရင် ပုံ (??) မှာလို ဖြစ်သွားပါမယ်။ Python နဲ့ VS Code ကိစ္စတော့ ပြီးသွားပြီ။ ကားရဲလ် example run ဖို့ ဆက်လုပ်ရမယ်။



ပုံ ၃/၁၉



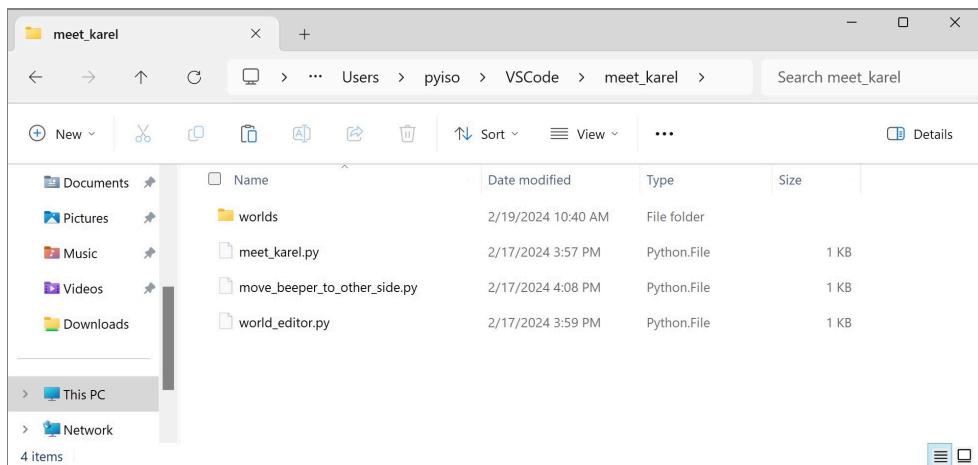
ပုံ ၃/၂၀

နမူနာ ကားရဲလ် ကဗ္ဗာနှင့် ပရိဂရမ်ကုဒ် ဖိုင်များထည့်ခြင်း

meet_karel.zip ဖိုင်ကို ဒီလင့် <http://tinyurl.com/3mm9c7j> ကနေ ဒေါင်းလုပ်လုပ်ပါ။ ငါး zip ဖိုင်ကို extract လုပ်ပါ။ meet_karel နံမည်နဲ့ ဖိုဒါတစ်ခု ရလာပါမယ်။ ဖိုဒါထဲ ဝင်ကြည့်ရင် အောက်ပါ အတိုင်း ရှိသင့်ပါတယ်။

■ worlds

- ▶ **meet_karel.w**
- ▶ **move_beeper_to_other_side.w**
- **meet_karel.py**
- **move_beeper_to_other_side.py**
- **world_editor.py**



ပုံ က/၂၁

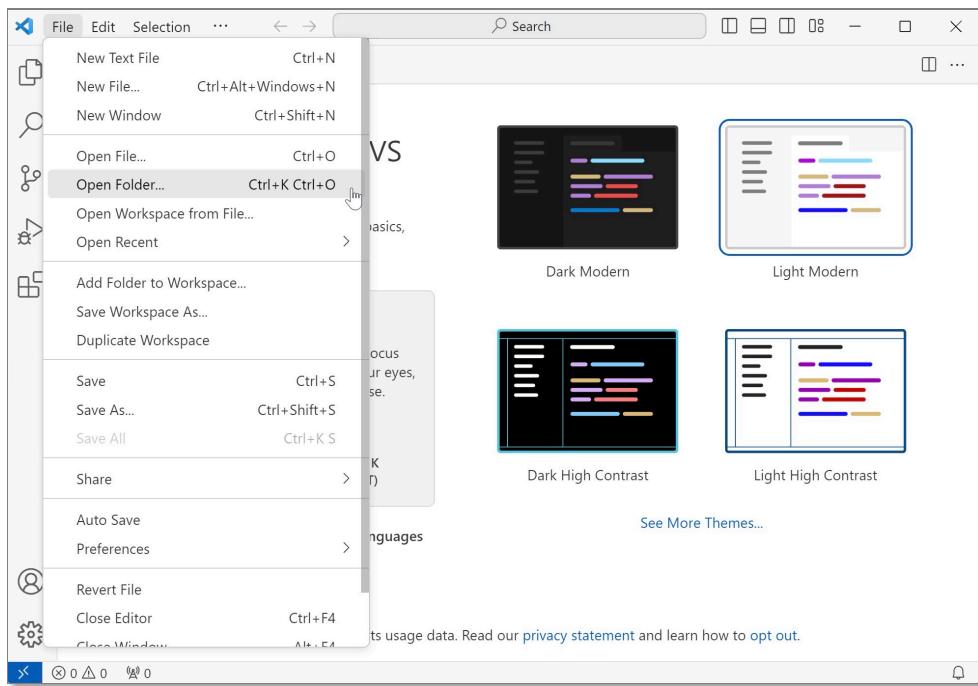
VS Code အတွက် ဖိုဒါတစ်ခုကို မိမိအတွက် အဆင်ပြေမဲ့နေရာမှာ သီးသန့်တည်ဆောက် ထားသင့်တယ်။ ဥပမာ

C:\Users\yourname\VS Code

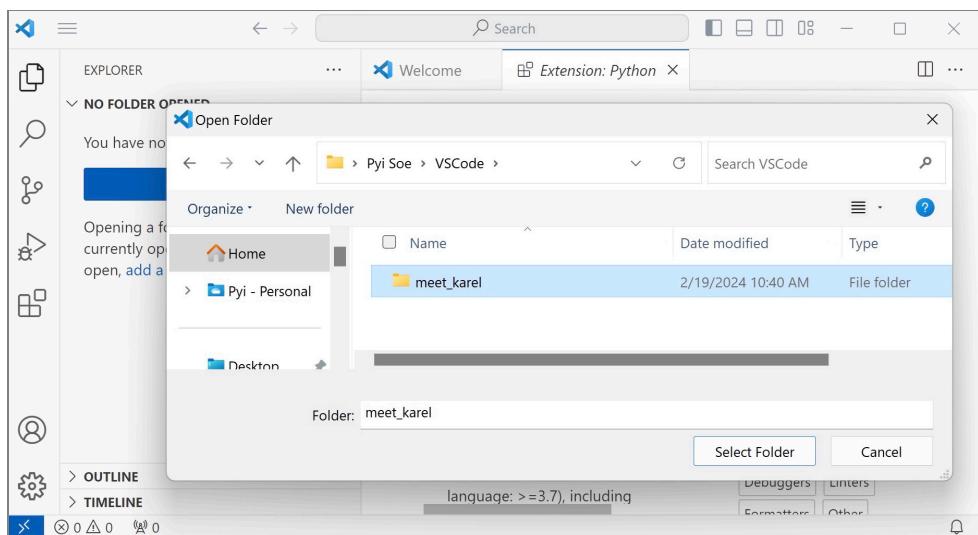
မိမိ လက်ရှိ Home ဖိုဒါကို C: drive ရဲ **Users** ဖိုဒါထဲမှာ တွေ့နိုင်ပါတယ်။ **Win + R** ကြောက်နှင့်ပြီး %userprofile% ရိုက်ထည့်၍ Ok လုပ်ပြီး Home ဖိုဒါကို သွားနိုင်ပါတယ်။ အကယ်၍ မသွားတတ်ရင်လည်း ပြဿနာမရှိပါဘူး။ ကိုယ့်အတွက် လွယ်ကူမဲ့ Desktop, Downloads, Documents တစ်ခုခုထဲမှာ VS Code အတွက် ဖိုဒါတစ်ခု ထားလည်းရတယ်။

meet_karel ဖိုဒါ (.zip ဖိုင်မဟုတ်ပါ) ကို အထက်ပါအတိုင်း အသစ် ဆောက်ထားတဲ့ VS Code သီးသန့်ဖို့တဲ့ကို ကော်ပိကူးထည့်ပါ။ ငါး **meet_karel** ဖိုဒါကို VS Code **File** ပြန်မှ **Open Folder** နှင့်ပြု၍ ဖွံ့ဖြိုးပါ။ ပုံ (??) တွင်ကြည့်ပါ။

အခန်းအလိုက် နမူနာ ကုဒ်ဖိုင်တွေ ထည့်ပေးထားတဲ့ .zip ဖိုင်တွေကိုလည်း အထက်ပါအတိုင်း လုပ်ရပါမယ်။ .zip ဖိုင်ကို ဖြေည့်၊ ရလာတဲ့ ဖိုဒါကို သီးသန့်ဖို့တစ်ခုထဲကို ကော်ပိကူးထည့်၊ VS Code နဲ့ အဲ ဒီဖိုဒါကို ဖွံ့ဖြိုးပါပဲ။



ပုံ က/JJ

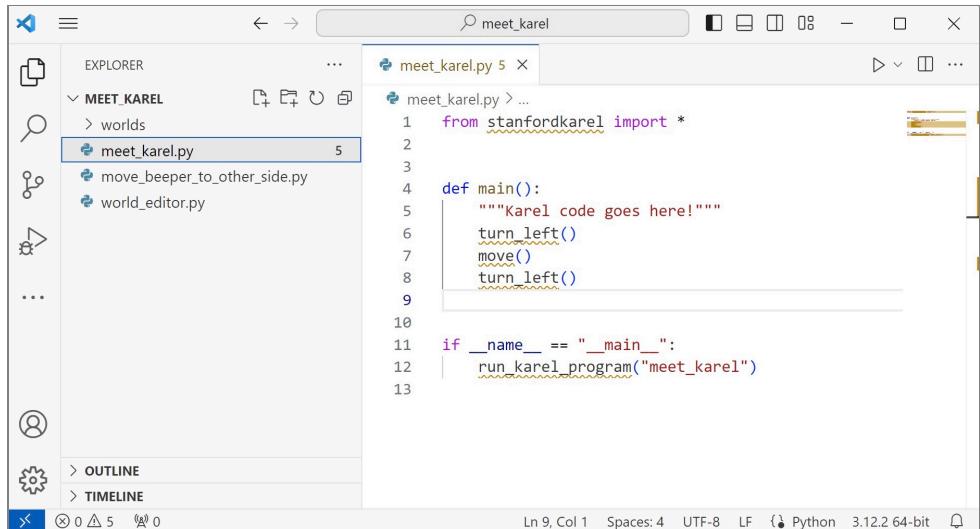


ပုံ က/JR

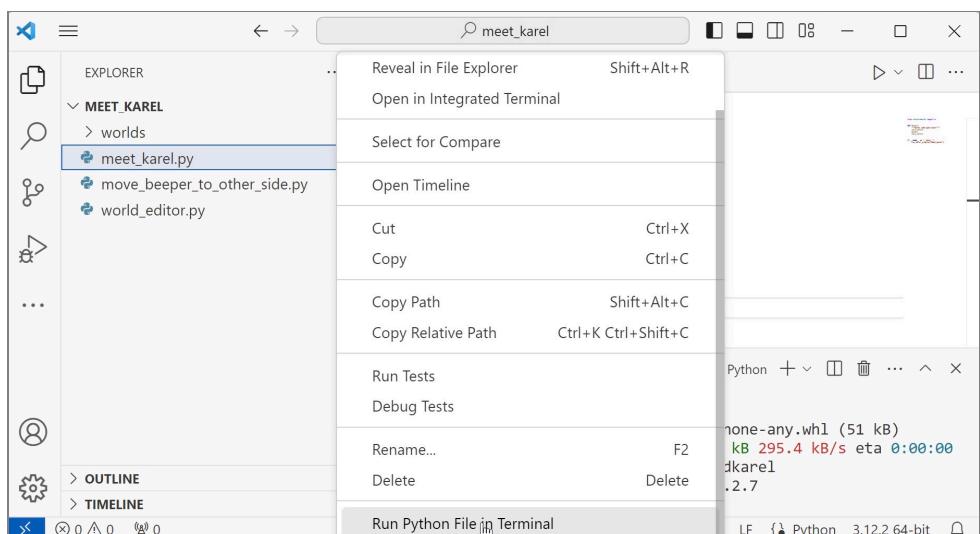
၁၅၁

stanfordkarel လိုက်ဘရီ အင်စတေလ်လုပ်ခြင်း

`meet_karel.py` ဖိုင်ကို ကလစ်နှစ်ချက်နိုင် ဖွံ့ဖြိုးပါ။ ကုဒ်အယ်ဒီတာ ပွင့်လာမယ် (ပု ??)။ အဲဒီ ကုဒ် အယ်ဒီတာပေါ် (သို့) `meet_karel.py` ဖိုင်ကို ညာကလစ်နိုင်ပြီး `Run Python File in Terminal` လုပ်ပါ (ပု ??)။ Terminal ပွင့်လာပြီး အယ်ရာမက်ဆောင်တွေ ပြလိမ့်မယ်။ ပု (?) မှာကြည့်ပါ။ ကားရဲ့လပ် ရှိရာမေးတွက် လိုအပ်တဲ့ stanfordkarel လိုက်ဘရီ အင်စတေလ် မလုပ်ရသေးပါဘူး။ ဒါကြောင့် အယ်ရာဖြစ်နေတာ။



ပု ၃/၂၄



ပု ၃/၂၅

ခုနကပွင့်လာတဲ့ Terminal မှာပဲ အောက်ပါကွန်မန်းကို run ပြီး stanfordkarel လိုက်ဘရီကို အင်စတေလ်လုပ်ပါ။

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project structure under "MEET_KAREL". The file "meet_karel.py" is selected, indicated by a blue icon and highlighted text.
- Code Editor (Center):** Displays the content of "meet_karel.py".

```
from stanfordkarel import *
def main():
    ...
```
- Terminal (Bottom):** Shows the command-line output of running the script.

```
PS C:\Users\pyiso\VSCode\meet_karel> & C:/Users/pyiso/AppData/Local/Programs/Python/Python312/python.exe c:/Users/pyiso/VSCode/meet_karel/meet_karel.py
Traceback (most recent call last):
  File "c:/Users/pyiso/VSCode/meet_karel/meet_karel.py", line 1,
    in <module>
      from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
PS C:\Users\pyiso\VSCode\meet_karel> pip install stanfordkarel
```

୧୯

```
pip install stanfordkarel
```

ပုံ (??) မှာ အနိဂင်းထားတာကို ကြည့်ပါ။ အဲဒီအတိုင်းရိုက်ထည့်ပြီး Enter ကိုနိပ်ပါ။ ခေါ်ကာတဲ့အခါ အခုလို မက်ဆွဲချကြတွေ ကျလာပါလိမ့်မယ်။

Installing collected packages: stanfordkarel

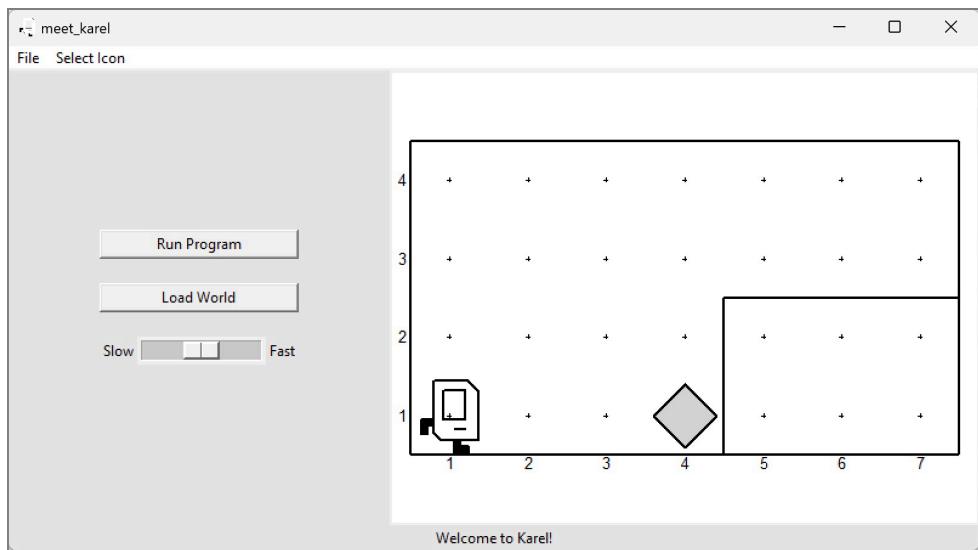
Successfully installed stanfordkarel-0.2.7

PS C:\Users\pyiso\VSCode\meet_karel>

meet_karel.py ဖိုင်ကို ညာကလေးနှင့်ပြီး Run Python File in Terminal ပြန်လုပ်ကြည့်ပါ။ ပုံ (??) မှာပြထားတဲ့ ကားရဲလုပ်ရိုက်ရမဲ့ ပွင့်လာသင့်ပါတယ်။ Run Program နှင့်ကြည့်ပါ။ စက်ရုပ်လေး ကားရဲပဲ နေရာရွှေသွားတာ တော်မယ်။ meet_karel.py ကို အောက်ပါအတိုင်း ဖြော်စွာကြရေးပါ။

```
from stanfordkarel import *
```

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



ဗို ၂/၂၇

```

pick_beeper()
turn_left()
move()
move()

turn_left()
turn_left()
turn_left()

move()
put_beeper()

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

```

`meet_karel.py` ဖိုင်ကို ညာကလစ်နိပ်ပြီး Run Python File in Terminal ပြန်လုပ်ကြည့်ပါ။ ကား ရဲ၍ပရိုကရမ် ပွင့်လာရင် `Run Program` နှိပ်ကြည့်ပါ။ ဘိပါလေးကို နေရာချွေးပါလိမ့်မယ်။ အကယ်၍ ကားရဲ၍ပရိုကရမ် မတက်လာရင် ကုဒ်ရေးတာမှားနေလို့ ဖြစ်နိုင်တယ်။ အပေါ်ကုဒ်နဲ့ ဦးယူဉ်ပြီး ကြည့်ပါ။ VS Code အယ်ဒီတာမှာ အနိလိုင်တွေ့လေးတွေ ပြတဲ့နေရင် အဲဒီနေရာတွေမှာ ဆင်းတက်မှုးနေတာ ဖြစ်နိုင်တယ်။

VS Code အယ်ဒီတာမှာ ပရိုကရမ်ကုဒ်ပြင်ပြီး ပြန် run တဲ့အခါ ပထမ run ထားတဲ့ ပရိုကရမ်ကို အရင်ပိတ်ဖို့လိုပါတယ်။ ဆိုလိုတာက meet_karel.py ကို run ထားတယ်ဆိုပါစွဲ။ ပုံ (??) က ဝင်း ဒီးပုံပုံလာမယ်။ meet_karel.py ကုဒ်ကို ပြင်ပြီး ပြန် run ချင်ရင် အဲဒီ ဝင်း ဒီးကို အရင်ပိတ်ရမယ်။ မဟုတ်ရင် ပြင်ထားတဲ့ ပရိုကရမ်က ချက်ချင်း ပွင့်မလာဘူး။ ပထမ ဟာကို ပိတ်တော့မှုပဲ နောက် run တဲ့ဟာ ပွင့်လာမှာ။

ဂိုက်ကွင်းကျို့နေတာက အဖြစ်များတဲ့ အမှားပါ။ ကျို့ခဲ့လို မရပါဘူး။ အင်ဒန်တေးရှင်း (indentation) လုပ်ရမဲ့နေရာမှာ မလုပ်ထားရင်လည်း ပြဿနာဖြစ်တယ်။ move, turn_left တွေကို ဘေးမျဉ်း ညာဘက်ဆွဲပြီး အင်ဒန်လုပ်ပေးရမယ်။ အဲဒါတွေ ကရာဇ်စိုက်မိရင် ဆင်းတက်စ်အမှားဖြစ်ပြီး ပရိုကရမ် run လို့ မရနိုင်ဘူး။

Terminal မှာ ထုတ်ပေးတဲ့ မက်ဆွဲချုပ်တွေကို ကြည့်ပြီးတော့လည်း ဘာပြဿနာဖြစ်နေလဲ မှန်းဆ လို့ရနိုင်တယ်။ ဘာကြောင့်ဖြစ်နိုင်လဲ ဆက်စပ်စဉ်းတားလို့ ရတယ်။ ဥပမာ ဖြစ်တဲ့ပြဿနာအလိုက် အခုလို တွေ့ရပါမယ်။

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 6
```

```
move(
```

```
^
```

```
SyntaxError: '(' was never closed
```

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 7
```

```
move()
```

```
IndentationError: unexpected indent
```

```
Traceback (most recent call last):
```

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 1,
```

```
    in <module>
```

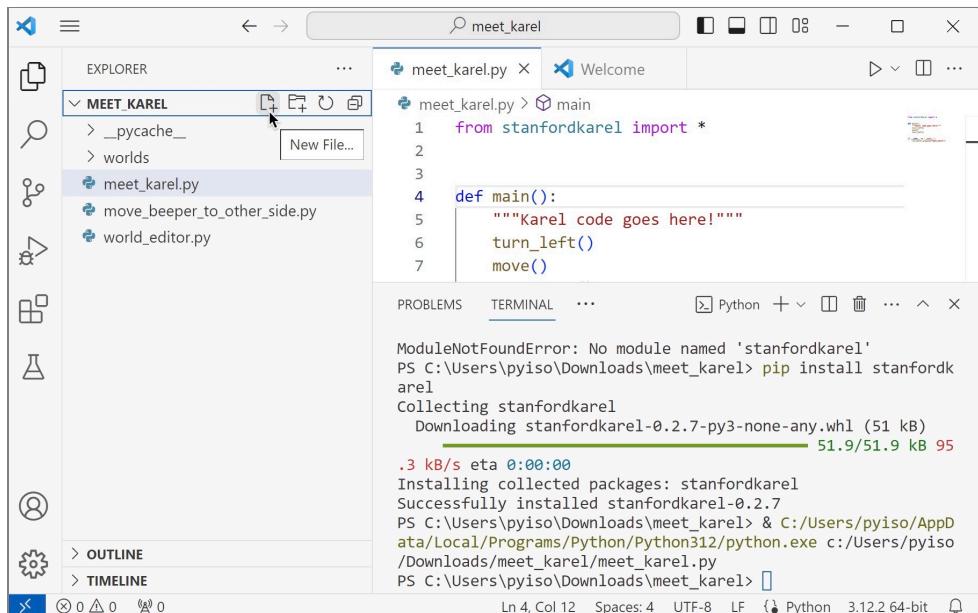
```
from stanfordkarel import *
```

```
ModuleNotFoundError: No module named 'stanfordkarel'
```

VS Code တွင် Python ဖိုင် အသစ်ယူခြင်း

MEET_KAREL ပင်မ ပရောဂျက်ဖိုဒ်အပေါ်မှာ ကလစ်နိုင်ပါ။ ပုံမှာ ပြထားတဲ့ New File အိုင်ကိုနိုင်ပါ။ ဖိုင်နံပည်ဖြည့်တဲ့ ဘောက်စံလေး ပေါ်လာမယ်။ Python ဖိုင်တွေက .py အိပ်စံတန်းရှင်းနဲ့ ဖြစ်ရပါမယ်။ ဒါကြောင့် နံပည် ဖြည့်တဲ့အခါ .py နဲ့ အဆုံးသတ်ပေးရပါမယ် (ဥပမာ hello.py)။ ကားချုပ်ပရီဂရမ်တစ်ခုကို Python ဖိုင်တစ်ခု ထားပါမယ်။ ပင်မ ပရောဂျက်ဖိုဒ်အပေါ်မှာပဲ တိုက်ရှိကြရပါမယ်။

နောက်ပိုင်း၊ အဆင့်မြင့်လာရင် ပရီဂရမ်တစ်ခုအတွက် ပရောဂျက်တစ်ခု ထားနိုင်တယ်။ ကုဒ်ဖိုင်တွေ အပြင် ပရီဂရမ်အတွက် လိုအပ်တဲ့ ရုပ်ပုံတော့၊ အခြားဖိုင်တွေ (config ဖိုင်၊ setting ဖိုင် စသည်ဖြင့်) လည်း ပါနိုင်တယ်။ ပင်မပရောဂျက် အောက်မှာပဲ ဖိုင်တွေက တိုက်ရှိကြရဖို့လည်း မလိုတော့ဘူး။ ဆက်စပ်ရာ ဖိုင်တွေကို အမျိုးအစားအလိုက်၊ ဖန်ရှင်အလိုက် ဖို့ဒါတွေခဲ့ပြီး စနစ်ကျ စီစဉ်ဖွဲ့စည်း ထားရမှာပါ။ ပရောဂျက်တစ်ခုမှာ ဖိုင်တွေကို စနစ်တကျ စွဲထားဖို့ အရေးကြီးပါတယ်။



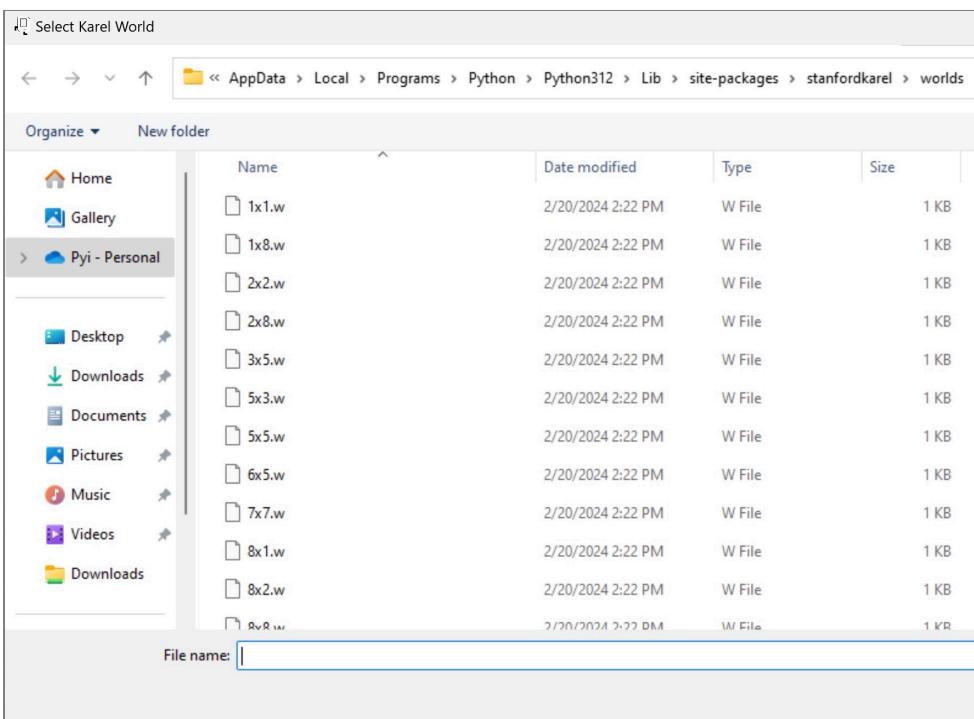
ပုံ က/၂၈

နောက်ဆက်တွဲ ခ

ကားရဲလ်ပရီဂရမ် ဖီချာများ

ကားရဲလ် ကဗ္ဗာဖိုင်များ

ကားရဲလ်ပရီဂရမ် ဝင်းခွဲ့မှာ **Load World** ခလုတ်နှုပ်ပြီး ကဗ္ဗာဖိုင်အသစ် တင်လိုရတယ်။ ခလုတ်နှုပ်လိုက် ရင် အခုလို ဖိုင် ခိုင်ယာလော့ဂုံး ပွင့်လာမယ်။



ပုံ ၁/၁

ဒါက stanfordkarel လိုက်ဘရဲ သူနိုင်အရှိအတိုင်း ပါတဲ့ worlds ဖို့ပါ။ ဖိုင်တွေက .w နဲ့ ဆုံးပါတယ်။ ကဗ္ဗာဖိုင်တွေကို ပင်မ ပရောဂျက်အောက် worlds ဖို့ပါထဲမှာ ထားလိုလည်းရတယ်။ အခြားနေရာတွေမှာ ထားလိုတော့ မရဘူး။

စာအုပ်ပါ ဥပမာတွေ၊ လေ့ကျင့်ခန်းတွေအတွက် ကဗျာဖိုင်တွေကို ပရောဂျက် တစ်ခုချင်းအလိုက် သီးခြား worlds ဖိုဒါထဲမှာ ထည့်ပေးထားမှာပါ။ ပရိုဂျမ်းတစ်ခုဟာ ကဗျာတစ်ခုတည်းမှာပဲ အလုပ်လုပ်တာမဟုတ်ဘဲ အလားတူ ကဗျာအမျိုးမျိုးအတွက် အလုပ်လုပ်အောင် ရေးပေးရတာ။ အခုပြာတာကို နား မလည်ရင် အခန်း (??) ဖတ်ပြီးရင် နားလည်သွားမှာပါ။

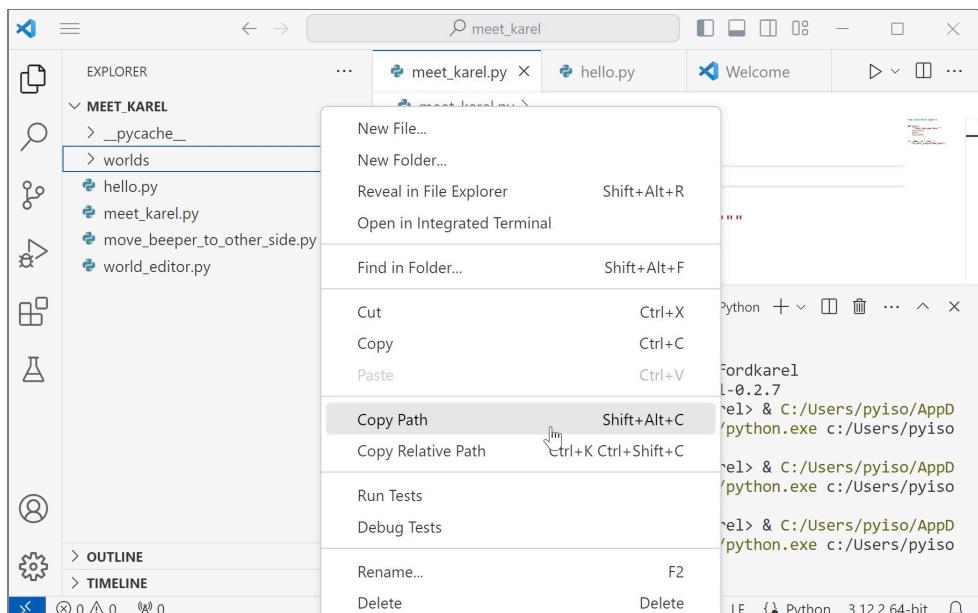
Load World လုပ်တဲ့အခါ ပွင့်လာတဲ့ ဖိုင် ခိုင်ယာလော်ကာ ကိုယ်လိုချင်တဲ့ worlds ဖိုဒါ မဖြစ် နေသူး။ သူနိုင်ပါတဲ့ worlds ဖိုဒါ ဖြစ်နေတယ်။ ကိုယ်ခေါ်တင်ချင်တဲ့ ဖိုင်တွေရှိတဲ့ လက်ရှိပရောဂျက်ရဲ့ worlds ဖိုဒါကို သွားရမယ်။ ဥပမာ PyCharm/VS Code အတွက် MeetKarel/meet_karel ပရောဂျက် worlds ဖိုဒါ လမ်းကြောင်း အပြည့်အစုံက

C:\Users\yourname\VSCode\meet_karel\worlds

C:\Users\yourname\PycharmProjects\MeetKarel\worlds

ဖြစ်ပါမယ်။ ဖိုင်ခိုင်ယာလော်ကနေ ဖော်ပြပါ လက်ရှိပရောဂျက် worlds ဖိုဒါကို တစ်ဆင့်ချင်း သွားပြီး တင်ချင်တဲ့ ကဗျာဖိုင် (.w ဖိုင်) ကို ရွေးရမှာပါ။

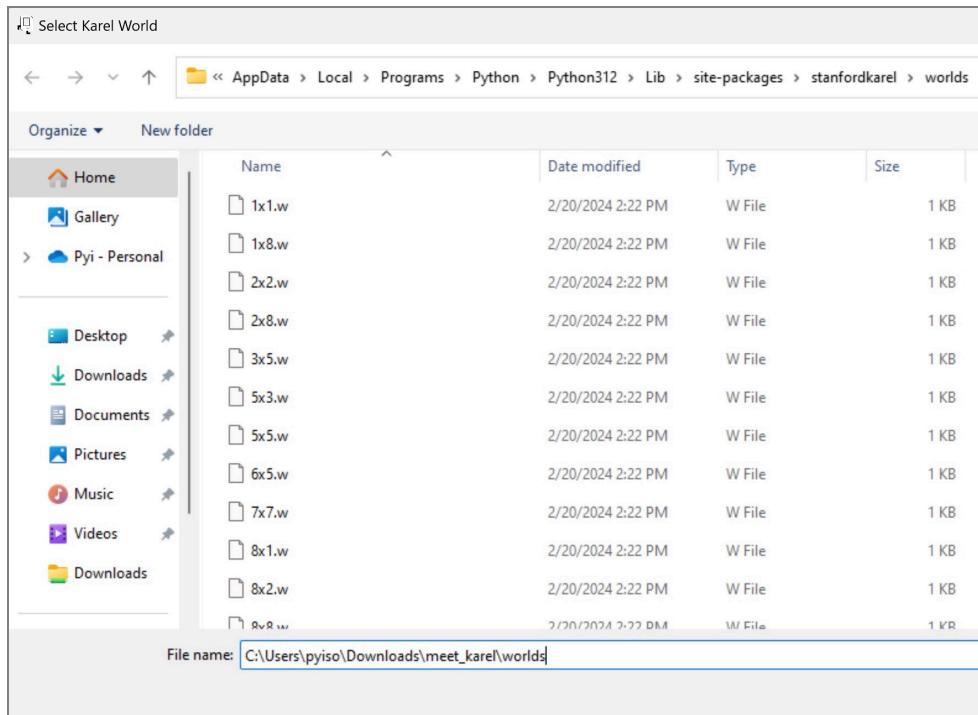
အပေါ်ကနည်း၊ အဆင်မပြော့ရင် အခုလုပ်မားကြည့်ပါ။ လက်ရှိပရောဂျက် worlds ဖိုဒါကို ညာကလစ်နိုပ်ပြီး Copy Path လုပ်ပါ (ပု ??)။ ဖိုင် ခိုင်ယာလော် File name မှာ ကူးထေည့်ပါ (ပု ??)။ Enter ကိုးနိုပ်ပါ။ ပရောဂျက် worlds ဖိုဒါကိုရောက်သွားပါမယ်။ လိုတဲ့ကဗျာဖိုင် ရွေးတင်ရှုပါပဲ။ ပု (??) မှာ meet_karel worlds ကို နမူနာပြထားပါတယ်။



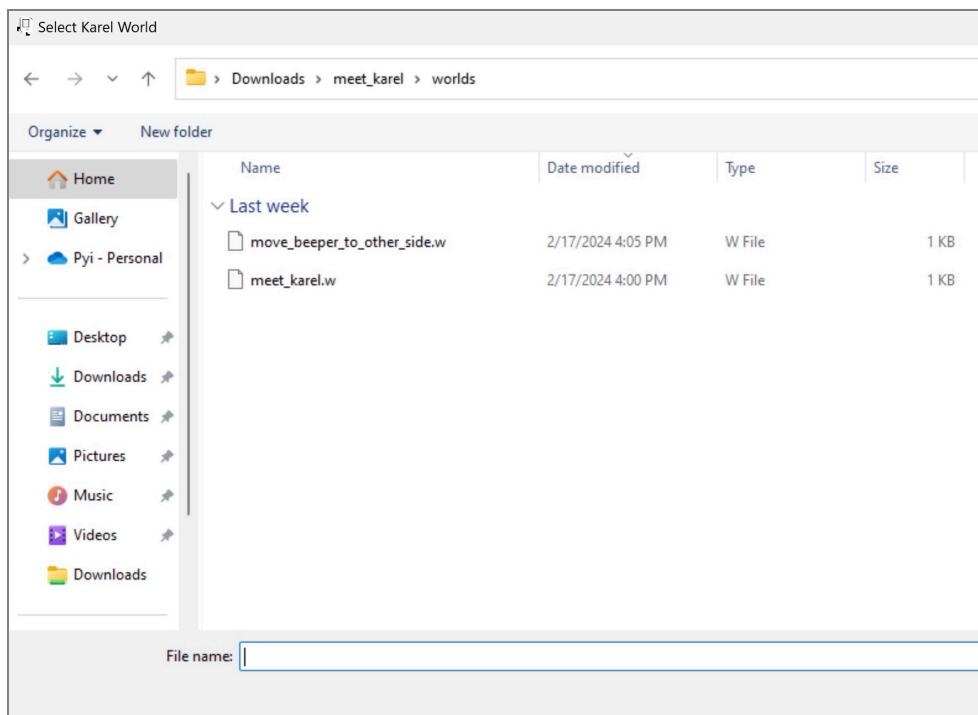
ပု ခ/J

အကယ်၍ အထက်ဖော်ပြပါနည်းတွေက ရှုပ်နေတယ်ထင်ရင် မှတ်ရ/သွားရ လွယ်တဲ့ Desktop, Downloads, Documents လို နေရာတစ်ခွဲမှာ သီးသန်ဖိုဒါတစ်ခု ဆောက်ပြီး ပရောဂျက်အားလုံး ထည့်ထားတာ အရှင်းဆုံးပါပဲ။ ပရောဂျက်ဖိုဒါနေရာ သိရင် ဖိုင်ခိုင်ယာလော်ကနေ ဘယ်လိုမဆိုရောက်အောင် သွားလို့ရပါတယ်။

၁၂၃



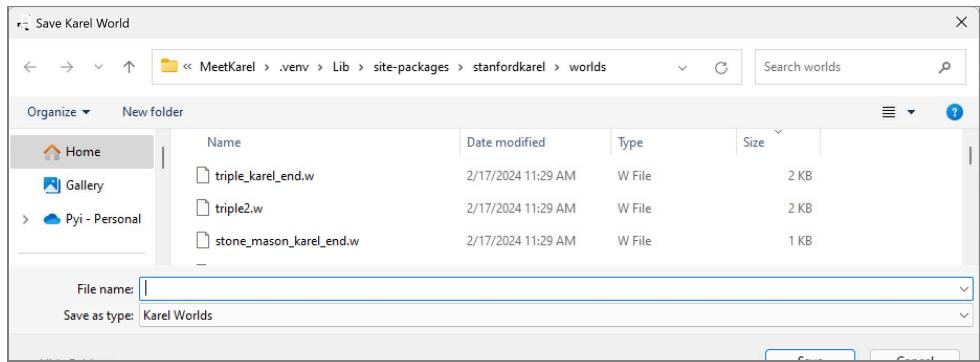
ပုံ ၅/၃



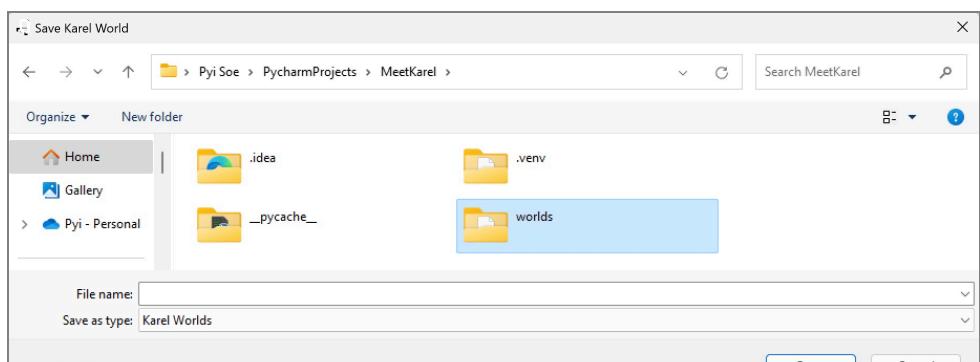
ပုံ ၆/၄

ကိုယ်ပိုင် ကားရဲလ်ကမ္ဘာ ဆောက်ခြင်း

ကားရဲလ်ရဲ ကမ္ဘာ အသစ်တစ်ခုဆောက်မယ်ဆိုရင် world_editor.py ဖိုင်ကို ညာကလစ်နှင့် Run ပါ။ Would you like to load an existing world? လို ပေါ်လာပြီး Yes/No ရွှေးချိုင်ပါလိမ့်မယ်။ No ကို နှိပ်ပါ။ ကမ္ဘာအရွယ်အစားအတွက် ကော်လံ ဘယ်နှစ်ခုလဲ row ဘယ်နှစ်ခုလဲ ထည့်ပေးပါ။ ကိုယ်ပိုင် ကားရဲလ်ကမ္ဘာ တည်ဆောက်လိုရတဲ့ ဝင်းဒီးပွင့်လာပါလိမ့်မယ်။ ကားရဲလ် မျက်နှာမူရာအပ်၊ ဘိပါအိတ်ထဲရှိ ဘိပါအရေအတွက်၊ နံရုံဆောက်/ဖျက်တာ၊ ဘိပါထည့်/ဖယ်ထုတ်တာ စတာတွေ လုပ်နိုင်ပါတယ်။ Save World နှိပ်ပြီး သိမ်းနိုင်ပါတယ် ဖိုင်ကိုသိမ်းတဲ့အခါ သုန္တရှိ သိမ်းခိုင်းတဲ့ဖို့ (default world folder) ထဲမှာ သို့မဟုတ် ပင်မ ပရောဂျက်ဖို့တဲ့က worlds ဖို့ဒါထဲမှာ သိမ်းရပါမယ်။



ပုံ ၉/၅



ပုံ ၉/၆

Python programming language

import, from, def, if

New term

fEnEmpstatement

ပါရာမီတာမပါရင်လည်း ပိုက်ကွင်းအလွတ် တစ်ဖုံး ‘()’ တော့ပါရမယ်။

colon ‘:’

ဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ကုဒ်ဘလောက် (*code block*)

quote သုံးခုတွဲ ‘“”’

ပရိုဂရမ်ဝင်းဒီးမှာ **Run Program** ခလုတ်

meet_karel.w ကဲ့သို့