

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 ပရီဂရမ်းမျိုးမျိုး ထည့်ပေးလိုက်တဲ့ အခါမာ သူ့အစွမ်းက အတိုင်းအဆမဲ့ပဲ။ နေရာမျိုးစံ၊ နယ်ပါယ်မျိုးစံ မှာ အကူးအညီပေးနိုင်တဲ့ စွယ်စုံသုံး ပစ္စည်းတစ်ခုဖြစ်သွားတယ်။ ဂိတ်သံစဉ်တွေကို ဖွင့်ပေးနိုင်သလို အသံလည်းသွင်းပေးနိုင်တယ်။ ရုပ်ရှင်တည်းဖြတ် လုပ်ချင်တာလား။ ပြဿနာမရှိဘူး၊ ကူညီပေးနိုင်တယ်။ နျှော လီးယား ပါတ်ပေါင်းဖို့တွေကို ဖီမံနိုင်သလို မောင်းသူမဲ့ အုံပျံတွေကိုလည်း ပဲထိန်းပေးနိုင်တယ်။

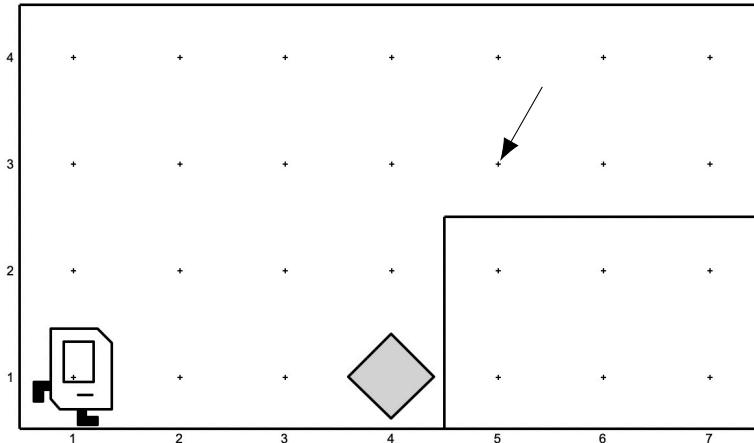
ကျွန်တော်တို့တွေ နိစ္စဓာဝ အသုံးပြုနေကြတဲ့ ကား၊ စမတ်နှင့်း၊ လက်ပါတ်နာရီ၊ မိုက်ခရှိဝေဖွံ့ဖို့ အဝတ်လျှော့စက် စတဲ့ စက်ပစ္စည်း အမျိုးမျိုးဟာလည်း ကွန်ပျိုတာတွေနဲ့ မကင်းပြန်ပါဘူး။ “ကွန်ပျိုတာနည်းပညာ အကူးအညီမပါတဲ့ အတိုင်းဆက်သံစုံထွင်မှုဆိုတာ မရှိဘူး” လို့ ဆိုနိုင်ပါတယ်။

တစ်ချက်တစ်ချက် ရိုက်ခတ်လိုက်တဲ့ ကွန်ပျိုတာနည်းပညာ လိုင်းလုံးကြီးတွေဟာ ကမ္မာတစ်စုံမှုးလုံး ပုံစံပြောင်းသွားလောက်အောင် အဟုန်ပြင်းထန်လာတယ်။ ဘီလီယံချွဲ့ခြုံတဲ့ လူတွေ ဆီရှယ်မီဒီယာတွေပေါ်က နေ ရုပ်သံတွေနဲ့ ချိတ်ဆက်ပြောဆိုသွေ့လို့ ရစေတာဟာလည်း ကွန်ပျိုတာစနစ်တွေပါပဲ။ Artificial Intelligence (AI) နည်းပညာကြောင့် သက်ရှိတွေမှာပဲတွေ့ရတဲ့ ညာတ်ရည်မျိုးကို ကွန်ပျိုတာတွေမှာလည်း တွေ့လာရပါပြီ။ သံချွဲ့ပွဲတွေ ဖြေရှင်းခြင်း၊ စစ်တုရင်ထိုးခြင်း စတဲ့ ကိစ္စမျိုးတွေအပြင် ပန်းချီဆွဲခြင်း၊ ကျွေးမှုရေးစပ်ခြင်း၊ သီချင်းရေးဖွဲ့ခြင်း ကဲ့သို့ အနုပညာဖန်တီးမှုတွေကိုပါ AI က လုပ်ဆောင်ပေးနိုင်ပါတယ်။ နှစ်ဆယ်တစ်ရာစုံ၊ အထူးမြှားဆုံး AI နည်းပညာလိုင်းဟာ အရှိန်အဟုန်ပြင်းပြင်း ရိုတ်ခတ်ဖို့ အားယူစုံပြုနေပါပြီ။

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

၁.၁ စက်ရုပ် ကားရဲလို

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



ပုံ ၁.၁ ခက်ရှုပ်လေး ကားရဲ့

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

ကားရဲ့လိုက်မန်းများ

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

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

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

ဘိပါကို ဘယ်လိုပြောခိုင်းမလဲ

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

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

၁.၂ Meet Karel ပရိုဂရမ်

ပရိုဂရမ် ရေးတယ်ဆိုတာ ကွန်ပူးတာကို ကိစ္စတစ်ခုခဲ့ အောက်ရှုက်ပေးဖို့ ခိုင်းစေတဲ့ ညွှန်ကြားချက်တွေ ရေးတာပါပဲ။ ဒီလို ညွှန်ကြားချက်တွေကို ပရိုဂရမ်ကုဒ် (program code) လို့ ခေါ်တယ်။ ပရိုဂရမ်ကုဒ် တွေကို ကွန်ပူးတာနားလည်တဲ့ programming language တစ်ခုခဲနဲ့ ရေးရတယ်။ ဒီစာအပ်မှာ အသုံးပြုမဲ့ programming language ကတော့ Python ပါ။ Programming language တစ်မျိုးပဲ ရှိတာ မဟုတ်ပါဘူး။ ရာနဲ့ချိပြီး ရှိတာပါ။ လူဘာသာစကားတွေ အမျိုးမျိုးရှိသလိုပေပါ။ Python ဟာ ဒီလို ရာနဲ့ချိတဲ့ထဲက လက်ရှိအသုံးအများဆုံး 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 လေ့လစွဲလိုပါမယ်။

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 ထုတ်လုပ်ဖြန့်ချုပ်ထားတဲ့ 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]

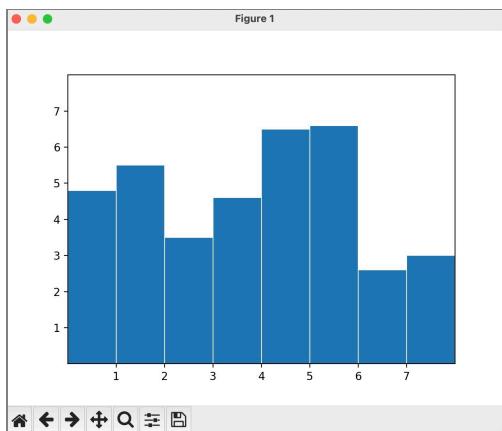
# plot
fig, ax = subplots()
bar(x, y, width=1, edgecolor="white", linewidth=0.7)

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

show()

```

ဘားချုတ်ကို ဒီလို ထုတ်ပေးပါတယ်။



ပုံ ၁၂

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

တံ့ကင်၊ ဓာတ်မန်နှင့် ဆင်းတာကို

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

ဖွဲ့စည်းထားတာပါ။ စတိတ်မန် ပုံစံတစ်မျိုးကို တွေ့ခဲ့ပြီးပါပြီ။ အဲဒါကတော့ ရှေ့စာမျက်နှာက အင်ပိုစတိတ်မန်ပဲ ဖြစ်ပါတယ်။

လူဘာသာစကားတွေမှာ ဖွဲ့စည်းတည်ဆောက်ပုံ စထရက်ချာရှိသလို programming language တွေမှာလည်း စထရက်ချာရှိဖို့ လိုအပ်တာပေါ့။ ဖွဲ့စည်းပုံ စထရက်ချာ မှန်/မမှန်ကို သွှေ့စည်းမျဉ်တွေနဲ့ ထိန်းကွပ်ထားတာပါ။ ပရိုကရမ်ကုဒ် 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 ပရိုကရမ်ကို Python က run ခွင့် ပြုမှုမဟုတ်ဘဲ အမှားနဲ့ သက်ဆိုင်တဲ့ အယ်ရာမက်ဆွဲချုပ်တွေ ပြုပေးမှာပါ။ ဖြစ်လေ့ရှိတဲ့ ဆင်းတက်စ်အမှားတွေကို မကြာခင် တွေ ရပါမယ်။

Keywords

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

main ဖန်ရှင်

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

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

    move()
    move()
    move()
    pick_beeper()
    turn_left()
    move()
    move()

    turn_left()
    turn_left()
    turn_left()

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

ဖန်ရှင် (function) ဆိုတာ ကိစ္စတစ်ခု လုပ်ဆောင်ပေးဖို့အတွက် ယူနစ်တစ်ခုအနေနဲ့ ဖွံ့ဖြိုးထားတဲ့ ပရီဂရမ်ကုတ် 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။

ဖန်ရှင်ခေါင်းစည်းအောက် ဒုတိယပိုင်းကတော့ main ဖန်ရှင် ကုဒ်ဘလောက် (code block) ပါ။ ကုဒ်ဘလောက် ဆိုတာ ကုဒ်တွေကို အပ်စုတစ်စုံ အဖြစ် ဖွံ့ဖြိုးထားတာကို 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 မညီတာနဲ့ ဆင်းတက်စုံအမှား ဖြစ်တယ်။ မလိုတဲ့နောရာမှာလည်း ခွာရေးလို့ မရဘူး။ ခေါင်းစီးကို ဘေးမျဉ်းနဲ့ ခွာထားကြည့်ပါ။ အယ်ရာဖြစ်တာကို တွေ့ရမယ်။ အင်ပို့စတိတ်မန့်လည်း ဘေးမျဉ်းနဲ့ ကွာနေ လို့မရဘူး။ အခြား language တွေမှားလည်း အင်ဒန်ထုတ်လုပ် ရေးကြပေမဲ့ Python မှာလောက် မတင်းကျပ်ဘူး။ အင်ဒန်ထုတ်မလုပ်လည်း ဆင်းတက်စုံမှားတာ မဖြစ်ဘူး။

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

Entry Point

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

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

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

```
run_karel_program()
```

ဆိုရင် 8×8 အရွယ် default ကဲ့ကို တင်ပေးပါတယ်။

ကားရဲလ်ကဲ့တစ်ခုကို လိုချင်တဲ့ပဲစဲ ဒီဇိုင်းဆွဲပြီး ဖိုင်နဲ့ သိမ်းထားရတာပါ။ ကဲ့ပုံစံချုတဲ့ ပရိုကရမ်လည်း ရှိတယ်။ ကဲ့ဖိုင်တွေက .w ဖိုင် အိပ်စာန်းရှင်းနဲ့ ဖြစ်တယ်။ ဒီစာအုပ်မှာပါတဲ့ ဥပမာတွေလေ့ကျင့်ခန်းတွေ အားလုံးအတွက် လိုအပ်တဲ့ ကဲ့တွေတွေကို အဆင်သင့်ပေးထားမှာပါ။ ကိုယ်ဟာကို လုပ်ဖို့ မလိုဘူး။ စိတ်ဝင်ထားရင် စမ်းကြည့်လိုကြအောင် စာမျက်နှာ (၁၀၂) နောက်ဆက်တဲ့ (ခ) မှာ အကျဉ်းဖော်ပြပေးထားပါတယ်။

၁.၃ ကားရဲလ် ပရိုကရမ် run ခြင်း

လိုအပ်တဲ့ဆော်ပဲတွေ ထည့်သွင်းနည်းကို စာမျက်နှာ (၈၁) နောက်ဆက်တဲ့ (က) မှာ တစ်ဆင့်ချင်း ဖော်ပြပေးထားပါတယ်။ အခုက Python ပရိုကရမ်တစ်ခုကို အရှုံးရှင်းဆုံး (လွှာယ်တယ်လို့ မဆိုလို) run လိုရတဲ့ နည်းကိုဖော်ပြပေးမှာပါ။ သဘောတရားပိုင်း နားလည်ဖို့ အထောက်အပံ့ဖြစ်မယ်။ အခုနည်းလမ်းကို အကြမ်းဖျော်း နားလည်အောင် ဖတ်ပြီးမှ နောက်ဆက်တဲ့ (က) ကို ဖတ်စေချင်ပါတယ်။

မိုက်ခရိုဆော့ဖို့ ဝင်းဒုးမှာ Notepad ၊ အက်ပဲလ် မက်ခံအက်စ်မှာTextEdit စတဲ့ တက်စ်အယ်ဒီတာတစ်ခုခုနဲ့ ပရိုကရမ်ကုံးရေးလိုရတယ်။ ကုံးဖိုင်ကို .py အိပ်စာန်းရှင်းနဲ့ သိမ်းရပါပယ်။ ပလိုန်းတက်စ် (plain text) ဖိုင် ပါပဲ။ Python ကုံးဖိုင်မို့လို့ .txt အစား .py နဲ့ သိမ်းတာပါ။ Python ဖိုင်နံမည်ကို စာလုံးအသေးနှုပ် ပေးတဲ့ ထုံးစံရှိတယ်။ စပေါ်နေရာမှာ _ (underscore) သုံးတဲ့ ထုံးစံရှိတယ်။ ဒါကြောင့် ‘Meet Karel’ ပရိုကရမ်ကုံးရေးလို့ ကုံးဖိုင်မှာ သိမ်းသင့်တယ်။

Python နဲ့ ရေးထားတဲ့ ပရိုကရမ်ကို run မယဆိုရင် Python ဆော်ပဲရမှာပါ။ ဒီဆော်ပဲ အင်စတောလ် လုပ်နည်းကို နောက်ဆက်တဲ့ (က) စာမျက်နှာ (၉၃) မှာ ဖော်ပြပေးထားပါတယ်။ Python ကုံးတွေကို ကွန်ပျူးတာက တိုက်ရှိက် နားမလည်ပါဘူး။ Python ဆော်ပဲဟာ Python ကုံးတွေကို တိုက်ရှိက် နားလည်ပြီး ကွန်ပျူးတာပေါ်မှာ run လိုကြအောင် ကြားခံဆောင်ရွက်ပေးတဲ့ ဆော်ပဲလို့ ယေဘုယျအားဖြင့် ယူဆနိုင်တယ်။

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

```
pip install stanfordkarel
```

ကားရဲလ်ပရိုကရမ်မှာ ခေါ်တင်ချင်တဲ့ ကဗျာဖိုင်တွေလည်း ရှိရပါမယ်။ `meet_karel.w` ကဗျာဖိုင်က `meet_karel.zip` ဖိုင် `worlds` ဖိုဒါထဲမှာ ရှိပါတယ်။ <http://tinyurl.com/3mmmm9c7j> လုပ်ကနေ `meet_karel.zip` ဖိုင်ကို အောင်လုပ်ပါ။ ဒါ zip ဖိုင်ထဲက `worlds` ဖိုဒါကို `meet_karel.py` ဖိုင်ရှိတဲ့ နေရာမှာ ကော်ပိုက္ခားထည့်ပါ။ ဝင်းဒီးမှာ Command Prompt မက်ခံအိုအက်စ်မှာ Terminal ဖွဲ့စြိုး `cd` ကွန်မန်းနဲ့ ကုဒ်ဖိုင်ထားတဲ့ ဖိုဒါထဲကို သွားပြီး အောက်ပါအတိုင်း `python` ကွန်မန်းနဲ့ ကုဒ်ဖိုင်ကို run ပေးရပါမယ် (လာမဲ့စာမျက်နှာမှာ နမူနာပြထားတာ ကြည့်ပါ။)

```
python meet_karel.py
```

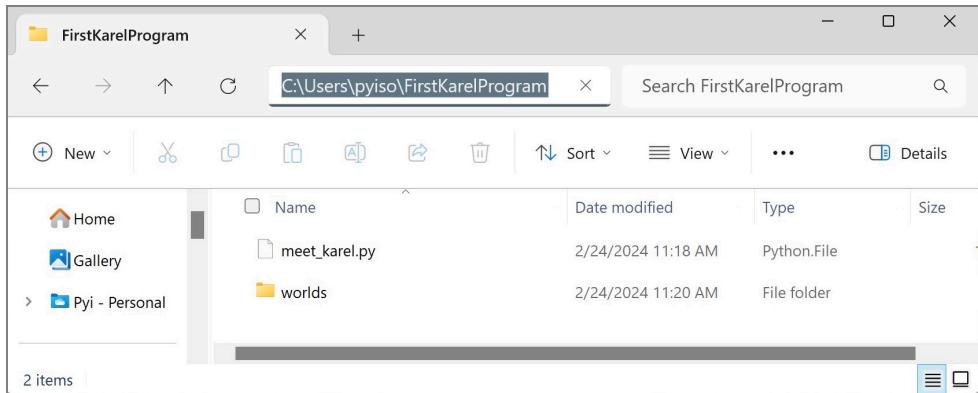
ကုဒ်ရေးထားတာ ဆင်းတက်စ်အမှား မရှိဘူးဆိုရင် ကားရဲပရိုကရမ် ပွင့်လာမှာပါ။

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

```
python your_karel_program.py
```

run ရုပ်ပါပဲ။

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

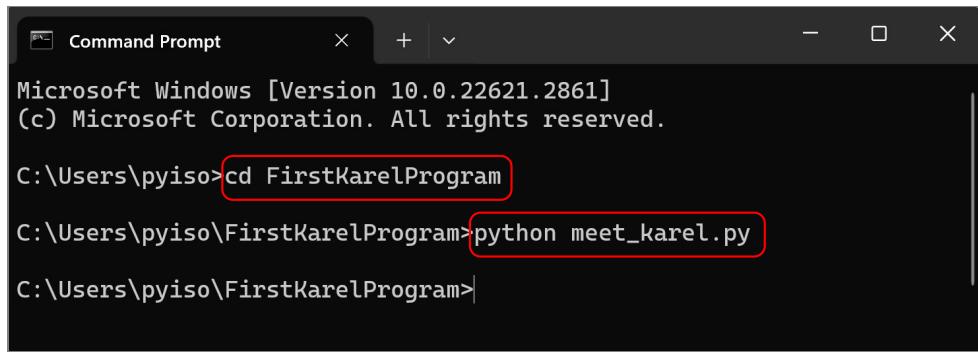


ပုံ ၁၃

၁.၄ Move Beeper to Other Side

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

ပုံ (၁.၆) မှ ဘိပါကို နံရံအဲ့တော်စာ် အောက်ခြေကို ရွှေ့ပေးတဲ့ ပရိုကရမ် ရေးကြည့်ပါ။ Python ထုံးစံအရ `move_beeper_to_other_side.py` ဖိုင်နဲ့ သိမ်းသင့်ပါတယ်။ `meet_karel.zip` ဖိုင်ထဲမှာပါတဲ့



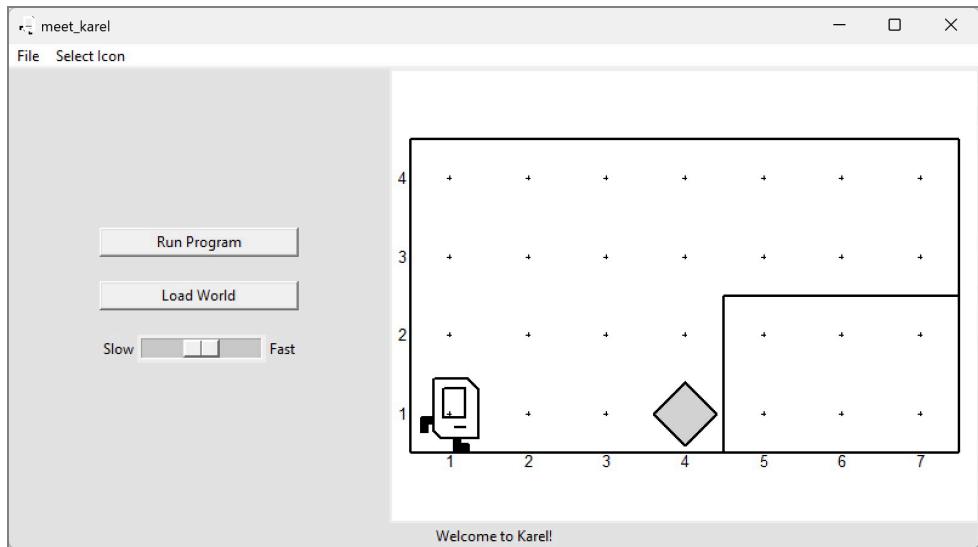
```

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 ၁.၅



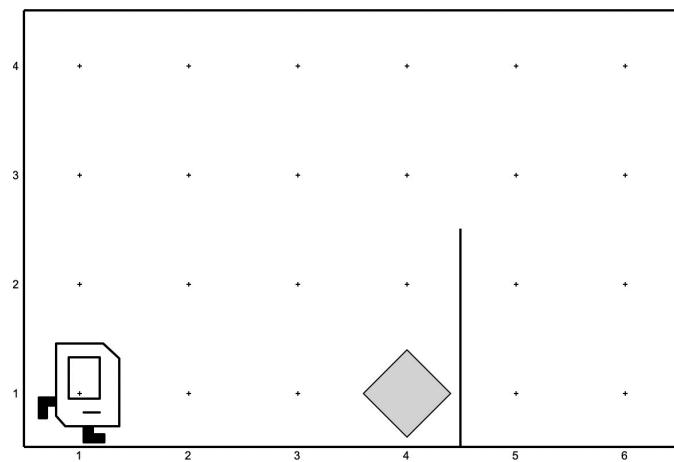
ပုံ ၁.၆

worlds ဖို့မှာပဲ အခု ကမ္ဘာဖိုင် ထည့်ပေးထားပါတယ်။ `move_beeper_to_other_side.w` နံမည့်ပါ။ အန်ထရိပိုင့်အတွက် အခုလိုရေးရပါမယ်။

```

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

```



٦٥٥٦

အခန်း J

ကားရဲလိန့် ကွန်ထရီးလ် စတိတ်မန္တိများ

ကားရဲလိန့်ပရိုကရမ်တစ်ခု ဖွဲ့စည်းတည်ဆောက်ပုံကို ရှေ့အခန်းမှာ လေ့လာခဲ့ကြပြီး ကွန်ထရီးလ် စတိတ်မန့် တွေ ဖြစ်ကြတဲ့ **for** loop, **while** loop, **if** နဲ့ **if...else** တိုကို အခုဆက်လက် လေ့လာကြပါမယ်။ ပရိုကရမ်တစ်ခုကို run တဲ့အခါ ပရိုကရမ်ကုဒ်ထဲက ညွှန်ကြားချက်တွေအတိုင်း ကွန်ပြုတာက လုပ်ဆောင် ပေးတာပါ။ ဒီလိုလုပ်ဆောင်ပေးတာကို အကွဲခံကျ၍ (execute) လုပ်တယ်လို့ ခေါ်တယ်။ ကွန်ထရီးလ် စတိတ်မန့်တွေဟာ ပရိုကရမ်တစ်ခုရဲ့ execution flow ကို ထိန်းချုပ်ပေးတာ ဖြစ်တဲ့အတွက် *control flow statements* တွေလိုလည်း ခေါ်ပါတယ်။

J.၁ **for** loop

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

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

ရေးရပါတယ်။ **put_beeper**, **move**, **turn_left** စတိတ်မန့် သုံးကြောင်းတစ်စုကို အကိုမ်တစ်ရာ ကျော်ချင်ရင် ဒီလိုပါ

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

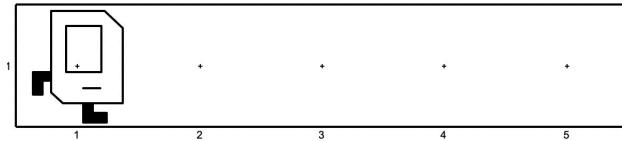
နှစ်ဆယ့်ငါးကြိမ်ကို **range(25)**, အကြိမ်တစ်ရာကို **range(100)** စသည်ဖြင့် တွေ့ရတယ်။ ယေ ဘုယျအားဖြင့် အကြိမ်အရေအတွက် N ကြိမ် ကျော်ချင်ရင်

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

ပုံစံနဲ့ သတ်မှတ်ရတာ။ N က အကြိမ်အရေအတွက်ကို ဖော်ပြတဲ့ ကိန်းပြည့်ကြန်းဖြစ်တယ်။ for loop ရေးတဲ့အခါ ကော်လံး : မကျွန်းခဲ့ဖို့ သတိပြုရပါမယ်။ x ဟာ ပေခိုရောဲလ်တစ်ခုဖြစ်ပြီး 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 (J.၁) ကားရဲလ်ကဲ့မှာ ကွန်နာအားလုံးမှာ ဘိပါတစ်ခုစီ ချထားပေးရမယ်။ ကွန်နာဝါးခုရှိတာမို့ လို့ စုစုပေါင်း ဘိပါဝါးခု ချထားရမှာပါ။ move ကတော့ လေးကြိမ်ပဲ လုပ်ရမယ် (ကားရဲလ်ရှေ့မှာ ကွန်နာ လေးခုပဲ ရှိတယ်)။ ဒီအတွက်ကို အခုလိုရေးနိုင်ပါတယ်။

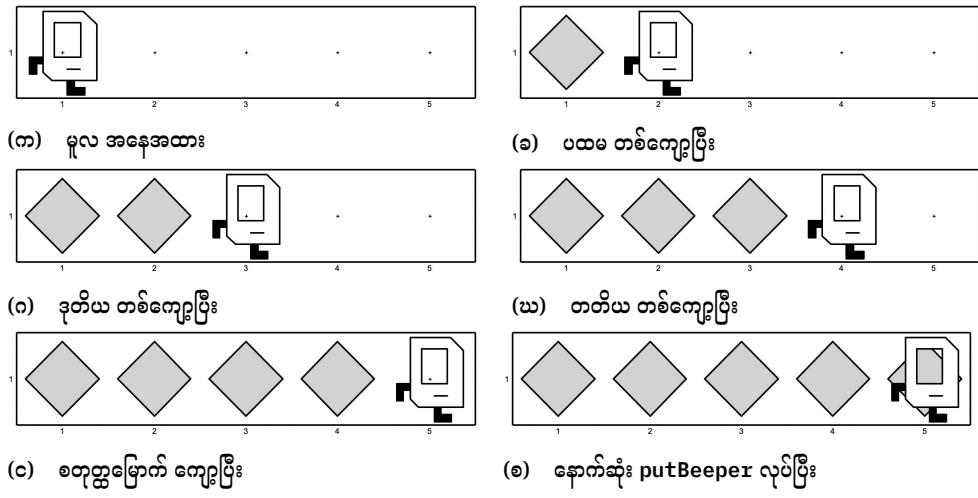
```
# File: make_row_of_5_beeper.py
from stanfordkarel import *

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

    put_beeper()

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

for loop ဟာ put_beeper နဲ့ move ကို လေးကြိမ်ကျော်ပေးမှာပါ။ တစ်ကြိမ်ပြီးတိုင်း ရှိနေမဲ့ အနေအထား တစ်ခုချင်းစီကို ပုံ (J.၂) မှာ တွေ့နိုင်ပါတယ်။ လေးကြိမ်မြောက် နောက်ဆုံးတစ်ကျော်အပြီး ပုံ (J.၂) (c) မှာ ဘိပါတစ်ခုလိုအနေသေးတာ တွေ့ရမယ်။ for loop ပြီးတဲ့အခါ ပေါ်ပြုပါတယ်။ တစ်တန်းလုံးအပြည့်ဖြစ်သွားမှာပါ။



ဦး J-J

Off-by-one bug

ပြီးခဲ့တဲ့ ဥပမားမှု ဒါ။ for loop အပြီး put_beeper မလုပ်မိရင် ပရီဂရမ်ဟာ လိုချင်တဲ့ အတိုင်း မဖြစ်ပါဘူး။ ဘိပါတစ်ခုလိုနေမယ်။ ဒီလိုပြဿနာမျိုးဟာ အဖြစ်များတဲ့ အမှားဖြစ်ပြီး off-by-one bug လိုခေါ်ပါတယ်။ Loop သုံးတဲ့ အခါ ကြိုရလေ့ရှိတဲ့ ဖြစ်တတ်တဲ့ အမှား (bug) ဖြစ်တယ်။ သတ်မှတ်ထားတဲ့ အတိုင်း ဖြစ်သင့်တဲ့ အတိုင်း ပရီဂရမ်က အလုပ်မလုပ်ဘဲ ဖြစ်နေတဲ့ အမှားကို ကွန်ပျိုတာအသုံးအနေးမှာ ပေးပို့လိုပါတယ်။ Bug ကိုလိုရှာပြီး မှန်အောင်ပြင်ပေးတာကိုတော့ debug လုပ်တယ်လိုခေါ်ပါတယ်။

ဘိပါတစ်ခုကျိုန်ခဲ့တဲ့ အမှားမျိုးဟာ off-by-one bug ပေးပို့လိုပါတယ်။ ခုနှစ်နဲ့ ဆယ့်သုံးကေား ကတ်နဲ့ ခုနှစ်လုံးရှိပါတယ် (ခုနှစ်နဲ့ ဆယ့်သုံးအပါဝင် ဆိုလျှင်)။ $13 - 7 = 6$ လိုတွက်မိရင် တစ်လုံး လိုနေပါလိမ့်မယ်။ သုံးပေါ်း တစ်တိုင် အလျားပေသုံးဆယ်ရှိ တပြောင့်တည်း ခြေစည်းရှိုးတစ်ခု အတွက် တိုင်စုစုပေါင်း တစ်ဆယ့်တစ်တိုင် လိုပါမယ်။ $30 \div 3 = 10$ လိုတွက်ရင် မမှန်ပါဘူး။ တစ်ခု လိုတာ အပြင် တစ်ခုပုံးနေတာလည်း ဖြစ်တတ်တယ်။ စောနက ခြေစည်းရှိုးမှာပဲ ထရံအချုပ်အရေအတွက် ကတော့ ကိုးချုပ်ဖြစ်ရမှာပါ။ $30 \div 3 = 10$ လိုတွက်ရင် တစ်ခုပုံးနေပါမယ်။ ခုနှစ်နဲ့ ဆယ့်သုံးကို ထည့်မစည်းစားရင် ကြားမှာ ကတ်နဲ့ လုံးရှိတာပါ။ $13 - 7 = 6$ လိုတွက်မိရင် တစ်လုံး ပို့နေပါလိမ့်မယ်။ ဒီ ဥပမာ အားလုံးဟာ အခြေခံသဘောတရားအားဖြင့် သိပ်မက္ခားခြားတဲ့ off-by-one bug ပုံစံကွဲအမျိုးမျိုး ဖြစ်တယ်။

‘Make Row of Five Beepers’ ခုတိယ ဗုံးရှင်း

ပရီဂရမ်ရေးတဲ့ အခါ ပရီဂရမ်မာတွေ တစ်ယောက်နဲ့ တစ်ယောက် စဉ်းစားပဲ စဉ်းစားနည်း ဖြေရှင်းနည်း ထပ်တူကျေလေ့မရှိပါဘူး။ ဘိပါဝါးခဲ့ အတန်းလိုက် ချေပေးတဲ့ ပရီဂရမ်ကိုပဲ နောက်ဟာရှင်းတစ်မျိုးနဲ့ ရေးနိုင်ပါတယ်။

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

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

J.၂ အင်ဒန်ထုတ်လုပ်ခြင်းနှင့် ကုဒ်ခွဲချေခြင်း

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

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

အခါ Python ကုဒ်မှာ အင်ဒန်ထုတ်လုပ်ထားဘဲ ဘယ်ဘက်စွန်း ကပ်ရေးထားတဲ့ **def main():** (ဖန် ရှင်ခေါင်းစည်း) ဟာ 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။

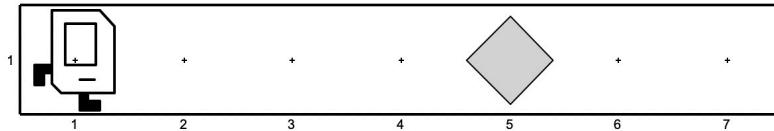
အောက်ဆုံး **turn_left()** ကလည်း အင်ဒန်ထုတ်လုပ် မထားတဲ့အတွက် အပေါ်ဆုံးအဆင့်ပဲ။ **def main():** နဲ့ အဆင့်တူတာပေါ့။ **main** ဖန်ရှင်ဘလောက် အပြင်ဘက်မှာလို့ ယူဆရမယ်။

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

J.၃ while loop

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

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



ပုံ၂၂

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

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

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

ကားရဲလ်ဟာ အခြေခံကွန်မန်း လေးခုအပြင် လက်ရှိ ကွန်နာမှာ ဘိပါရှိ/မရှိ၊ ရှေ့မှာ နံရုပ်တန်/မနေ့၊ အရောက်ကို ပျောက်နာမှုထား/မထား စတဲ့ သူနဲ့ (သို့) သူ့ရဲ့ကဲ့ဒဲ့ သက်ဆိုင်တဲ့ အခြေအနေတစ်ရပ်ရပ် မှန်/မမှန် စစ်နိုင်ပါတယ်။ 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 နောက်ထပ်တစ်ကြိမ် ထပ်ကျော်ပါတယ်။ ကွန်ဒီရှင်စစ်လိုက်၊ မှန်ရင် တစ်ခါထပ်ကျော်လိုက်၊ ကွန်ဒီရှင်ပြန်စစ်လိုက်၊ မှန်ရင် နောက်တစ်ခါ ထပ်ကျော်လိုက်၊ ... ဒီအတိုင်းဆက်ပြီး ကွန်ဒီရှင်မှန်နေသ၍ while loop က move ကို ပြန်ကျော်ပေး မှာပါ။ ကွန်ဒီရှင် စစ်လိုက်လို့ မှားသွားပြီဆိုရင်တော့ ထပ်မကျော်တွေ့ဘဲ ရပ်သွားမှာဖြစ်တယ်။ while loop တစ်ကြိမ်ကျော်ပြီးတိုင်း ရှိနေမဲ့ အနေအထား တစ်ခုချင်းကို ပုံ (၂.၄) မှာ လေ့လာကြည့်ပါ။ လေးကြိမ်မြောက် ကျော်ပြီးနောက် ကွန်ဒီရှင်စစ်လိုက်တဲ့အခါ မှားနေပြီ။ ဒီအခါ while loop က ထပ်မကျော်တော့ဘူး။ ဒီလို loop က ပြန်ကျော်နေတာ ရပ်သွားရင် loop ကနေ ထွက်တယ် (loop exits) လို့ ဖြေ လေ့ရှိတယ်။

အပြည့်အစုံ ဖော်ပြပေးထားတဲ့ ပရိုက်ရမ်းကို လေ့လာကြည့်ပါ။

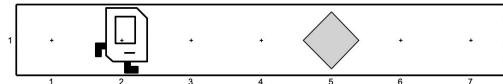
```
# File: go_pick_beeper.py
from stanfordkarel import *

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

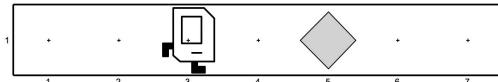
    pick_beeper()
```



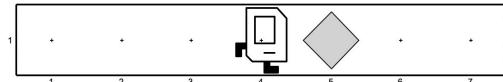
(က) မူလ အနေအထား



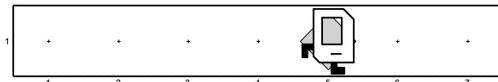
(ခ) ပထမ တစ်ကျော်ပြီး



(က) ဒုတိယ တစ်ကျော်ပြီး



(ဃ) တတိယ တစ်ကျော်ပြီး



(c) စတုတွေ့မြှာက် ကျော်ပြီး

ပုံ J-၄

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

ကားရဲလ် ကွန်ဒီရှင်များ

beepers_present ကွန်ဒီရှင် ကျတော့ လက်ရှိကွန်နာမှာ ဘိပါရိုရင် အမှန်၊ မရှိရင် အမှား ရလဒ်ထဲက် တယ်။ no_beeper_present နဲ့ ဆန်ကျင်ဘက်ပေါ့။ ကားရဲလ် ကွန်ဒီရှင်တွေအားလုံးကို တေဘာလ် (၂.၃) မှာ ကြည့်ပါ။ ပုံမှန်စစ်တာနဲ့ အငြင်းပုံစစ်တာ နှစ်မျိုးကို ယှဉ်တဲ့ပြထားပါတယ်။

ကွန်ဒီရှင်	ဆန်ကျင်ဘက် ကွန်ဒီရှင်	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_beeper_present	လက်ရှိကွန်နာမှာ ဘိပါရိုမရှိ
beepers_in_bag	no_beeper_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	မြာက်ဘက် မျက်နှာမှုလျက် ရှိမရှိ

တေဘာလ် ၂.၁ ကားရဲလ် စစ်ဆေးသည့် ကွန်ဒီရှင်များ

while loop ဆင်းတက်ခဲ့

while loop ရေးတဲ့ ပုံစံက ယေဘုယျအားဖြင့် ဒီလိုပါ။

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

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

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

ရေးပါတယ်။

‘Make Beeper Row’ ဥပမာ

‘Make Row of Five Beepers’ ဥပမာမှာ လမ်းခွဲအလျားဟာ မပြောင်းလဲဘူး ယူဆတာမိုလို for loop ကို အသုံးပြုတာ ဆိုလျှပ်ပါတယ်။ 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()

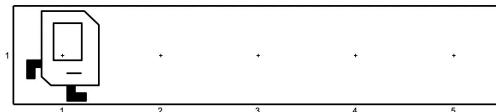
    put_beeper()

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

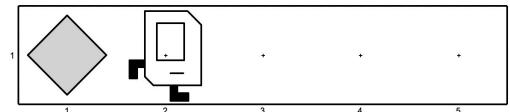
တစ်ကြိမ်ကျော့ပြီးတိုင်း ရှိနေမဲ့ အနေအထားကို ပုံ (၂၅) မှာ ပြည့်ပါ။ လေးကြိမ်မြောက်အပြီး front_is_clear စစ်တဲ့အခါ မှုးနေပြီဖြစ်လို့ ထပ်မကျော့တော့ဘဲ loop ကနေ ထွက်သွားမယ်။ ဒါ အခါမှာ ဘိပါတစ်ခု လိုနေသေးတယ်။ put_beeper ထပ်လုပ်ရမယ်။ တစ်ခါပဲ လုပ်ရမှာပါ။ ဒါကြောင့် while loop ထဲမပါအောင် ပထမအဆင့် အင်ဒန်ထုတ်ပဲ ဖြစ်ရမယ်။

J.၄ if စတိတ်မန်

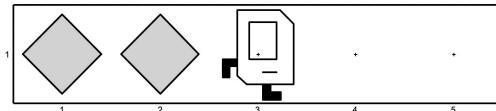
တစ်ခု (သို့) တစ်ခုထက်ပိုတဲ့ စတိတ်မန်တွေကို အခြေအနေတစ်ရပ် မှန်တော့မှုပဲ လုပ်ဆောင်စေချင်တဲ့ အခါ if ကို အသုံးပြုနိုင်တယ်။ ဥပမာ



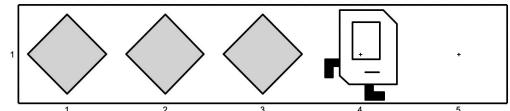
(က) မူလ အနေအထား



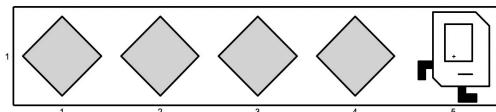
(ခ) ပထမ တစ်ကျော်ပြီး



(ဂ) ဒုတိယ တစ်ကျော်ပြီး



(ဃ) တတိယ တစ်ကျော်ပြီး

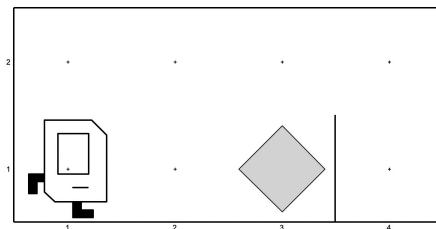


(င) စတုတ္ထပြောက် ကျော်ပြီး

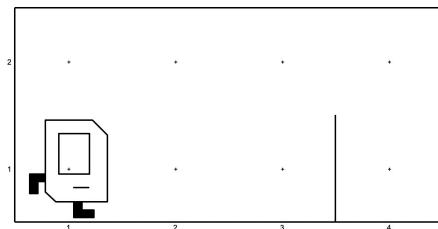
ပုံ J.၅

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

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



(က)



(ခ)

ပုံ J.၆

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

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

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

J.၁

```
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 (၂.၃) မှာ ကားရဲလ်ကို စစ်ခိုင်းလို့ရတဲ့ ကွန်ဒါရှင်အားလုံး ပြထားပါတယ်။

J.၂ if...else စတိတ်မန့်

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

```
from stanfordkarel import *
```

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

```

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

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

```



```

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

```

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

```

if condition :
    statement1a
    statement2a
    statement3a etc.

else:
    statement1b
    statement2b
    statement3b etc.

```

J.၆ Nested ကွန်ထရိုးလ် စတိတ်မန်များ

ကွန်ထရိုးလ် စတိတ်မန် ဘလောက်ထဲမှာ ကွန်ထရိုးလ် စတိတ်မန် ရှိလိုက်ပါ။ Nested ကွန်ထရိုးလ် စတိတ်မန်လို့ ခေါ်ပါတယ်။

Nested for loop

ဘိပါသုံးချခုလိုက် 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 ရရှိသောပေါ့။ ဒါလို ဖြစ်သွားမှုပါ

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

for loop နှစ်ခု ဆင့်ရေးထားလို nested for loop လိုခေါ်ပါတယ်။ Loop တစ်ခုချင်းကို သီးခြား ရည်ညွှန်းချင်တဲ့အခါ အပြင် for loop နဲ့ အတွင်း for loop လို ပြောလေ့ရှိတယ်။ အင်ဒန်ထဲ 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 ဘိပါခြိစည်းရှိုး ခတ်ပေးဖို့အတွက် for နဲ့ while nest လုပ်ထားတာကို လေ့လာ ကြည့်ပါ။ (၁, ၁) ကွန်နာမှာ အရှေ့ဘက်လည့် အနေအထားကနေ စမယ်လို့ ယူဆပါ။ ကဗျာအရွယ်အစား အမျိုးမျိုးကို ခြိစည်းရှိုး ခတ်ပေးနိုင်ပါမယ်။

တောင်ဘက် နံရုံတလျောက် ဘိပါတွေချွေားမယ် (နောက်ဆုံး ကွန်နာမှာ ဘိပါမချဲဘဲ ချိန်ထားမယ်)။ ပြီးရင် အရှေ့ဘက် နံရုံအတွက် အဆင်သင့်ဖြစ်အောင် ဘယ်ဘက်လှည့်မယ်။ ပုံ (၂.၇) (က) နဲ့ (ခ) ကို ကြည့်ပါ။ အရှေ့ဗြို့မြောက် နဲ့ အနောက်ဘက် နံရုံတွေအတွက်လည့် ဒီအတိုင်း လုပ်သွားရှုပါပဲ။ ပုံ (၂.၇) (ဂ)၊ (ဃ)၊ (၁)၊ (၁)၊ (၁) အသီးသီးကို ကြည့်ပါ။

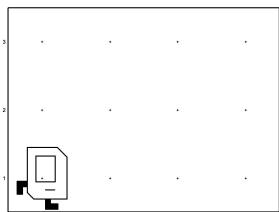
နံရုံတစ်ဘက် အတွက် ဆိုရင် အခုလို

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

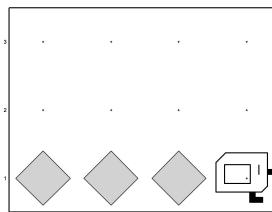
ဖြစ်မယ်။ (while ဘလောက်မှာ turn_left မပါတာ ဂရုပြုပါ)။ နံရုံလေးဘက်အတွက် လေးကြိမ် လုပ် ရမှာဆိုတော့ for နဲ့ အခုလို

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

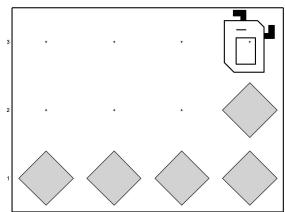
ရေးရပါမယ်။



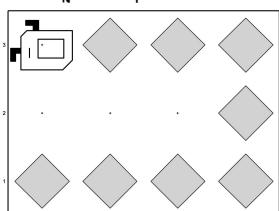
(က) မူလ အနေအထား



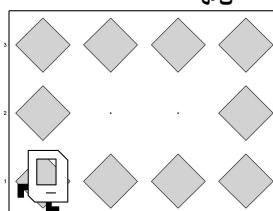
(ခ) ပထမ တစ်ကျော်ပြီး



(ဂ) ဒုတိယ တစ်ကျော်ပြီး



(ဃ) တတိယ တစ်ကျော်ပြီး



(၁) စတုတွေ့မြောက် ကျော်ပြီး

ပုံ ၂.၇

ပရီဂရမ် တစ်ခုလုံး အတွက်ဆိုရင် အခုလိုပါ

```

# File: beeper_fence.py
from stanfordkarel import *

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

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

```

Nested while and if

while နဲ့ if nest လုပ်လိုလည်း ရတာပေါ့။ လမ်းပေါ့မှာ 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 ဘိပါရှိတော့မှာပဲ ကောက်ပေးဖို့အတွက်။

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

```
# File: toggle beepers.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 beepers")
```

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

Nested if

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

```
if right_is_clear():
    if no beepers_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_beepers_present အခြေအနေ နှစ်ခုလုံး မှန်မှုပဲ လုပ်မှာပါ။

ပုံ (၂.၈) (က) မှာ ဒုတိယ လမ်းတစ်လျှောက် လမ်းပြင်တဲ့အလုပ်ကို ကားရဲလှုပိုကို တာဝန်ပေးတယ် လို့ ယူဆပါ။ ဘိပါရေး ညာဘက်နဲ့ရုံးပါ 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 beepers_present():
```

```

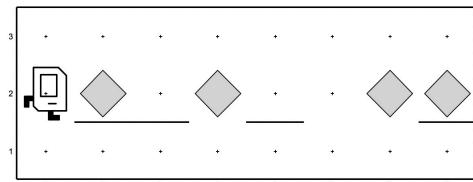
        put_beeper()
move()

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

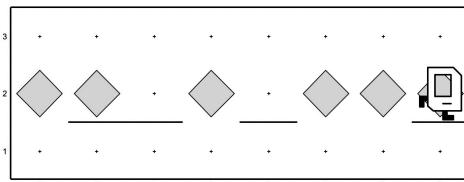
if __name__ == "__main__":
    run_karel_program("repair_street")

```

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



(က) မူလ အနေအထား



(ခ) လမ်းပြင်ပြီး အနေအထား

ပုံ J.7

အခန်း ၃

ဖန်ရှင်များ (Functions)

ဖန်ရှင် (function) တွေဟာ 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 တိုက်ရိုက် ရေးလို့ရရင် ပိုပြီးတော့ အဆင်ပြေမှုပါ။ ဒီလို လိုအပ်ချက်မျိုးကို ဖြည့်ဆည်း ပေးဖို့အတွက် ဟာ ဖန်ရှင်တွေရဲ့ အဓိက ရည်ရွယ်ချက်တွေထဲက တစ်ခုဖြစ်တယ်။ `turn_right` ဖန်ရှင်ကို အခုလို သတ်မှတ်နိုင်ပါတယ်။

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

ဒီလို သတ်မှတ်ထားပြီးရင် ညာဘက်လှည့်ချင်တဲ့အခါ

```
turn_right()
```

လို တိုက်ရိုက်ပြေလို့ ရသွားမှာ ဖြစ်ပါတယ်။ `turn_left` သုံးခါ ရေးဖို့ မလိုတော့ပါဘူး။

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

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

```
def turn_right():
```

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

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

ဖန်ရှင် ဟက်ဒီအောက် အင်ဒန်ထုပ်လုပ်ထားတဲ့ လိုင်းအားလုံးဟာ ငှင်းဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ကုံးဘေးလောက် ဖြစ်တယ်။ အခု turn_right ဖန်ရှင် ဘောက်မှာ turn_left သုံးကြိုးမြှင့်ပါတယ်။

| turn_right()

လုပ်ခိုင်းတာက (သတ်မှတ်ထားတဲ့) ဖန်ရှင်ကို အသုံးပြုတာ ဖြစ်တယ်။ ဒီအခါမှာ ငှင်းဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ဘောက်ကို 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 ကြားမှာ underscore (_) ခြားပေးလေ့ ရှိတယ်။

def name_of_function():
 statement₁
 statement₂
 statement₃ etc.

| name_of_function()

ဖန်ရှင်နံမည် အဓိပ္ပာယ် အရေးကြီးပါတယ်

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

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

ဖန်ရှင်သတ်မှတ်ချက်နဲ့ ဖန်ရှင်ကောလ ဥပမာ တရာ့ကို လေ့လာကြည့်ပါ။ ဘိပါ နှစ်ဆယ့်ငါးခု ချပေးတဲ့ put_25_beeper ဖန်ရှင်ပါ

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

put_25_beeper()

ဒါကတော့ ကွန်နာတစ်ခုမှာ ရှိတဲ့ ဘိပါအားလုံးကောက်ပေးတဲ့ ဖန်ရှင်ဖြစ်ပါတယ်

```

def pick_all beepers():
    while beepers_present():
        pick_beepers()

pick_all beepers()

```

ဖန်ရှင် အသုံးပြုတဲ့အခါ ကိစ္စတစ်ခုကို ပိုပြီး အလယ်တကူ လုပ်လိုက်သားတယ်။ ဘိပါအားလုံး ကောက် မယ်ဆုံးရင် pick_all_beepers ဖန်ရှင်ခေါ်လိုက်ရမဲ့။ ဘိပါ နှစ်ဆယ့်ငါးခု ချချင်ရင် sum_25_beepers ဖန်ရှင်ခေါ်လိုက်ရမဲ့။ ဖန်ရှင်တစ်ခုကို အသုံးပြုတဲ့အခါ အဲဒီဖန်ရှင်ကို ဘယ်လိုသတ်မှတ်ထားလဲ ပြန် စဉ်းစားနေဖို့ မလိုပါဘူး။ ဥပမာ ...

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

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

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

ဖန်ရှင် အခြေခံအုတ်ချုပ်များ

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

```

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

pick_all beepers()

```

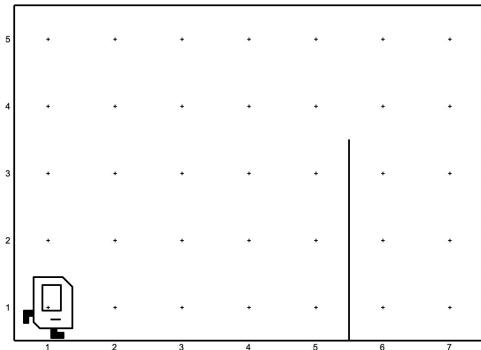
လမ်းတစ်လမ်းလုံး ရှင်းဖို့အတွက် ကွန်နာတစ်ခုက ဘိပါအားလုံး ကောက်ပေးတဲ့ pick_all_beepers ကို အခြေခံ အုတ်ချုပ်သွေ့ယူ အသုံးပြုထားတာပါ။

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

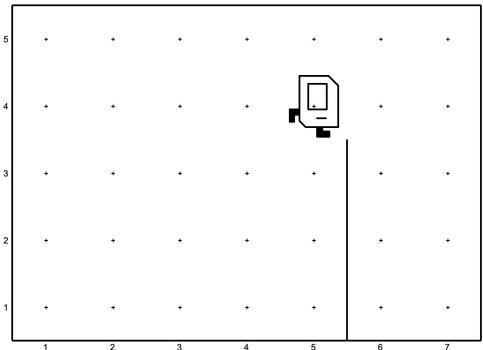
၃.၂ ပရီကွန်ဒီရှင်နှင့် ပို့ခိုကွန်ဒီရှင်

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 ၃.၁ (က) မှ (ခ) အနေအထားသို့ ကားရဲလ်က တိုင်ထိပ်အရောက် တက်သွားရပါမယ်။ တိုင်အကွား အဝေး၊ အမြင့် အမျိုးမျိုးနဲ့ အလားတူ ကွဲ့ပွဲတွေမှုလည်း အလုပ်လုပ်ရပါမယ်။ (နံရုံကို တိုင်ဟု ယူဆပါ)။ တိုင်အောက်ခြေကိုသွားတာနဲ့ တိုင်ထိပ်တက်တာ အလုပ်နှစ်ခု ပါဝင်တယ်လို့ မြင်နိုင်တယ်။ ဒီအတွက် ဖန်



(က) မတိုင်မဲ့ အခြေအနေ



(ခ) ပြီးဆောက် အခြေအနေ

ပုံ ၃.၁ တိုင်ထိပ်သို့

ရှင်နှစ်ခု သတ်မှတ်ပေးပါမယ်။

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

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

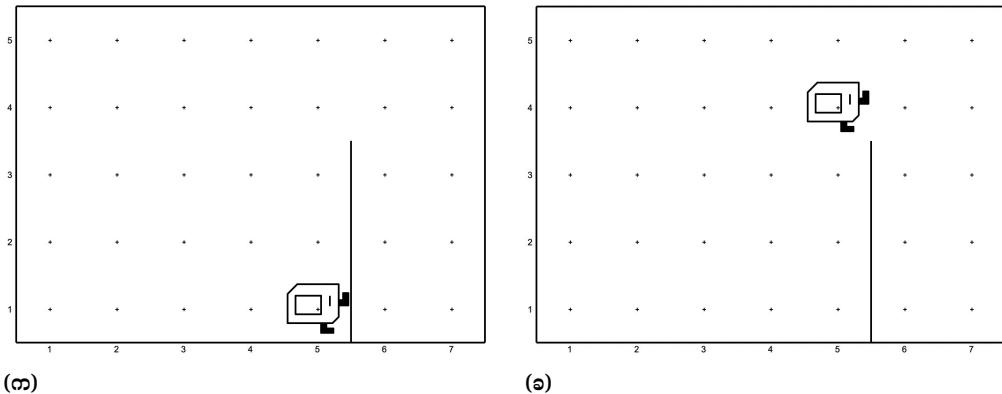
အထက်ပါ ဖန်ရှင်နှစ်ခုနဲ့ go_to_top ကို ဆက်လက် သတ်မှတ်ပါမယ်

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

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

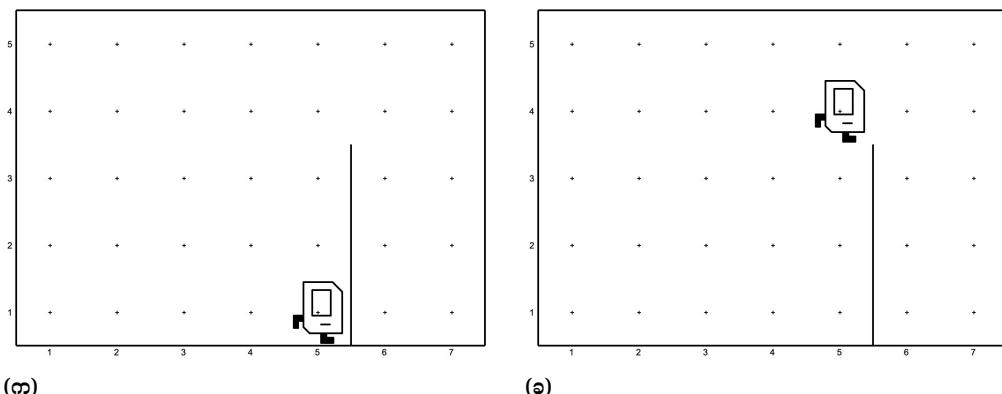
ဖန်ရှင် စတင်မလုပ်ဆောင်မီ ကြိုတင်ရှိနေရမဲ့ အခြေအနေကို ပရီကွန်ဒီရှင်လို့ ပြောခဲ့ပါတယ်။ ဖန်ရှင် သတ်မှတ်တဲ့အခါ ပရီကွန်ဒီရှင်ကို တိတိကျကျ စဉ်းစားဖို့ လိုပါတယ်။ ဖန်ရှင်အသုံးပြုတဲ့အခါမှာလည်း သတ်မှတ်ထားတဲ့ ပရီကွန်ဒီရှင် အတိုင်းကိုက်ညီဖို့ လိုတယ်။ `ascend_pole` ပရီကွန်ဒီရှင်းဟာ တိုင်ခြုံမှာ ကားခဲ့လဲ အပေါ်ဘက်ကို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

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



ပုံ ၃.၂ `ascend_pole` ပရီကွန်ဒီရှင်းနှင့် ပိုစ်ကွန်ဒီရှင်း

ဖန်ရှင် ပရီကွန်ဒီရှင် ပိုစ်ကွန်ဒီရှင်ကို သင့်တော်သလို သတ်မှတ်နိုင်ပါတယ်။ ပုံ (၃.၃) တွင် `ascend_pole` အတွက် အခြား ရွေးချယ်နှင့် ပရီ နဲ့ ပိုစ် ကွန်ဒီရှင်ကို ကြည့်ပါ။



ပုံ ၃.၃ `ascend_pole` ပရီ နဲ့ ပိုစ် ကွန်ဒီရှင်

အထက်ပါ ပရီ နဲ့ ပိုစ် ကွန်ဒီရှင်အရ `ascend_pole` ကို အခုလို

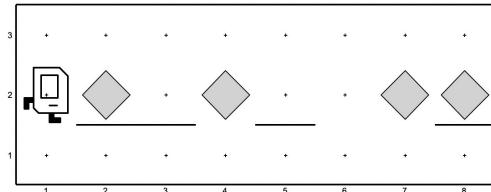
```
def ascend_pole():
    turn_left()
    while right_is_blocked():
        move()
        turn_right()
```

သတ်မှတ်ရမှာပါ။ `go_to_top` ကလည်း ဒီလို ဖြစ်သွားမယ်

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

၃.၃ ဖန်ရှင်များဖြင့် abstraction များ တည်ဆောက်ခြင်း

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



ပုံ ၃.၆

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

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

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

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

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

```

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

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

၃.၄ Bottom-up Programming

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

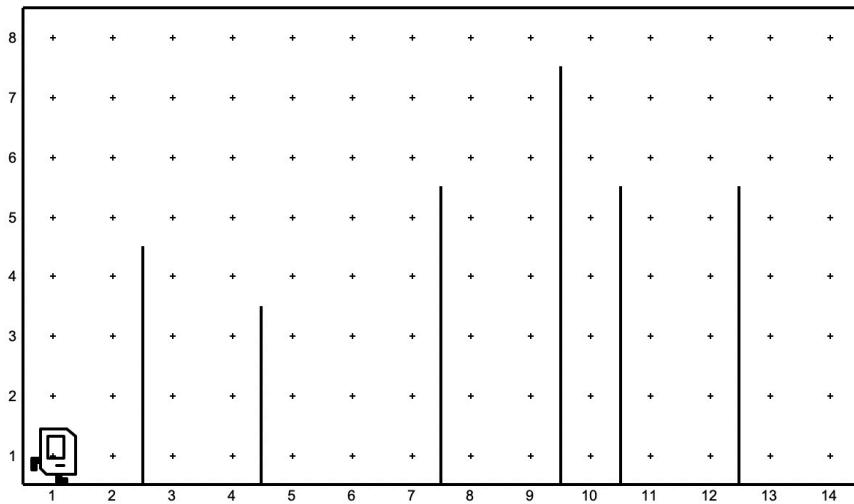
ခက်ခဲရှုပ်ထွေးတဲ့ ကိစ္စတစ်ခုကို ဖြေရှင်းတဲ့အခါ အပိုင်းတွေခဲ့ပြီး တစ်ပိုင်းချင်းကို သီး၌ဗြားဖြေရှင်းလေ့ရှိတယ်။ အပိုင်းတွေခဲ့လိုက်ခြင်းအားဖြင့် တစ်ပိုင်းစီဟာ မူလဖြေရှင်းရမဲ့ ကိစ္စလောက် မခက်ခဲတော့ ပါဘူး။ အရွယ်အစားအားဖြင့်လည်း နိဂုံထက် ငယ်သွားမယ်။ အပိုင်း တစ်ပိုင်းချင်းစီကို ဖြေရှင်းတဲ့အခါ မှာလည်း အလားတူပဲ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

ပုံ (၃.၅) ကမ္ဘာမှာ ကားရဲလ် တန်းကော်ပြီးပြုပါမယ်။ ထောင်လိုက်နံရုံတွေကို တန်းတွေလို့ယူဆပါ။ တန်းတွေဟာ အနိမ့်အမြင့် အမျိုးမျိုးဖြစ်နိုင်ပါတယ်။ ကမ္ဘာရဲ့ 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 နိဂုံမူလကိစ္စတစ်ခုကိုနေ တစ်ဆင့်ထက်တစ်ဆင့် ပိုသေးငယ်တဲ့ အပိုင်းလေးတွေဖြစ်လာအောင် ခွဲထုတ်တာကို *problem decomposition* လုပ်တယ်လို့ ခေါ်ပါတယ်။

Bottom-up programming နည်းနဲ့ ပရိုကရမ်ရေးတဲ့အခါ 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 အခုလိုတွေ့ရမှာ ဖြစ်ပါတယ်။

- တန်းကျော်ပြေးပြိုင်ပွဲဝင်ခြင်း
 - ▶ တန်းတစ်ခုကျော်ခြင်း
 - ▶ အပေါ်တက်ခြင်း
 - ▶ အောက်ဆင်းခြင်း

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

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

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

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

    turn_right()
    while front_is_clear():
```

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

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

```
def jump_over_hurdle():
```

```
    """

```

တန်းတစ်ခုကို ကျော်သည်

Precondition: တန်းသယ်ဘက် ခြေရှင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ

Postcondition: တန်းညာဘက် ခြေရှင်းမှာ အရှေ့ဘက်မျက်နှာမူပြီး ရှိနေ

```
    """

```

```
ascend()
```

```
move()
```

```
descend()
```

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

```
def compete_hurdle_race():
```

```
    """

```

တန်းကျော်ပြီးပြုင့်ပဲ ပန်းဝင်သည်ထိ ယုံးပြိုင်သည်

Precondition: (၁,၁) ကွန်နာတွင် အရှေ့ဘက်မျက်နှာမူနေ

Postcondition: (၁၄,၁) ကွန်နာတွင် အရှေ့ဘက်မျက်နှာမူနေ

```
    """

```

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

```
    if front_is_clear():
```

```
        move()
```

```
    else:
```

```
        jump_over_hurdle()
```

ပရီဂရမ် အစအဆုံးကို လေ့လာကြည့်ပါ။ ဖန်ရှင်တစ်ခုချင်း ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေကို တိတိကျကျ မြင်အောင် ရရှိကိုကြည့်ဖို့လည်း အရေးကြီးတယ်။ ဖန်ရှင် docstring မှာ ငှါးဖန်ရှင် 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():
```

```
    """

```

თანა: ჭრილი გრძელებული არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Precondition: თანა: გრძელებული არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Postcondition: თანა: გრძელებული არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

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

def descend():

"""

თანა: დანართი გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Precondition: თანა: გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Postcondition: თანა: გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

"""

```
turn_right()
```

while front_is_clear():

```
    move()
```

turn_left()

def jump_over_hurdle():

"""

თანა: გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Precondition: თანა: გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Postcondition: თანა: გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

"""

ascend()

move()

descend()

def compete_hurdle_race():

"""

თანა: გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Precondition: (0,0) გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

Postcondition: (0,0) გრძელებული არ არის და მას უნდა გადასახლოს და გრძელებული არ არის და მას უნდა გადასახლოს

"""

for i in range(13):

if front_is_clear():

```
    move()
```

else:

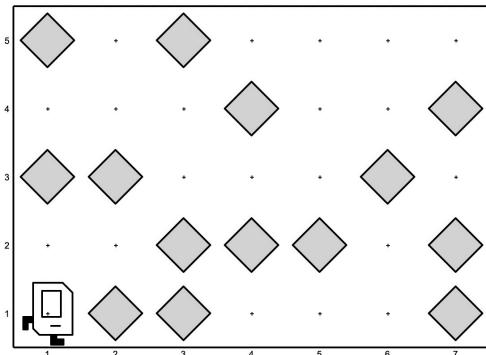
jump_over_hurdle()

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

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

အမိန်သိမ်းတဲ့ ကားရဲလ်

အခုတ်စံခါမှာ ကားပေါ်က အမိုက်တော်ရှင်းပေးပါမယ်။ ပုံ (၃၆) နှုန်းက ကမ္မာမှာ ဖြေရာကျန်း (random) ပြန်ကျေနေတဲ့ ဘိပါတွေကို အမိုက်လို ယူဆပါ။ ကမ္မာအရွယ်အစား အမျိုးမျိုးအတွက် အမိုက်ရှင်းပေးတဲ့ ပရိုဂရမ်တစ်ခု ရေးပေးရမှာပါ။

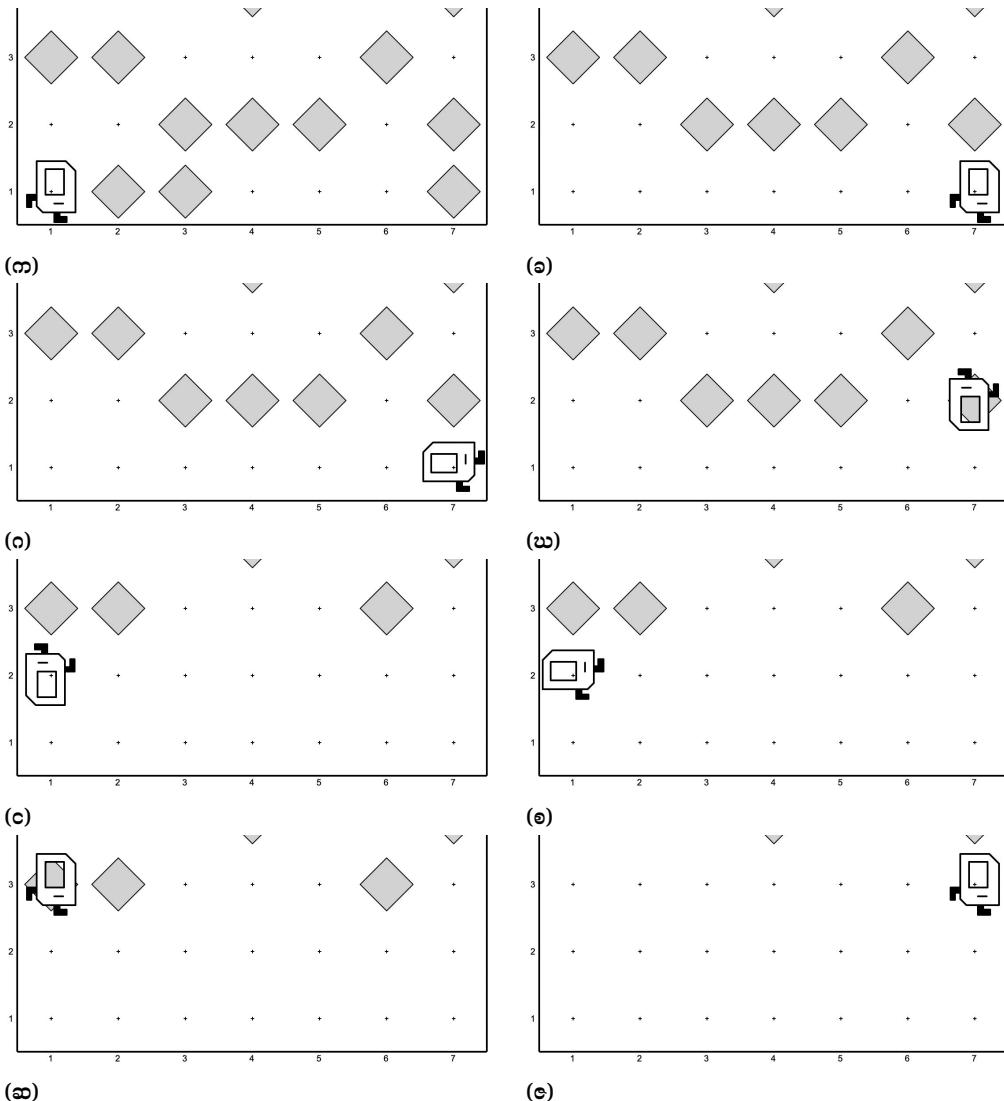


१२६

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

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

လမ်းတားလမ်း ရှင်းတဲ့ ကိစ္စကို ထပ်ခဲ့ကြည့်ရင် ကွန်းနာတစ်ခုရှင်းတာကို တွေ့ရမှာပါ။ Bottom-up နည်းအရ အောက်ဆုံးအလွှာက ကွန်းနာတစ်ခု အမိုက်ရှင်းတဲ့ ကိစ္စအတွက် ဖန်ရှင်ကို ပထမဆုံး သတ်မှတ်

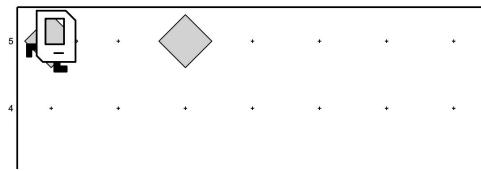


१०

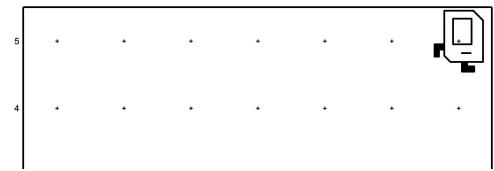
ରପିମନ୍ୟ॥

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

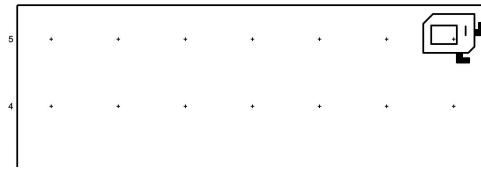
`clean_corner` ສູ່ ລັບ:ຕັ້ງລັບ:ຄຸນ:ຕະໜີ ພັນຍົມ ພັນຍົມ



(a)



(b)



(c)

ပုံ ၃.၈

```
def clean_street():
```

```
    """

```

လမ်းတလျောက် ဘိပါအားလုံးရှင်းပေးသည်

Precondition: လမ်းအစွမ်းတစ်ဖက်မှာ အခြားဘက်စွန်းကို မျက်နှာမျှ၍ ရှိမယ်

Postcondition: လမ်းပေါ်ဘိပါအားလုံး ရှင်းပြီး အခြားဘက်စွန်း ရောက်နေမယ်

```
    """

```

```
    while front_is_clear():

```

```
        clean_corner()

```

```
        move()

```

```
        clean_corner()
```

မြောက်ဘက်လှည့်တဲ့ ဖန်ရှင်

```
def turn_north():

```

```
    while not_facing_north():

```

```
        turn_left()
```

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

လမ်းတစ်လမ်းရှင်းပြီး နောက်တစ်လမ်းကူးတဲ့ ဖန်ရှင်

```
def change_street():

```

```
    """

```

လမ်းတစ်လမ်း၏ အစွမ်းတစ်ဖက်မှာ အပေါ်လမ်းသို့ ကူးသည်

Precondition: လမ်းအစွမ်းတစ်ဖက်မှာ ကပ်လျက် နံရံကို မျက်နှာမျှ၍ ရှိမယ်

Postcondition: အပေါ်တစ်လမ်းကူးပြီး အခြားဘက်စွန်းကို မျက်နှာမျှ၍ ရှိမယ်

```
    """

```

```
        move()

```

```
        if right_is_blocked():

```

```
            turn_left()

```

```
        else:

```

```
            turn_right()
```

တစ်လမ်းချင်း အမိုက်အကုန် ရှင်းတဲ့ ဖန်ရှင်ကို သတ်မှတ်လို့ ရပါပြီ

```

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

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

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



```

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

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

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

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

```

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

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

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

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

```

၃.၅ Top-down Programming

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

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

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

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

```

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

```

ဒီနေရာမှ ပရီနဲ့ ပိုစ် ကွန်ဒါရှင်တွေ တိတိကျကျ သတ်မှတ်ထားဖို့ အလွန်အရေးပါတယ်။ clean_street, turn_north, change_street ဖန်ရှင်တစ်ခုချင်းခဲ့ ပရီနဲ့ ပိုစ် 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 မခဲ့သေးဘဲ ရုပ်ထားရပါမယ်။ clean_street ဖန်ရှင်အတွက် clean_corner ရှိထားပြီး ဖြစ်တယ်လို့ မှတ်ယူရမှာပါ။

```

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

```

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

```

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

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

```

ဒုတိယတစ်ဆင့် ဖြေရှင်းပြီးသွားပါပြီ။

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

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

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

```

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

```

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

```

def main():
    clean_world()

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

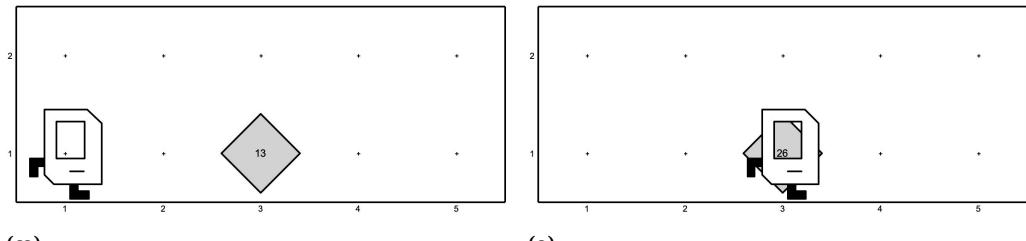
```

(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 လက်ရှိဖန်ရှင်ကို ဘယ်လိုရေးမလဲ စဉ်းစားရဲ တာ ဦးနှောက်ခြောက်စရာ ဖြစ်နေတာပါ။ ဥပမာ နောက်တစ်ခုလောက် ထပ်ကြည့်ရင် ပိုပြီး သဘောပေါက် သွားမယ် ထင်ပါတယ်။

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



ပုံ ၃.၉

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

ရှေ့ကွန်နာမှာ နိုင်ဘိပါပုံရဲ့ နှစ်ဆဖြစ်အောင် လုပ်လိုရတဲ့နည်းတော့ စဉ်းစားလိုပြီး။ ပြီးတဲ့အခါ အဲဒီ ဘိပါတွေကို နိုင်ဘိပါပုံနေရာကို ရွှေ့ပိုပါ။ 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 ရွှေ့ကွန်နာတွင် နှစ်ဆဖြစ်အောင် ရွှေ့ပေးသည်
    Precondition: ဘိပါပုံပေါ်ပွဲတွင် အရောက် မျက်နှာမှု၏ ရှိနေမယ်
    Postcondition: နှစ်ဆဘိပါပုံပေါ်ပွဲတွင် အရောက် မျက်နှာမှု၏
    """
    pass

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

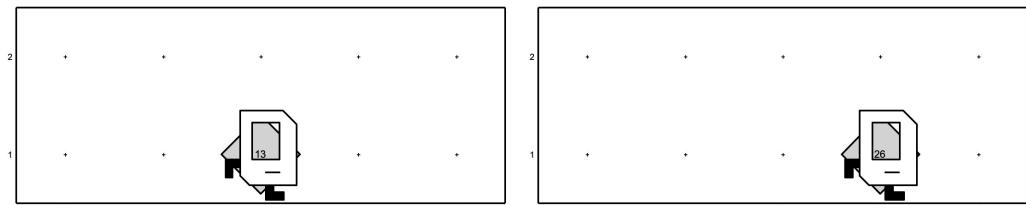
Python မှာ ဗလာဖန်ရှင်ဆိုပေမဲ့ စတိတ်မန့်တစ်ခုတော့ ပါရမယ်။ မဟုတ်ရင် ဆင်းတက်စ် အမှားဖြစ်မှပါ။ ဒါကြောင့် pass စတိတ်မန့်ကို ယာယီအနေနဲ့ သုံးလေ့ရှိတယ်။ ဒေါ်မိစတိတ်မန့် (dummy statement) လို့ ခေါ်ပါတယ်။ ဖန်ရှင်နှစ်ခုလဲ ပရီနဲ့ ပိုစ် ကွန်ဒီရှင်တွေကို ပုံးပို့ (၃.၁၀) နဲ့ (၃.၁၁) မှာ တွေ့နိုင်ပါတယ်။ နှစ်ခုလဲး မျက်နှာမှုတဲ့ဘက်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

မူလနေရာမှာပဲ ဘိပါနှစ်ဆဖြစ်အောင် လုပ်တဲ့ ဖန်ရှင်က ဒီလိုဖြစ်ပါမယ်

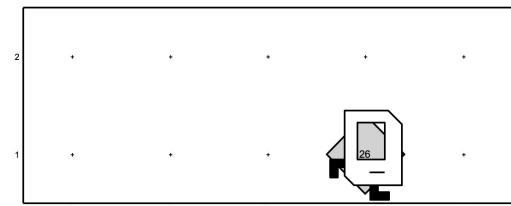
```
def double_beeper_pile():
    """
    ဘိပါပုံကို မူလနေရာတွင်ပင် နှစ်ဆတိုးပေးသည်
    Precondition: ဘိပါပုံပေါ်ပွဲတွင် အရောက်မျက်နှာမှုလျက် ရှိနေ
    Postcondition: နှစ်ဆတိုးပြီး ဘိပါပုံပွဲတွင် မူလအတိုင်း အရောက်မျက်နှာမှုလျက် ရှိနေ
    """
    double_beeper_pile_next()
    turn_left()
    turn_left()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

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

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

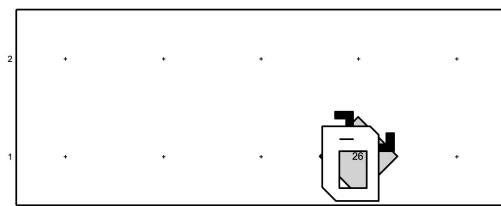


(က)

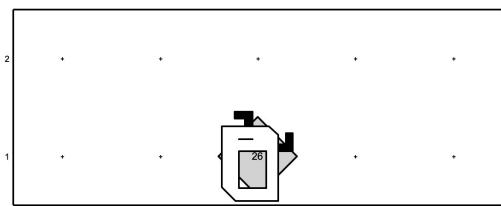


(ခ)

ပို ၃.၀၀



(က)



(ခ)

ပို ၃.၀၀

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

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

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

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

```

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

    move()

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

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

    move()

# File: double_beeper_pile.py
from stanfordkarel import *

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

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

```

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

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

    move()

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

    move()

def turn_around():
    turn_left()
    turn_left()

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

အခန်း ၄

ကားရဲလ်နှင့် ရီကားဆိုစ် ဖန်ရှင်များ

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

ဖန်ရှင် သတ်မှတ်ချက်ထဲမှာ ငှုံးဖန်ရှင်ကိုယ်တိုင်ကို ပြန်ခေါ်လို ရပါတယ်။ ရီကားဆိုစ် ဖန်ရှင် (recursive function) လို ခေါ်တယ်။ ရီကားဆိုစ် ဖန်ရှင်တွေဟာ ဘီကင်နာ 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 ပြသနာမရှိပါဘူး။ အကြမ်းဖျဉ်းလောက် ဖတ်ကြည့်ပြီး နောက်လာမဲ့ အခန်းတွေကို ကျော်သွားနိုင်ပါတယ်။ နောင်တစ်ခို့ကျော် ပြန်လာဖတ်ပေါ့။

၄.၁ ရီကားဆိုစ် ဖန်ရှင် ဘယ်လို အလုပ်လုပ်လဲ

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

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

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

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



ပို ၄.၁

ဖန်ရှင် စတင် လုပ်ဆောင်ပါမယ်။ ရှေ့မှာရှင်းနေတဲ့ အတွက် if ဘလောက်ကို လုပ်မှာပါ

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

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

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

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

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

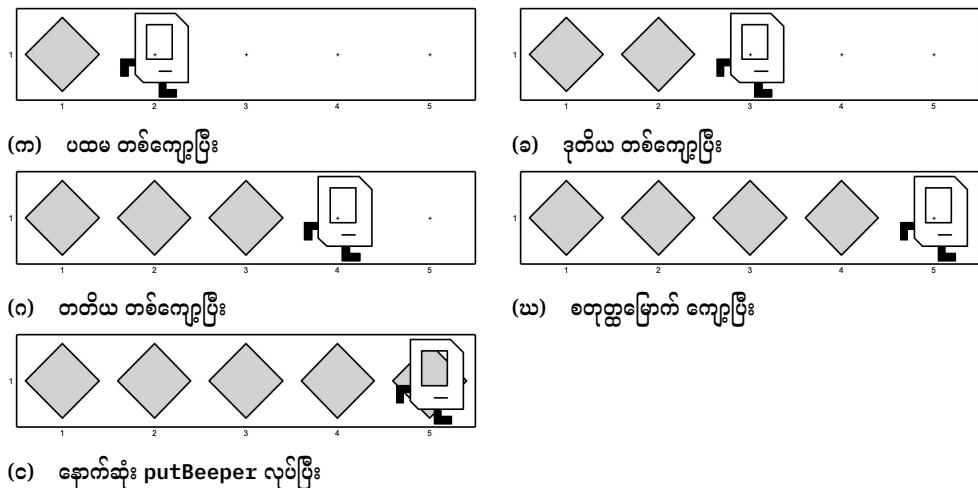
ပုံ (၄.၂) (ဂ) နေရာရောက်ပြီး သူကိုယ်သူ ထပ်ခေါ်ထားပြန်တယ်။ သုံးကြိမ်ရှိပြီ။ ဒီတစ်ခါလည်း if အပိုင်းပဲ ထပ်လုပ်

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

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

```
put_beeper()
```

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



(c) နောက်ဆုံး putBeeper လုပ်ပြီး

ပုံ ၄၂

ခေါ်ပါတယ်။ ဒက်စ်နေးရှင်းအရ 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()
```

ရိုကားဆစ်စ်ကောလ် ဖြစ်တာကို စိတ်ကုံးပံ့ဖော်ကြည့်ဖို့ ပြထားတာပါ။ အပိုင်ဆုံး မြှားအနက်က ကန် ဦး ဖန်ရှင်ကောလ် စတင်တာဖြစ်ပေါ်တာကို ဖော်ပြထာ်။ ရိုကားဆစ်စ်ကောလ် မဟုတ်သေးဘူး။ မြှား အပြာာက ရိုကားဆစ်စ်ကောလ်ကြောင့် ဖန်ရှင်အစ ပြန်ရောက်သွားတာကို ပြထာ်။ ပထမနဲ့ ဒုတိယ ရိုကားဆစ်စ်ကောလ် နှစ်ခုအတွက်ပြထားတာပါ။ ရွှေ့က ဥပမာအတွက် မြှားအပြာာ လေးခု ရိုရှုံးပါ (ရိုကား ဆစ်စ်ကောလ် လေးကြိမ်အတွက်)။ မြှားအပြာာ နောက်ထပ် နှစ်ခုရှိတယ် မှတ်ယူပါ (ပုံမှာထပ်ထည့်ရင် ကြပ်ညပ်ပြီး ကြည့်ရှုရှုပဲလို့ မဆွဲပြတာ)။ လေးကြိမ်မြှားကို ရိုကားဆစ်စ်ကောလ် ထပ်မဖြစ်တော့ဘူး (if အပိုင်းကို မလုပ်တော့တဲ့အတွက်)။ ဘို့ပါချုပြုး ဖန်ရှင်ကောလ် စခဲ့တဲ့နေရာကို ပြန်ရောက် သွားမယ် (မြှားအနဲ့)။ ဘယ်လို့ ပြန်ရောက်သွားတာလဲ ဆက်ကြည့်ရအောင်။

နောက်ဆုံး ရိုကားဆစ်စ်ကောလ်မှ ပြန်လာခြင်း

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

```

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

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

```

ပထမ turn_right ကော်လုပ်ဆောင်တဲ့အခါ ကော်လုပ်တဲ့ 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 ဖန်ရှင်ကော်လွှေမှုလည်း ဒီသဘောတရား အတိုင်းပါပဲ။ အောက်ဖော်ပြပါ ပရီ ဂရမ်ကုဒ်ကို ကြည့်ပါ။ main ဖန်ရှင်ထဲကနေ do_tricks ဆိုကို ရောက်သွားမယ်။ do_tricks ထဲက ငါ put_two ထဲကို ရောက်သွားမယ်။

```

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

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

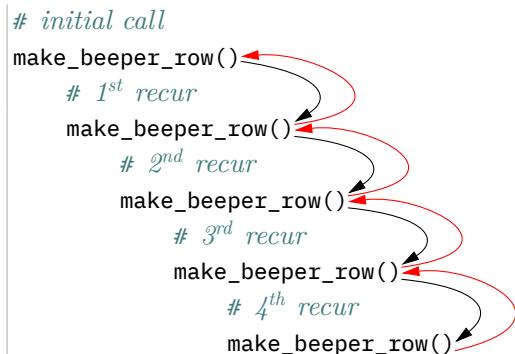
def put_two():
    put_beeper() ←
    put_beeper() →

```

put_two ပြီးသွားတဲ့အခါ do_tricks ထဲပြန်ရောက်သွားမယ်။ turn_left, move ဆက်လုပ်ပြီး do_tricks ခေါ်ခဲ့တဲ့နေရာ main ထဲပြန်ရောက်သွားမယ်။ နောက်ဆုံး main ထဲက move ကိုဆက်လုပ်ပါတယ်။

ဖန်ရှင် အဆင့်ဆင့် ခေါ်ထားတဲ့အခါ နောက်ဆုံးခေါ်တဲ့ ဖန်ရှင်က အရင်ဆုံး return ပြန်ပါတယ်။ main ကနေ do_tricks ကိုခေါ်။ do_tricks ကနေ put_two ကိုခေါ်ထားရင် put_two ကနေ do_tricks ဆိုကို အရင် return ပြန်တယ်။ ပြီးတော့မှာ do_tricks ကနေ main ကို ပြန်ရောက်မှာ ပါ။ ဒီသော့အရ put_two return မပြန်မချင်း do_tricks ဖန်ရှင်မပြီးသေးဘူး။ put_two ကနေ ပြန်လည်ပြီးမှ ကျွန်ုတ်တဲ့ turn_left, move ဆက်လုပ်တယ်။ ပြီးတော့မှာ do_tricks ဖန်ရှင်က return ပြန်ပါတယ်။

ရိုကားဆစ်စ် ဖန်ရှင်ကောလ်တွေ ဘယ်လို့ return ပြန်လာ။ ရွှေကလို့ မြှားတွေနဲ့ ဆဲပြလို့ ရပေမဲ့ ကြည့်ရတာ ရှုပ်ရှက်ခတ်နေမှာပါ။ အခုလို့ မြင်ကြည့်ရင် ပို့ရှင်းပါတယ်။



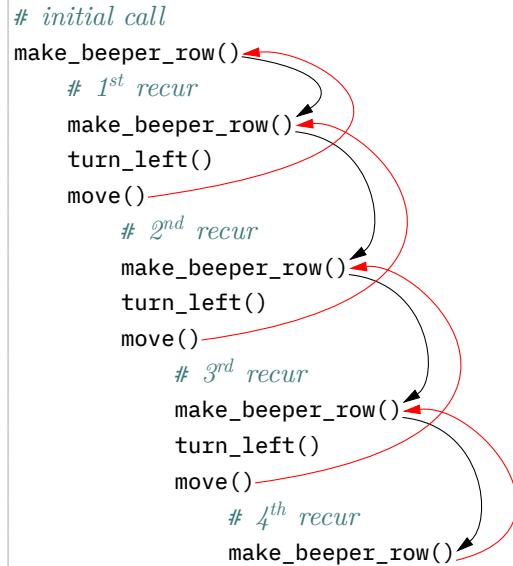
မြှေးအနက်တွေက ဖန်ရှင်ကောလ် တစ်ဆင့်ပြီးတစ်ဆင့် ဖြစ်တာကို ပြတာပါ။ အထက်မှအောက် အစိအစဉ်အတိုင်း ဖြစ်ပါတယ်။ လေးကြိမ်မြှောက်မှာ နောက်ထပ် ရိုကားဆစ်ဖိုကောလ် ထပ်မဖြစ်တော့ဘဲ နောက်ဆုံး ရိုကားဆစ်ဖိုကောလ်က အရင်ဆုံး return စပိုပါတယ် (အောက်ဆုံး မြှေးအနီး၊ ပြထား)။ ဒါ အခါ တတိယ ရိုကားဆစ်ဖိုကောလ်ကို ပြန်ရောက်သွားမှာပါ။ ဒါအတိုင်း တစ်ဆင့်ပြီးတစ်ဆင့် အထက်ကို return ပြန်ပြီး နောက်ဆုံးမှာ ပထမဆုံး ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားမှာပါ။ (အခုပြထားတာကို တကယ့် Python ကုဒ် အနေနဲ့ မယူဆရပါ။ ဖြစ်စဉ် နားလည်အောင် ပြခြင်းသာဖြစ်ပါတယ်)။ နေ့ပြင်ထားတဲ့ make_beeper_row ဟာရင်းမှာ return ပြန်တဲ့ ဖြစ်စဉ်ကို ကြည့်ရအောင်။

```
make_beeper_row(
```

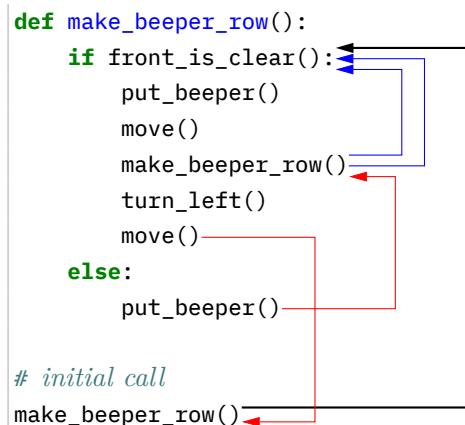
```
def make_beeper_row():
    if front_is_clear():
        put_beeper()
        move()
        make_beeper_row()
        turn_left()
        move()
    else:
        put_beeper()
```

if အပိုင်း ရိုကားဆစ်စကေလ်ပြီး turn_left နဲ့ move ထပ်ဖြည့်ထားတာ။ ရိုကားဆစ်စကေလ် return ပြန်လာပြီးမဲ ဒီနှစ်ခဲ ဆက်လုပ်မှာပါ။ နောက်ဆုံး ရိုကားဆစ်စကေလ်က return စဖြစ်တယ်။ ဒီ

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



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



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

```

# File: make_beeper_row2.py
def make_beeper_row():

```

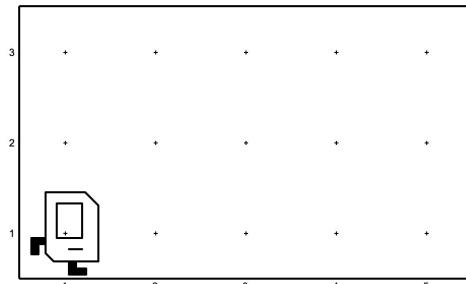
```

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

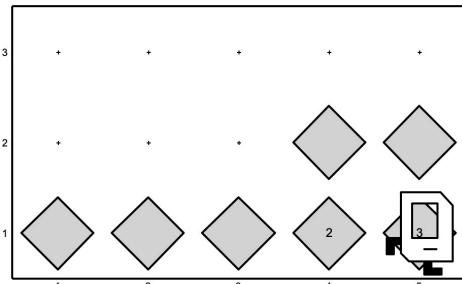
# initial call
make_beeper_row()

```

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



(က)



(ခ)

ပုံ ၄၃

၄.၂ ရီကားဆစ်ဖိနည်းဖြင့် ပုံစံဖြေရှင်းခြင်း

ရှေ့စက်ရှင်မှာ လေ့လာခဲ့တာက ရီကားဆစ်ဖိန်ရှင် ဘယ်လို အလုပ်လုပ်လဲပဲ ရီပါဘေးတယ်။ တစ်နည်းအားဖြင့် မကြန်စ်မေ (mechanism) ကို လေ့လာတာပါ။ အခုက်ခုခြင်း ရီကားဆစ်ဖိန်ရှင်တွေနဲ့ ပုံစံတွေ ဘယ်လိုဖြေရှင်းမလဲ ဆက်လက်လေ့လာပါမယ်။ ‘ရီကားဆစ်ဖိန်ရှင်တွေနဲ့ ပုံစံတွေ ဘယ်လိုဖြေရှင်းမလဲ’ ဆိုမယူတဲ့ ‘ရီကားဆစ်ဖိနည်းဖြင့် ပုံစံဖြေရှင်းခြင်း’ (solving problems recursively) ကို လေ့လာမှာပါ။

ရီကားဆစ်ဖိနည်းခြင်း ဥပမာ (၁)

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

```

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

```

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

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

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

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

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

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

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

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

ရှိကားဆစ်ဖန်ရှင် မှန်မမှန် စ်ဆေးခြင်း

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

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

if ဘလောက် မလုပ်ဆောင်ဘဲ return ဖြစ်သွားမှာပါ (ရီကားဆစ်စောင် မဖြစ်လိုက်ဘူး)။ သူည့်ဘိပါအတွက် ဖြစ်သင့်တဲ့အတိုင်း ဖြစ်ပါတယ်။ ဘိပါတစ်ခုပဲ ရှိရင်ရော ဘယ်လိုဖြစ်မလဲ။ ဘိပါကောက်တယ် သူညာဘိပါဖြစ်သွားပြီး ရီကားဆစ်စောင် ဖြစ်မယ်။ တစ်ကြိမ်ပဲ ဖြစ်မယ်။ if ဘလောက် အလုပ်မလုပ်ဘဲ return ပြန်မယ်။

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

သုံးခုရှိရင် ရီကားဆစ်စောင် နှစ်ခါဖြစ်ပြီးမှ return ပြန်မှာပါ။

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

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

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

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

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

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

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

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

ရိုကားဆုံး စဉ်းစားခြင်း ဥပမာ (၂)

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

ဖြေရှင်းမဲ့ ကိုစွဲကို ရှင်းကိုယ်တိုင်းနဲ့ သဏ္ဌာန်တူပြီး အရွယ်အစားအားဖြင့် တစ်ဆင့်ထက်တစ်ဆင့် သေး ယောက်တွေအဖြစ် ခွဲခြမ်းကြည့်ရမှာပါ။

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

clean_street ဖန်ရှင်ဟာ လက်ရှိဖြေရှင်းမဲ့ အရွယ်အစားထက် တစ်ဆင့်ယောက်တဲ့ ကိုစွဲကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူရပါမယ်။

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

လက်ရှိအရွယ်အစားကို သုထက်တစ်ဆင့်ယောက်တဲ့ ကိုစွဲဖြစ်သွားအောင် ဘာလုပ်ရမလဲ စဉ်းစားရတယ်။ ကွန်နာငါးခဲ့ လမ်းကိုရှင်းတဲ့ အခါကိုရှင်းတဲ့ အခါကို ကွန်နာ လေးခဲ့ ရှင်းစရာကျွန်အောင် ဘာလုပ်မလဲ။ ယော်သူယျာပြာရင် n ကွန်နာဆိုရင် $n - 1$ ကွန်နာ ရှင်းဖို့လို့တော့အောင် ဘာလုပ်မလဲ။

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

တစ်ဆင့်ယောက်တဲ့ ကိုစွဲ $n - 1$ ကို ဖြေရှင်းနိုင်ပြီးသားလို့ မှတ်ယူပြီး လက်ရှိကစွဲ n ကို ဘယ်လို့ ဖြေရှင်းမလဲ စဉ်းစားရပါမယ်။

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

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

```

အသေးငယ်ဆုံး ကိုစွဲကို ခွင့်ချက်အနေနဲ့ စဉ်းစားရမယ်။ ကွန်နာတစ်ခုဟာ အသေးငယ်ဆုံးကိုစွဲဖြစ်တယ်။ ဒီကိုစွဲကို ဘယ်လို့ ဖြေရှင်းမလဲ။

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

$n = 1$ ဖြစ်နေပြီ။

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

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

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

ရိုကားဆုံးနည်းနဲ့ လမ်းတစ်လမ်း ရှင်းလို့ရပါပြီ။ ကားရဲလ်ကမ္ဘာတစ်ခုလုံး ရှင်းဖို့ ရိုကားဆုံးဆုံးပါမယ်။

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

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

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

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

```

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

```

ဖြစ်ရမယ်။

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

```

# File: clean_world_recur1.py
from stanfordkarel import *

def main():
    clean_world()

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

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

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

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

```

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

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

# File: checkerboard_recur.py
from stanfordkarel import *

def main():
    mk_checkerboard()

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

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

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

def switch_row():
```

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

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

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

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

အခန်း ၅

ဒေတာ၊ အိပ်စ်ပရက်ရှင်းနဲ့ ဖေရီရောဘလ်များ

ရှုပိုင်း ကားရဲလ်အခန်း လေးခုမှာ လေ့လာခဲ့ကြတဲ့ အကြောင်းအရာတွေဟာ ပရိုကရမ်မင်း ဘာသာရပ်ရဲ့ အခြေခံအကျဆုံး ပင်မထောက်တိုင် သဘောတရားတွေလို့ ဆုံးရမှာပါ။ ဒီသဘောတရားတွေ မကျောက် ဘဲ ပရိုကရမ်ရေးလို့ မရပါဘူး။ ကွန်ဒီရှင်နယ်တွေဖြစ်တဲ့ if, if...else ပြန်ကျော်ခြင်းအတွက် for နဲ့ while loop၊ ဖန်ရှင်တွေ၊ top-down, bottom-up ပရိုကရမ်းမင်း၊ ရီကားရှင်းနဲ့ ရီကားဆစ်စ် စဉ်းစားခြင်း စတာတွေနဲ့ ပရိုကရမ်းမင်း ပူဇာတွေ ဖြေရှင်းဆိုင်ရင် ပရိုကရမ်မာလေ့ကား ပထမ တစ်ထစ် တက်လှမ်းအောင်ပြင်ပြီ ပြောနိုင်ပါတယ်။ ဒီသဘောတရားတွေကို ဘိက်နှာတွေ အရိုးရှင်းဆုံးနည်းနဲ့ နားလည်အောင် လေ့ကျင့်လို့ရအောင် ကားရဲလ်က ထောက်ကူးပေးတာပါ။ စက်ရှပ်လေး ကားရဲလ်ကို နှုတ်ဆက်ခဲ့ပြီး အခုံဆက်လက်လေ့လာကြမှာကတော့ ဒေတာ၊ အိပ်စ်ပရက်ရှင်းနဲ့ ဖေရီရောဘလ်တွေ အကြောင်းပါ။

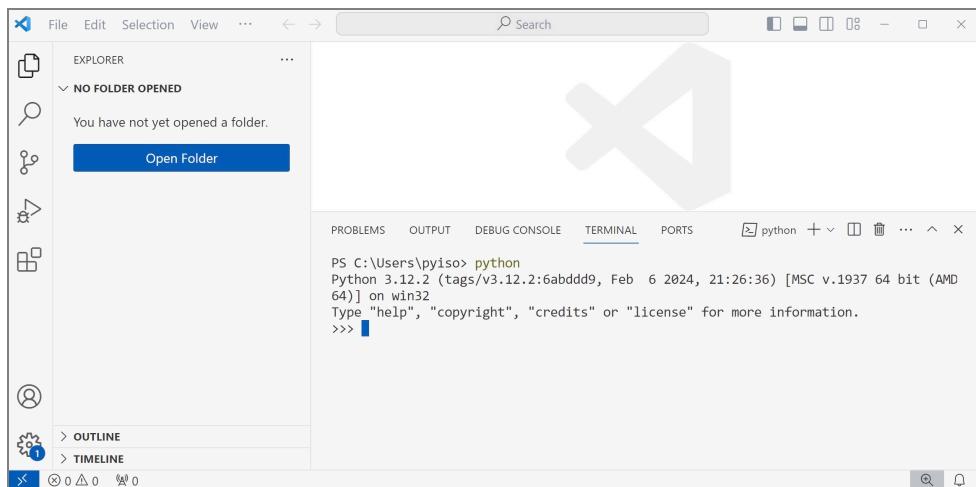
ကွန်ပျိုးတွေဟာ အချက်အလက် (ဒေတာ) အမျိုးမျိုးကို ကိုင်တွေယ်ဆောင်ရွက်ပေးနိုင်တယ်။ ကိုန်းကဏ်းတွေအပြင် စာသား၊ ရုပ်သံ စတာတွေကိုပါ လက်ခံ အလုပ်လုပ်ပေးနိုင်တယ်။ ဒီလိုလုပ်ဆောင်နှင့်စွမ်းဟာ ကွန်ပျိုးတွေကို နယ်ပယ်ပေါင်းစုံမှာ တွင်တွင်ကျယ်ကျယ် အသုံးချလာရခြင်းရဲ့ အဓိက အကြောင်းအရင်း ဆုံးရှင်လည်း မမှားဘူး။

ဒေတာ အမျိုးအစား အများအပြားရှိပေမဲ့ အခြေခံအကျဆုံးက ကိုန်းကဏ်းတွေပါ။ ကွန်ပျိုးတွေကို စတင်တိထွင်ဖို့ ကြိုးစားလာကြတဲ့ အဓိက အကြောင်းအရင်းကလည်း ကဏ်းသချာ အတွက်အချက်တွေကို လုပ်ဆောင်ရာမှာ လူတွေကို အကုအညီ ပေးဖို့အတွက်ပဲလို့ ဆုံးဖို့ကြပ်ပါတယ်။ ဒါ့အပြင် စာသား၊ ရုပ်သံ စတဲ့ အခြားဒေတာ အမျိုးအစားတွေကို ကွန်ပျိုးတွာထဲမှာ ဖော်ပြသမ်းဆည်းထားဖို့အတွက် ကိုန်းကဏ်းတွေကိုပဲ အသုံးပြုထားတယ်ဆိုတာ နောက်ပိုင်းမှာ နားလည်သိမြင် လာပါလိမ့်မယ်။ ဒါကြောင့်လည်း ယနေ့ခေတ် ကွန်ပျိုးတွေကို အစိုးပြုစိတ်တယ် ကွန်ပျိုးတွေလို့ ခေါ်တာဖြစ်တယ်။ ကိုန်းကဏ်းကို အခြေခံပြီး အလုပ်လုပ်တဲ့ ကွန်ပျိုးတွေပေါ့။

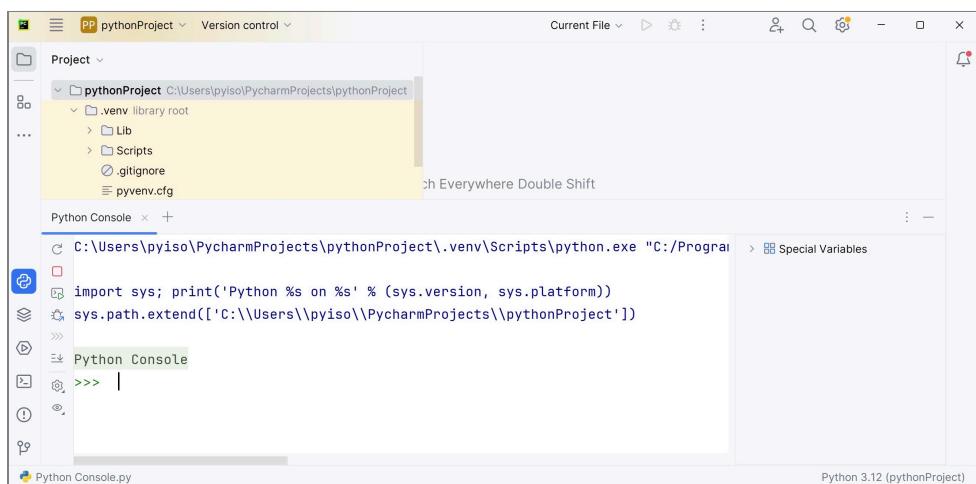
၅.၁ ကိုန်းကဏ်းများ

အခုံအခန်းအတွက် ပရိုကရမ်ကို အပိုင်းအစ အတိအတွေလေးတွေကို အလွယ်တကူ လက်တွေ စမ်းသပ်ကြည့်လို့ရအောင် Python Console ကို အသုံးပြုမယ်။ VS Code မှာ View မီနဲးမှာ Terminal ကိုဖွဲ့ (Ctrl + ` ရှေ့ကတ်ကိုနဲ့ ဖွင့်လို့လည်းရတယ်) ပြီး python ကွန်းများနဲ့ ကွန်ဆုံးလို့လို့ကို run ပါ။ PyCharm မှာဆိုရင် Python Console အိုင်ကွန်နှင့်ပြီး run ပါ ။ ပုံ (၅.၁)၊ (၅.၂)။

ကွန်ဆုံးလို့ 2 + 5 ထည့်ပြီး Enter ကိုနိုပ်ပါ။ အဖြေ 7 ထုတ်ပေးတာ တွေ့ရပါမယ်။



ጀ.၁၁ VS Code Python Console



ጀ.၁၂ PyCharm Python Console

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

အောက်ပါအတိုင်း တစ်ခုပြီးတစ်ခု ဆက်လက်စမ်းသပ်ကြည့်ပါ။

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

Values and Types

float တိုက်ပော်ပါ၏ လိုသလောက် တိကျလို့မရတဲ့ သဘောရှိတယ်။ အောက်ပါအတိုင်း စမ်းကြည့်ရင် 0.3 နဲ့ 1.0 ရသင့်တာ ဖြစ်ပေမဲ့ အတိအကျ အဖြမထဲက်ပါဘူး။

```
>>> 0.1 + 0.1 + 0.1
0.3000000000000004
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
0.999999999999999
```

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

float ကတော့ int နဲ့ မတူဘဲ အကြီးဆုံး/အင်ယ်ဆုံး ကန့်သတ်ချက်ရှိပါတယ်။ ဒီကန့်သတ်ချက် ကလည်း ကွန်ပျိုးတာစနစ်ပေါ် မူတည်ပါတယ်။ တစ်ကိုယ်ရောသံး ကန့်ပျိုးတာ အများစုအတွက် 1×10^{400} (၁ နောက် သုည အလုံး ၄၀၀) သို့မဟုတ် -1×10^{400} (အနှစ် ၁ နောက် သုည အလုံး ၄၀၀) ဟာ ကန့်သတ်ချက်ကျော်လွန်ပါတယ်။

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

ဒဿမကိန်းကို e အမှတ်အသားနဲ့ရေးထားတာပါ။ အကွားရာ e နောက် လိုက်တဲ့ ကိန်းပြည့်ကဏ္ဍးကို 10 ထပ်ကိန်းလို့ မှတ်ယူရပါတယ်။ $e3$ က 1×10^3 , $e-3$ က 1×10^{-3} ပါ။ အကွားရာ E အကြီးနဲ့လည်း ရေးနိုင်တယ်။ ထပ်ကိန်း ကြီးလွန်းတဲ့အတွက် ကန့်သတ်ချက် ကျော်လွန်ရင် `inf` (အနှစ်ကိန်းဆိုရင် `-inf`) ရပါမယ်။ `Infinity` ကို ဆိုလိုတာပါ။ (၁) မပြည့်တဲ့ ပမာဏ သေးငယ်လွန်းတဲ့ ကဏ္ဍးတွေကိုလည်း အနီးစပ်ဆုံး သုညအနေနဲ့ ယူပါတယ်။ အနီးစပ်ဆုံးယူတဲ့အခါ အပေါင်း/အနှစ်ကိုတော့ ခွဲခြားပေးပါတယ်။ e အမှတ်အသားနဲ့ရေးထားတဲ့ နောက်ထပ် ဥပမာလေး နှစ်ခုပါ

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

ကဏ္ဍးအလုံးအရေအတွက် များရင် 100,500 (တစ်သိန်းငါးရာ)၊ 1,500,000 (တစ်သိန်းငါးသိန်း) စသဖြင့် သုံးလုံးတစ်ဖြတ် ကော်မာခံရေးလေ့ရှိတယ်။ Python မှာတော့ ကော်မာအစား `_`(underline) နဲ့ သုံးလုံးတစ်ဖြတ် ခြားရေးနိုင်ပါတယ်။

```
>>> 1_500_000 + 100_500
1600500
>>> 200_000.33 + 3_800_000.22
4000000.55000000003
```

တိုက်ပတူသည့် ကိန်းဂဏ္ဍး အပိုစ်ပရက်ရှင်များ

အပိုစ်ပရက်ရှင်တွေမှာ ပါဝင်တဲ့ တိုက်ပ တစ်မျိုးတည်း ဖြစ်ရမယ် မရှိပါ။ တိုက်ပတူတဲ့ ကိန်းဂဏ္ဍးတွေ အပိုစ်ပရက်ရှင်တစ်ခုမှာ ရောပြီး ပါဝင်နိုင်ပါတယ်။

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

`int` နဲ့ `float` ရောနေရင် အိပ်စံပရက်ရှင် ရလဒ်သည် `float` တိုက်ပဲ ဖြစ်မှာပါ။ အစား (division) မှာတော့ အင်တီဂျာအချင်းချင်း စားတဲ့အခါမှာလည်း ရလဒ်က `float` ဖြစ်ပါမယ်။

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

အင်တီဂျာ ဒီပီးရှင်း၊ မောဒ္ဒါလို နှင့် ထပ်ကိန်းတင်ခြင်း

အကယ်၍ ဒုသေမကိန်းမထွက်ဘဲ ကိန်းပြည့် လိုချင်ရင် // ကို သုံးရပါမယ်။ ဒီအခါ အကြောင်းကိုဖယ်ပြီး စားလဒ်ကိုပဲ ကိန်းပြည့်အနေနဲ့ ရှုံးရမှာပါ။

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

သချုပ်မှာ ဒီလိုမျိုး၊ အစားကို အင်တီဂျာ ဒီပီးရှင်း (integer division) လိုခေါ်ပါတယ်။ အကြောင်းရှာမယ် ဆိုရင် % အော်ပရိတ်တာ ရှိပါတယ်။ % ကို မောဒ္ဒါလို (modulo) အော်ပရိတ်တာလို့ ခေါ်တယ်။ remainder အော်ပရိတ်တာလို့လည်း ခေါ်တယ်။

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

အင်တီဂျာ ဒီပီးရှင်းနဲ့ မောဒ္ဒါလိုကို အနှုတ်ကိန်းတွေနဲ့ သုံးမယ်ဆိုရင် သတိပြုပါ။ စားလဒ် အနှုတ် ကိန်း ဖြစ်ရင် // က ပိုင်ယ်တဲ့ အနှုတ်ကိန်းကို အနီးစပ်ဆုံး ယူမှာပါ။ တစ်နှစ်းအားဖြင့် 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 သူထက်ပိုင်ယတဲ့ -2 ကို အနီးစပ်ဆုံး ယူတယ်။ -3.1, -3.5, -3.8 တို့ကိုလည်း အနီးစပ်ဆုံး -4 ယူတာပါ။

မော်ဒူးလို့ အော်ပရိတ်တာ % သုံးတဲ့အခါ ရလဒ်ဟာ စားကိန်းရဲ့ sign အပေါ် မူတည်တယ်။ (အပေါင်း အနူတ်ကို ဆိုလိုတာပါ)။

```

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

```

မော်ဒူးလို့နဲ့ အင်တိဂျာ ဒီပါးရင်း အော်ပရိတ်တာ နှစ်ခုက အောက်ပါညီမျှခြင်းအရ ဆက်စပ်နေတာပါ။ စားကိန်း $B \neq 0$ ဖြစ်ပါတယ်။

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

ဒါကြောင့် $B = 10, A = -17$ 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 ပြောည်စေမှာ 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 အော်ပရီတောက $\star\star$ ပါ။ 2^4 နဲ့ $(3.3)^3$ ကို အခါ လို တွက်ပါတယ်။

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

သချုပန်ရှင်များ

ကိန်းကြောင်းတွေအကြောင်း လေ့လာလက်စနဲ့ `math` လိုက်ဘရီ သချုပန်ရှင်တေချို့ကိုလည်း တစ်ခါတည်း ဆက်ကြည့်လိုက်ရအောင်။ အဓိကက သချုပန်ရှင်ဆိုတာထက် ဖန်ရှင် အခြေခံအသုံးပြုပုံကို စပြီးလေ့လာ မှုပါ။ `math` လိုက်ဘရီက Python မှာ တစ်ခါတည်း ထည့်ထားပေးပြီးသား (built-in) လိုက်ဘရီပါ။ အင်စတောလ်လုပ်စရာ မလိုဘဲ အင်ပို့လုပ်ပြီး သုံးလို့ရတယ်။

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

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

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

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

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

အကြီးဆုံးဘုံးဆုံးကိန်းကို `gcd` ဖန်ရှင်နဲ့ ရှာတာပါ။ အင်တိဂျာ `input` တစ်ခုနဲ့အထက် လက်ခံတဲ့ ဖန်ရှင် ဖြစ်တယ်။ `input` ကြောင်းအားလုံးကို စားလို့ပြတ်တဲ့ အကြီးဆုံးကိန်းကို ရှာပေးတယ်။ ကိန်းပြည့်မဟုတ် တာ 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
```

လောကရှစ်သမ်၊ ထရီဂိန့်မေတရီ 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 သဘောတရား ပါရှိပါတယ်။ `int` နဲ့ `float` တိုက်ပ်မှာ ပါဝင်တဲ့ ကိန်းပြည့်တွေနဲ့ သယ်ယူနိုင်တဲ့ မိတ်ဆက်ပြီးတဲ့အခါ ‘တိုက်ပ်’ ဆိုတာဘာလဲ တိတိကျကျရှင်းပြလိုပါပြီ။ တိုက်ပ်တစ်ခုဟာ

- တန်ဖိုးတွေပါဝင်တဲ့ အစု (set) တစ်ခု နဲ့
- ငှုံးတန်ဖိုးများအပေါ်တွင် အသုံးချိန်ငါးတဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစုတစ်ခု

ဖြစ်ပါတယ်။ ဥပမာ `int` တိုက်ပ်ကို ကြည့်ရင် ကိန်းပြည့်တွေ ပါဝင်တဲ့ အစုနဲ့ ကိန်းပြည့်တွေအပေါ်မှာ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစု

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

ဖြစ်ပါတယ်။ `float` တိုက်ပ်ကတော့ ကိန်းစစ် (real numbers) တွေပါဝင်တဲ့ အစုနဲ့ ငှုံးတို့အပေါ်မှာ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစုတို့ ဖြစ်ပါတယ်။ `int` နဲ့ `float` တို့ကိုလည်း အော်ပရေးရှင်းတွေ တူတူဖြစ်နေတာ တွေ့ရမှာပါ။ ဒီလို့အမြဲဖြစ်မယ်လို့ မယူဆရပါဘူး။ တိုက်ပ် မတူတဲ့အခါ အသုံးချိန်တဲ့ အော်ပရေးရှင်းတွေ ကွားမြေးရှိပါတယ်။ ဥပမာ `str` တိုက်ပ် အော်ပရေးရှင်းတွေက `int` တို့ `float` တို့နဲ့ မတူပါဘူး။ `str` က `string` ရဲ့ အတိုကောက်ဖြစ်ပြီး စာသားတွေအတွက် အသုံးပြုပါတယ်။ မကြာခင် လေ့လာကြမှာပါ။

အပေါ်က 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 ပကဗောတန်ဖိုး ရှုံးပေးပါတယ်။ လိုအပ်ရင် ကိုယ်ပိုင်ဖန်ရှင်တွေ သတ်မှတ်ပြီး တိုက်ပ်တစ်ခုရဲ့ အော်ပရေးရှင်းတွေကို ဖြည့်စွက်တိုးချဲ့နိုင်ပါတယ်။

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

၅.၃ ဖေရီရောဲလ်များ

ဖေရီရောဲလ်ဆိုတာ တန်ဖိုးတစ်ခုကို ကိုယ်စားပြုတဲ့ နံမည်ပါပဲ။ နံမည်နဲ့ ငင်းကိုယ်စားပြုတဲ့ တန်ဖိုး တွဲဖက်ပေးဖို့ အဆိုင်းမနဲ့ (assignment) စတိတ်မနဲ့ကို သုံးရပါတယ်။

```
>>> age = 12
>>> weight = 35.5
```

age နဲ့ weight ဟာ ဖေရီရောဲလ်တွေ ဖြစ်ပါတယ်။ ညီမျှခြင်းသက်တာ (=) ကတော့ အဆိုင်းမနဲ့ အော်ပရိတ်တာပါ။ ဖေရီရောဲလ်နဲ့ တန်ဖိုး တွဲဖက်ပေးတဲ့ အော်ပရိတ်တာ ဖြစ်တယ်။ ဖေရီရောဲလ်နံမည်ကို variable identifier လို့လည်း ခေါ်ပါတယ်။ Identifier က နည်းပညာ အခေါ်အခြေပြီ။ Variable name က သာမန်လူ နားလည်တဲ့ နည်းနဲ့ ပြောတာပါ။ ဖေရီရောဲလ် တစ်ခုချင်း ထည့်ကြည့်ရင် ငင်းကိုယ်စားပြုတဲ့ တန်ဖိုးကို ပြန်ထုတ်ပေးတာ တွေ့ရမှာပါ။

```
>>> age
12
>>> weight
35.5
```

အိပ်စ်ပရ်ကျင်တွေက ဖေရီရောဲလ်တွေနဲ့ ဖြစ်နိုင်ပါတယ်။ အိပ်စ်ပရ်ကျင် တွက်ချက်ရင် ဖေရီရောဲလ် တန်ဖိုးနဲ့ အစားထိုး တွက်ချက်တယ်လို့ ယူဆရမှာပါ။ ဥပမာ

```
>>> age + 1
13
>>> weight / 2
17.75
```

ဖေရီရောဲလ်တစ်ခုကို အိပ်စ်ပရ်ကျင်ရလဒ်နဲ့ အဆိုင်းမနဲ့ လုပ်လို့ရပါတယ်။ rect_area ကို အောက်တွင် ကြည့်ပါ။ အလျား အနဲ့ မြှောက်လဒ်ကို အဆိုင်းမနဲ့ လုပ်ထားတာ တွေ့ရပါမယ်။

```
>>> rect_width = 22.5
>>> rect_length = 10
>>> rect_area = rect_width * rect_length
>>> rect_area
225.0
```

အဆိုင်းမနဲ့ ခတိတ်မနဲ့

ဖေရီရောဲလ်တစ်ခုဟာ အချိန်တစ်ချိန်မှာ တန်ဖိုးတစ်ခုကိုပဲ ကိုယ်စားပြုနိုင်တယ်။ ဒါပေမဲ့ အချိန်ကာလပေါ်မူတည်ပြီး တန်ဖိုးပြောင်းနိုင်တယ်။ (တစ်ချိန်တည်းမှာ တန်ဖိုးနှစ်ခု မဖြစ်နိုင်ဘူး)။ ဥပမာ x တန်ဖိုးဟာ

ပထမ 10 ပါ။ ဒုတိယ အဆိုင်းမန်လုပ်ပြီးတဲ့ အချိန်မှာ အဲဒီ x ကပဲ 1000 ဖြစ်နေမှာပါ။

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

၅.၄ စာသားများ

စာသား (text) ဟာ အသုံးအများဆုံး ဆက်သွယ်ဆောင်ရွက်ရေး ကြားခံနယ်တစ်ခုပါ။ ဝက်ဘ်ဆိုက် စာမျက်နှာ၊ အီးမေးလု၊ အီးဘွတ်ခုနဲ့ အီးလက်ထရွန်းနှစ် စာရွက်စာတမ်း (e-documents) စတာတွေမှာ ရုပ်သံတွေ အသုံးပြုလာကြပေမဲ့ စာသား အဓိကဖြစ်နေဆဲပါပဲ။ ဆိုရယ်မိဒီယာ၊ ဂိမ်းနဲ့ အခြားအပ်ပွဲတွေ ဟာလည်း စာသားနဲ့ မက်င်းနိုင်ကြပါဘူး။ ဒါကြောင့် ပရိုကရမ်းမင်းအတွက် စာသားဟာ ဘယ်လောက်ထိ အရေးပါကြောင်း အများကြီးပြောစရာ လို့မယ်မထင်ပါဘူး။

ပရိုကရမ်းမင်းမှာ စာသားကို *string* လို့ခေါ်ပြီး ကာရ်ကာ (character) တွေနဲ့ စီတန်ဖွဲ့စည်းထားတယ်။ ကာရ်ကာဆုံးတာ အခြေခံ သတင်းအချက်အလက် ယူနစ်တစ်ခုပါပဲ။ အကွားရာ၊ ဂဏန်း (digit)၊ သက်တ သို့မဟုတ် ကွန်ထရိုးလုံကုံး တစ်ခုခဲ့ ဖြစ်နိုင်ပါတယ်။ ဥပမာ A, B, C, \$, @, #, 1, 3, _ စသည်ဖြင့်။ Double quotes ("") တစ်စုံကြား ညှပ်ရေးထားတဲ့ ကာရ်ကာတွေ အသီအတန်းလိုက်ကို စာသားအနေနဲ့ ယူဆတယ်။ Python မှာ စာသားရဲ့ တိုက်ပ်ဟာ *str* ဖြစ်တယ်။ *string* ကို အတိုကောက် ယူထားတာပါ။

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

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

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

စာသားတစ်ခုမှာ ပါဝင်တဲ့ ကာရ်ကာ အရေအတွက်ကို *len* ဖန်ရှင်နဲ့ စစ်ကြည့်နိုင်ပါတယ်။ ကာ ရ်ကာ 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 တိုက်ပ်ရဲ့ အခြေခံကျတဲ့ အော်ပရေးရှင်းတစ်ခုက စာသားတစ်ခုနဲ့ တစ်ခု ဆက်တာပါ။ + အော် ပရိုတ်တာနဲ့ စာသားတွေကို ဆက်နိုင်ပါတယ်။

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

စာသားအချင်းချင်းပဲ ဆက်လိုက်ပါတယ်။ စာသားနဲ့ ကိန်းကဏ္န်း ဆက်လို့မရပါဘူး။ အောက်ပါအတိုင်း စမ်းကြည့်တဲ့အခါ str နဲ့ float ဆက်လို့မရဘူးလို့ အယ်ရာမက်ဆွဲချုပ် 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 ကိန်းကဏ္န်းတစ်ခုကနေ စာသားကို ထုတ်ပေးပါတယ်။ မူးရှင်းကိန်းကဏ္န်းကို စာသားဖြစ်အောင် ပြောင်းလိုက်တာ မဟုတ်ပါဘူး။ ကိန်းကဏ္န်း တန်ဖိုးကနေ ငြင်းကိုဖော်ပြတဲ့ စာသားကို ဖန်ရှင်က ပြန်ထုတ်ပေးတာပါ။

```
>>> str(pi)
'3.141592653589793'
```

ထွက်လာတဲ့ တန်ဖိုးဟာ စာသားဖြစ်တဲ့အတွက် single quote ပါနေတာ သတိပြုပါ။ pi တန်ဖိုးနဲ့ စာသား အခုံလို့ ဆက်ရပါမယ်။

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

str ဖန်ရှင်နဲ့ စာသားရအောင် အရှင်လုပ်ပြီးမှ ဆက်ထားတာပါ။
စာသားကနေ ကိန်းကဏ္န်း လိုချင်ရင် int နဲ့ float ဖန်ရှင် သုံးနိုင်ပါတယ်။

```
>>> int('1024')
1024
>>> int('1024') * 2
2048
>>> float('2.4') * 3
7.19999999999999
```

ကဏ္န်းပြောင်းလို့မရတဲ့ စာသားဖြစ်နေရင် အယ်ရာဖြစ်မှာပါ။

```
>>> 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 ဒသမ ပါနေတာကြောင့် int ပြောင်းလို့ မရတဲ့အတွက် အယ်ရာတက်တာပါ။ စာသားကို * အော်ပရိတ်တာနဲ့ ပွဲးယူလို့ရတယ်။

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

'hello' သုံးခါ ဆက်လိုက်တာပါ။ string နဲ့ အကြမ်အရေအတွက် ကိန်းပြည့် ဖြစ်ရပါမယ်။ သုည သို့မဟုတ် အနှစ်ကိန်း ဖြစ်နေရင် empty string ရမှာပါ။

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

စာသားနဲ့ ဂဏန်း ဖလှယ်လို့ရပါတယ်

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

string ဖော်ရောဲလ်များ

ဖော်ရောဲလ်တွေကို string တန်ဖိုးတော့အတွက်လည်း အသုံးပြန်တယ်။ အောက်ပါ အိပ်စာရက်ရှင်တွေ ကို နားလည်နိုင်မလား ကြိုးစားကြည့်ပါ။ ဖော်ရောဲလ် တစ်ခုချင်းကို သူ့ရဲ့တန်ဖိုးနဲ့ အစားထိုးပြီး +, *, * အော်ပရိတ်တာတွေ အလုပ်လုပ်ပုံနဲ့ ဆက်စပ်စဉ်းစားရင် ဘာကြောင့် အခုလို့ အဖြေထွက်လဲ ခန့်မှန်းနိုင်မှာပါ။ 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 စာသားအပိုင်း ကိုဘုံးကနေ ကာရက်တာ တစ်လုံး ချင်း ရိုက်ရုံပါပဲ။ စပယ်ရှယ် ကာရက်တာ တချို့ကိုတော့ ကိုဘုံးကနေ တိုက်ရိုက် ရိုက်ထည့်လို့ မရဘဲ သီးခြားနည်းလမ်းတစ်ခုနဲ့ ရေးပေးရပါတယ်။ ဥပမာ စာသားထဲမှာ tab ကာရက်တာအတွက် \t နဲ့ newline အတွက် \n ရေးရမှာပါ။ ကိုဘုံးကနေ tab ကိုး၊ enter/return ကိုး နှိပ်ပြီး တိုက်ရိုက်ထည့်လို့မရပါဘူး။ သီးခြားအဓိပ္ပာယ် တစ်ခုအတွက် \ နဲ့တဲ့ ကာရက်တာအတွဲလိုက်ကို escape sequence လို့ခေါ်ပြီး \ ကိုတော့ escape character လို့ ခေါ်ပါတယ်။

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

```
>>> tabs_eg
'Line 1\t\t1,000,000\nLine 1000\t10,000'
```

Escape sequence တွေကို **bold** ဖော်နဲ့ ပြထားပါတယ်။ Python ကွန်ဆိုလိမှာ \t နဲ့ \n ကို အရှိ အတိုင်း ပြန်ပါတယ်။ 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 \ " လိုပေးပါမယ်။ Single quote တစ်စုံနဲ့ စာသားထဲမှာ ' ပါနေရင်လည်း \ ' လိုပေးပါမယ်။

```
>>> '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 သိမဟုတ် 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။ ကျော်တဲ့တစ်မျိုးကတော့ ပါရင်လည်းရာ မပါလည်း ပြသုနာမရှိဘူး။ အောက်ပါတို့က ဂရိစိုက် 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?"'
```

၅.၅ အိပ်စပ်ရက်ရှင်များ

တိုက်ပဲ ဆိုတာဘာလဲ အကြမ်းဖျဉ်း ရှင်းပြဲ ပြီးပါပြီ။ ကိုယ်တော်များနဲ့ တာသား တိုက်ပဲ တရာ့တော်များနဲ့ လေ့လာခဲ့ပြီးပြီ။ တကယ်တော့ အိပ်စပ်ရက်ရှင် (expression) ဆိုတာလည်း အသစ်အဆန်း မဟုတ်ပါဘူး။ တန်ဖိုးတစ်ခုဟာ အရှိုးရှင်းဆုံး အိပ်စပ်ရက်ရှင်လို့ ဆိုနိုင်ပါတယ်။ "Hello", 2.3 စတဲ့ တန်ဖိုးတွေ့ဟာ အိပ်စပ်ရက်ရှင်တွေပါပဲ။ ဗေဒရောဘာလည်း တန်ဖိုးကို ကိုယ်စားပြုတဲ့အတွက် အိပ်စပ်ရက်ရှင် လို့ ယူဆရမှာပါ။

ရှိုးရှင်းတဲ့ အိပ်စပ်ရက်ရှင်တွေကနေတစ်ဆင့် ပေါင်းစပ် အိပ်စပ်ရက်ရှင် (compound expression) တွေ ဖွဲ့စည်းတည်ဆောက် ယူနိုင်ပါတယ်။

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

အိပ်စပ်ရက်ရှင်ကို ရှုတောင့်အမျိုးမျိုးကနေ အဓိပါယ်ဖွဲ့စည်းကြတာ တွေ့ရှုပါတယ်။ “အိပ်စပ်ရက်ရှင် ဆိုတာ တန်ဖိုးတစ်ခု ပြန်ပေးတဲ့ အော်ပရေးရှင်း အတွဲအဆက်ဖြစ်တယ်” လို့ဆိုရင် အတိုင်းအတာတစ်ခုအထိ တိတိကျကျရှိပြီး နားလည်ရလည်း လွယ်ပါတယ်။ တရာ့တော်မှာတော့ အိပ်စပ်ရက်ရှင်ကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်လို့ သတ်မှတ်ပါတယ်။

ပထမအဓိပါယ်အရ အိပ်စပ်ရက်ရှင်နဲ့ စတိတ်မန့် မတူဘူးလို့ ယူဆပါတယ်။ ဒီရှုတောင့်မှာ စတိတ်မန့်ဟာလည်း အော်ပရေးရှင်း အတွဲအဆက်ဖြစ်ပေမဲ့ တန်ဖိုးပြန်မပေးဘူး။ အိပ်စပ်ရက်ရှင်ကတော့ တန်ဖိုးပြန်ပေးရမှာပါ။ $3 * 2$ ကို လုပ်ဆောင်တဲ့အခါ 6 ရပါတယ်။ ဒါကြောင့် $3 * 2$ ဟာ အိပ်စပ်ရက်ရှင် ဖြစ်တယ်။ `result = 3 * 2` က စတိတ်မန့်ဖြစ်တယ်။ အဆိုင်းမန့်ဟာ ဗေဒရောဘာလို့ တန်ဖိုးတစ်ခုနဲ့ တွဲဖက်ပေးဘာ။ တန်ဖိုးပြန်မပေးဘူး။

ဒုတိယအဓိပါယ်အရ အိပ်စပ်ရက်ရှင်သည်လည်း စတိတ်မန့်ပဲ။ စတိတ်မန့်တွေကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်နဲ့ ပြန်မပေးတဲ့ စတိတ်မန့် အုပ်စုနှစ်စု ခွဲခြားတယ်။ တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်တွေကို အိပ်စပ်ရက်ရှင် သို့မဟုတ် အိပ်စပ်ရက်ရှင်စတိတ်မန့်လို့ ဒုတိယအဓိပါယ် သတ်မှတ်ချက်အရ ခေါ်တာပါ။

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

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

`print(3)` ကတော့ အိပ်စ်ပရက်ရှင် မဟုတ်ပါဘူး။ အခုလို စမ်းကြည့်ရင် ၃ ထုတ်ပေးတဲ့အတွက် အိပ်စ်ပရက်ရှင်လို့ ထင်စရာ အကြောင်းရှိပါတယ်။

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

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

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


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။

ပရိုကရမ်လက်တွေ့ရေး လေ့ကျင့်ဖို့အတွက် လိုအပ်တဲ့ ဆောဖံ့ဌတွေ ထည့်ထားရပါမယ်။ Python programming language အတွက် Python ဆောဖံ့ဌ ထည့်ရပါမယ်။ Python ဆောဖံ့ဌမရှိဘဲ Python ကုဒ်တွေ၊ Python ပရိုကရမ်တွေ run လို မရပါဘူး။ Python အပြင် ပရိုကရမ်ကုဒ် ရေးဖို့ အတွက် အထောက်အကူဗြာ ဆောဖံ့ဌပဲတစ်ခုလည်း လိုအပ်တယ်။

ပရိုကရမ် ကုဒ်ရေးဖို့အတွက် PyCharm သိမဟုတ် Visual Studio Code (VS Code) ကို အသုံးပြုနိုင်ပါတယ်။ အက်ဆေးတစ်ပုဒ်ရေးတဲ့အခါ မိုက်ခရိုဆောဖံ့ဌ Word နဲ့ရေးလိုရသလို ရှိုးရှိုးရှင်းရှင်း Notepad လောက်နဲ့ ရေးလိုလည်း ရတာပါပဲ။ အက်ဆေးရဲ့ အဓိပ္ပာယ်ကသာ အစိုကပါ။ ပရိုကရမ်ကုဒ် ရေးတဲ့အခါမှာလည်း ခီးခေါ်ပါပဲ။ ဆောဖံ့ဌပဲတွေ ပရောဂျက်ကြိုးတွေ တည်ဆောက်တဲ့အခါ လိုအပ်မဲ့ ဖီချာတွေ အားလုံး စုံစုံလင်လင်ပါပြီးသား PyCharm လို Integrated Development Environment (IDE) ဆောဖံ့ဌပဲနဲ့ ကုဒ်တွေရေးလိုရသလို ပါ့ပေါ့ပါ့ပါ့နဲ့ လိုအပ်မှုပဲ လိုတဲ့ဖီချာအတွက် extensions (plugin လိုလည်းခေါာယ်) ထည့်သွင်းရတဲ့ Visual Studio Code (VS Code) လို ကုဒ်အသိဒ်ဘာ (Code Editor) ဆောဖံ့ဌပဲ သုံးပြီး ရေးရင်လည်း ရတာပါပဲ။ နောက်ဆုံး ကုဒ်နည်နည်း ဖိုင်နည်းနည်း ရှိုးရှင်းတဲ့ ပရိုကရမ်လေးတွေဆုံးရင် Notepad နဲ့ ရေးလိုတောင် ရပါတယ်။ ကုဒ်တွေများမယ် ဖိုင်တွေ များမယ်၊ အသင်းအစွဲလိုက် ပူးပေါင်းရေးရတဲ့ ပရိုကရမ်မျိုးတွေ ဆိုရင်တော့လည်း Notepad လောက်နဲ့ အဆင်မပြနိုင်တော့ဘူးပေါ့။

PyCharm ရော VS Code ထည့်သွင်းနည်းပါ ဖော်ပြပေးပါမယ်။ မိမိ နှစ်သက်ရာ အဆင်ပြောရာ သို့မဟုတ် နီးစပ်ရာ ပရိုကရမ်ဟာ အသိမိတ်ဆွေ အကြုပြုတဲ့ တစ်ခုကို ရွေးချယ်သုံးပါ။ စလေ့လာသူအနေနဲ့ PyCharm ကို အသုံးပြုတာ ပိုအဆင်ပြောမယ်လို ထင်တယ်။ PyCharm သုံးကြည့်လို မိမိကွန်ပျိုးတာ မှာ နေးလွန်းတယ်ဆိုရင် VS Code ကိုစမ်းကြည့်ပါ။ နှစ်ခုလုံး စက်အရမ်းကောင်း/မြင့် ဖို့ မလိုပါဘူး။ တော်ရုံး အတန်အသင့်ကောင်းတဲ့ စက်လောက်နဲ့ အဆင်ပြောပါတယ်။

မိမိကိုယ်တိုင်က ကွန်ပျိုးတာ အသုံးပြုပဲ အခြေခံ အားနည်းပြီး ဖော်ပြပေးထားတဲ့ အတိုင်း တစ်ဆင့် ချင်း အင်စတောလ် လုပ်တာလည်း အဆင်မပြောဖြစ်နေရင် ဒီဘာအုပ်ရဲ့ အောက်ပါ ဖော်သွာတ်ခံနဲ့ ယူကျိုး။

ချုပ်နယ်တွေမှာ ကြည့်ရှုမေးမြန်း အကူအညီ တောင်းနိုင်ပါတယ်။ ဒါမှမဟုတ် အတွေးအကြိုးတဲ့ နီးစပ်ရာ အသိမိတ်ဆေး/ညီကိုမောင်နှစ်မဲ တစ်ယောက်ယောက်ရဲ့ အကူအညီယူပြီး အင်စတောလ်လုပ်ပါ။

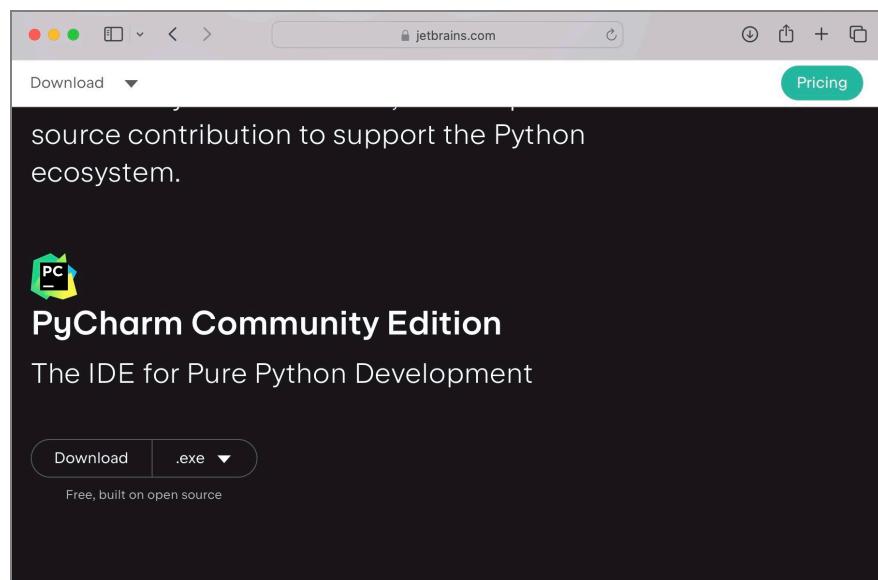
<https://www.facebook.com/bpwp>

<https://www.youtube.com/bpwp>

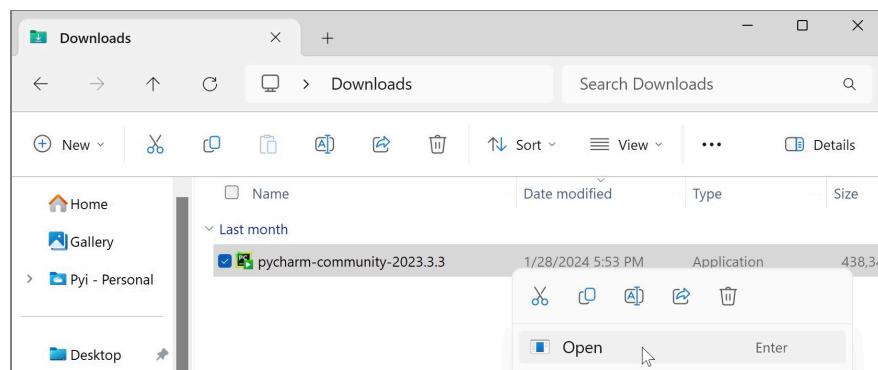
PyCharm အတွက် အရင်ဖော်ပြပေးပါမယ်။ VS Code အတွက် စာမျက်နှာ ၉၆ မှာ ကြည့်ပါ။

Python နှင့် PyCharm IDE ထည့်သွင်းခြင်း

<https://www.jetbrains.com/pycharm/download/> လင့်ကိုဖွင့်ပါ။ ဝဘ်စာမျက်နှာ အောက်ဘက် နည်းနည်း ဆွဲချုပ်ကိုရင် PyCharm Community Edition ဒေါ်င်းလုပ်ခလုတ်ကို တွေ့ရပါမယ်။ ပုံ (က/ခ) ကိုကြည့်ပါ။



ပုံ ၃/၁၁



ပုံ ၃/၂၂

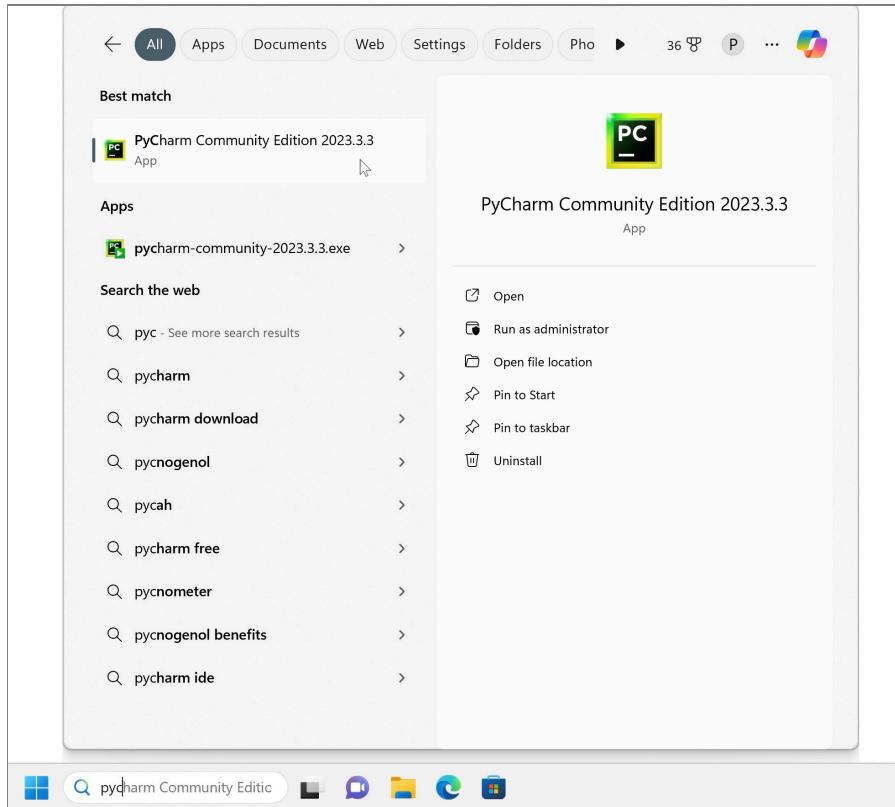
PyCharm Community Edition ကို အောင်လုပ်လုပ်ပါ။ (ဝဘ်စာမျက်နှာ အပေါ်ပိုင်းက ၀၂၍ သုံးရတဲ့ PyCharm Professional ကို အောင်လုပ် မှားမလုပ်မိန့် သတိပြုပါ)။ အင်စတော်လာဖိုင်ကို ညာကလစ်နှုပ်ပြီး Open လုပ်ပါ (ပုံ ၂/၂ ကိုကြည့်ပါ)။ Yes/No မေးတဲ့အခါ Yes နှုပ်ပါ။ Next > ကို နှုပ်၍ ရှေ့ကိုဆက်သွားပြီး နောက်ဆုံးမှာ Install နှုပ်ပြီး ကွန်ဖန်လုပ်ပါ။ အင်စတော်ပြီးသွားရင် Finish နှုပ်ပါ။

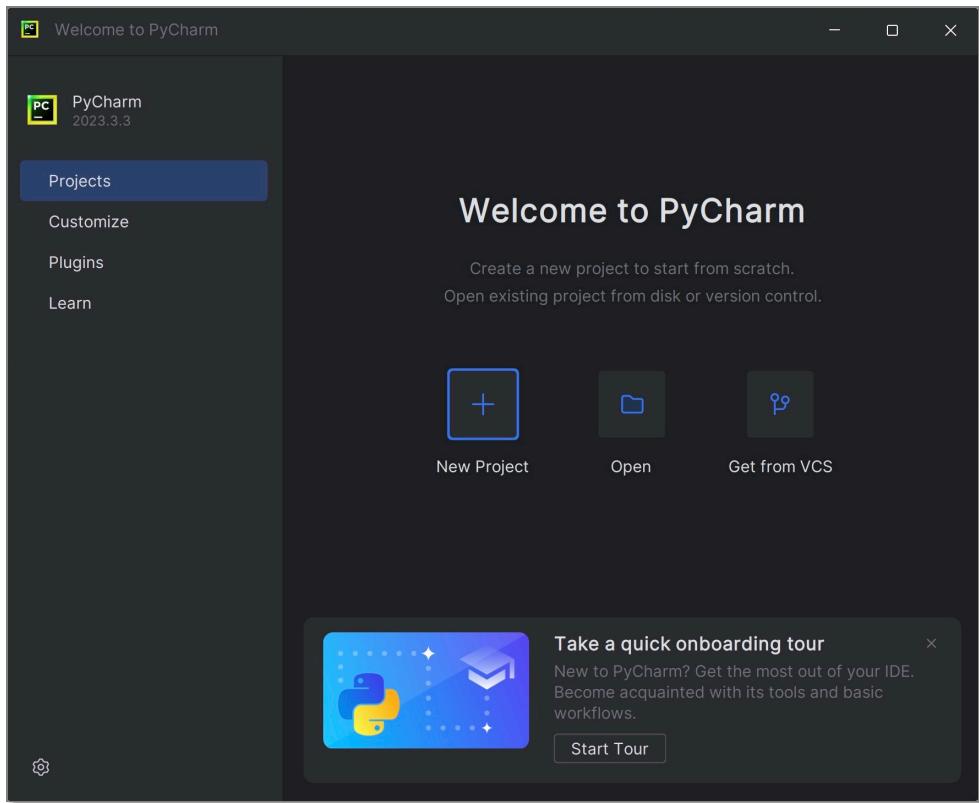
၀၂:၃၀ နာရီ: Taskbar Search ကနေ PyCharm ကိုရှာပြီးဖွင့်ပါ (ပုံ ၂/၃ ကို ကြည့်ပါ)။ သဘောတူ ကြောင်း ကွန်ဖန်လုပ်ခိုင်းရင် ချက်ချက်ဘောက်စ် ချက်ချက်လုပ်ပြီး Continue နှုပ်ပါ။ ဒေတာပိုချင်လား ထပ် မေးပါလိမ့်မယ်။ Don't Send နှုပ်ပါ။ Welcome စခရင်ကိုပေါ်လာမယ်။ ပုံ (၂/၄) မှာ ကြည့်ပါ။

ဒီစာရေးနေချိန် လက်ရှိ PyCharm ဗားရှင်းက ၂၀၂၃ ပါ။ သိပ်မကြာခင် ၂၀၂၄ ထွက်ပါတော့မယ်။ အကယ်၍ လက်ရှိဗားရှင်းထက် နိမ့်တဲ့ဗားရှင်းတွေကို အောင်လုပ် လုပ်ချင်ရင် အောက်ပါ လင့်ကို သွားပါ။

<https://www.jetbrains.com/pycharm/download/other.html>

ဗားရှင်း ၂၀၂၄/၂၂ ထွက်ပြီးတဲ့ အချိန်မှာ ၂၀၂၃ ဗားရှင်းကို လိုချင်ရင် ဝဘ်စာမျက်နှာမှ ရှာပြီး အောင်လုပ်လုပ်ပါ။ ၂၀၂၃ မှာလည်း ဗားရှင်းအခွဲတွေ ရှိပါသေးတယ်။ လက်ရှိအမြဲ့အမွှေ့ဗားရှင်းအခွဲ (ဥပမာ ၂၀၂၃.၃.၃) ကို သုံးလို့ရပါတယ်။ ၂၀၂၄/၂၂ သုံးမယ်ဆိုရင်လည်း ပြဿနာတော့ မရှိပါဘူး။ Update ဗားရှင်းဖြစ်တဲ့အတွက် ပုံတွေမှာ ပြထားတာနဲ့တော့ ကွားခြားချက်တရာ့၍ ရှိကောင်းရှိနိုင်ပါတယ်။

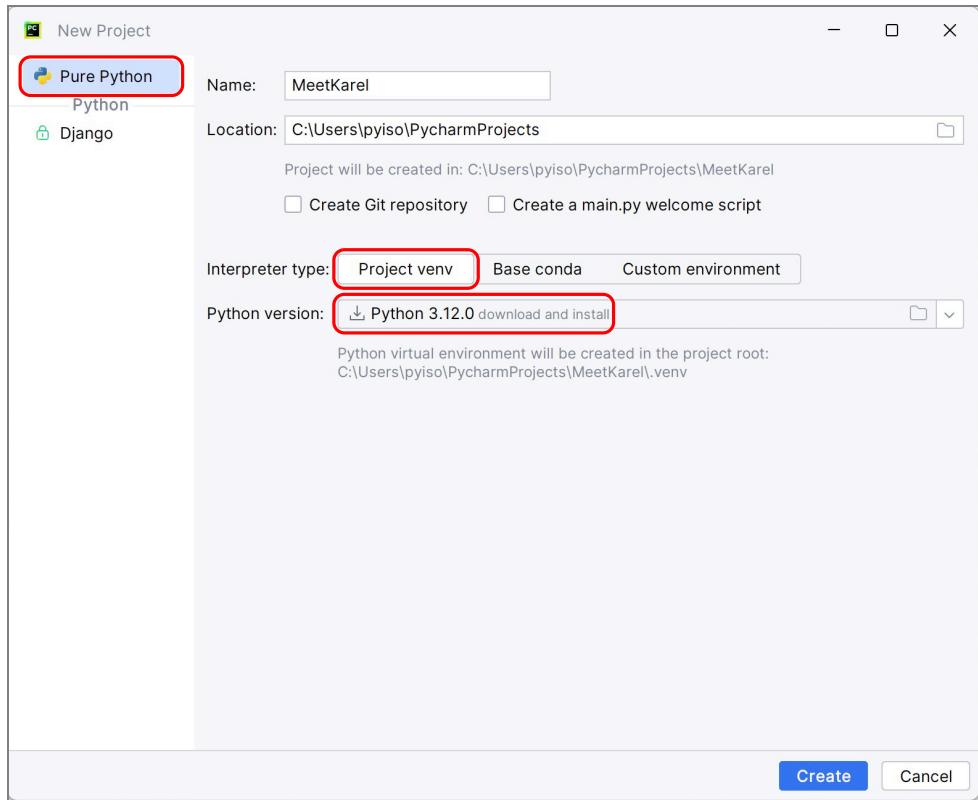




ပုံ ၂/၄

PyCharm IDE ဆိတာဘာလ

PyCharm ဟာ Python နဲ့ ဆောင်ဝဲ ရေးဖို့အတွက် အထောက်အကူဗြိ Integrated Development Environment(IDE) ဆောင်ဝဲဖြစ်ပါတယ်။ စာစီတရုက်လုပ်တဲ့အခါ Microsoft Word ကို အသုံးပြု ကြသလိုပဲ Python ကုဒ်ရေးဖို့ PyCharm ကိုထုတဲ့ သဘောပေါ့။ PyCharm IDE ၏ Python ပရောဂျက်တွေ အတွက် အခိုက်ရည်ရွယ်တယ်။ အထောက်အဦးတစ်ခု၊ တံတားတစ်ခု ဆောက်လုပ်တာ ကို ပရောဂျက်လို့ ပြောလေ့ရှိသလို ပရိုကရမ်/ဆောင်ဝဲတစ်ခု တည်ဆောက်တာကိုလည်း ပရောဂျက်လိုပဲ သုံးနှုန်းပါတယ်။ တစ်ဦးတစ်ယောက်တည်း ရေးတဲ့ ပရိုကရမ်အသေးလေးတွေ အတွက် PyCharm ကို အသုံးပြုနိုင်သလို ပရိုကရမ်မာတွေ အဖွဲ့လိုက်နဲ့ တည်ဆောက်ရတဲ့ ပရောဂျက်ကြီးတွေ အတွက်လည်း သုံးပါတယ်။ ဒီစာအုပ်မှာတော့ PyCharm ရဲ့ အဆင့်မြင့်ဖို့ချာတွေကို အသုံးပြုမှာ မဟုတ်ပါဘူး။ ပရိုကရမ်းမင်း စလေ့လာသူတွေကို လွှာယ်ကူးအဆင်ပြေစေတဲ့ အခြေခံ ဒီချာတွေလောက်ပဲ အသုံးပြုမှုပါ။



ပုံ ၂/၅ ပရောဂျက် အသစ်ယူခြင်း

PyCharm ပရောဂျက်ဆောက်ခြင်း

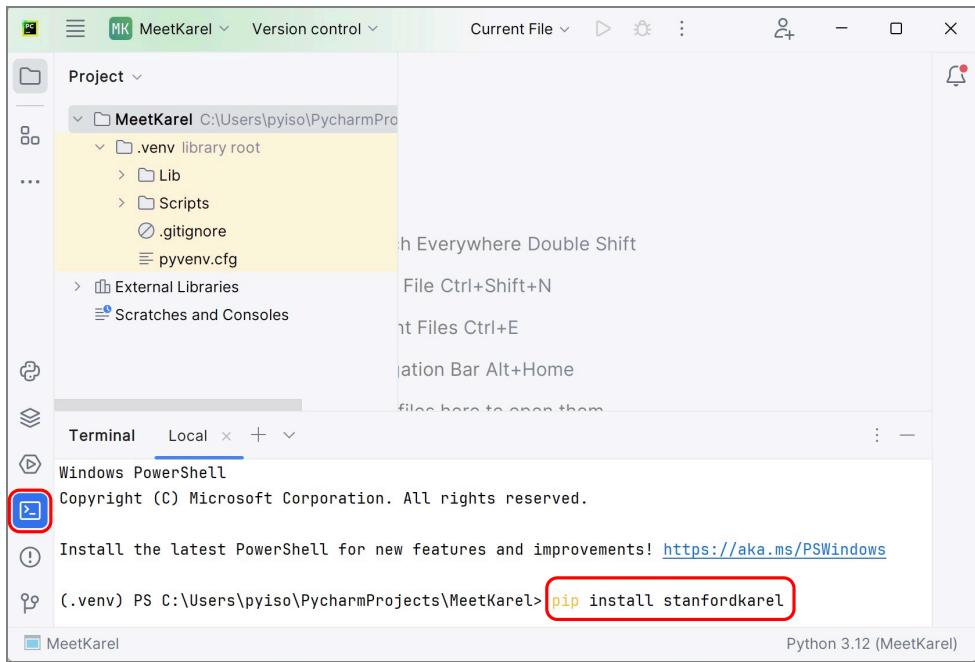
အင်စတောလုပ်ပြီးရင် PyCharm IDE ကိုဖွံ့ဖြိုး စေရင်မှ သို့မဟုတ် File မိန္ဒီးမှ New Project နှင့်ပြီး ပရောဂျက် အသစ်ယူပါ (ပြီးခဲတဲ့ ပုံ ၂/၅ ကို ပြန်ကြည့်ပါ)။ နံမည်ကို MeetKarel သေးပါ။ ပုံ (၂/၅) မှာ တွေ့ရတဲ့အတိုင်း Pure Python, Proj venv နဲ့ Python ဗားရှင်း 3.12.xx ကို ရွေးပါ။ xx က အမြားကဏ္ဍား ဖြစ်နေနိုင်တယ်။ အစိက ဗားရှင်း 3.12 သာ ဖြစ်ပါခေါ်။ Create ခလုတ်နှင့်ပါ။ ပရောဂျက်အတွက် Python ကို အင်စတောလုပ်ပါလိမ့်မယ်။

မိမိလေ့လေနေတဲ့ အခန်းတစ်ခုချင်းစီအတွက် ပရောဂျက်တစ်ခု ဆောက်နိုင်ပါတယ်။ အခန်း (၂) အတွက် Chapter02၊ အခန်း (၃) အတွက် Chapter03 စသည်ဖြင့်။ သက်ဆိုင်ရာအခန်းအလိုက် ကုဒ္ဓိုင်တွေကို ပရောဂျက်တစ်ခုစီမှာ ထားတဲ့အတွက် ဖိုင်တွေများပြီး ရှုပ်ထွေဖောင်းပွဲနေတဲ့ ပြဿနာ မရှိတော့ဘူး။ ကုဒ္ဓိုင်တွေကို ပရောဂျက်တစ်ခုခဲ့မှာပဲ ဖိုဒ်အလိုက်ခဲ့ထားလို့ရပေမဲ့ စလေ့လာသူအနေနဲ့ ပရောဂျက်တစ်ခုချင်း ခဲ့ထားတာလောက် မလွယ်ကူဘူး။

ပရောဂျက် အသစ်ယူတဲ့အခါ တည်နေရာ Location: ကို သူ့ရှိခိုက်အတိုင်း ထားနိုင်သလို မိမိထားချင်တဲ့နေရာကို ညာဘာက်စွန်း ဖိုဒ်အရိုင်ကွန်လေးနှင့်ပြီး ပြောင်းလို့ရတယ်။

stanfordkarel လိုက်ဘရီ အင်စတောလုပ်ခြင်း

ပုံ (၂/၆) မှာ အနီရောင် ဝိုင်းပြထားတဲ့ အိုင်ကွန်ကို နှင့်ပြီး Terminal ကိုဖွံ့ဖြိုး ဖြစ်ပါ။ Terminal မှာ အောက်ပါ ကွန်မန်းဖြင့်



ပုံ ၂/၆ Karel လိုက်ဘရိ အင်စတောလ်လုပ်ခြင်း

```
pip install stanfordkarel
```

ကားရဲလိုက်ဘရိကို အင်စတောလ်လုပ်ပါ။ ပုံ (၂/၆) မှာ အနီရောင် စိုင်းပြထားပါတယ်။ ခက္ကာတဲ့ အခါ အခုလို မက်ဆော်ချုတွေ 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 မက်ဆော်ချု တွေ့ရရင် အင်စတောလ်လုပ်တာ အောင်မြင်လိုပါ။

မှတ်ချက်။ ပေါ်ရပါမယ်။ ကားရဲလိုက်ဘရိ အသစ်ဆောက်တိုင်း လိုအပ်တဲ့ လိုက်ဘရိကို တစ်ခါတပ်ပြီး အင်စတောလ် လုပ်ရပါမယ်။ ကားရဲလိုက်ဘရိ ပရောဂျက်တစ်ခုစီအတွက် stanfordkarel လိုက်ဘရိကို အထက် ဖော်ပြပါအတိုင်း အင်စတောလ် လုပ်ဖိုလိုတယ်။

နမူနာ ကားရဲလ် ကဗ္ဗာနှင့် ပရိုကရမ်ကုဒ် ဖိုင်များထည့်ခြင်း

meet_karel.zip ဖိုင်ကို ဒီလင့် <http://tinyurl.com/3mmmm9c7j> ကနေ ဒေါင်းလုဒ်လုပ်ပါ။ ငှါး zip ဖိုင်ကို extract လုပ်ပါ။ meet_karel နံမည်နဲ့ ဖိုဒါတစ်ခု ရလာပါမယ်။ ငှါးဖိုဒါထဲမှ အောက်ပါ worlds ဖိုဒါနှင့် .py ဖိုင်အားလုံးကို ကော်ပီလုပ်ပါ။

- worlds
 - ▶ meet_karel.w
 - ▶ move_beeper_to_other_side.w
- meet_karel.py
- move_beeper_to_other_side.py
- world_editor.py

MeetKarel ပရောဂျက်ထဲတွင် ကူးထည့်ပါ။ ပင်မ ပရောဂျက် MeetKarel (ပုံ က/၂ မှာ မြှားပြထား) ပေါ်မှာ ညာကလစ်နှင့်ပြီး Paste လုပ်ရမှာပါ။ ကော်ပီကူးထည့်လိုက်တဲ့ ဖိုင်တွေက ပုံမှာ တွေ့ရတဲ့ အတိုင်း MeetKarel ဖိုဒါအောက်မှာ ရှိသင့်ပါတယ်။

အခန်းအလိုက် နမူနာ ကုဒ်ဖိုင်တွေ ထည့်ပေးထားတဲ့ .zip ဖိုင်တွေကိုလည်း အထက်ပါအတိုင်း အလားတူ လုပ်ရပါမယ်။ ပရောဂျက်အသစ်ဆောက်၊ .zip ဖိုင်ကို ဖြည့်၊ ရလာတဲ့ ဖိုဒါထဲက ဖိုင်တွေကို ပင်မ ပရောဂျက် ဖိုဒါထဲ ကော်ပီကူးထည့် ရုံပါပဲ။

meet_karel.py ဖိုင်ကို ကလစ်နှစ်ချက်နှင့်ပါ။ ပုံ (က/၂) မှာ အနီစိုင်းထားတဲ့ ကုဒ်အယ်ဒီတာ (code editor) ပွင့်လာပါမယ်။ အဲဒီ မူရင်းဖိုင်မှာ အောက်ပါအတိုင်း ဆက်လက် ပြင်ဆင်ဖြည့်စွက်ပါ။

```
from stanfordkarel import *

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

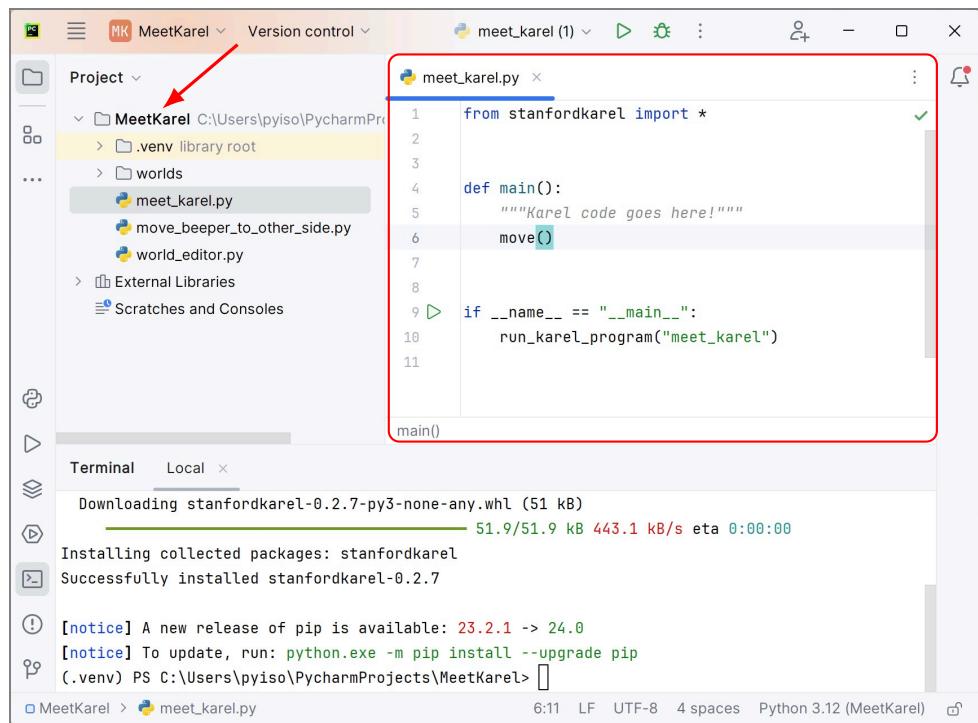
    turn_left()
    turn_left()
    turn_left()

    move()
    put_beeper()

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

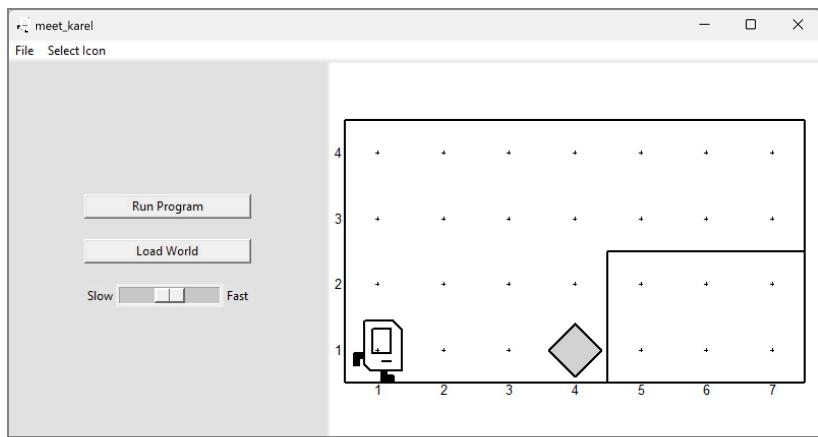
run ထားတဲ့ ပရိုကရမ်တစ်ခုကို တစ်ခါထပ် run ရင် Cancel (သို့) Stop and Rerun လုပ်မှုလား မေးတယ်။ Stop and Rerun လုပ်ရပါတယ်။

meet_karel.py ဖိုင်ကို ညာကလစ်နှုမြို့း Run 'meet_karel' လုပ်ပါ (အယ်ဒီတာ ဧရိယာမှာ ညာကလစ်နှုမြို့း Run 'meet_karel' လုပ်လိုလည်း ရပါတယ်)။ ပုံ (က/၈) က ကားရဲ့ပဲ ပရိုကရမ် တက်လာရင် Run Program ခလုတ်ကိုနှိပ်ပါ။ ဘိပါကို ကားရဲ့ပဲက ခွဲပေးပါလိမ့်မယ်။



ပုံ ၃/၄

၁၉



ပုံ ၃/၈

ဆင်းတက်စ်အမှားများ

အကယ်၍ ပရိုကရမဲ့ run လို့မရရင် ကုသရေးတာမှားနေလို့ ဖြစ်နိုင်တယ်။ မိမိရေးထားတာကို စာမျက်နှာ (၈၂) က ပရိုကရမဲ့ကုပ်နဲ့ နှင့်ယူဉ် စစ်ဆေးကြည့်ပါ။ PyCharm အယ်ဒီတာမှာ အနီလိုင့်တွန်လေးတွေ (ပုံ ၂/၉) ပြတဲ့နေရင် အဲဒီနေရာတွေမှာ ဆင်းတက်စ်မှားနေလို့ (သို့) လိုက်ဘရီမထည့်ရသေးလို့ပဲ။

ဂိုက်ကွင်းကျို့နေတာက အဖြစ်များတဲ့ အမှားပါ။ ကျို့ခဲ့လို့ မရပါဘူး။ အင်ဒန်တေးရှင်း (indentation) လုပ်ရမဲ့နေရာမှာ မလုပ်ထားရင်လည်း ပြဿနာဖြစ်တယ်။ move, turn_left တွေကို ဘေးမျဉ်း ညာဘက်ခွဲပြီး အင်ဒန်လုပ်ပေးရမယ်။ အဲဒီတွေ ဂရုမစိုက်ပိုရင် ဆင်းတက်စ်အမှားဖြစ်ပြီး ပရိုကရမဲ့ run လို့ မရနိုင်ဘူး။

Terminal မှာ ထုတ်ပေးတဲ့ မက်ဆွဲချုပ်တွေကို ကြည့်ပြီးတော့လည်း ဘာပြဿနာဖြစ်နေလဲ မှန်းဆ လို့ရနိုင်တယ်။ ဘာကြောင့်ဖြစ်နိုင်လဲ ဆက်စပ်စဉ်းစားလို့ ရတယ်။ ဥပမာ ဖြစ်တဲ့ပြဿနာအလိုက် အခုလို တွေ့ရပါမယ်။

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 6
```

```
    move(
```

```
    ^
```

```
SyntaxError: '(' was never closed
```

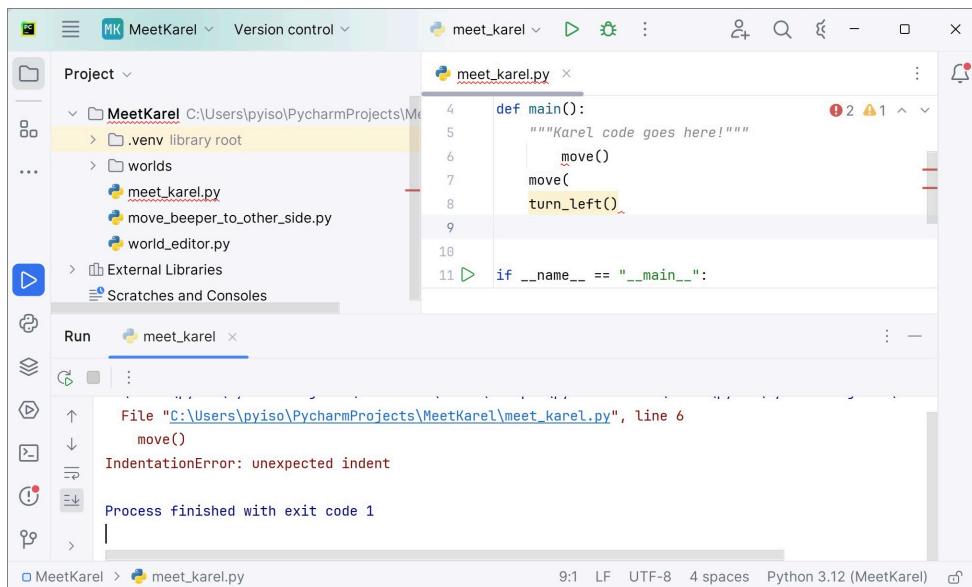
```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 7
```

```
    move()
```

```
IndentationError: unexpected indent
```

Traceback (most recent call last):

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 1,
      in <module>
```



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

if __name__ == "__main__":
    """Karel code goes here!"""
    move()
    move()
    turn_left()

Process finished with exit code 1
```

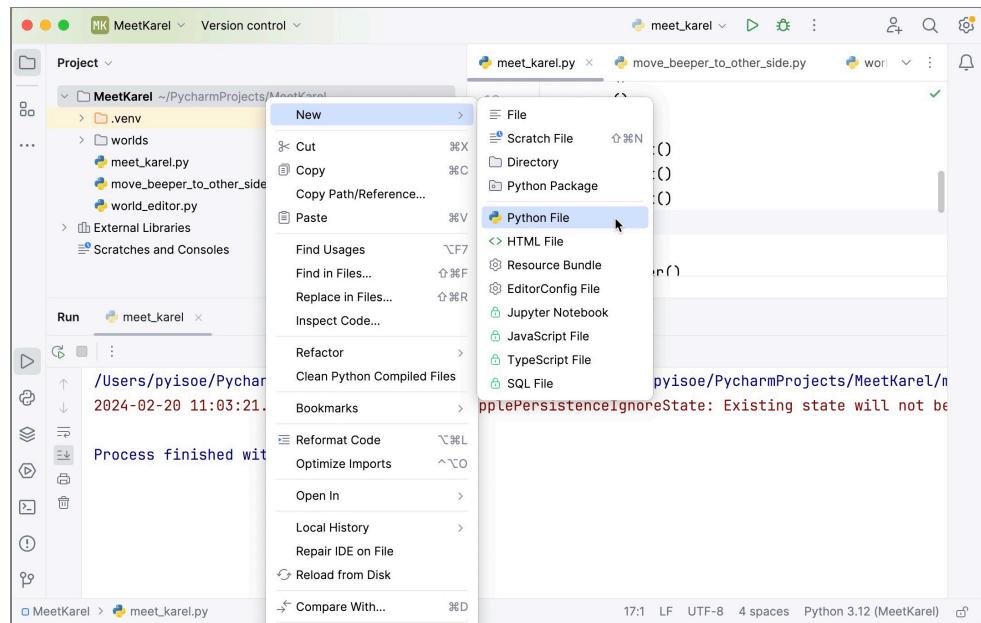
60

```
from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
```

Python ဖိုင် အသစ်ယူခြင်း

MeetKarel ပင်မ ပရောဂျက်ဖို့ဒါပေါ်မှာ ညာကလစ်နှုပ်ပြီး Python ဖိုင် အသစ်ယူနိုင်ပါတယ်။ Python ဖိုင်တွေက .py အိုင်စာန်းရှင်းနှုပါ။ ကားရဲလ်ပရိုဂရမ်တစ်ခုကို Python ဖိုင်တစ်ခု ထားပါမယ်။ ပင်မ ပရောဂျက်ဖို့ဒါ အောက်မှာပဲ တိုက်ရိုက်ရှိရပါမယ်။

နောက်ပိုင်း အဆင့်မြင့်လာရင် ပရိုဂရမ်တစ်ခုအတွက် ပရောဂျက်တစ်ခု ထားနိုင်တယ်။ ကုဒ်ဖိုင်တွေ အပြင် ပရိုဂရမ်အတွက် လိုအပ်တဲ့ ရုပ်ပုံတော့၊ အခြားဖိုင်တွေ (config ဖိုင်၊ setting ဖိုင် စသည်ဖြင့်) လည်း ပါနိုင်တယ်။ ပင်မပရောဂျက် အောက်မှာပဲ ဖိုင်တွေက တိုက်ရိုက်ရှိဖို့လည်း မလိုတော့ဘူး။ ဆက် စပ်ရာ ဖိုင်တွေကို အမျိုးအစားအလိုက် ဖန်ရှင်အလိုက် ဖို့ဒါတွေခဲ့ပြီး စနစ်ကျ စီစဉ်ဖွဲ့စည်း ထားရမှာပါ။ ပရောဂျက်တစ်ခုမှာ ဖိုင်တွေကို စနစ်တကျ စုဖွဲ့ထားဖို့ အရေးကြီးပါတယ်။



ပုံ က/၁၀

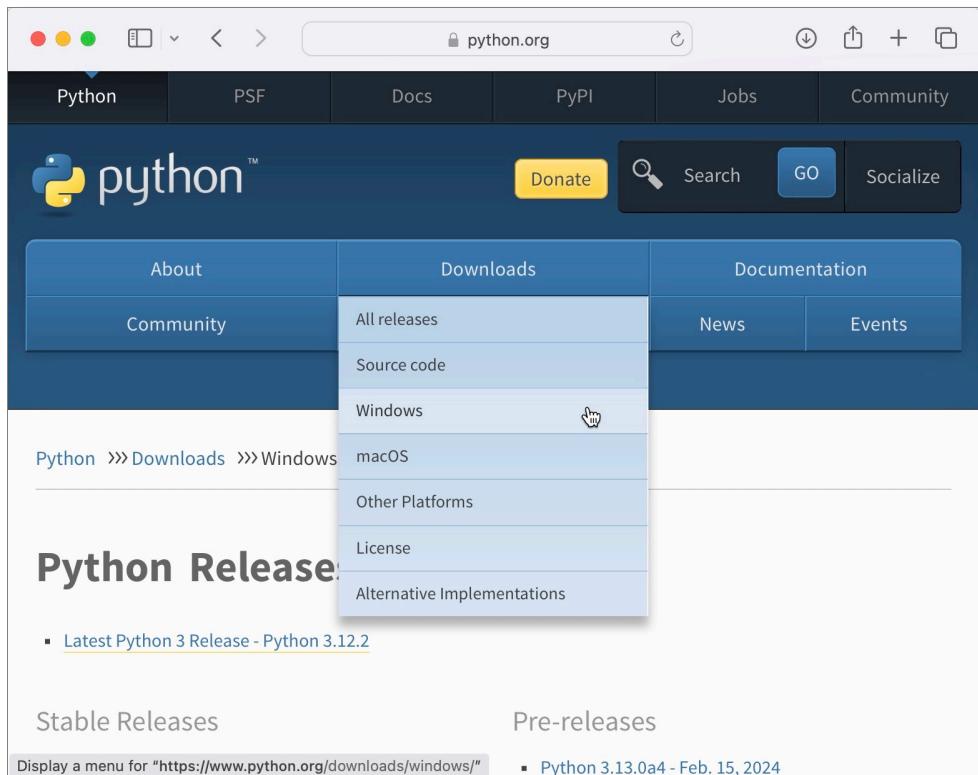
Visual Studio Code နှင့် Python အင်စတောလ်လုပ်ခြင်း

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 အင်စတောလ် လုပ်လို့ရတယ်။ VS Code နဲ့ Python ရေးမယ်ဆိုရင် Python Programming Language ကို သီး၌ ဒေါင်းလုဒ်လုပ်ပြီး အင်စတောလ် လုပ်ရပါမယ်။

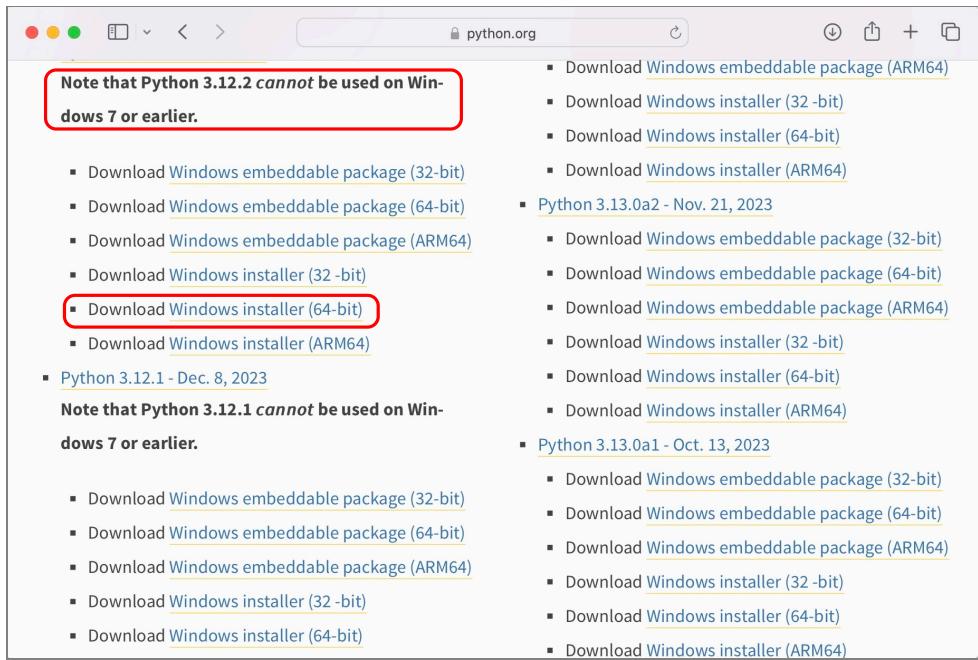
Python အင်စတောလ်လုပ်ခြင်း

ဒီလုပ် <https://www.python.org/> ကိုဖွေ့စ်ပါ။ Download မိန္ဒီးမှ Windows နိုပ်ပါ (ပုံ က/၁၁ ကို ကြည့်ပါ)။ Apple ကွန်ပျိုးတာအတွက် ဆိုရင် macOS ရွှေးရပါမယ်။ လူများစုသုံးတဲ့ ပိုက်ခရီးဆော် ဝင်း ဒီးအတွက် အင်စတောလ်လုပ်နည်းကို အခိုက်ပြေားမှာပါ။ ပုံ က/၁၂ မှာလို့ တွေ့ရပါလိမ့်မယ်။ အင်စတော်လာ ဗားရှင်း 3.12 ထဲက လက်ရှိအမြင့်ဆုံး ကိုရွေးပါ။ ဒီစာရေးချိန်မှာ 3.12.2 ဟာ အမြင့်ဆုံးဗားရှင်းပါ။ Windows Installer (64-bit) ကိုနိုပ်ပြီး ဒေါင်းလုဒ်လုပ်ပါ။



ပုံ က/၀၁

ဒေါင်းလုဒ်ပြီးရင် အင်စတော်လာဖိုင်ကို ညာကလစ်နိုပ်ပြီး Run as administrator လုပ်ပါ။ ပုံ (က/၁၃) ကိုကြည့်ပါ။ ပုံ (က/၁၄) မှာလို့ ဒိုင်ယာလော် ဆောက်စ် ပွင့်လာပါမယ်။ အနိဂုင်းထားတဲ့ ချက်ချေ ဆောက်စ်နှစ်ခုကို ချက်ချေလုပ်ပြီး Install Now နိုပ်ပါ။ အင်စတောလ် ပြီးလို့ Setup was successful



Note that Python 3.12.2 cannot be used on Windows 7 or earlier.

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)**
- Download Windows installer (ARM64)

Python 3.12.1 - Dec. 8, 2023

Note that Python 3.12.1 cannot be used on Windows 7 or earlier.

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)

Python 3.13.0a2 - Nov. 21, 2023

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)
- Download Windows installer (ARM64)

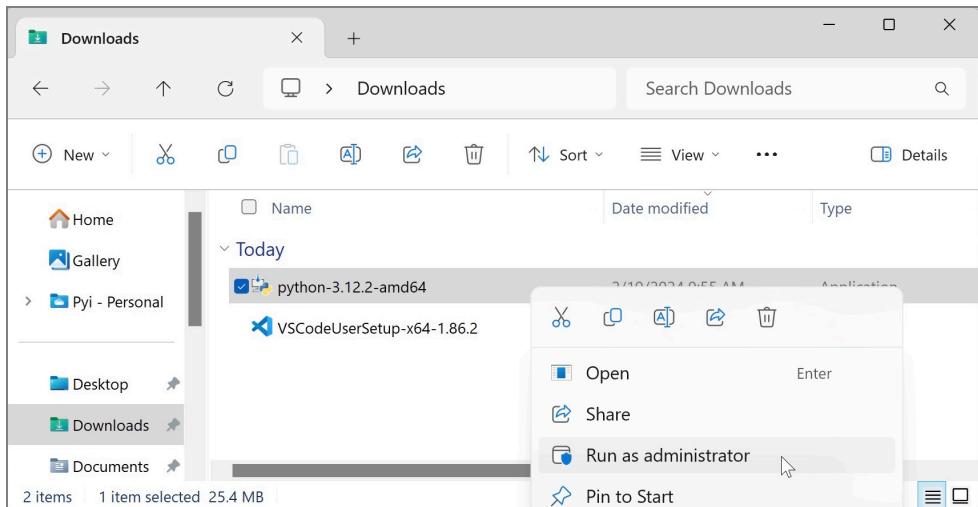
Python 3.13.0a1 - Oct. 13, 2023

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows embeddable package (ARM64)
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)
- Download Windows installer (ARM64)

ဗိ ၂/၁၂

ပေါ်လာရင် Close ခလုတ်နိုပ်ပြီး ပိတ်ပါ။

ဝင်းဒီး command prompt (cmd) မှာ `python --version` run ရင် ဖုံး (၂/၁၃) မှာလို အင် စတော်လုပ်ထားတဲ့ ဗားရှင်းကို ပြပေးသင့်ပါတယ်။



Downloads

Name Date modified Type

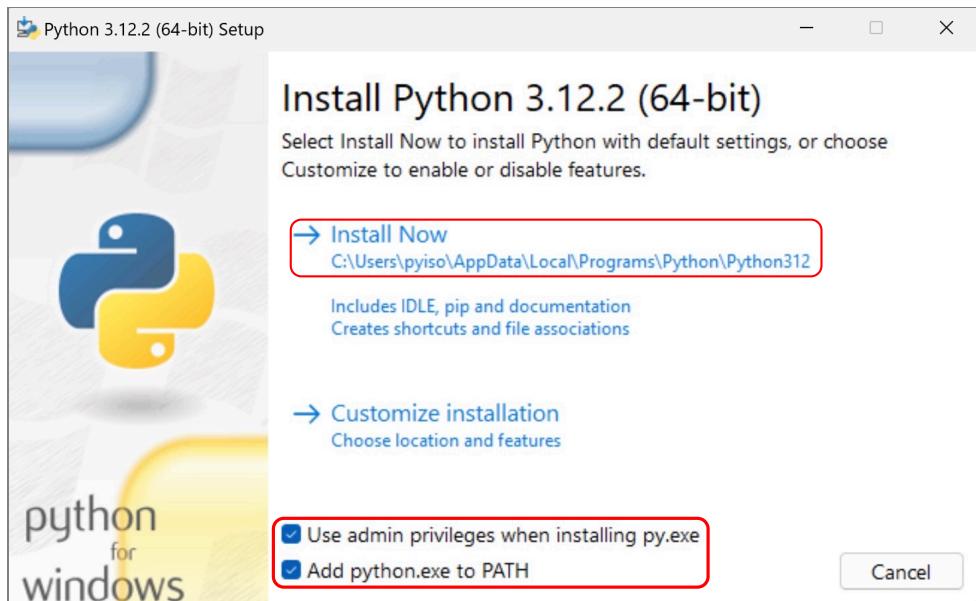
python-3.12.2-amd64

VSCodeUserSetup-x64-1.86.2

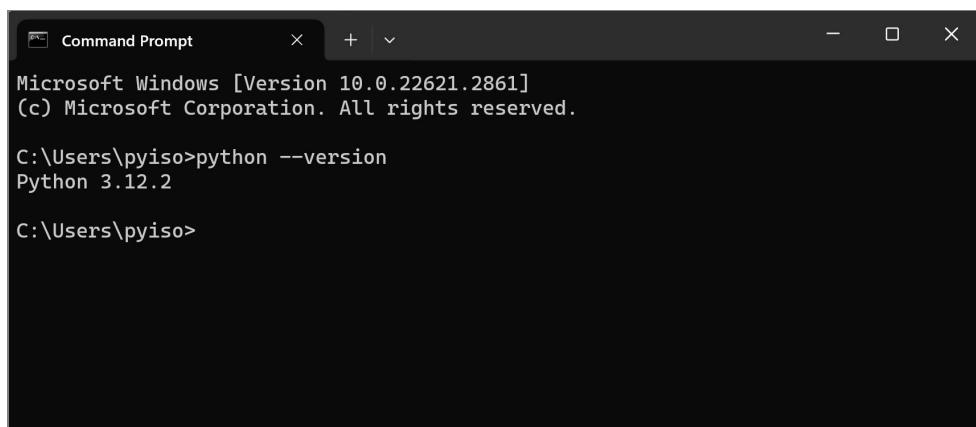
Run as administrator

ဗိ ၃/၁၃

၃၂



ပုံ ၃/၀၄



ပုံ ၃/၀၅

VS Code အင်စတေလုပ်ခြင်း

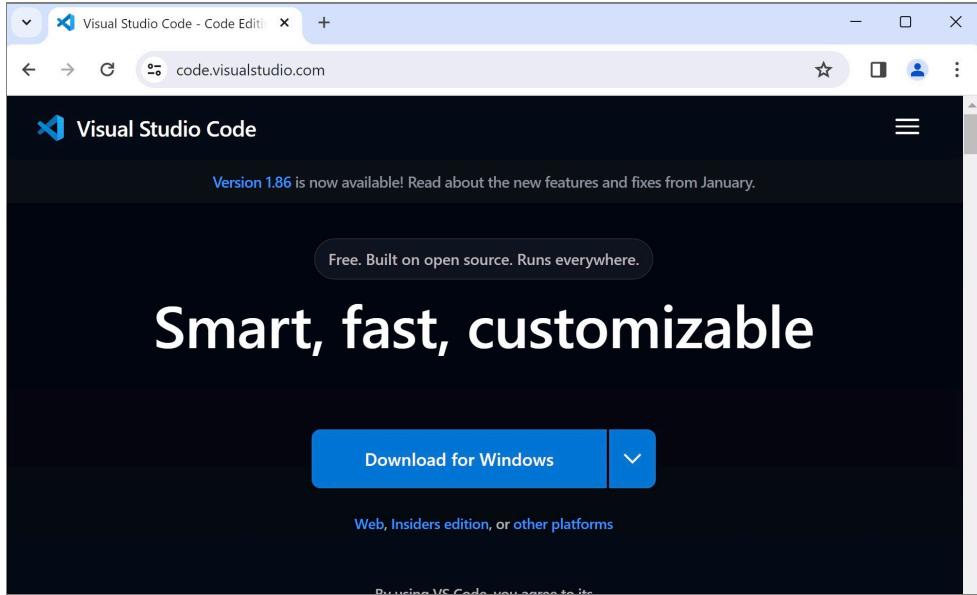
အခုနိရင် Python programming language အောင်မြင်စွာ ထည့်ပြီးသွားပါပြီ။ VS Code အင်စတေလုပ်ခြင်း ဆက်လုပ်ပါမယ်။ အင်စတေလုပ်ခြင်း အောင်မြင်စွာ ထည့်ပြီးသွားပါပြီ။

<https://code.visualstudio.com>

မှတစ်ဆင့် သွားပါ။ ပုံ (က/၁၆) ဝဘ်စာမျက်နှာကို တွေ့ရပါမယ်။ [Download for Windows](#) နှင့် ဒေါင်းလုပ်လုပ်ရန် ဝဘ်စာမျက်နှာကို အောက်ပါလုပ်ပါ။

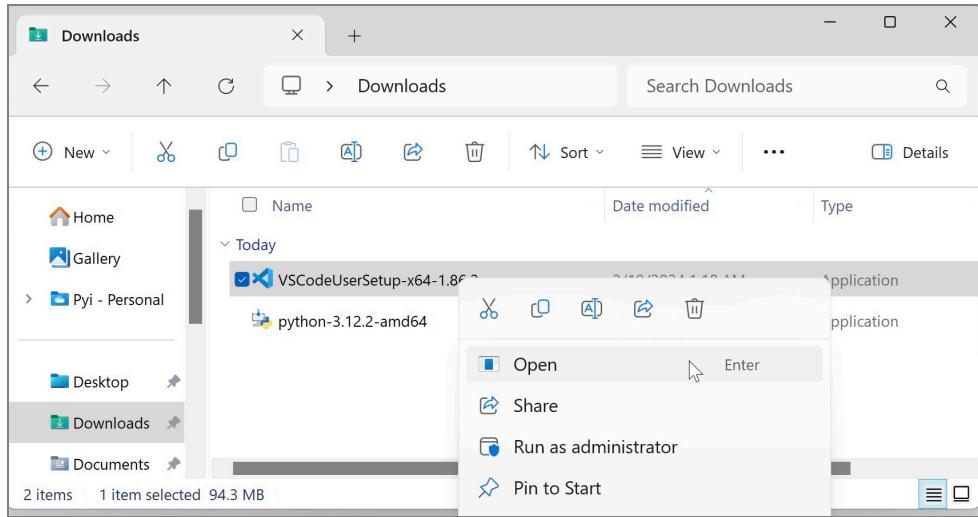
ပြီးတဲ့အခါ အင်စတေလုပ်ခြင်းကို ညာကလစ်နှင့်ပြီး ဖွင့်ပါ (ပုံ က/၁၇ မှာ ပြထားပါတယ်)။ အင်စတေလုပ်ခြင်းကို အောက်ဖော်လေ့ရှိဘေးကို ပွင့်လာရင် [I accept the agreement](#) ကို ချက်ချွဲလုပ်၍ [Next >](#) တစ်ခြားတစ်ခု ဆက်နှိပ်သွားပြီး နောက်ဆုံးမှာ [Install](#) နှင့်ပါ။ အင်စတေလုပ်ခြင်းနောက် ခေါ်စောင့်ပြီး၊ ပြီးသွားရင် [Finish](#) နှင့်ပါ။

Welcome စခရင်ကို ပုံ (က/၁၈) လို တွေ့ရပါမယ်။ [Dark Modern](#) (သို့) [Light Modern](#) နှစ်သက်ရာ သီးမှတ် ရွှေးပါ။ ဒါဆိုရင် VS Code လည်း အင်စတေလုပ်ခြင်းနှင့်ပါ။ ကားရဲ့လုပ်ရှိရမဲ့ ရေးဖို့ ဆက်လုပ်ရပါမယ်။

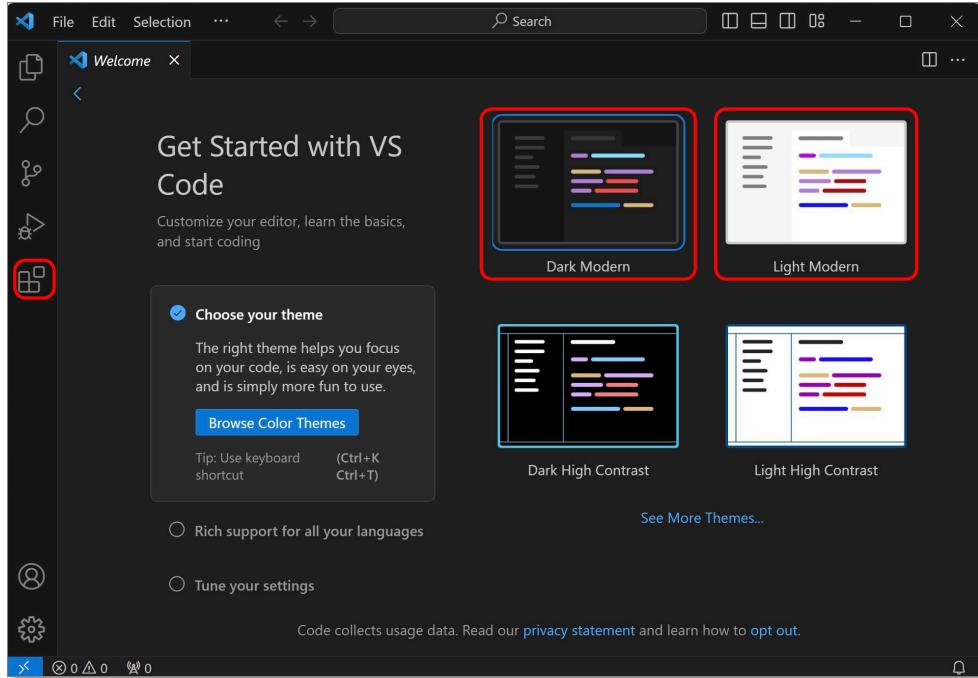


ပုံ က/၁၆

82



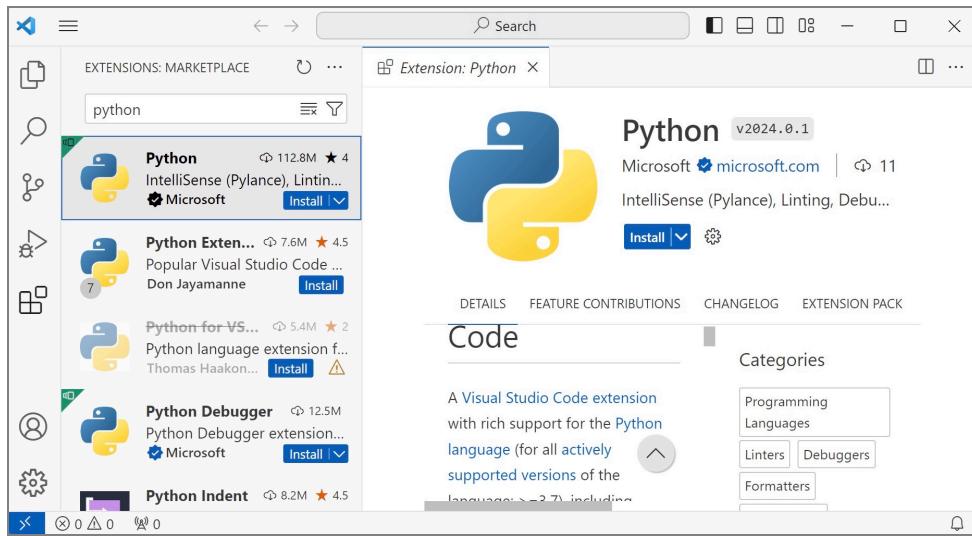
ပုံ ၈/၁၇



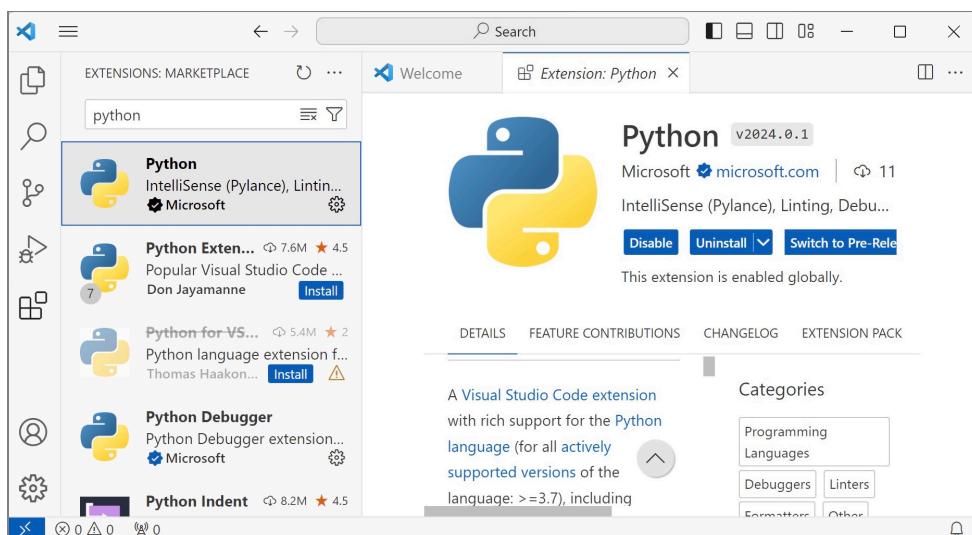
ပုံ ၈/၁၇

VS Code Python Extension တည်ခြင်း

Python ပရိဂုဂ္ဂရေးဖို့အတွက် VS Code က ဒီအတိုင်းဆိုရင် သိပ်အဆင်မပြေသေးပါဘူး။ Python အတွက် extension အင်စတော်လုပ်ပေးရပါအိုးမယ်။ ပုံ (က/၁၈) ဘယ်ဘက်သော်နားမှာ အနီးပိုင်းထားတဲ့ အိုင်ကွန်လေးကို နှိပ်ပါ။ Python extension ကိုရှာပါ။ ပုံ (က/၁၉) မှာ ဘယ်လိုရှာရမလဲ ပြထားပါတယ်။ ပုံမှာတွေ့ရတဲ့ Microsoft က ထုတ်တဲ့ extension ကို အင်စတော်လုပ်ပါ။ အောင်မြင်ရင် ပုံ (က/၂၀) မှာလို ဖြစ်သွားပါမယ်။ Python နဲ့ VS Code ကိစ္စတော့ ပြီးသွားပြီ။ ကားရဲ့လဲ example run ဖို့ ဆက်လုပ်ရမယ်။



ပုံ က/၁၉



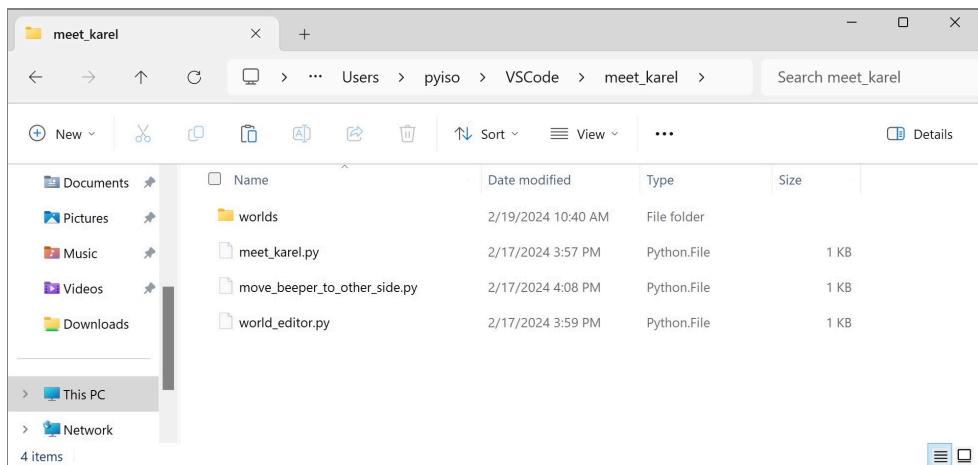
ပုံ က/၂၀

နမူနာ ကားရဲလ် ကဗ္ဗာနှင့် ပရိုဂရမ်ကို ပိုင်များထည့်ခြင်း

meet_karel.zip ဖိုင်ကို ဒီလင့် <http://tinyurl.com/3mm9c7j> ကနေ ဒေါင်းလုပ်လုပ်ပါ။ ငါး zip ဖိုင်ကို extract လုပ်ပါ။ meet_karel နံမည်နဲ့ 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/၁၁

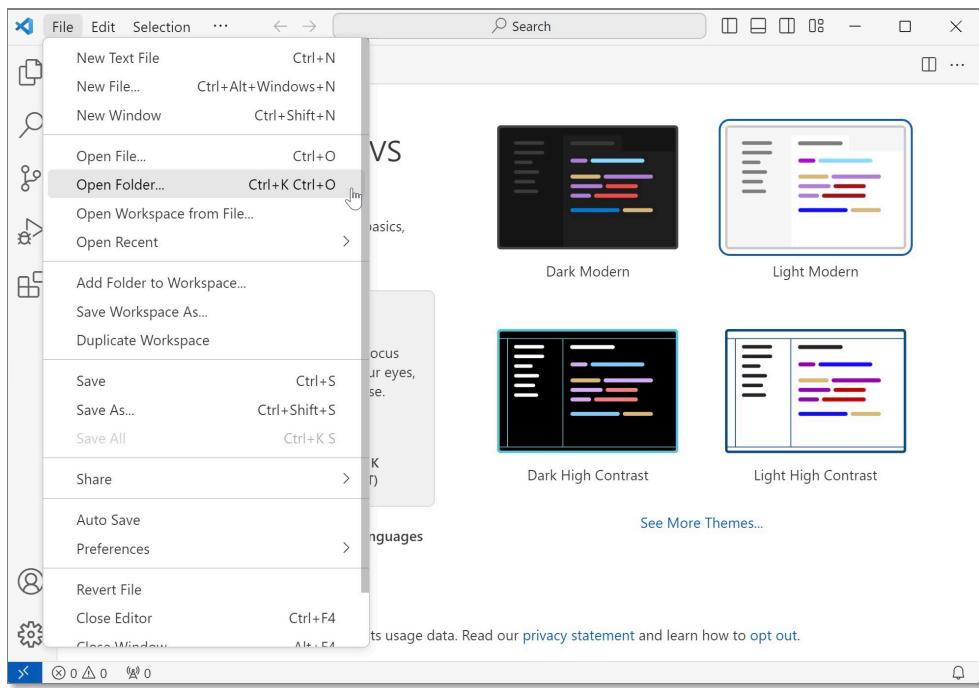
VS Code အတွက် ဖိုဒါတစ်ခုကို မိမိအတွက် အဆင်ပြေမဲ့နေရာမှာ သီးသန့်တည်ဆောက် ထားသင့်တယ်။ ဥပမာ

C:\Users\yourname\VS Code

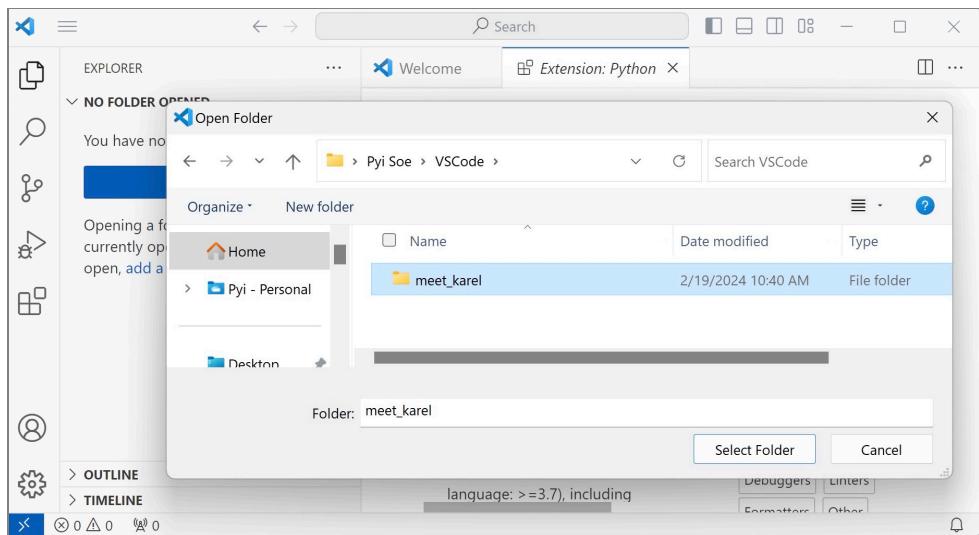
မိမိ လက်ရှိ Home ဖိုဒါကို C: drive ရဲ့ Users ဖိုဒါထဲမှာ တွေ့နိုင်ပါတယ်။ Win + R ကြောက်နှင့်ပြီး %userprofile% ရိုက်ထည့်၍ Ok လုပ်ပြီး Home ဖိုဒါကို သွားနိုင်ပါတယ်။ အကယ်၍ မသွားတတ်ရင်လည်း ပြဿနာမရှိပါဘူး။ ကိုယ့်အတွက် လွယ်ကူမဲ့ Desktop, Downloads, Documents တစ်ခုခုထဲမှာ VS Code အတွက် ဖိုဒါတစ်ခု ထားလည်းရတယ်။

meet_karel ဖိုဒါ (.zip ဖိုင်မဟုတ်ပါ) ကို အထက်ပါအတိုင်း အသစ် ဆောက်ထားတဲ့ VS Code သီးသန့်ဖိုဒါတဲ့ကို ကော်ပိကူးထည့်ပါ။ ငါး meet_karel ဖိုဒါကို VS Code File ပြန်မှု Open Folder နှင့်ပြု ဖွင့်ပါ။ ပုံ (က/၂၂) တွင်ကြည့်ပါ။

အခန်းအလိုက် နမူနာ ကုဒ်ဖိုင်တွေ ထည့်ပေးထားတဲ့ .zip ဖိုင်တွေကိုလည်း အထက်ပါအတိုင်း လုပ်ရပါမယ်။ .zip ဖိုင်ကို ဖြည့်၊ ရလာတဲ့ ဖိုဒါကို သီးသန့်ဖိုဒါတစ်ခုထဲကို ကော်ပိကူးထည့်၊ VS Code နဲ့ အဲဒီဖိုဒါကို ဖွင့်ရုံပါပဲ။



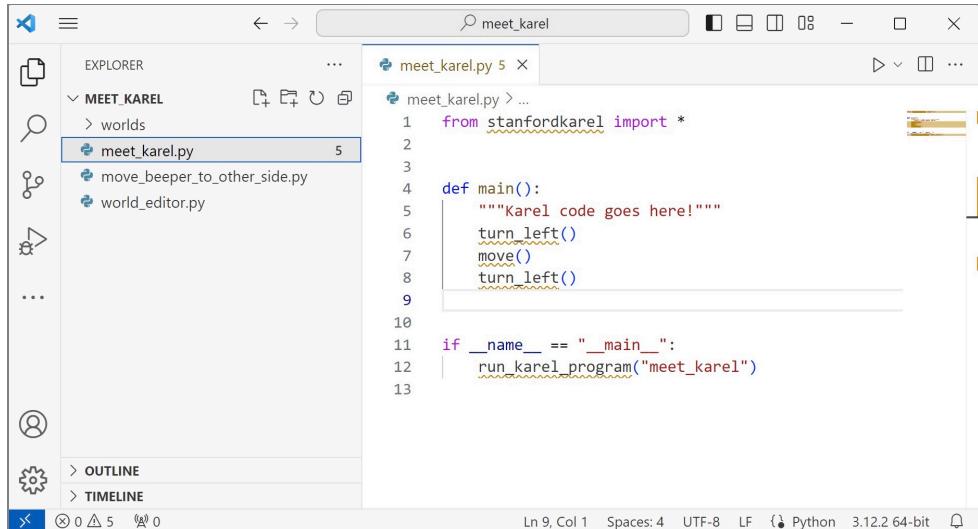
ဗုဒ္ဓန/၂၂



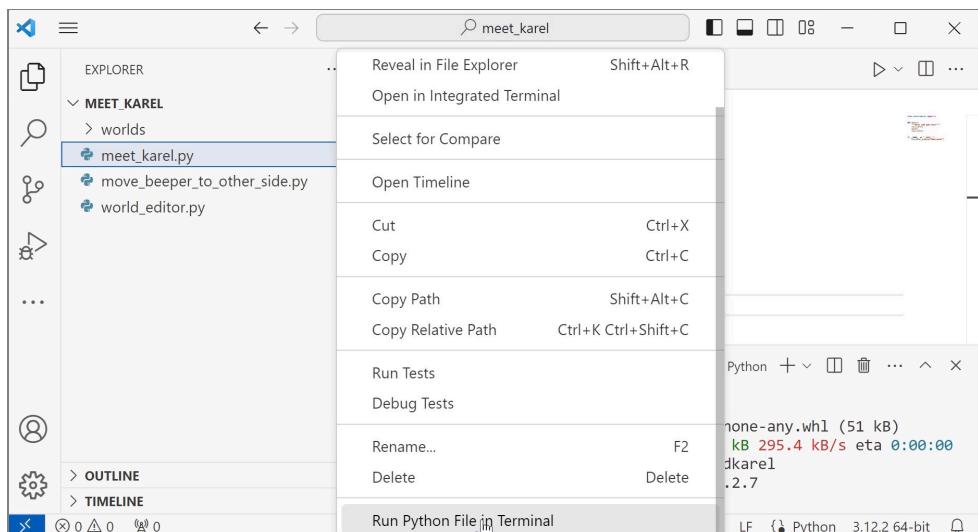
ဗုဒ္ဓန/၂၃

stanfordkarel လိုက်ဘရီ အင်စတောလ်လုပ်ခြင်း

`meet_karel.py` ဖိုင်ကို ကလစ်နှစ်ချက်နှင့် ဖွင့်ပါ။ ကုဒ်အယ်ဒီတာ ပွင့်လာမယ် (ပုံ ၂/၂။)။ အဲဒီ ကုဒ် အယ်ဒီတာပေါ် (သို့) `meet_karel.py` ဖိုင်ကို ညာကလစ်နှင့်ပြီး `Run Python File in Terminal` လုပ်ပါ (ပုံ ၂/၂။)။ Terminal ပွင့်လာပြီး အယ်ရာမက်ဆောင်တွေ ပြလိမ့်မယ်။ ပုံ (၂/၂) မှာ ကြည့်ပါ။ ကားရုံးပရှုံးရပ်အတွက် လိုအပ်တဲ့ stanfordkarel လိုက်ဘရီ အင်စတောလ် မလုပ်ရသေးပါဘူး။ ဒါ ကြောင့် အယ်ရာဖြစ်နေတာ။



ပုံ ၂/၂



ပုံ ၂/၂

ခုနကပွင့်လာတဲ့ Terminal မှာပဲ အောက်ပါကွန်မန်းကို run ပြီး stanfordkarel လိုက်ဘရီကို အင်စတောလ်လုပ်ပါ။

```

EXPLORER              meet_karel.py 5
MEET_KAREL
  worlds
  meet_karel.py
  move_beeper_to_other_side.py
  world_editor.py

PROBLEMS 5   OUTPUT  TERMINAL  ...
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

```

ပုံ ၃/၂၆

`pip install stanfordkarel`

ပုံ (၃/၂၆) မှာ အနိဂုင်းထားတာကို ကြည့်ပါ။ အဲဒီအတိုင်းရှိက်ထည့်ပြီး Enter ကိုနှိပ်ပါ။ ခေါက်တဲ့ အခါ အခုလို မက်ဆောင်တွေ ကျလာပါလိမ့်မယ်။

`ModuleNotFoundError: No module named 'stanfordkarel'`

`PS C:\Users\pyiso\VSCode\meet_karel> pip install stanfordkarel`

`Collecting stanfordkarel`

`Downloading stanfordkarel-0.2.7-py3-none-any.whl (51 kB)`

`51.9/51.9 kB 295.4 kB/s eta 0:00:00`

`Installing collected packages: stanfordkarel`

`Successfully installed stanfordkarel-0.2.7`

`PS C:\Users\pyiso\VSCode\meet_karel>`

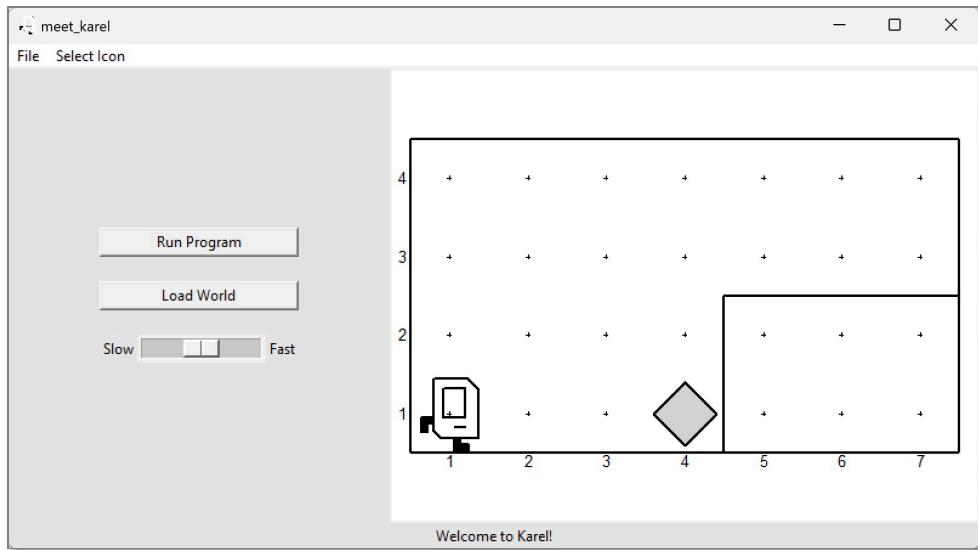
ဟိုက်လိုက်ပြထားတဲ့ မက်ဆောင်တွေရှုရင် အင်စတောလ အောင်မြင်လိုပါ။ ပုံ (၃/၂၆) အယ်ဒီတာမှာလို သတိပေး လိုင်းတွေနဲ့မျှပွဲတွေ မရှိသင့်တော့ဘူး။ stanfordkarel လိုက်ဘရှိ အင်စတောလ လုပ်ပြီးပြီ့မိုလို သတိပေးတာတွေ ပျောက်သွားသင့်တယ်။

`meet_karel.py` ဖိုင်ကို ညာကလစ်နှုပ်ပြီး `Run Python File in Terminal` ပြန်လုပ်ကြည့်ပါ။ ပုံ (၃/၂၇) မှာပြထားတဲ့ ကားရဲ့ပရိုကရမ် ပွင့်လာသင့်ပါတယ်။ `Run Program` နှိပ်ကြည့်ပါ။ စက်ရှုပ်လေး ကားရဲ့လုပ် နေရာရွှေသွားတာ တွေ့ရမယ်။ `meet_karel.py` ကို အောက်ပါအတိုင်း ဖြည့်စွက်ရေးပါ။

```
from stanfordkarel import *
```

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

    move()
    move()
```



ဗို ၂/၂၇

```

move()
pick_beeper()
turn_left()
move()
move()

turn_left()
turn_left()
turn_left()

move()
put_beeper()

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

```

`meet_karel.py` ဖိုင်ကို ညာကလစ်နိုပ်ပြီး Run Python File in Terminal ပြန်လုပ်ကြည့်ပါ။ ကား ရဲလ်ပရိုကရမ် ပွင့်လာရင် `Run Program` နှိပ်ကြည့်ပါ။ ဘိပါလေးကို နေရာခွဲ့ပေးပါလိမ့်မယ်။ အကယ်၍ ကားရဲလ်ပရိုကရမ် မတက်လာရင် ကုဒ်ရေးတာမှားနေလို့ ဖြစ်နိုင်တယ်။ အပေါ်ကုဒ်နဲ့ ဦးယျာဉ်ပြီး ကြည့်ပါ။ VS Code အယ်ဒီတာမှာ အနိလိုင်တွေ့လေးတွေ ပြတဲ့နေရင် အဲဒီနေရာတွေမှာ ဆင်းတက်မှားနေတာ ဖြစ်နိုင်တယ်။

VS Code အယ်ဒီတာမှာ ပရိုကရမ်ကုဒ်ပြင်ပြီး ပြန် run တဲ့အခါ ပထမ run ထားတဲ့ ပရိုကရမ်ကို အရင်ပိတ်ဖို့လိုပါတယ်။ ဆိုလိုတာက meet_karel.py ကို run ထားတယ်ဆိုပါစူး။ ပုံ (က/ဂျ) က ဝင်း ဒီးပင့်လာမယ်။ 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 ဘေးမျဉ်းညာဘက်ဆွဲပြီး အင်ဒန်လုပ်ပေးရမယ်။ အဲဒီတွေ ကရာဇ်စိုက်မိရင် ဆင်းတက်စ်အမှားဖြစ်ပြီး ပရိုကရမ် run လို မရနိုင်ဘူး။

Terminal မှာ ထုတ်ပေးတဲ့ မက်ဆွဲချုပ်တွေကို ကြည့်ပြီးတော့လည်း ဘာပြဿနာဖြစ်နေလဲ မှန်းဆလိုရနိုင်တယ်။ ဘာကြောင့်ဖြစ်နိုင်လဲ ဆက်စပ်စဉ်းတားလို ရတယ်။ ဥပမာ ဖြစ်တဲ့ပြဿနာအလိုက် အခုလို တွေ့ရပါမယ်။

```
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 6
    move(
        ^
SyntaxError: '(' was never closed

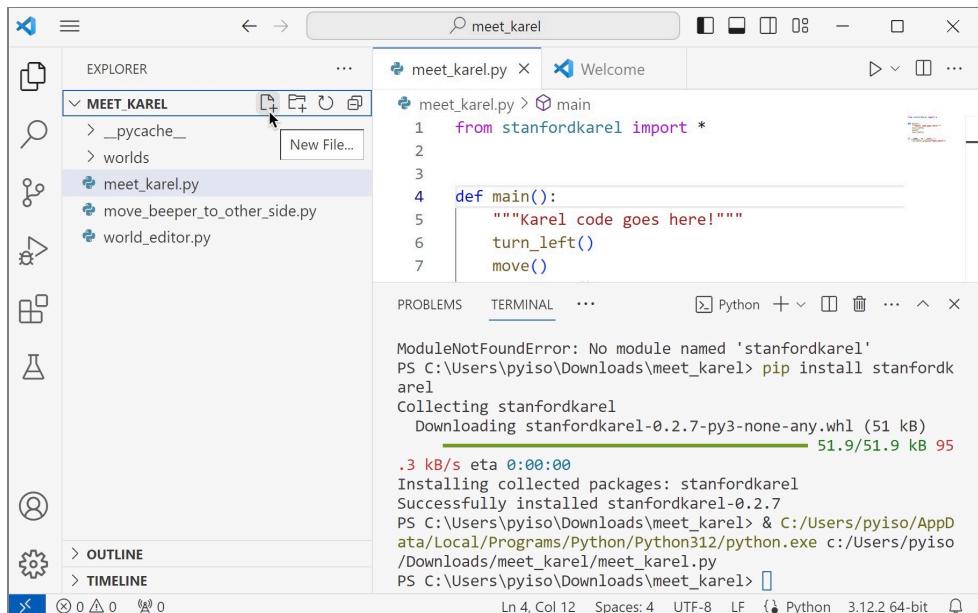
File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 7
    move()
        ^
IndentationError: unexpected indent

Traceback (most recent call last):
  File "c:\Users\pyiso\VS Code\meet_karel\meet_karel.py", line 1,
    in <module>
      from stanfordkarel import *
ModuleNotFoundError: No module named 'stanfordkarel'
```

VS Code တွင် Python ဖိုင် အသစ်ယူခြင်း

MEET_KAREL ပင်မ ပရောဂျက်ဖိုဒ်အပေါ်မှာ ကလစ်နိုင်ပါ။ ပုံမှာ ပြထားတဲ့ **New File** အိုင်ကိုနိုင်ပါ။ ဖိုင်နံပည်ဖြည့်တဲ့ ဘောက်စံလေး ပေါ်လာမယ်။ Python ဖိုင်တွေက .py အိပ်စံတန်းရှင်းနဲ့ ဖြစ်ရပါမယ်။ ဒါကြောင့် နံပည် ဖြည့်တဲ့အခါ .py နဲ့ အဆုံးသတ်ပေးရပါမယ် (ဥပမာ hello.py)။ ကားချုလ်ပရိုဂရမ်တစ်ခုကို Python ဖိုင်တစ်ခု ထားပါမယ်။ ပင်မ ပရောဂျက်ဖိုဒ်အပေါ်မှာပဲ တိုက်ရှိကြရပါမယ်။

နောက်ပိုင်း အဆင့်မြင့်လာရင် ပရိုဂရမ်တစ်ခုအတွက် ပရောဂျက်တစ်ခု ထားနိုင်တယ်။ ကုဒ်ဖိုင်တွေ အပြင် ပရိုဂရမ်အတွက် လိုအပ်တဲ့ ရုပ်ပုံတွေ၊ အခြားဖိုင်တွေ (config ဖိုင်၊ setting ဖိုင် စသည်ဖြင့်) လည်း ပါနိုင်တယ်။ ပင်မပရောဂျက် အောက်မှာပဲ ဖိုင်တွေက တိုက်ရှိကြရဖို့လည်း မလိုတော့ဘူး။ ဆက်စပ်ရာ ဖိုင်တွေကို အမျိုးအစားအလိုက် ဖန်ရှင်အလိုက် ဖို့ဒါတွေခဲ့ပြီး စနစ်ကျ စီစဉ်ဖွဲ့စည်း ထားရမှာပါ။ ပရောဂျက်တစ်ခုမှာ ဖိုင်တွေကို စနစ်တကျ စွဲထားဖို့ အရေးကြီးပါတယ်။



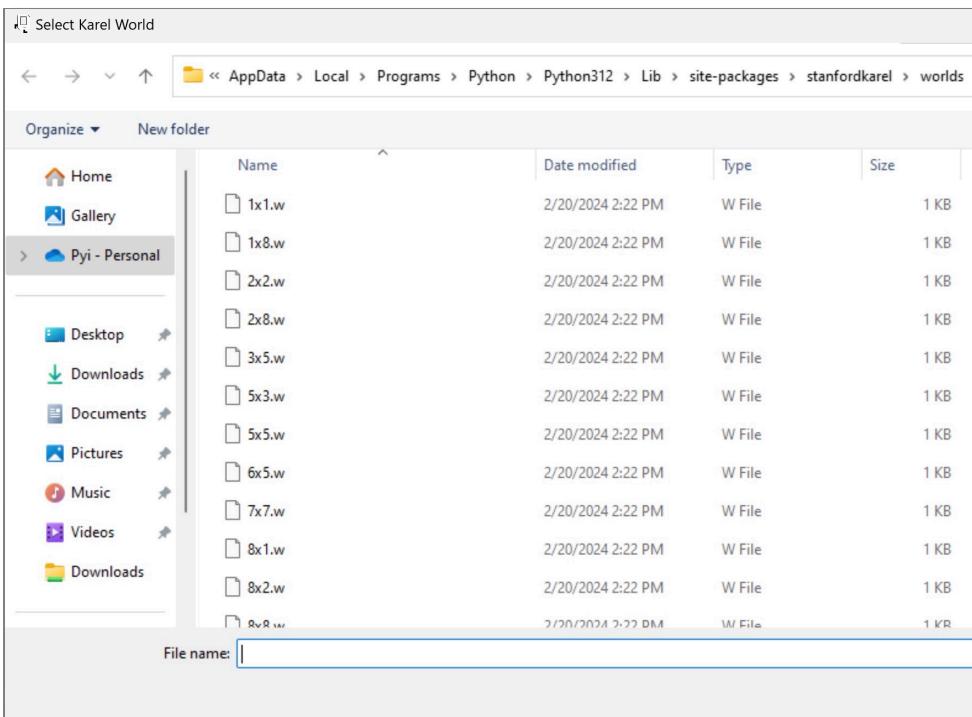
ပုံ က/ဂ

နောက်ဆက်တွဲ ခ

ကားရဲလ်ပရီဂရမ် ဖီချာများ

ကားရဲလ် ကဗျာဖိုင်များ

ကားရဲလ်ပရီဂရမ် ဝင်းခွဲးမှာ **Load World** ခလုတ်နှုပ်ပြီး ကဗျာဖိုင်အသစ် တင်လိုရတယ်။ ခလုတ်နှုပ်လိုက် ရင် အခုလို ဖိုင် ခိုင်ယာလော့ဂုံးပုံင့်လာမယ်။



ပုံ ၁/၁

ဒါက stanfordkarel လိုက်ဘရဲ သူနိုင်အရိုအတိုင်း ပါတဲ့ worlds ဖို့ပါ။ ဖိုင်တွေက **.w** နဲ့ ဆုံးပါတယ်။ ကဗျာဖိုင်တွေကို ပင်မ ပရောဂျက်အောက် worlds ဖို့ပါထဲမှာ ထားလိုလည်းရတယ်။ အခြားနေရာတွေမှာ ထားလိုတော့ မရဘူး။

စာအုပ်ပါ ဥပမာတွေ၊ လောကျင့်ခန်းတွေအတွက် ကမ္မာဖိုင်တွေကို ပရောဂျက် တစ်ခုချင်းအလိုက် သီးခြား worlds ဖို့ဒါထဲမှာ ထည့်ပေးထားမှာပါ။ ပရိုဂ်ရမ်တစ်ခုဟာ ကမ္မာတစ်ခုတည်းမှာပဲ အလုပ်လုပ်တာမဟုတ်ဘဲ အလားတူ ကမ္မာအမျိုးမျိုးအတွက် အလုပ်လုပ်အောင် ရေးပေးရတာ။ အခုပြောတာကို နား မလည်ရင် အခန်း (၂) ဖတ်ပြီးရင် နားလည်သွားမှာပါ။

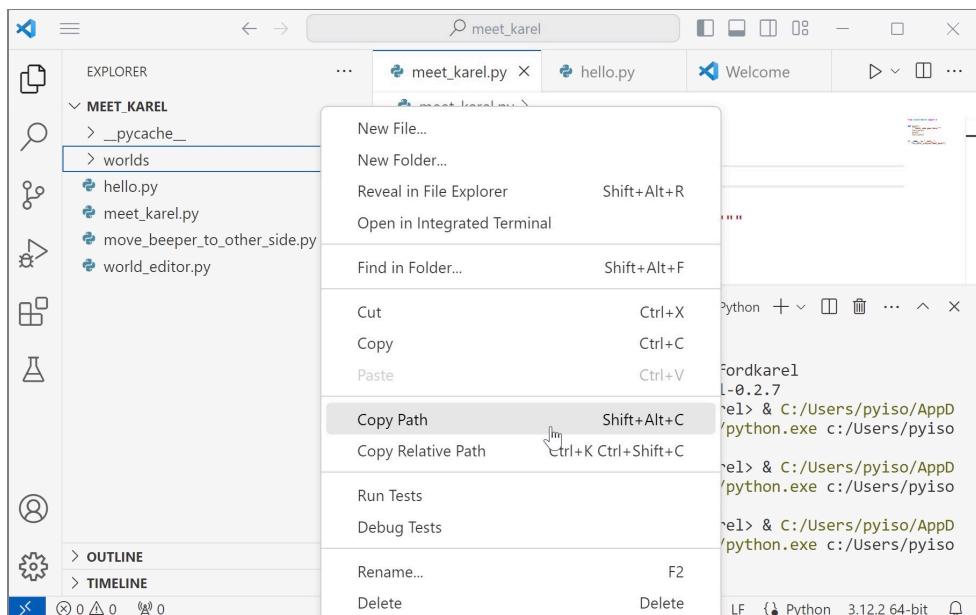
Load World လုပ်တဲ့အခါ ပွင့်လာတဲ့ ဖိုင် ခိုင်ယာလော်ကာ ကိုယ်လိုချင်တဲ့ worlds ဖို့ဒါ မဖြစ် နေဘူး။ သူနိုင်ပါတဲ့ worlds ဖို့ဒါ ဖြစ်နေတယ်။ ကိုယ်ခေါ်တင်ချင်တဲ့ ဖိုင်တွေရှိတဲ့ လက်ရှိပရောဂျက်ရဲ့ worlds ဖို့ဒါကို သွားရမယ်။ ဥပမာ PyCharm/VS Code အတွက် MeetKarel/meet_karel ပရောဂျက် worlds ဖို့ဒါ လမ်းကြောင်း အပြည့်အစုံက

C:\Users\yourname\VSCode\meet_karel\worlds

C:\Users\yourname\PycharmProjects\MeetKarel\worlds

ဖြစ်ပါမယ်။ ဖိုင်ခိုင်ယာလော်ကနေ ဖော်ပြပါ လက်ရှိပရောဂျက် worlds ဖို့ဒါကို တစ်ဆင့်ချင်း သွားပြီး တင်ချင်တဲ့ ကမ္မာဖိုင် (.w ဖိုင်) ကို ရွေးရမှာပါ။

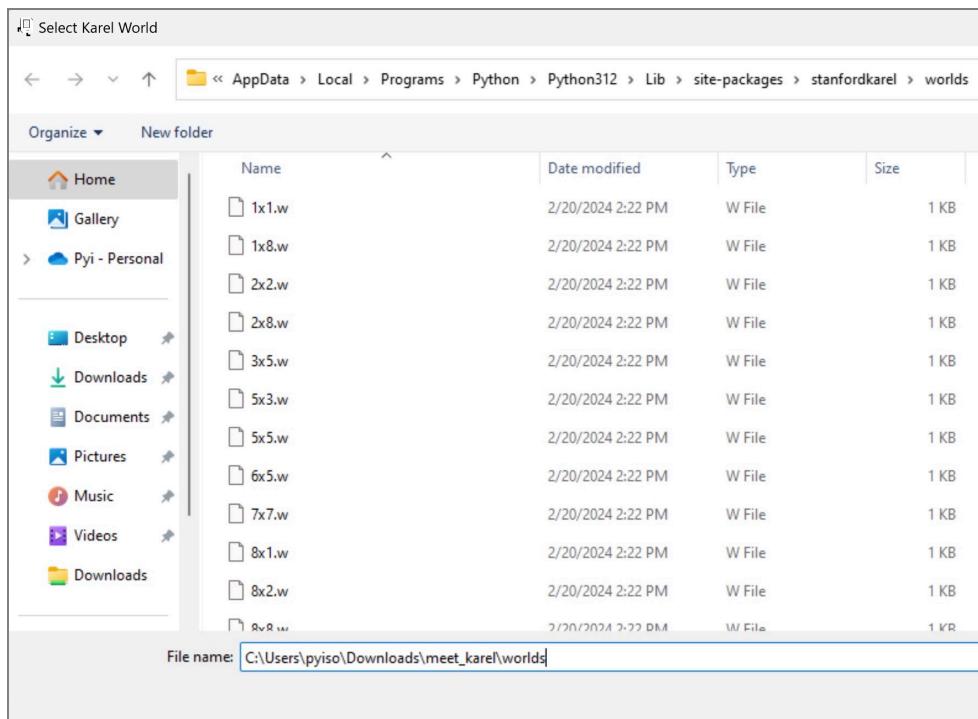
အပေါ်ကနည်းနဲ့ အဆင်မပြော့ရင် အခုလိုစမ်းကြည့်ပါ။ လက်ရှိပရောဂျက် worlds ဖို့ဒါကို ညာ ကလစ်နိုပ်ပြီး Copy Path လုပ်ပါ (ပုံ ခ/၂)။ ဖိုင် ခိုင်ယာလော် **File name** မှာ ကူးထည့်ပါ (ပုံ ခ/၃)။ **Enter** ကိုနိုပ်ပါ။ ပရောဂျက် worlds ဖို့ဒါကိုရောက်သွားပါမယ်။ လိုတဲ့ကမ္မာဖိုင် ရွေးတင်ရုံပါပဲ။ ပုံ (ခ/၄) မှာ meet_karel worlds ကို နှုနာပြထားပါတယ်။



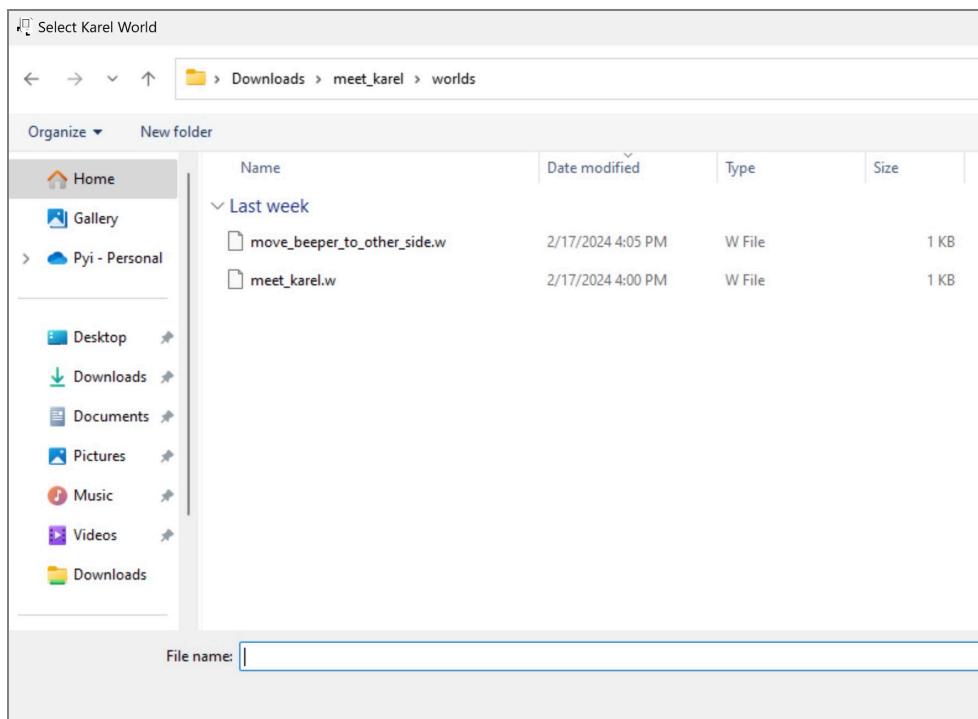
ပုံ ခ/၂

အကယ်၍ အထက်ဖော်ပြပါနည်းတွေက ရှုပ်နေတယ်ထင်ရင် မှတ်ရ/သွားရ လွယ်တဲ့ Desktop, Downloads, Documents လို နေရာတစ်ခွဲမှာ သီးသန်ဖို့ဒါတစ်ခု ဆောက်ပြီး ပရောဂျက်အားလုံး ထည့်ထားတာ အရှင်းဆုံးပါပဲ။ ပရောဂျက်ဖို့ဒါနေရာ သိရင် ဖိုင်ခိုင်ယာလော်ကနေ ဘယ်လိုမဆို ရောက်အောင် သွားလို့ရပါတယ်။

၁၀၆



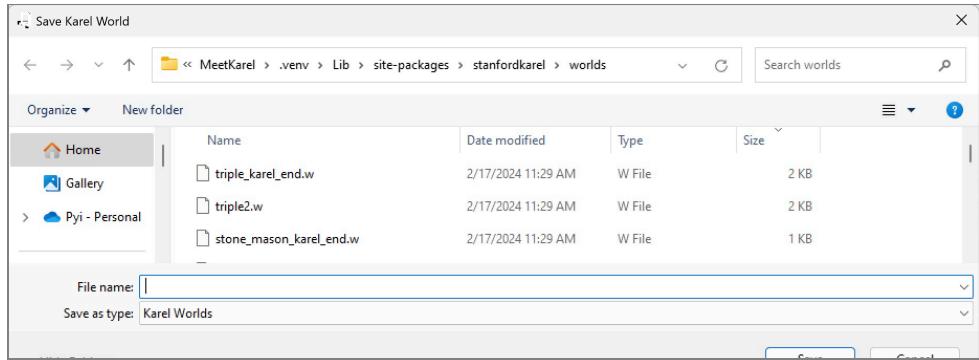
ပုံ ၅/၃



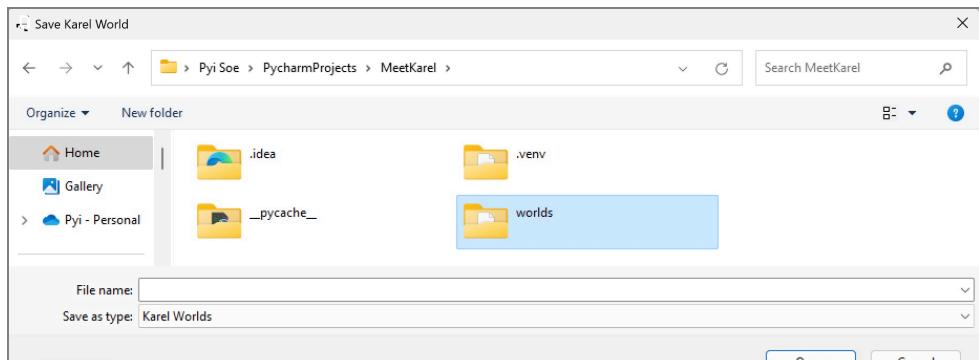
ပုံ ၅/၄

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 ရွှေးချင်ပါလိမ့်မယ်။ No ကို နှိပ်ပါ။ ကမ္ဘာအရွယ်အစားအတွက် ကော်လံ ဘယ်နှစ်ခုလဲ row ဘယ်နှစ်ခုလဲ ထည့်ပေးပါ။ ကိုယ်ပိုင် ကားရဲလ်ကမ္ဘာ တည်ဆောက်လိုရတဲ့ ဝင်းဒီးပွင့်လာပါလိမ့်မယ်။ ကားရဲလ် မျက်နှာမူရာအရပ်၊ 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 တစ်စုံ ‘()’ တော့ပါရမယ်။

colon ‘:’

ဖန်ရှင်နဲ့ သက်ဆိုင်တဲ့ ကုဒ်ဘလောက် (*code block*)

quote သုံးခုတွဲ ‘“”’

ပရိုဂရမ်ဝင်းဒီးမှာ **Run Program** ခလုတ်

meet_karel.w ကမ္ာကို