Begin Modern Programming with Python

Pyi Soe

အခန်း ၁

ဒေတာများနှင့် ဖန်ရှင်များ

ရှေ့ပိုင်း ကားရဲလ်အခန်း လေးခုမှာ လေ့လာခဲ့ကြတဲ့ အကြောင်းအရာတွေဟာ ပရိုဂရမ်မင်း ဘာသာရပ်ရဲ့ အခြေခံအကျဆုံး ပင်မထောက်တိုင် သဘောတရားတွေလို့ ဆိုရမှာပါ။ ဒီသဘောတရားတွေ မကျေညက်ဘဲ ပရိုဂရမ်ရေးလို့ မရပါဘူး။ ကွန်ဒီရှင်နယ်တွေဖြစ်တဲ့ `if`, `if...else`၊ ပြန်ကျော့ခြင်းအတွက် `for` နဲ့ `while` loop၊ ဖန်ရှင်တွေ၊ top-down, bottom-up ပရိုဂရမ်းမင်း၊ ရီကားရှင်းနဲ့ ရီကားဆစ်ဖ် စဉ်းစားခြင်း စတာတွေနဲ့ ပရိုဂရမ်းမင်း ပုစ္ဆာတွေ ဖြေရှင်းနိုင်ရင် ပရိုဂရမ်မာလောကား ပထမ တစ်ထစ် တက်လှမ်းအောင်မြင်ပြီ ပြောနိုင်ပါတယ်။ ဒီသဘောတရားတွေကို ဘီဂင်နာတွေ အရိုးရှင်းဆုံးနည်းနဲ့ နားလည်အောင်၊ လေ့ကျင့်လို့ရအောင် ကားရဲလ်က ထောက်ကူပေးတာပါ။ စက်ရုပ်လေး ကားရဲလ်ကို နှုတ်ဆက်ခဲ့ပြီး အခုဆက်လက်လေ့လာကြမှာကတော့ ဒေတာ၊ အိပ်စ်ပရက်ရှင်းနဲ့ ဗေရီရေဘဲလ်တွေ အကြောင်းပါ။

ကွန်ပျူတာတွေဟာ အချက်အလက်(ဒေတာ) အမျိုးမျိုးကို ကိုင်တွယ်ဆောင်ရွက်ပေးနိုင်တယ်။ ကိန်းဂဏန်းတွေအပြင် စာသား၊ ရုပ်သံ စတာတွေကိုပါ လက်ခံ အလုပ်လုပ်ပေးနိုင်တယ်။ ဒီလိုလုပ်ဆောင်နိုင်စွမ်းဟာ ကွန်ပျူတာတွေကို နယ်ပယ်ပေါင်းစုံမှာ တွင်တွင်ကျယ်ကျယ် အသုံးချလာရခြင်းရဲ့ အဓိက အကြောင်းအရင်း ဆိုရင်လည်း မမှားဘူး။

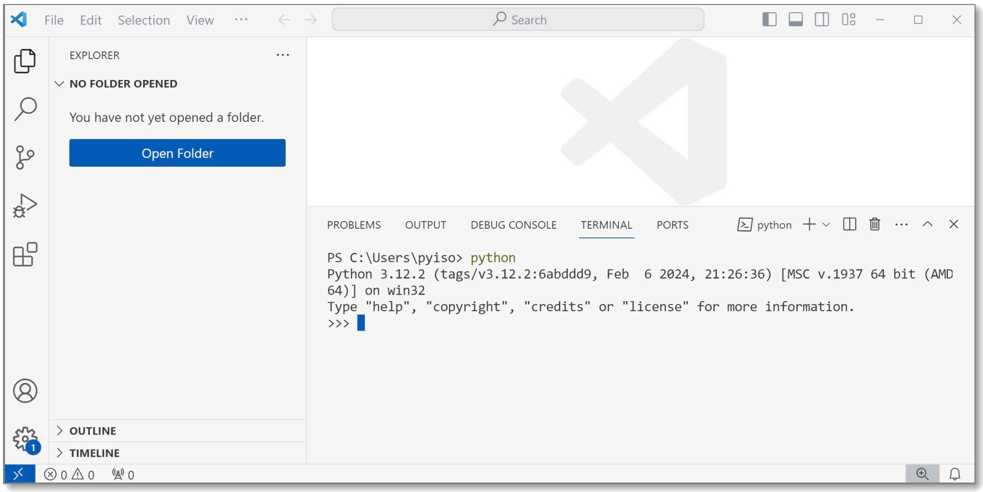
ဒေတာ အမျိုးအစား အများအပြားရှိပေမဲ့ အခြေခံအကျဆုံးက ကိန်းဂဏန်းတွေပါ။ ကွန်ပျူတာတွေကို စတင်တီထွင်ဖို့ ကြိုးစားလာကြတဲ့ အဓိက အကြောင်းအရင်းကလည်း ဂဏန်းသင်္ချာ အတွက်အချက်တွေကို လုပ်ဆောင်ရာမှာ လူတွေကို အကူအညီ ပေးဖို့အတွက်ပဲလို့ ဆိုနိုင်ပါတယ်။ ဒါ့အပြင် စာသား၊ ရုပ်သံ စတဲ့ အခြားဒေတာ အမျိုးအစားတွေကို ကွန်ပျူတာထဲမှာ ဖော်ပြသိမ်းဆည်းထားဖို့အတွက် ကိန်းဂဏန်းတွေကိုပဲ အသုံးပြုထားတယ်ဆိုတာ နောက်ပိုင်းမှာ နားလည်သိမြင် လာပါလိမ့်မယ်။ ဒါကြောင့်လည်း ယနေ့ခေတ် ကွန်ပျူတာတွေကို ဒစ်ဂျစ်တယ် ကွန်ပျူတာတွေလို့ ခေါ်တာဖြစ်တယ်။ ကိန်းဂဏန်းကို အခြေခံပြီး အလုပ်လုပ်တဲ့ ကွန်ပျူတာတွေပေါ့။

၁.၁ ကိန်းဂဏန်းများ

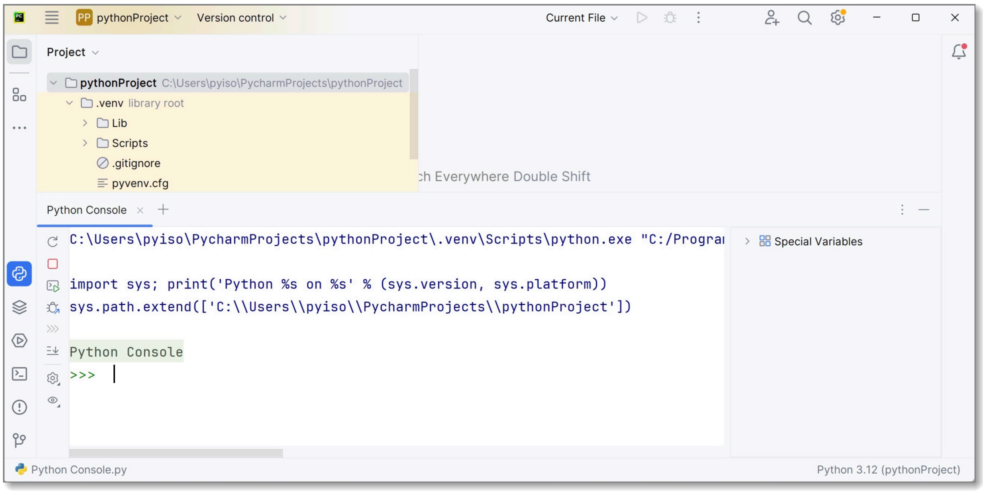
Python အင်တာပရက်တာနဲ့ Python ကုဒ်တွေကို run လို့ရတဲ့နည်း နှစ်ခုရှိပါတယ်။ တစ်နည်းက ကုဒ်တွေကို **.py** ဖိုင်နဲ့သိမ်းပြီး **python** ကွန်မန်းနဲ့ run တာပါ။ ဒါက ကားရဲလ် ပရိုဂရမ်တွေမှာ သုံးခဲ့တဲ့နည်း။ နောက်တစ်နည်းကတော့ ကုဒ်တစ်ကြောင်းချင်း ရိုက်ထည့်ပြီး run တဲ့နည်းပါ။ ဒီနည်းလမ်းက interactive mode နဲ့ အင်တာပရက်တာကို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

ကြည့်ရတာ လွယ်တဲ့အတွက်ကြောင့်ပါ။

ဝင်းဒိုး Command Prompt သို့မဟုတ် မက်ခ်အိုအက်စ် Terminal မှာ **python** ကွန်မန်း run ပြီး interactive mode ကို ဝင်နိုင်ပါတယ်။ VS Code မှာပဲ သုံးချင်လည်း ရတယ်။ **View** မီနူးမှ **Terminal** ကိုဖွင့် (Ctrl + ` ရှော့ကတ်ကီးနဲ့ ဖွင့်လို့လည်းရတယ်) ပြီး **python** ကွန်မန်း run ရုံပဲ။ PyCharm မှာဆိုရင် **Python Console** အိုင်ကွန်နှိပ်ပြီး ဝင်ရပါမယ်။ ပုံ (၁.၁)၊ (၁.၂) တွင်ကြည့်ပါ။



ပုံ ၁.၁ VS Code Python Console



ပုံ ၁.၂ PyCharm Python Console

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

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

အောက်ပါအတိုင်း တစ်ခုပြီးတစ်ခု ဆက်လက်စမ်းသပ်ကြည့်ပါ။

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

3×3 ကို `3 * 3` လို့ ရိုက်ထည့်ပေးရပြီး $5 \div 2$ အတွက် `5 / 2` လို့ ရေးရတာကို သတိပြုမိမှာပါ။ Programming language အများစုမှာ `*` (asterisk) အမြှောက်သင်္ကေတအဖြစ် အသုံးပြုပြီး `/` (forward slash) ကို အစားသင်္ကေတအနေနဲ့ အသုံးပြုလေ့ရှိတယ်။

Values and Types

တန်ဖိုးတိုင်းဟာ တိုက်ပ် (*type*) တစ်မျိုးမျိုးမှာ ပါဝင်ပါတယ်။ `-3, 0, 2` စတဲ့ တန်ဖိုးတွေဟာ `int` (integer ရဲ့ အတိုကောက်) တိုက်ပ်ဖြစ်ပြီး `-3.0, 0.1, 3.3333` စတာတွေက `float` တိုက်ပ် ဖြစ်ပါတယ်။ ဒဿမကိန်းတွေကို ကွန်ပျူတာနဲ့ ဖော်ပြဖို့ *floating point* လို့ခေါ်တဲ့ နည်းစနစ်ကို အသုံးပြုတယ်။ ဒဿမကိန်း အတွက်အချက်တွေကိုလည်း ဒီနည်းစနစ်ကို အခြေခံပြီး ကွန်ပျူတာက လုပ်ဆောင်တာပါ။ ဒါကြောင့် floating point ဟာ ဒဿမကိန်းတွေကို ဖော်ပြဖို့နဲ့ ဒဿမကိန်း အတွက်အချက်တွေ လုပ်ဆောင်ဖို့ တီထွင်ထားတဲ့ နည်းစနစ်တစ်ခုလို့ ဆိုနိုင်ပါတယ်။ ဒီစနစ်ကို အခြေခံထားတဲ့ ဒဿမကိန်းတွေကို programming language တွေမှာ `float` တိုက်ပ်လို့ ခေါ်တာပါ။

`float` တိုက်ပ်ဟာ လိုသလောက် တိကျလို့မရတဲ့ သဘောရှိတယ်။ အောက်ပါအတိုင်း စမ်းကြည့်ရင် `0.3` နဲ့ `1.0` ရသင့်တာ ဖြစ်ပေမဲ့ အတိအကျ အဖြေမထွက်ပါဘူး။

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

ကွာခြားချက်က မဆိုစလောက် သေးငယ်တယ် ဆိုပေမဲ့ ဒီအချက်ကို ပရိုဂရမ်မာ အနေနဲ့ ဂရုပြုမိဖို့ လိုပါတယ်။ Floating point စနစ်ဟာ လုံးဝကြီးတိကျဖို့ မလိုအပ်တဲ့ (တနည်းအားဖြင့် မဆိုစလောက် သေးငယ်တဲ့ ကွာဟချက်ကို လက်ခံနိုင်တဲ့) ကိန်းဂဏန်းအတွက်အချက် ကိစ္စတွေအတွက် ရည်ရွယ်တာပါ။ သိပ္ပံနဲ့ နည်းပညာဆိုင်ရာ တိုင်းတာ တွက်ချက်မှုတွေအတွက် အသုံးပြုလေ့ရှိတယ်။ ဒဿမကိန်းတွေ လုံးဝအတိအကျ ဖြစ်ဖို့ လိုအပ်တဲ့ ကိစ္စမျိုးတွေ (ဥပမာအားဖြင့် ငွေကြေးကိစ္စ အတွက်အချက်) မှာ အသုံးမပြုသင့်ပါဘူး။ ဆယ်ပြားကို `0.1` နဲ့ဖော်ပြရင် ဆယ်ပြားစေ့ ဆယ်စေ့ဟာ တစ်ကျပ် ဖြစ်ကို ဖြစ်သင့်ပြီး `0.9999999999999999` မဖြစ်သင့်ဘူး။ ဒီလိုကိစ္စမျိုးတွေအတွက် Python မှာ `Decimal` ကို အသုံးပြုနိုင်ပါတယ်။ လောလောဆယ် ကိန်းဂဏန်းတွေနဲ့ ပါတ်သက်ပြီး စကတည်းက သိထားသင့်တာတချို့ကို ကြိုပြောထားတာပါ။ `Decimal` တိုက်ပ် အကြောင်း မကြာခင် လေ့လာမှာပါ။

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

`int` တိုက်ပ် ဖော်ပြနိုင်တဲ့ အကြီးဆုံး အပေါင်းကိန်းပြည့် သို့မဟုတ် အငယ်ဆုံး အနုတ်ကိန်းပြည့် တန်ဖိုးကို ကန့်သတ်ထားတာ မရှိဘူး။ သီအိုရီအရ ကြိုက်သလောက် ကြီးလို့/ငယ်လို့ ရပါတယ်။ လက်တွေ့

မှာတော့ အသုံးပြုတဲ့ ကွန်ပျူတာစနစ်ပေါ် မူတည်ပြီး ကန့်သတ်ချက်ရှိမှာပါ။ ဒီစာရေးနေတဲ့ ကွန်ပျူတာမှာ အောက်ပါကိန်းပြည့်တွေကို အသာလေး တွက်ချက်ပေးနိုင်ပါတယ်။ ဒီ့ထက်အများကြီး ကြီးတဲ့ဟာတွေကိုလည်း ကိုင်တွယ်တွက်ချက်နိုင်အုံးမှာပါ။ တော်ရုံကိစ္စတွေအတွက် ကွန်ပျူတာစနစ်တစ်ခုရဲ့ လက်တွေ့ကန့်သတ်ချက်ကို ကျော်လွန်သွားဖို့ မလွယ်ပါဘူး။

```
>>> 9000000000000000000000000000 + 5000000000
9000000000000000005000000000
>>> -9000000000000000000000000000 + 1
-8999999999999999999999999999
```

float ကတော့ int နဲ့ မတူဘဲ အကြီးဆုံး/အငယ်ဆုံး ကန့်သတ်ချက်ရှိပါတယ်။ ဒီကန့်သတ်ချက်ကလည်း ကွန်ပျူတာစနစ်ပေါ် မူတည်ပါတယ်။ တစ်ကိုယ်ရေသုံး ကွန်ပျူတာ အများစုအတွက် 1×10^{400} (၁ နောက် သုည အလုံး ၄၀၀) သို့မဟုတ် -1×10^{400} (အနှုတ် ၁ နောက် သုည အလုံး ၄၀၀) ဟာ ကန့်သတ်ချက်ကျော်လွန်ပါတယ်။

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

ဒသမကိန်းကို e အမှတ်အသားနဲ့ရေးထားတာပါ။ အက္ခရာ e နောက် လိုက်တဲ့ ကိန်းပြည့်ဂဏန်းကို 10 ထပ်ကိန်းလို့ မှတ်ယူရပါတယ်။ e3 က 1×10^3, e-3 က 1×10^{-3} ပါ။ အက္ခရာ E အကြီးနဲ့လည်း ရေးနိုင်တယ်။ ထပ်ကိန်း ကြီးလွန်းတဲ့အတွက် ကန့်သတ်ချက် ကျော်လွန်ရင် `inf` (အနှုတ်ကိန်းဆိုရင် `-inf`) ရပါမယ်။ Infinity ကို ဆိုလိုတာပါ။ (၁) မပြည့်တဲ့ ပမာဏ သေးငယ်လွန်းတဲ့ ဂဏန်းတွေကိုလည်း အနီးစပ်ဆုံး သုညအနေနဲ့ ယူပါတယ်။ အနီးစပ်ဆုံးယူတဲ့အခါ အပေါင်း/အနှုတ်ကိုတော့ ခွဲခြားပေးပါတယ်။ e အမှတ်အသားနဲ့ရေးထားတဲ့ နောက်ထပ် ဥပမာလေး နှစ်ခုပါ

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

ဂဏန်းအလုံးအရေအတွက် များရင် $100,500$ (တစ်သိန်းငါးရာ)၊ $1,500,000$ (တစ်သန်းငါးသိန်း) စသဖြင့် သုံးလုံးတစ်ဖြတ် ကော်မာခံရေးလေ့ရှိတယ်။ Python မှာတော့ ကော်မာအစား _(underscore) နဲ့ သုံးလုံးတစ်ဖြတ် ခြားရေးနိုင်ပါတယ်။

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

တိုက်ပ်မတူသည့် ကိန်းဂဏန်း အိပ်စ်ပရက်ရှင်များ

အိပ်စ်ပရက်ရှင်တွေမှာ ပါဝင်တဲ့ တိုက်ပ် တစ်မျိုးတည်း ဖြစ်ရမယ် မရှိပါ။ တိုက်ပ်မတူတဲ့ ကိန်းဂဏန်းတွေ အိပ်စ်ပရက်ရှင်တစ်ခုမှာ ရောပြီး ပါဝင်နိုင်ပါတယ်။

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

`int` နဲ့ `float` ရောနေရင် အိပ်စ်ပရက်ရှင် ရလဒ်သည် `float` တိုက်ပ် 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 ဒီဗီးရှင်း၊ မော်ဂျူလို နှင့် ထပ်ကိန်းတင်ခြင်း

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

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

သင်္ချာမှာ ဒီလိုမျိုး အစားကို အင်တီဂျာ ဒီဗီးရှင်း (integer division) လို့ခေါ်ပါတယ်။ အကြွင်းရှာမယ်ဆိုရင် `%` အော်ပရိတ်တာ ရှိပါတယ်။ `%` ကို မော်ဂျူလို (modulo) အော်ပရိတ်တာလို့ ခေါ်တယ်။ remainder အော်ပရိတ်တာလို့လည်း ခေါ်တယ်။

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

အင်တီဂျာ ဒီဗီးရှင်းနဲ့ မော်ဂျူလိုကို အနှုတ်ကိန်းတွေနဲ့ သုံးမယ်ဆိုရင် သတိပြုပါ။ စားလဒ် အနှုတ်ကိန်း ဖြစ်ရင် `//` က ပိုငယ်တဲ့ အနှုတ်ကိန်းကို အနီးစပ်ဆုံး ယူမှာပါ။ တစ်နည်းအားဖြင့် round down လုပ်တာ ဖြစ်တယ်။

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

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

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

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

မော်ဒျူလိုနဲ့ အင်တီဂျာ ဒီဗီးရှင်း 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 ဒသမကိန်းတွေနဲ့လည်း သုံးလို့ရပေမဲ့ ရလဒ်တွေက အထက်ပါ ညီမျှခြင်းကို ပြေလည်စေမှာ မဟုတ်ပါဘူး။ `float` တိုက်ပ်ဟာ

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

ထပ်ကိန်းတင် (exponentiation) ဖို့ အတွက် အော်ပရိတ်တာက `**` ပါ။ 2^4 နဲ့ $(3.3)^3$ ကို အခုလို တွက်ပါတယ်။

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

သင်္ချာဖန်ရှင်များ

ကိန်းဂဏန်းတွေအကြောင်း လေ့လာလက်စနဲ့ `math` လိုက်ဘရီ သင်္ချာဖန်ရှင်တချို့ကိုလည်း တစ်ခါတည်း ဆက်ကြည့်လိုက်ရအောင်။ အခုကက သင်္ချာဖန်ရှင်ဆိုတာထက် ဖန်ရှင် အခြေခံအသုံးပြုပုံကို စပြီးလေ့လာမှာပါ။ `math` လိုက်ဘရီက Python မှာ တစ်ခါတည်း ထည့်ထားပေးပြီးသား (built-in) လိုက်ဘရီပါ။ အင်စတောလ်လုပ်စရာ မလိုဘဲ အင်ပို့လုပ်ပြီး သုံးလို့ရတယ်။

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

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

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

သင်္ချာဖန်ရှင်အားလုံးဟာ input တန်ဖိုးတစ်ခု သို့မဟုတ် တစ်ခုထက်ပို၍ လက်ခံပြီး output တန်ဖိုးတစ်ခု 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
```

အကြီးဆုံးဘုံဆခွဲကိန်းကို gcd ဖန်ရှင်နဲ့ ရှာတာပါ။ အင်တီဂျာ input တစ်ခုနဲ့အထက် လက်ခံတဲ့ ဖန်ရှင်ဖြစ်တယ်။ input ဂဏန်းအားလုံးကို စားလို့ပြတ်တဲ့ အကြီးဆုံးကိန်းကို ရှာပေးတယ်။ ကိန်းပြည့်မဟုတ်တာ ထည့်ရင် အယ်ရာဖြစ်ပါတယ်။

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

လော့ဂရစ်သမ်၊ ထရီဂိုနိုမေထရီ 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
```

၁.၂ 'တိုက်ပ်' ဆိုတာ ဘာလဲ

Programming language အားလုံးမှာ တိုက်ပ် သို့မဟုတ် ဒေတာတိုက်ပ် သဘောတရား ပါရှိပါတယ်။ `int` နဲ့ `float` တိုက်ပ်မှာ ပါဝင်တဲ့ ကိန်းပြည့်တွေနဲ့ ဒသမကိန်းတွေကို မိတ်ဆက်ပြီးတဲ့အခါ 'တိုက်ပ်' ဆိုတာဘာလဲ တိတိကျကျ ရှင်းပြလို့ရပါပြီ။ တိုက်ပ်တစ်ခုဟာ

- တန်ဖိုးတွေပါဝင်တဲ့ အစု (set) တစ်ခု နဲ့
- ၎င်းတန်ဖိုးများအပေါ်တွင် အသုံးချနိုင်တဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစုတစ်ခု

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

$$\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$$
$$\{+, -, *, /, //, \%, **, \ldots\}$$

ဖြစ်ပါတယ်။ `float` တိုက်ပ်ကတော့ ကိန်းစစ် (real numbers) တွေပါဝင်တဲ့ အစုနဲ့ ၎င်းတို့အပေါ်မှာ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေ ပါဝင်တဲ့ အစုတို့ ဖြစ်ပါတယ်။ `int` နဲ့ `float` တိုက်ပ်မှာ အော်ပရေးရှင်းတွေ တူတူဖြစ်နေတာ တွေ့ရမှာပါ။ ဒီလိုအမြဲဖြစ်မယ်လို့ မယူဆရပါဘူး။ တိုက်ပ် မတူတဲ့အခါ အသုံးချလို့ ရနိုင်တဲ့ အော်ပရေးရှင်းတွေ ကွာခြားနိုင်ပါတယ်။ 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 အသုံးချပုံပါ။ နှစ်ထပ်ကိန်းရင်းနဲ့ ပကတိတန်ဖိုး ရှာပေးပါတယ်။ လိုအပ်ရင် ကိုယ်ပိုင်ဖန်ရှင်တွေ သတ်မှတ်ပြီး တိုက်ပ်တစ်ခုရဲ့ အော်ပရေးရှင်းတွေကို ဖြည့်စွက်တိုးချဲ့နိုင်ပါတယ်။

> အော်ပရေးရှင်းနဲ့ အော်ပရိတ်တာ ရောထွေးစရာ ရှိပါတယ်။ $+, -, *, /, //, \%, **$ စတဲ့ သင်္ကေတတွေကို အော်ပရိတ်တာလို့ ခေါ်ပါတယ်။ အော်ပရေးရှင်း လုပ်ဆောင်ဖို့အတွက် အသုံးပြုတဲ့ သင်္ကေတတွေကို အော်ပရိတ်တာလို့ ခေါ်တာပါ။ ဥပမာ "$*$ သင်္ကေတဟာ အမြှောက်အော်ပရေးရှင်း လုပ်ဆောင်ဖို့ သတ်မှတ်ထားတဲ့ အော်ပရိတ်တာ" လို့ ပြောတယ်။ အမြှောက် 'အော်ပရေးရှင်း' ကျတော့ မြှောက်တဲ့အလုပ် ဆောက်ရွက်တာကို ဆိုလိုတာ။

၁.၃ ဗေရီရေဘဲလ်များ

ဗေရီရေဘဲလ်ဆိုတာ တန်ဖိုးတစ်ခုကို ကိုယ်စားပြုတဲ့ နံမည်ပါပဲ။ နံမည်နဲ့ ၎င်းကိုယ်စားပြုတဲ့ တန်ဖိုး တွဲဖက်ပေးဖို့ အဆိုင်းမန့် (*assignment*) စတိတ်မန့်ကို သုံးရပါတယ်။

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

`age` နဲ့ `weight` ဟာ ဗေရီရေဘဲလ်တွေ ဖြစ်ပါတယ်။ ညီမျှခြင်းသင်္ကေတ (=) ကတော့ အဆိုင်းမန့် အော်ပရိတ်တာပါ။ ဗေရီရေဘဲလ်နဲ့ တန်ဖိုး တွဲဖက်ပေးတဲ့ အော်ပရိတ်တာ ဖြစ်တယ်။ ဗေရီရေဘဲလ်နံမည်ကို variable *identifier* လို့လည်း ခေါ်ပါတယ်။ Identifier က နည်းပညာ အခေါ်အဝေါ်ပေါ့။ Variable name က သာမန်လူ နားလည်တဲ့ နည်းနဲ့ ပြောတာပါ။ ဗေရီရေဘဲလ် တစ်ခုချင်း ထည့်ကြည့်ရင် ၎င်းကိုယ်စားပြုတဲ့ တန်ဖိုးကို ပြန်ထုတ်ပေးတာ တွေ့ရမှာပါ။

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

အိပ်စ်ပရက်ရှင်တွေက ဗေရီရေဘဲလ်တွေနဲ့ ဖြစ်နိုင်ပါတယ်။ အိပ်စ်ပရက်ရှင် တွက်ချက်ရင် ဗေရီရေဘဲလ် တန်ဖိုးနဲ့ အစားထိုး တွက်ချက်တယ်လို့ ယူဆရမှာပါ။ ဥပမာ

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

ဗေရီရေဘဲလ်တစ်ခုကို အိပ်စ်ပရက်ရှင်ရလဒ်နဲ့ အဆိုင်းမန့် လုပ်လို့ရပါတယ်။ `rect_area` ကို အောက်တွင် ကြည့်ပါ။ အလျား အနံ မြှောက်လဒ်ကို အဆိုင်းမန့် လုပ်ထားတာ တွေ့ရပါမယ်။

```
>>> rect_width = 22.5
>>> rect_length = 10
```

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

အဆိုင်းမန့် စတိတ်မန့်

ဗေရီရေဘဲလ်တစ်ခုဟာ အချိန်တစ်ချိန်မှာ တန်ဖိုးတစ်ခုကိုပဲ ကိုယ်စားပြုနိုင်တယ်။ ဒါပေမဲ့ အချိန်ကာလပေါ် မူတည်ပြီး တန်ဖိုးပြောင်းနိုင်တယ်။ (တစ်ချိန်တည်းမှာ တန်ဖိုးနှစ်ခု မဖြစ်နိုင်ဘူး)။ ဥပမာ x တန်ဖိုးဟာ ပထမ 10 ပါ။ ဒုတိယ အဆိုင်းမန့်လုပ်ပြီးတဲ့ အချိန်မှာ အဲ့ဒီ x ကပဲ 1000 ဖြစ်နေမှာပါ။

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

၁.၄ စာသားများ

စာသား (text) ဟာ အသုံးအများဆုံး ဆက်သွယ်ဆောင်ရွက်ရေး ကြားခံနယ်တစ်ခုပါ။ ဝက်ဘ်ဆိုက် စာမျက်နှာ၊ အီးမေးလ်၊ အီးဘွတ်ခ်နဲ့ အီလက်ထရွန်းနစ် စာရွက်စာတမ်း (e-documents) စတာတွေမှာ ရုပ်သံတွေ အသုံးပြုလာကြပေမဲ့ စာသား အဓိကဖြစ်နေဆဲပါပဲ။ ဆိုရှယ်မီဒီယာ၊ ဂိမ်းနဲ့ အခြားအပ်ပ်တွေ ဟာလည်း စာသားနဲ့ မကင်းနိုင်ကြပါဘူး။ ဒါကြောင့် ပရိုဂရမ်းမင်းအတွက် စာသားဟာ ဘယ်လောက်ထိ အရေးပါကြောင်း အများကြီးပြောစရာ လိုမယ်မထင်ပါဘူး။

ပရိုဂရမ်းမင်းမှာ စာသားကို *string* လို့ခေါ်ပြီး ကာရက်တာ (*character*) တွေနဲ့ စီတန်းဖွဲ့စည်းထား တယ်။ ကာရက်တာဆိုတာ အခြေခံ သင်္ကေတအချက်အလက် ယူနစ်တစ်ခုပါပဲ။ အက္ခရာ၊ ဂဏန်း (digit)၊ သင်္ကေတ သို့မဟုတ် ကွန်ထရိုးလ်ကုဒ် တစ်ခုခု ဖြစ်နိုင်ပါတယ်။ ဥပမာ A, B, C, $, @, #, 1, 3, _ စသည်ဖြင့်။ Double quotes(") တစ်စုံကြား ညှပ်ရေးထားတဲ့ ကာရက်တာတွေ 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 ကာရက်တာ အရေအတွက်ကို `len` ဖန်ရှင်နဲ့ စစ်ကြည့်နိုင်ပါတယ်။ ကာ ရက်တာ 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။ + အော်ပရိတ်တာနဲ့ စာသားတွေကို ဆက်နိုင်ပါတယ်။

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

စာသားအချင်းချင်းပဲ ဆက်လို့ရပါတယ်။ စာသားနဲ့ ကိန်းဂဏန်း ဆက်လို့မရပါဘူး။ အောက်ပါအတိုင်း စမ်းကြည့်တဲ့အခါ str နဲ့ float ဆက်လို့မရဘူးလို့ အယ်ရာမက်ဆေ့ချ် 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 ကိန်းဂဏန်းတစ်ခုကနေ စာသားကို ထုတ်ပေးပါတယ်။ မူရင်းကိန်းဂဏန်းကို စာသားဖြစ်အောင် ပြောင်းလိုက်တာ မဟုတ်ပါဘူး။ ကိန်းဂဏန်း တန်ဖိုးကနေ ၎င်းကိုဖော်ပြတဲ့ စာသားကို ဖန်ရှင်က ပြန်ထုတ်ပေးတာပါ။

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

ထွက်လာတဲ့ တန်ဖိုးဟာ စာသားဖြစ်တဲ့အတွက် single quote ပါနေတာ သတိပြုပါ။ pi တန်ဖိုးနဲ့ စာသားအခုလို ဆက်ရပါမယ်။

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

str ဖန်ရှင်နဲ့ စာသားရအောင် အရင်လုပ်ပြီးမှ ဆက်ထားတာပါ။

စာသားကနေ ကိန်းဂဏန်း လိုချင်ရင် int နဲ့ float ဖန်ရှင် သုံးနိုင်ပါတယ်။

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

ဂဏန်းပြောင်းလို့မရတဲ့ 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 ပွားယူလို့ရတယ်။

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

`'hello'` သုံးခါ ဆက်လိုက်တာပါ။ string နဲ့ အကြိမ်အရေအတွက် ကိန်းပြည့် ဖြစ်ရပါမယ်။ သုည သို့မဟုတ် အနှုတ်ကိန်း ဖြစ်နေရင် empty string ရမှာပါ။

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

စာသားနဲ့ ဂဏန်း ဖလှယ်လို့ရပါတယ်

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

string ဇေရီရေဘဲလ်များ

ဇေရီရေဘဲလ်တွေကို string တန်ဖိုးတွေအတွက်လည်း အသုံးပြုနိုင်တယ်။ အောက်ပါ အိပ်စ်ပရက်ရှင်တွေကို နားလည်နိုင်မလား ကြိုးစားကြည့်ပါ။ ဗေရီရေဘဲလ် တစ်ခုချင်းကို သူ့ရဲ့တန်ဖိုးနဲ့ အစားထိုးပြီး +, * အော်ပရိတ်တာတွေ အလုပ်လုပ်ပုံနဲ့ ဆက်စပ်စဉ်းစားရင် ဘာကြောင့် အခုလို အဖြေထွက်လဲ ခန့်မှန်းနိုင်မှာပါ။ သိပ်ခက်ခက်ခဲခဲ မဟုတ်ပါဘူး။

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

Escape Character and Escape Sequence

String တစ်ခု ရေးတဲ့အခါ ပုံမှန်အားဖြင့် လိုချင်တဲ့ စာသားအတိုင်း ကီးဘုဒ်ကနေ ကာရက်တာ တစ်လုံးချင်း ရိုက်ရုံပါပဲ။ စပယ်ရှယ် ကာရက်တာ တချို့ကိုတော့ ကီးဘုဒ်ကနေ တိုက်ရိုက် ရိုက်ထည့်လို့ မရဘဲ သီ

ခြားနည်းလမ်းတစ်ခုနဲ့ ရေးပေးရပါတယ်။ ဥပမာ စာသားထဲမှာ tab ကာရက်တာအတွက် `\t` နဲ့ newline အတွက် `\n` ရေးရမှာပါ။ ကီးဘုဒ်ကနေ tab ကီး၊ enter/return ကီး နှိပ်ပြီး တိုက်ရိုက်ထည့်လို့မရပါဘူး။ သီးခြားအဓိပ္ပါယ် တစ်ခုအတွက် \ နဲ့စတဲ့ ကာရက်တာအတွဲလိုက်ကို *escape sequence* လို့ခေါ်ပြီး \ ကိုတော့ *escape character* လို့ ခေါ်ပါတယ်။

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

Escape sequence တွေကို **bold** ဖောင့်နဲ့ ပြထားပါတယ်။ Python ကွန်ဆိုးလ်မှာ `\t` နဲ့ `\n` ကို အရှိအတိုင်း ပြနေပါတယ်။ ဒါပေမဲ့ အခုလို စမ်းကြည့်ရင် သိသာပါလိမ့်မယ်။

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

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

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

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

Double quotes နဲ့ စာသားထဲက single quote သို့မဟုတ် 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။ ကျန်တဲ့တစ်မျိုးကတော့ ပါရင်လည်းရ၊ မပါလည်း ပြဿနာမရှိဘူး။ အောက်ပါတို့ကို ဂရုစိုက် လေ့လာကြည့်ပါ။

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

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

တေဘဲလ် ၁.၁ Python Escape Sequences

၁.၅ အိပ်စ်ပရက်ရှင်များ

တိုက်ပ် ဆိုတာဘာလဲ အကြမ်းဖျဉ်း ရှင်းပြခဲ့ ပြီးပါပြီ။ ကိန်းဂဏန်းနဲ့ စာသား တိုက်ပ် တချို့ကိုလည်း လေ့လာခဲ့ပြီးပြီ။ တကယ်တော့ အိပ်စ်ပရက်ရှင် (*expression*) ဆိုတာလည်း အသစ်အဆန်း မဟုတ်ပါဘူး။ တန်ဖိုးတစ်ခုဟာ အရိုးရှင်းဆုံး အိပ်စ်ပရက်ရှင်လို့ ဆိုနိုင်ပါတယ်။ "Hello", 2.3 စတဲ့ တန်ဖိုးတွေဟာ အိပ်စ်ပရက်ရှင်တွေပါပဲ။ ဗေရီရေဘဲလ်ဟာလည်း တန်ဖိုးကို ကိုယ်စားပြုတဲ့အတွက် အိပ်စ်ပရက်ရှင်လို့ ယူဆရမှာပါ။

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

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

အိပ်စ်ပရက်ရှင်ကို ရှုထောင့်အမျိုးမျိုးကနေ အဓိပ္ပါယ်ဖွင့်ဆိုကြတာ တွေ့ရပါတယ်။ “အိပ်စ်ပရက်ရှင်ဆိုတာ တန်ဖိုးတစ်ခု ပြန်ပေးတဲ့ အော်ပရေးရှင်း အတွဲအဆက်ဖြစ်တယ်” လို့ဆိုရင် အတိုင်းအတာတစ်ခုအထိ တိတိကျကျရှိပြီး နားလည်ရလည်း လွယ်ပါတယ်။ တချို့စာအုပ်တွေမှာတော့ အိပ်စ်ပရက်ရှင်ကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်လို့ သတ်မှတ်ပါတယ်။

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

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

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

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

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

၁.၆ ဘူလီယန် တိုက်ပ်နှင့် ဘူလီယန် အိပ်စ်ပရက်ရှင်

Python မှာ `bool` တိုက်ပ်ဟာ ဘူလီယန် (boolean) တိုက်ပ်ကို ဆိုလိုတာဖြစ်ပြီး `True` နဲ့ `False` တန်ဖိုး နှစ်ခုပဲ ပါဝင်တယ်။ မှန်ခြင်း/မှားခြင်း၊ ရှိခြင်း/မရှိခြင်း၊ ဖြစ်ခြင်း/မဖြစ်ခြင်း စတဲ့အချက်အလက်မျိုးတွေကို ဖော်ပြဖို့ (boolean) တိုက်ပ်ကို အသုံးပြုနိုင်ပါတယ်။

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

တန်ဖိုးရှာတဲ့အခါ ဘူလီယန်တိုက်ပ် ရလာမဲ့ အိပ်စ်ပရက်ရှင်တွေကို ဘူလီယန် အိပ်စ်ပရက်ရှင်လို့ ခေါ်တယ်။ အောက်ပါတို့ကို ကြည့်ပါ

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

<, >, <=, >=, ==, != စတဲ့ သင်္ကေတတွေဟာ comparison operator တွေပါ။ Relational operator တွေလို့လည်း ခေါ်ကြတယ်။ အလယ်တန်းသင်္ချာမှာ သင်ခဲ့ရတဲ့ $<$, $>$, $\leq$, $\geq$, $=$, $\neq$ စတာတွေနဲ့ အဓိပ္ပါယ်တူပါတယ်။ ညီ/မညီ စစ်ချင်ရင် ညီမျှခြင်းသင်္ကေတနှစ်ခု == နဲ့ စစ်ရပါမယ်။ Comparison operator တွေကို ဒေတာတိုက်ပ် အမျိုးမျိုးနဲ့ တွဲဖက် အသုံးပြုနိုင်တာကို တွေ့ရမှာပါ။ တန်ဖိုးတစ်ခုနဲ့ တစ်ခု နှိုင်းယှဉ်နိုင်ခြင်းဟာ အရေးပါတဲ့ ကိစ္စဖြစ်ပါတယ်။

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

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

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

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

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

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

ကိန်းဂဏန်းတစ်ခုက တစ်နဲ့ တစ်ဆယ်ကြား ရှိ/မရှိ အခုလိုစစ်နိုင်ပါတယ်

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

တစ်ထက်လည်းကြီး၊ တစ်ဆယ်ထက်လည်း ငယ်တယ်ဆိုရင် တစ်နဲ့တစ်ဆယ်ကြား ဖြစ်ပါတယ်။ (Python မှာ `1 < x < 10` နဲ့ စစ်လို့လည်း ရတယ်)။ ယဉ်မောင်းလိုင်စင် ဖြေဖို့ အသက် ဆယ့်ရှစ်နှစ်နဲ့ အထက် ဖြစ်ရမယ်၊ မှတ်ပုံတင်လည်း ရှိရမယ်ဆိုပါစို့။ ဘူလီယန် အိပ်စ်ပရက်ရှင်နဲ့ အခုလို ဖော်ပြနိုင်တယ်

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

(အခု ဥပမာမှာ `True/False` တွေ ပုံသေထည့်ထားပေမဲ့ နောက်ပိုင်းမှာ ပရိုဂရမ် input ပေါ် မူတည်ပြီး တွက်ချက်လုပ်ဆောင်တာတွေ တွေ့ရမှာပါ)။

ဘူလီယန်တန်ဖိုး/အိပ်စ်ပရက်ရှင် နှစ်ခုအနက် အနည်းဆုံး တစ်ခု မှန်/မမှန် `or` နဲ့ စစ်နိုင်တယ်။ နှစ်ခုလုံး အမှားဖြစ်မှ `False` ရပါတယ်။ တစ်ခုမှန်တာနဲ့ အမှန်ထွက်ပါတယ်။

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

ကိန်းတစ်ခုဟာ တစ်နဲ့ တစ်ဆယ်ရဲ့ ပြင်ပမှာရှိလား အခုလို စစ်နိုင်တယ်

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

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

တစ်ထက်ငယ်ရင် (သို့) တစ်ဆယ်ထက်ကြီးရင် တစ်နဲ့တစ်ဆယ်ကြားမှာ မဟုတ်လို့ပါ။ (Python မှာ တန်ဖိုးနှစ်ခုကြားမှာရှိလား သို့မဟုတ် တန်ဖိုးနှစ်ခု ပြင်ပမှာရှိလား စစ်လို့ရတဲ့နည်း တစ်ခုမကရှိပါတယ်။ အခုနည်းလမ်းက and နဲ့ or ကို အသုံးချတဲ့ ဥပမာတချို့ကို ပြခြင်းသာဖြစ်တယ်)။

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

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

တန်ဖိုးနှစ်ခု ကြား (သို့) ပြင်ပမှာ ရှိ/မရှိ စစ်တဲ့အခါ အဲ့ဒီတန်ဖိုးနှစ်ခု အပါအဝင်လား (inclusive)၊ ဒါမှမဟုတ် မပါဝင်ဘူးလား (exclusive) ဂရုစိုက်ဖို့ လိုပါတယ်။ တစ်နဲ့ တစ်ဆယ်ကြား (၎င်းတို့ အပါအဝင်) ဆိုရင် အခုလို စစ်ရမှာပါ

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

not အော်ပရိတ်တာကတော့ True/False တန်ဖိုးကို ပြောင်းပြန် ဖြစ်စေတယ်။

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

တစ်နဲ့ တစ်ဆယ် ပြင်ပ (၎င်းတို့မပါ) မှာ ရှိ/မရှိကို အခုလိုလည်း စစ်လို့ရတယ်

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

'ငါးရာအောက်' လို့ ပြောတာဟာ 'ငါးရာနှင့်အထက် မဟုတ်' လို့ ပြောတာနဲ့ အဓိပ္ပါယ်တူတူပါပဲ။

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

not အော်ပရိတ်တာ သုံးခြင်းအားဖြင့် အကြောင်းအရာတစ်ခုကိုပဲ အဓိပ္ပါယ်အားဖြင့် ညီမျှတဲ့ အိပ်စ်ပရက်ရှင် အမျိုးမျိုးနဲ့ ဖော်ပြလို့ရလာတာကို တွေ့နိုင်တယ်။

အခန်း ၂

အော့ဘ်ဂျက်များ

"အော့ဘ်ဂျက် (*object*) ဆိုတာဘာလဲ" ရှုထောင့် အမျိုးမျိုးကနေ ရှင်းပြနိုင်ပါတယ်။ အပြည့်စုံဆုံး၊ အမှန်ကန်ဆုံး ဥပမာ သို့မဟုတ် အဓိပ္ပါယ်ဖွင့်ဆိုချက် ဆိုတာ မရှိပါဘူး။ သူ့နည်း သူ့ဟန်နဲ့ မှန်ကန်ကြတာပါပဲ။ ဒီအခန်းမှာတော့ အော့ဘ်ဂျက်ဆိုတာ ဘာလဲ၊ ဘယ်လိုမျိုးလဲ ခံစားလို့ရအောင်နဲ့ အခြေခံ အသုံးချတတ်ရုံလောက်ပဲ အဓိကထား လေ့လာကြမှာပါ။ လက်ရှိအခြေအနေနဲ့ သင့်တော်မဲ့ အဓိပ္ပါယ်ဖွင့်ဆိုချက် တချို့ကိုလည်း ဖော်ပြပေးသွားမှာပါ။ သိထားသင့်တဲ့ ဘာသာရပ်ဆိုင်ရာ စကားလုံး အသုံးအနှုန်းတွေကိုလည်း မိတ်ဆက်ပါမယ်။

ဆော့ဖ်ဝဲ အော့ဘ်ဂျက်တွေဟာ အပြင်မှာ တကယ်ရှိတဲ့ အရာတွေရော တကယ်မရှိဘဲ စိတ်ကူးသက်သက်ဖြစ်တဲ့ အိုင်ဒီယာတွေကိုပါ ပရိုဂရမ်ထဲမှာ ထင်ဟပ် ဖော်ပြတယ်။ ဥပမာ ကေသီ့ဘဏ်အကောင့်၊ $\frac{7}{13}$ (အပိုင်းဂဏန်း တစ်ခု)၊ 1948-01-04 (မြန်မာပြည် လွတ်လပ်ရေးရတဲ့နေ့)၊ စန္ဒီပိုင်တဲ့ အနီရောင် တိုယိုတာကား စတာတွေကို အော့ဘ်ဂျက်တွေနဲ့ ဖော်ပြနိုင်တယ်။

အော့ဘ်ဂျက်မှာလည်း တိုက်ပ်သဘောတရား ရှိတယ်။ တိုက်ပ်တူတဲ့ အော့ဘ်ဂျက်အားလုံး ဒေတာဖွဲ့စည်းထားပုံ တူတယ်။ လုပ်ဆောင်လို့ရတဲ့ အော်ပရေးရှင်းတွေလည်း တူပါတယ်။ ဘဏ်အကောင့် အော့ဘ်ဂျက်အားလုံးဟာ လက်ကျန်ငွေနဲ့ အကောင့်နံပါတ် ပါရှိပြီး ငွေသွင်း၊ ငွေထုတ်၊ ငွေလွှဲ အော်ပရေးရှင်းတွေ လုပ်ဆောင်လို့ ရပါမယ်။

အော့ဘ်ဂျက်တွေရဲ့ တိုက်ပ်နဲ့ နီးနီးစပ်စပ် ဆက်နွယ်နေတာကတော့ ကလပ်စ် (*class*) သဘောတရားပါ။ ကလပ်စ်ကို တိုက်ပ်တူအော့ဘ်ဂျက်တွေ ဖန်တီးဖို့အတွက် သတ်မှတ်ထားတဲ့ ပရိုဂရမ်ကုဒ် အစုအဝေးလို့ ယေဘုယျ ပြောနိုင်ပါတယ်။ `Account` ကလပ်စ်၊ `date` ကလပ်စ်၊ `Fraction` ကလပ်စ် စသည်ဖြင့် အော့ဘ်ဂျက် တိုက်ပ် တစ်မျိုးအတွက် ကလပ်စ်တစ်ခု ရှိမှာပါ။ တိုက်ပ်တူ အော့ဘ်ဂျက်တွေ အားလုံးမှာ ပါဝင်မဲ့ အချက်အလက်တွေ၊ အော်ပရေးရှင်းတွေနဲ့ အော့ဘ်ဂျက် ဖန်တီးယူတဲ့ ဖန်ရှင်တွေကို ကလပ်စ်တစ်ခုနဲ့ သတ်မှတ်ရတာပါ။ ကလပ်စ်ကနေ အော့ဘ်ဂျက်တွေ (တိုက်ပ် တူပါမယ်) ထုတ်ယူရတာ ဖြစ်တဲ့အတွက် ကလပ်စ်ကို အော့ဘ်ဂျက် စက်ရုံလို့လည်း ဆိုနိုင်ပါတယ်။ ပုံစံတူ အော့ဘ်ဂျက်တွေ ထုတ်ပေးတာမို့လို့ ကလပ်စ်ဆိုတာ အော့ဘ်ဂျက်တည်ဆောက်တဲ့ blueprint သို့မဟုတ် template ပဲလို့ ယူဆတာဟာလည်း သဘာဝကျတယ် ဆိုရမှာပါ။

အော့ဘ်ဂျက်တွေကို အက်ဘ်စရက်ရှင်း (*abstraction*) အနေနဲ့လည်း ရှုမြင်နိုင်တယ်။ ဘယ်လိုဖန်တီး တည်ဆောက်ထားလဲ မသိဘဲ အသုံးပြုလို့ရတဲ့ အရာအားလုံးကို အက်ဘ်စရက်ရှင်းလို့ ဆိုနိုင်တယ်။ အပြင်မှာသုံးကြတဲ့ ကား၊ တီဗွီ၊ ကွန်ပျူတာ စတာတွေဟာ အက်ဘ်စရက်ရှင်းတွေ ဖြစ်တယ်။ ဖန်ရှင်တွေဟာလည်း အက်ဘ်စရက်ရှင်းတွေပါပဲ။ အော့ဘ်ဂျက်တွေကတော့ ဒေတာနဲ့ အော်ပရေးရှင်း တွဲဖက်ပေါင်းစပ်ထားတဲ့ အက်ဘ်စရက်ရှင်းတွေပါ။ အော့ဘ်ဂျက်တစ်ခုနဲ့ တွဲဆက်ထားတဲ့ အော်ပရေးရှင်းတွေဟာ

အဲ့ဒီအော့ဘ်ဂျက်ရဲ့ ဒေတာတွေကို အသုံးပြုတယ်။ အဲ့ဒီအော့ဘ်ဂျက်ရဲ့ ဒေတာအပေါ် သက်ရောက်မှု ရှိနိုင်တယ်။ အခြားအော့ဘ်ဂျက်ရဲ့ ဒေတာကို မသုံးဘူး။ သက်ရောက်မှုလည်း မရှိစေဘူး။ အော့ဘ်ဂျက် အတွင်းပိုင်း ဒေတာတွေ ဖွဲ့စည်းထားပုံနဲ့ တိုက်ပ်ကို မသိဘဲ အော့ဘ်ဂျက်ကို အသုံးပြုလို့ရတယ်။ အော်ပရေးရှင်းတွေဟာ တကယ်ကတော့ အော့ဘ်ဂျက်ဒေတာ အသုံးပြုတဲ့ ဖန်ရှင်တွေပါပဲ။ ဒီဖန်ရှင်တွေ ဘယ်လိုရေးထားလဲ၊ ဒေတာကို ဘယ်ပုံဘယ်နည်း အသုံးပြုတာလဲ သိစရာမလိုဘဲ အသုံးပြုလို့ ရပါတယ်။

ဖန်ရှင် အသင့်ရှိပြီးသား ဆိုရင် အသုံးပြုလို့ ရသလို ကလပ်စ် အသင့်ရှိပြီးသား ဆိုရင် အော့ဘ်ဂျက်တွေ ဖန်တီးအသုံးပြုနိုင်ပါတယ်။ ဖန်ရှင်ရေးရတာ ခက်ခဲနိုင်ပါတယ်။ ရှိပြီးသား ဖန်ရှင်သုံးတာကတော့ မခက်ပါဘူး။ ဒီသဘောပါပဲ။ ကလပ်စ်သတ်မှတ်ရတာ၊ ဒီဇိုင်းလုပ်ရတာ ရှုပ်ထွေး ခက်ခဲနိုင်ပါတယ်။ ရှိပြီးသား ကလပ်စ်ကနေ အော့ဘ်ဂျက် ဖန်တီးအသုံးပြုရတာ မခက်ပါဘူး။ အသုံးပြုသူ လွယ်ကူအဆင်ပြေစေဖို့ တည်ဆောက်သူက အဓိကစဉ်းစား ဖြေရှင်းရတာပါ။ သုံးစွဲသူအဆင့်ကနေ စတင်ပြီး တည်ဆောက်သူ ပရိုဂရမ်မာ ဖြစ်လာအောင် တစ်ဆင့်ချင်း တက်လှမ်းဖို့ဟာ အဓိကပန်းတိုင်ပါ။ အော့ဘ်ဂျက်မိတ်ဆက် ခဏရပ်၊ အခုပဲ လက်တွေ့စမ်းသပ် ကြည့်လိုက်ရအောင် ...

၂.၁ date, time and datetime

ဆော့ဖ်ဝဲ အပ်ပလီကေးရှင်းတွေမှာ အချိန်နာရီ၊ နေ့ရက်တွေနဲ့ တွက်ချက်ဆုံးဖြတ်ရတာတွေ အမြဲလိုလိုပါတယ်။ ဒါကြောင့် အချိန်နဲ့သက်ဆိုင်တဲ့ အချက်အလက်တွေကို စနစ်တကျ ကိုင်တွယ်ဖြေရှင်းတတ်ဖို့ လေ့လာထားရပါမယ်။ အချိန်နဲ့ နေ့ရက်အတွက် `date`, `time`, `datetime` ကလပ်စ်တွေ ထောက်ပံ့ပေးထားတဲ့ `datetime` လိုက်ဘရီ အသုံးပြုပါမယ်။ `date` ကလပ်စ်ကနေ ဖန်တီးယူတဲ့ အော့ဘ်ဂျက်တစ်ခုဟာ အနောက်တိုင်းပြက္ခဒိန် နေ့ရက်တစ်ရက်ကို ဖော်ပြတယ်။ ခုနှစ်၊ လ၊ ရက် အချက်အလက် သုံးခုပါဝင်တယ်။ မြန်မာပြည် လွတ်လပ်ရေးရခဲ့တဲ့ နေ့ရက်ကို ဖော်ပြရင် အခုလိုပါ

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

ဒုတိယလိုင်းက အော့ဘ်ဂျက် ဖန်တီးတာပါ။ ဖန်ရှင်ခေါ်တာနဲ့ ပုံစံတူတာ တွေ့ရတယ်။ အော့ဘ်ဂျက်ဖန်တီးဖို့ စပယ်ရှယ် ဖန်ရှင်တစ်ခု ခေါ်ထားတယ်လို့ ယူဆနိုင်ပါတယ် (နောက်ပိုင်း ကလပ်စ်အခန်းမှာ အသေးစိတ်လေ့လာရမှာပါ)။ ကလပ်စ်ကနေ အော့ဘ်ဂျက် ဖန်တီးယူတာကို *instantiation* လို့ခေါ်ပြီး ရရှိလာတဲ့ အော့ဘ်ဂျက်ကို အဲဒီကလပ်စ်ရဲ့ *instance* လို့လည်း ခေါ်ပါတယ်။

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

အော့ဘ်ဂျက်ကို ဗေရီရေဘဲလ်နဲ့ အဆိုင်းမန့်လုပ်တာပါ။ အော့ဘ်ဂျက်ကို `mmid` ဗေရီရေဘဲလ်နဲ့ ရည်ညွှန်းအသုံးပြုလို့ ရမှာဖြစ်တယ်။

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

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

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

အော့ဘ်ဂျက်တစ်ခုမှာ ပါဝင်တဲ့ ဒေတာကို *attribute* လို့ ခေါ်တယ်။ Attribute တွေဟာ အော့ဘ်ဂျက်ရဲ့ လက်ရှိအခြေအနေ (*state*) ကိုဖော်ပြတယ်။ စတိတ် ပြောင်းလဲနိုင်တဲ့ အော့ဘ်ဂျက်တွေကို *mutable object* လို့ ခေါ်တယ်။ စတိတ် မပြောင်းလဲနိုင်တဲ့ အော့ဘ်ဂျက်တွေကို *immutable object* လို့ ခေါ်တယ်။ `date` အော့ဘ်ဂျက်တွေဟာ immutable object တွေပါ။ ဆိုလိုတာက attribute တွေဖြစ်တဲ့ `year`, `month`, `day` တန်ဖိုးတွေ မပြောင်းလဲနိုင်ပါဘူး။

ခုနှစ်၊ လ၊ ရက် အတွက် attribute သုံးခုဟာ `date` အော့ဘ်ဂျက် တစ်ခုစီတိုင်းအတွက် ကိုယ်ပိုင်ပါရှိမှာပါ။ ဆိုလိုတာက အောက်ပါ `usid` အော့ဘ်ဂျက်ရဲ့ attribute တွေနဲ့ ခုနက `mmid` ရဲ့ attribute တွေဟာ သီးခြားစီပဲ။ နံမည်တူပေမဲ့ တစ်ခုနဲ့တစ်ခု မရောယှက်ဘူး။

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

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

အခုလို တန်ဖိုးပြန်ယူကြည့်ရင်လည်း ဖြစ်သင့်တဲ့အတိုင်း သက်ဆိုင်ရာ အော့ဘ်ဂျက်ရဲ့ attribute တန်ဖိုးတွေပဲ ပြန်ရတာပေါ့။ လွတ်လပ်ရေး ရခဲ့တဲ့နေ့က ဘာနေ့ဖြစ်မလဲ

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

ဖန်ရှင်တစ်ခုတည်းကို မတူညီတဲ့ အော့ဘ်ဂျက်နှစ်ခုအပေါ်မှာ အသုံးချတာ ဖြစ်တယ်။ ဒေါ့ထ်ကိုပဲ သုံးတယ်။ ပထမတစ်ခုက `usid` အော့ဘ်ဂျက်၊ နောက်တစ်ခုက `mmid` အော့ဘ်ဂျက်အပေါ်မှာ သုံးထားတာပါ။ အမေရိကန် လွတ်လပ်ရေးရခဲ့တာ ကြာသာပတေးနေ့၊ မြန်မာကတော့ တနင်္ဂနွေနေ့ပါ (တနင်္လာက တစ်၊ တနင်္ဂနွေက ခုနှစ်ပါ)။ `isoweekday` ဖန်ရှင်ကို အော့ဘ်ဂျက်တစ်ခုအပေါ် အသုံးပြုတဲ့အခါ ၎င်းအော့ဘ်ဂျက်နဲ့ သက်ဆိုင်တဲ့ ဒေတာနဲ့ ဖန်ရှင်က အလုပ်လုပ်သွားတာပါ။ ဒါကြောင့်လည်း attribute မတူတဲ့ အော့ဘ်ဂျက်တွေအပေါ်မှာ အသုံးချတဲ့အခါ မတူညီတဲ့ ရလဒ်တွေ ထွက်လာရတာပေါ့။ 'ဒေတာနဲ့ အော်ပရေးရှင်း တွဲဖက်ထားတယ်' ဆိုတာ ဒီသဘောတရားကို ဆိုလိုတာပါ။ အော့ဘ်ဂျက်ဒေတာနဲ့ တွဲဖက်အလုပ်လုပ်တဲ့ ဖန်ရှင်တွေကို မက်သဒ် (*method*) လို့ ခေါ်တယ်။ နောက် မက်သဒ်တစ်ခုက `isoformat` ပါ။ နေ့ရက်ကို စားသားအဖြစ် `'yyyy-mm-dd'` ဖော့မတ်နဲ့ ပြန်ပေးတယ်။

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

`date` တစ်ခု ဖန်တီးတဲ့အခါ ခုနှစ်၊ လ၊ ရက် နေရာ မှန်ဖို့ အရေးကြီးပါတယ်။ `date(1948,4,1)` လို့ ရေးမိရင် လေးလပိုင်း တစ်ရက်နေ့ ဖြစ်သွားမှာပါ။ ဒါပေမဲ့ Python မှာ အာ့ဂုမန့်တွေကို နံမည်နဲ့ တွဲပြီး ထည့်ပေးလို့ရတယ်။

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

ဒီနည်းနဲ့ ဆိုရင်တော့ year, month, day ကြိုက်သလို အစီအစဉ်နဲ့ ထည့်လို့ရမှာပါ။

replace မက်သဒ် ဘယ်လို အလုပ်လုပ်လဲ ကြည့်ရအောင်

```
>>> usid = date(1776, 7, 4)
>>> usid100 = usid.replace(year=1876)
```

နဂို usid နေ့ရက်ရဲ့ ခုနှစ်ကို 1876 နဲ့ အစားထိုးထားတဲ့ အော့ဘ်ဂျက် အသစ်တစ်ခု ပြန်ရပါတယ်။ နဂို ရက်စွဲက မပြောင်းသွားဘူး (date အော့ဘ်ဂျက် ဟာ immutable ဖြစ်တာ သတိပြုပါ)။

```
>>> usid
datetime.date(1776, 7, 4)
>>> usid100
datetime.date(1876, 7, 4)
```

ခုနှစ်၊ လ၊ ရက် သုံးခုလုံး အစားထိုးချင်ရင်

```
>>> dt1 = date(2000,2,21)
>>> dt2 = dt1.replace(2010,10,10)
>>> dt3 = dt1.replace(day=20,month=12,year=2020)
```

အာ့ဂုမန့် နံမည် မပါရင် ခုနှစ်၊ လ၊ ရက် အစဉ်အတိုင်း ဖြစ်ရပါမယ်။ ရလဒ်တွေ ကြည့်ရင်

```
>>> dt1
datetime.date(2000, 2, 21)
>>> dt2
datetime.date(2010, 10, 10)
>>> dt3
datetime.date(2020, 12, 20)
```

ခုနှစ်၊ လ၊ ရက် တခုခု ချန်ထားကြည့်ပါ

```
>>> dt4 = dt1.replace(2020)
>>> dt5 = dt1.replace(2030,11)
```

ချန်ထားခဲ့တာတွေ နဂိုအတိုင်းရှိပါမယ်

```
>>> dt4
datetime.date(2020, 2, 21)
>>> dt5
datetime.date(2030, 11, 21)
```

ရက်တစ်ခုတည်း အစားထိုးမယ်ဆိုရင် နံမည်နဲ့တွဲတဲ့ နည်းကပဲ အဆင်ပြေပါမယ်

```
>>> dt6 = dt1.replace(day=28)
>>> dt6
datetime.date(2000, 2, 28)
```

နံမည်မပါရင် ခုနှစ်၊ လ၊ ရက် အစဉ်အတိုင်းဖြစ်တာကြောင့် ခုနှစ်ကို အစားထိုးမှာပါ

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

`time` and `datetime`

နေ့ရက်နဲ့ အချိန် တွဲရက်ကို `datetime`, ရက်စွဲမလိုဘဲ အချိန်ပဲဆိုရင် `time` သုံးပါတယ်

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

`timedelta`

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

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

(`...` က အော်တို ထည့်ပေးသွားတာ။ ကိုယ်တိုင် ရိုက်ထည့်စရာမလိုဘူး။ တစ်လိုင်းချင်း Enter ခေါက်သွားရုံပဲ။ အပိတ်ဝိုက်ကွင်းမှာ စတိတ်မန့် ဆုံးတယ်ဆိုတာ အင်တာပရက်တာက နားလည်တယ်။)

အော့ဘ်ဂျက် အသုံးပြုသူအနေနဲ့ အချိန်ကာလ ကြာမြင့်ချိန်ကို `days`, `weeks`, `hours ...` စတာတွေနဲ့ သတ်မှတ်လို့ရတယ်။ ၎င်းတို့ကို ရက်၊ စက္ကန့်၊ မိုက်ခရိုစက္ကန့် ဖွဲ့ပြီး အော့ဘ်ဂျက် အတွင်းပိုင်း `days`, `seconds`, `microseconds` attributes ဒေတာအနေနဲ့ သိမ်းမှာပါ။

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

ဖော်ပြခဲ့ပြီးတဲ့ အော့ဘ်ဂျက်တွေနဲ့ အပေါင်း၊ အနှုတ် အော်ပရေးရှင်းတွေ လုပ်လို့ရပါတယ်။ ဥပမာ တချို့ လေ့လာကြည့်ပါ

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

ဒီလိုစစ်ကြည့်ပါ

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

၂.၂ `list`

List ဆိုတာ အိုက်တမ် item တွေ အတွဲလိုက် စုစည်းထားဖို့ အသုံးပြုတဲ့ စထရက်ချာတစ်မျိုး ဖြစ်တယ်။ Python မှာ `list` အော့ဘ်ဂျက်တွေကို item တွေ အတွဲလိုက် စုစည်းထားဖို့ သုံးတယ်။ ဘာအိုက်တမ်မှ မပါတဲ့ `list` အသစ်တစ်ခု လိုချင်ရင် ဒီလို

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

ဒီ `lsit` ထဲမှာ ပါတွေပါလဲ

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

လေးထောင့်ကွင်းနဲ့ `list` ကိုပြပေးတယ်။ လက်ရှိ `list` ထဲမှာ အိုက်တမ် မရှိသေးဘူး။ အိုက်တမ် တစ်ခုချင်း ထည့်ချင်ရင် `append` မက်သဒ်ရှိတယ်

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

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

ပါဝင်တဲ့ အိုက်တမ်တစ်ခုစီကို ကော်မာခြားပြီး ပြတယ်။ နောက်ထပ် နည်းလမ်းတစ်ခုနဲ့လည်း `list` ဖန်တီးလို့ ရတယ်။ လေးထောင့်ကွင်း သုံးတဲ့နည်းပါ

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

`list` ဟာ mutable ဖြစ်တယ်။ အိုက်တမ် နောက်တစ်ခု ထပ်ထည့်ကြည့်ပါ

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

နဂို အော့ဘ်ဂျက်မှာ အိုက်တမ်တစ်ခု ထပ်တိုးသွားတာ။ အော့ဘ်ဂျက်ရဲ့ စတိတ် ပြောင်းသွားတယ်။ အိုက်တမ်တစ်ခုကို ဖယ်ထုတ်ချင်ရင်

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

`list` အိုက်တမ်တွေရဲ့ တည်နေရာကို index လို့ခေါ်တယ်။ သုညနဲ့ စတယ်။ ဒုတိယက တစ်၊ တတိယက နှစ် စသည်ဖြင့် ဖြစ်မယ်။ အခု odds မှာ အိုက်တမ် လေးခုရှိနေတယ်။ 7 ဖြုတ်ချင်ရင် index နံပါတ် 2 ကို pop လုပ်ရမှာ

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

ဖယ်လိုက်တဲ့ အိုက်တမ်တွေ နဂိုနေရာမှာ ပြန်ထည့်ချင်တယ်ဆိုပါစို့။ `insert` ရှိပါတယ်

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

အိုက်တမ် အစားထိုးတာ၊ index နဲ့ နေရာတစ်ခုက အိုက်တမ်ကို ပြန်ထုတ်ကြည့်တာကို လေးထောင့်ကွင်းနဲ့ ရေးနည်းလည်းရှိတယ်

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

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

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

replace နဲ့ သဘောတရားတူတာကတော့

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

နောက်ဆုံး နေရာ (index နံပါတ် 5) ကို 14 လဲထည့်လိုက်တာ။

ဗေရီရေဘဲလ် နှစ်ခုက အော့ဘ်ဂျက်တစ်ခုတည်းကို ရည်ညွှန်းနေရင် သတိပြုသင့်တဲ့ ထူးခြားချက်တွေ ရှိလာပါတယ်။

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

ဒီလိုဆိုရင် အော့ဘ်ဂျက် တစ်ခုတည်းကို fruits ရော myfav နဲ့ပါ သုံးလို့ရပါမယ်။ fruits နဲ့ အိုက်တမ်တစ်ခု ထပ်ထည့်ကြည့်မယ်

```
>>> fruits.append('orange')
```

myfav ပါ လိုက်ပြောင်းတာကို တွေ့ရမှာပါ

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

Mutable အော့ဘ်ဂျက်တွေမှာ ဒီအချက်ကို သတိပြုဖို့ လိုပါတယ်။ ဗေရီရေဘဲလ် တစ်ခုကနေ အော့ဘ်ဂျက် စတိတ်ကို ပြောင်းလဲတဲ့အခါ အဲ့ဒီအော့ဘ်ဂျက်ကို ရည်ညွှန်းတဲ့ အခြား ဗေရီရေဘဲလ် အားလုံးကလည်း အပြောင်းအလဲကို မြင်ရမှာ ဖြစ်တယ်။ Immutable ဆိုရင်တော့ စတိတ်မပြောင်းနိုင်တာကြောင့် ဒီလိုကိစ္စမျိုး စဉ်းစားစရာ မလိုဘူး။

အခန်း ၃

ကွန်ထရိုးလ် စတိတ်မန့်များ

ကွန်ထရိုးလ် စတိတ်မန့်တွေက ကားရဲလ်မှာ တွေ့တော့ တွေ့ခဲ့ပြီးသားပါ။ ဒါပေမဲ့ ကားရဲလ်ပရိုဂရမ်မင်းအတွက် လိုသလောက် အခြေခံကိုပဲ ကန့်သတ်ဖော်ပြခဲ့တာပါ။ ဒီအခန်းမှာ ပြည့်စုံအောင် အသေးစိတ်ဆက်လက် လေ့လာကြပါမယ်။ လက်တွေ့အသုံးချ ဥပမာတွေ၊ လေ့ကျင့်ခန်းတွေ ဂရုတစိုက် စီစဉ်ပေးထားတယ်။ စတိတ်မန့် အသစ်တချို့လည်း တွေ့ရမယ်။ စာအုပ်တွေမှာ ဖော်ပြတာ သိပ်မတွေ့ရပေမဲ့ ဘီဂင်နာအများစု အခက်အခဲတွေ့ကြတဲ့ နေရာတွေ၊ တိတိကျကျ နားလည်ဖို့ လိုတဲ့ ပွိုင့်တွေကိုလည်း အလေးပေးရှင်းပြထားတယ်။ အထူးခြားဆုံးကတော့ ရုပ်ပုံတွေဆွဲတာနဲ့ အန်နီမေးရှင်း အခြေခံကို Arcade ဂိမ်းလိုက်ဘရီ အသုံးပြုပြီး စတင်မိတ်ဆက်ထားတာပဲ ဖြစ်တယ်။ စာသားတွေချည်းပဲထက် စိတ်လှုပ်ရှားဖို့ ပိုကောင်းမယ်ထင်ပါတယ်။

၃.၁ `if` စတိတ်မန့်

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

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

`amt > 50_000` ဘူလီယန် အိပ်စ်ပရက်ရှင် `true` ဖြစ်မှပဲ `if` ဘလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဒစ်စကောင့် ဘယ်လောက်ရလဲရော ကျသင့်ငွေပါ ပြပေးမယ်ဆိုရင် ဒီလို

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

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

Python ဗားရှင်း 3.6 ကစပြီး f-string (formatted string) ခေါ်တဲ့ string အသစ်တစ်မျိုး ပါလာပါတယ်။ String ရှေ့မှာ `f` နဲ့ စရင် f-string လို့ သတ်မှတ်တယ်။ Single/double quote ရှေ့မှာ `f` ထည့်ပေးရတာပါ။

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

F-string နဲ့ဆိုရင် ဗေရီရေဘဲလ် (သို့) အိပ်စ်ပရက်ရှင် တွေကို တွန့်ကွင်းထဲမှာ ထည့်ရေးလို့ရတယ်။ ၎င်းတို့ကို f-string က တန်ဖိုးရှာပြီး အစားထိုးပေးမှာပါ။ ဒါကြောင့် `{2 + 3}` က `5` ဖြစ်သွားတာပါ။ ရိုးရိုး string နဲ့ဆို အခုလို

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

ရေးနေရမယ်။ F-string နဲ့ဆို ပိုအဆင်ပြေတယ်။ နမူနာတချို့ကို လေ့လာကြည့်ပါ

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

ဒစ်စကောင့်ပေးပြီး ပရိုမိုးရှင်းလုပ်တဲ့အခါ သတ်မှတ်ပမာဏ မပြည့်သေးရင် ဘယ်လောက်ဖိုးထပ်သုံးတာနဲ့ ဒစ်စကောင့် ရမှာဖြစ်ကြောင်းပြောပြီး ဆွဲဆောင်လေ့ရှိတယ်။ ဒစ်စကောင့်ရအောင် ဘယ်လောက်ထပ်သုံးရမလဲ ပရိုဂရမ်က ပြပေးချင်တယ် ဆိုပါစို့။ `if...else` သုံးနိုင်ပါတယ်။

```
from math import *

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

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

`amt >= 50_000` မှန်ရင် `if` ဘလောက်၊ မှားရင် `else` ဘလောက် လုပ်ဆောင်မှာဖြစ်တယ်။

`if` နဲ့ `if...else` ယေဘုယျပုံစံကို ကြည့်ရင် အခုလိုရှိပါတယ်

> **if** *test*:
> $statement_1$
> $statement_2$
> $statement_3$
>   ...etc.

> **if** *test*:
> $statement_{1a}$
> $statement_{2a}$
> $statement_{3a}$
>   ...etc.
> **else**:
> $statement_{1b}$
> $statement_{2b}$
> $statement_{3b}$
>   ...etc.

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

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

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

တေဘဲလ် ၃.၁ Score and Grading

ကျောင်းသူ/သား နံမည်နဲ့ ရမှတ်ကို ထည့်ပေးရင် ပရိုဂရမ်က အခုလို ပြပေးရပါမယ်။

> Student name: *Amy*
> Score: *95*
> *Amy* get grade *A*

ဒီပရိုဂရမ် အတွက် cascading `if` သုံးထားတာ ကြည့်ပါ

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

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

print(f'{stu_name} get grade {grade}')
```

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

နောက်ဆုံး else အပိုင်းက မပါလို့လည်းရတယ်။ အပေါ်မှာ တစ်ခုမှ True မဖြစ်တော့မှ else ဘလောက်ကို လုပ်ဆောင်တယ်။ နောက်ဆုံးမှာ else အပိုင်းမပါဘူး၊ အပေါ်မှာလည်း ဘယ်တစ်ခုကမှ True မဖြစ်ဘူး ဆိုရင်တော့ လုပ်ဆောင်ပေးစရာ ဘလောက်လည်း မရှိဘူးပေါ့။ ဒီတော့ အားလုံး False ဖြစ်ခဲ့ရင် လုပ်ချင်တဲ့ကိစ္စ ရှိ/မရှိ အပေါ်မူတည်ပြီး else အပိုင်း လို/မလို ဆုံးဖြတ်ရတယ်။

အခု grading ပရိုဂရမ်မှာ ရမှတ်ဟာ သုညနဲ့ တစ်ရာကြား ဖြစ်သင့်တယ်။ အကယ်၍ ထည့်ပေးတာမှားရင် မှားတယ်လို့ ပြပေးချင်တယ်။ ဥပမာ

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

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

အခုပရိုဂရမ်မှာ သိပ်စိတ်တိုင်းကျစရာ မကောင်းတဲ့ ပြဿနာတစ်ခုတွေ့ရပါတယ်။ အောက်ပါအတိုင်း စမ်းကြည့်ရင် Sandy က grade F ရတယ်လို့ ပြနေပါတယ်

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

အမှတ်ထည့်ပေးတာ မှားနေရင် grade ကို မပြပေးသင့်ပါဘူး။ ဒီလို ပြင်ရေးလိုက်မယ် ဆိုရင်

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

if 0 <= score <= 100:
    grade = 'F'
    if 90 <= score <= 100:
        grade = 'A'
    elif 80 <= score <= 89:
        grade = 'B'
    elif 70 <= score <= 79:
        grade = 'C'
    elif 60 <= score <= 69:
        grade = 'D'
    elif 0 <= score <= 59:
        grade = 'F'

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

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

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

```python
stu_name = input("Student name: ")
score = int(input("Score: "))
if 0 <= score <= 100:
    grade = 'F'
    if score >= 90:
        grade = 'A'
    elif score >= 80:
        grade = 'B'
    elif score >= 70:
        grade = 'C'
    elif score >= 60:
        grade = 'D'
    else:
        grade = 'F'
    print(f'{stu_name} get grade {grade}.')
else:
```

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

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

Cascading if မသုံးဘဲ if...else တွေနဲ့လည်း ရေးလို့တော့ ရပါတယ်။ Nesting လုပ်တာတွေ အရမ်းများပြီး ဖတ်ရမလွယ်ကူတာ တွေ့ရမှာပါ။ Grading ပရိုဂရမ်ကို cascading if မသုံးဘဲ ရေးထားတာပါ

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

၃.၂ for Loop

Python for loop ဟာ အဆင့်မြင့် အက်ဘ်စရက်ရှင်း တစ်ခု ဖြစ်ပါတယ်။ စတိတ်မန့်တစ်စုကို သတ်မှတ်ထားတဲ့ အကြိမ်အရေအတွက် ပြည့်အောင် ထပ်ခါထပ်ခါ လုပ်ဆောင်ဖို့ လိုတဲ့အခါ for loop ကို အသုံးပြုတယ်။ Loop ကို စတင် လုပ်ဆောင်တဲ့အချိန်မှာ ဘယ်နှစ်ကြိမ် ပြန်ကျော့မလဲ အတိအကျကြိုသိရင် *definite loop* လို့ သတ်မှတ်တယ်။ for loop ဟာ definite loop ဖြစ်ပါတယ်။ 'အဆင့်မြင့် အက်ဘ်စရက်ရှင်း' လို့ ပြောရတာက list, dictionary, set, range စတဲ့ စထရက်ချာ အမျိုးမျိုးနဲ့ အသုံးပြုလို့ရတဲ့ အတွက်ကြောင့်ပါ။

for loop နဲ့ list ထဲက အိုက်တမ်တစ်ခုချင်း ထုတ်ယူအသုံးပြုနိုင်ပါတယ် . . .

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

Output:

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

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

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

Output:

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

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

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

Output:

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

List နှစ်ခုရဲ့ cartesian product ပါ (ဖြစ်နိုင်တဲ့ အတွဲအားလုံး ရှာတာပါ)

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

Output:

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

Dictionary နဲ့လည်း သုံးလို့ရတာပေါ့

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

Output:

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

ဖော်ပြခဲ့တဲ့ ဥပမာတွေကို ကြည့်ခြင်းအားဖြင့် Python `for` loop ဟာ list, dictionary, string စတဲ့ စထရက်ချာအမျိုးမျိုးနဲ့ အလုပ်လုပ်နိုင်တာ တွေ့ရမှာပါ။ တစ်ခါကျော့တိုင်း အိုက်တမ်တစ်ခုကို loop ဗေရီရေဘဲလ်ထဲမှာ ထည့်ပေးထားတယ်။ အိုက်တမ်တွေအားလုံး ပြီးတဲ့အခါ `for` loop ရပ်သွားမှာ ဖြစ်တယ်။ ပြန်ကျော့တဲ့ အကြိမ်အရေအတွက်ဟာ အိုက်တမ်အရေအတွက်ပဲ ဖြစ်တယ်။

`range` ဖန်ရှင်နှင့် `for` loop

သုညကနေ တစ်ဆယ်ထိ အစဉ်အတိုင်း ရေတွက်ချင်ရင် နည်းလမ်းတစ်ခုက

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

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

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

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

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

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

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

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

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

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

for loop အသုံးချ ဥပမာများ

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

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

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

ပရိုဂရမ် run တဲ့ အချိန်ကျတော့မှ အကြိမ်အရေအတွက် သတ်မှတ်လို့လည်း ရတယ်။ ဥပမာ

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

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

Loop တစ်ကျော့ပြီး တစ်ကျော့ ဗေရီရေဘဲလ် တန်ဖိုးတွေ ဘယ်လောက်ဖြစ်နေလဲ အခုလို လိုက်ကြည့်နိုင်ပါတယ်။

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

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

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

3rd iter:
i = 2
```

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

4th iter:

```
i = 3
val = float(input("? "))    # 11 ထည့်တယ် ယူဆပါ
tot += val                  # 20 (လက်ရှိ tot တန်ဖိုး)

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

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

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

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

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

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

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

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

list နဲ့ ဥပမာ ပြထားတာပါ။

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

```
grand_tot = 0

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

        # အတွင်း for loop
        1st iter:
        val = 1
        row_tot += val          # 1

        2nd iter:
        val = 3
        row_tot += val          # 4

        3rd iter:
        val = 5
        row_tot += val          # 9

    print('Row total: ' + str(row_tot))
    grand_tot += row_total   # 9
    # အပြင်ဘလောက် တစ်ကျော့ပြီး row_tot စုကုပ် အဆုံး

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

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

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

        3rd iter:
        val = 6
```

```
        row_tot += val          # 12

    print('Row total: ' + str(row_tot))
    grand_tot += row_total  # 21
    # အပြင်ဘလောက် နောက်တစ်ကျော့ပြီး row_tot စကုပ် အဆုံး

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

ခရစ်စမတ် သစ်ပင်လေး ဆွဲကြည့်ရအောင်။ တစ်တန်းမှာ စပေ့စ်ဘယ်နှစ်ခုပါလဲ ကြည့်ရလွယ်အောင် □ လေးတွေနဲ့ ပြထားတယ်။

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

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

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

```
□□□□□□□*
□□□□□□***
□□□□□*****
□□□□*******
□□□*********
□□***********
□*************
***************
□□□□□□***
□□□□□□***
□□□□□□***
```

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

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

၃.၃ Python Arcade Library

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

```
pip install arcade
```

Arcade နဲ့ ပုံဆွဲဖို့အတွက် ဂရပ်ဖစ် ဝင်းဒိုးတစ်ခုကို အောက်ပါအတိုင်း ယူရပါတယ်။ Run လိုက်ရင် ပုံ (၃.၁) မှာ တွေ့ရတဲ့ နောက်ခံရောင်နဲ့ ဝင်းဒိုးအလွတ် တစ်ခု တက်လာမှာပါ။

```
# File: arcade_starter.py
import arcade

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

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

# Get ready to draw
arcade.start_render()

# Finish drawing
arcade.finish_render()

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

အပေါ်ဆုံးမှာ arcade လိုက်ဘရီ အင်ပို့လုပ်ထားတာပါ။ ရှေ့ပိုင်းမှာသုံးတဲ့ from *lib* import * ပုံစံ နဲ့ ကွာခြားတာက အခုနည်းနဲ့ အင်ပို့လုပ်ထားရင် လိုက်ဘရီမှ ပါတဲ့ အစိတ်အပိုင်းတွေကို ဒေါ့ထ် အမှတ်အသားနဲ့ အသုံးပြုရပါမယ်။ ဥပမာ open_window ဖန်ရှင်ကို

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

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

ပုံ ၃.၁

`set_viewport` ဖန်ရှင်က ဝင်းဒိုးနေရာယူထားတဲ့ စခရင်ဧရိယာမှာ ကိုသြဒိနိတ်စစ်စတမ် သတ်မှတ်ပေးတာပါ။ Origin အမှတ်နဲ့ x, y ဒါရိုက်ရှင် သတ်မှတ်ပေးတာပါ။

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

ဝင်းဒိုး ဘယ်ဘက်စွန်းကို $x = 0$, ညာဘက်စွန်းကို $x = 300$ သတ်မှတ်ထားတယ်။ အပေါ်ဘက်စွန်း (တိုက်တယ်လ်ဘားမပါ) ကို $y = 0$, အောက်ဘက်စွန်းကို $y = 200$ သတ်မှတ်ထားပါတယ်။ ဝင်းဒိုးရဲ့ ဘယ်ဘက်အပေါ်ထောင့်စွန်းက origin အမှတ် $(0, 0)$ ဖြစ်တယ်။ ဘယ်ဘက်ကို သွားရင် x တန်ဖိုး တိုးသွားပြီး အောက်ဘက်ကို ဆင်းရင် y တန်ဖိုး တိုးသွားမှာဖြစ်တယ်။ အခုလို ကိုယ်တိုင် မသတ်မှတ်ပေးဘဲ သူ့နဂိုအရှိအတိုင်းဆိုရင် 'အောက်ခြေ' ဘယ်ဘက်စွန်းက origin ဖြစ်နေမှာပါ။ အောက်ခြေကနေ အပေါ်ကိုတက်သွားရင် y တန်ဖိုး များလာမှာပါ။ သူ့နဂိုအတိုင်းက `set_viewport` ကို အခုလိုခေါ်ထားတာဖြစ်မယ်။ `top=200`, `bottom=0` ပါ။

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

အသုံးများတဲ့ ဂိမ်းလိုက်ဘရီတွေမှာ ဒီစနစ်ကို သုံးကြတာ မတွေ့ရဘူး။ ဒီစနစ်နဲ့ အကျင့်ဖြစ်သွားရင် အခြားလိုက်ဘရီတွေ လေ့လာတဲ့အခါ အခက်အခဲရှိနိုင်တယ်။ အခုလေ့လာကြမဲ့ ဥပမာတွေ အတွက်လည်း အပေါ်မှာပြောတဲ့နည်းက ပိုအဆင်ပြေတယ်။ (Arcade ကိုပဲ တစိုက်မတ်မတ်သုံးမယ်၊ လေ့လာမယ်ဆိုရင်တော့ သူ့နဂိုအတိုင်းသုံးတာ အကောင်းဆုံးဖြစ်မှာပါ)။

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

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

အခုလို အင်ပို့လုပ်ထားရင် အရောင်တွေ သုံးရတာ ပိုအဆင်ပြေတယ်

```
import arcade
from arcade.color import *
...
arcade.set_background_color(PINK_PEARL)
```

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

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

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

```
arcade.run()
```

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

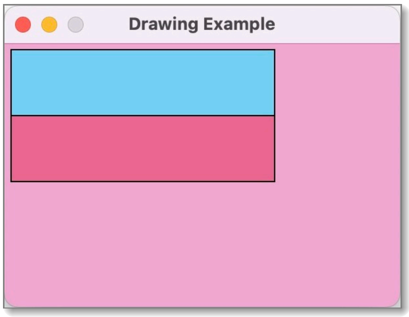
Arcade မှာ ပါတဲ့ အခြေခံ ပုံဆွဲဖန်ရှင် တချို့ကို ဆက်ကြည့်ရအောင်။ ထောင့်မှန်စတုဂံဆွဲတဲ့ ဖန်ရှင်တွေထဲက နှစ်ခု သုံးပြထားတယ်။ နှစ်ခုလုံးက ထောင့်မှန်စတုဂံရဲ့ ဘယ်ဘက်အပေါ်ထောင့်စွန်းနဲ့ တည်နေရာကို သတ်မှတ်ပြီး အရွယ်အစားကို အကျယ်၊ အမြင့်နဲ့ သတ်မှတ်ပေးရတာပါ။ draw_xywh_rectangle_filled က အတွင်းပိုင်း အရောင်နဲ့ ဆွဲပေးတယ်။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

```
import arcade
from arcade.color import *

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

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

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

အပေါ် ထောင့်မှန်စတုဂံပုံကို ဒီနှစ်ခုနဲ့

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

ဆွဲထားတာပါ [ပုံ (၃.၂)]။ ပါရာမီတာတွေက $x, y, width, height, color$ အစဉ်အတိုင်းပဲ။ ဘယ်ဘက်ဘောင်ကနေ $x = 5$ ယူနစ် ခွာထားတယ်။ အပေါ်ဘောင်ကနေလည်း $y = 5$ ယူနစ်ခွာထားတယ်။ သုညထားပြီး စမ်းကြည့်ပါ။ အခြားတန်ဖိုးတွေ ထည့်ပြီး စမ်းကြည့်ပါ။ ပိုပြီးသဘောပေါက် လာပါလိမ့်မယ်။ ဒုတိယ ထောင့်မှန်စတုဂံက ဒီနှစ်ခုနဲ့



ပုံ ၃.၂

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

ဆွဲထားတာပါ။ ဘယ်ဘက်ကို ခွာထားတာက အပေါ်ပုံနဲ့ တူတူပဲ ($x = 5$)။ အပေါ်ပုံရဲ့ အောက်ခြေအနားနဲ့ ကပ်နေအောင် $y = 5 + 50(height) = 55$ ထားရပါမယ်။

> သူ့နဂိုအတိုင်းဆိုရင် Arcade ကိုဩဒိနိတ်စနစ်မှာ origin က အောက်ခြေဘယ်ဘက်စွန်းလို့ ပြောခဲ့ပါတယ်။ `set_viewport` နဲ့ အပေါ်ဘယ်ဘက်စွန်းကို origin အဖြစ်ပြောင်းလဲ သတ်မှတ်ထားတယ်။ ဒီအတွက်ကြောင့် အထက်အောက် ပြောင်းပြန်ဖြစ်သွားပါတယ်။ ဖန်ရှင် documentation မှာကြည့်ရင် အခုလိုတွေ့ရမှာပါ (VS Code/PyCharm မှာ မောက်စ်ပွိုင်တာကို ဖန်ရှင်နံမည်ပေါ်မှာထားရင် documentation ပြပေးပါလိမ့်မယ်)
>
> ```
> def draw_xywh_rectangle_filled(bottom_left_x: float,
>                                bottom_left_y: float,
>                                width: float,
>                                height: float,
>                                color: tuple[int, int, int]
>                                    | list[int]
>                                    | tuple[int, int, int, int]) -> None
> ```
>
> Documentation မှာ bottom ဆိုရင် ကိုယ့်အတွက် top လို့ ပြောင်းပြန် စဉ်းစားရမယ်။ သူ့နဂိုစနစ်ကို ပြောင်းလိုက်လို့ ဒီအခက်အခဲ ကြုံရတာပါ။ ဒါကြောင့် Arcade ကိုပဲ အဓိကသုံးမယ်ဆိုရင် သူ့အရှိအတိုင်းသုံးတာ အကောင်းဆုံးပဲ။ အထက်အောက် ပြောင်းပြန် စဉ်းစားစရာ မလိုတော့ဘူး။

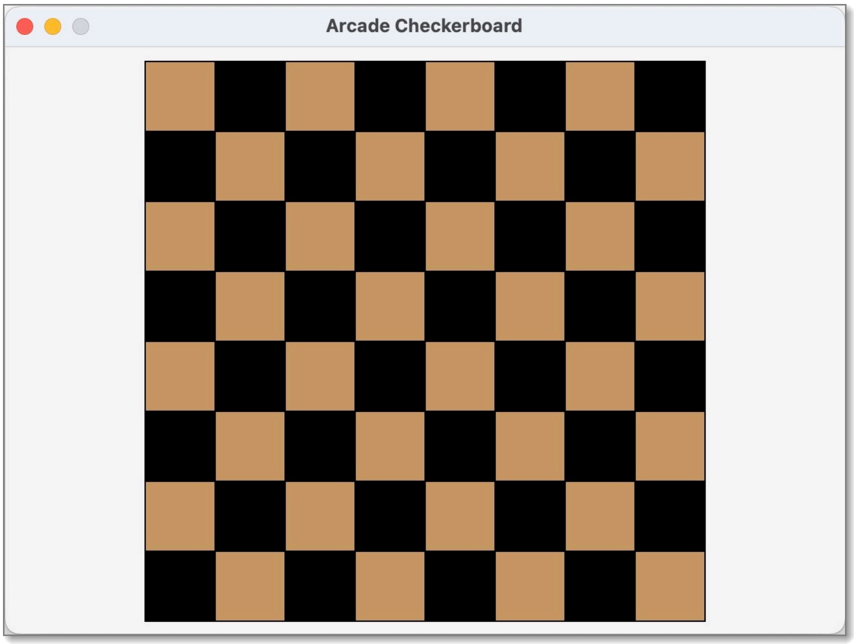
```
import arcade
from arcade.color import *

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

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

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

for i in range(ROWS):
    for j in range(COLS):
        x = X_LFT + SQ_SIZE * i
        y = Y_TOP + SQ_SIZE * j
        if (i + j) % 2 == 0:
            arcade.draw_xywh_rectangle_filled(x,
                                              y,
                                              SQ_SIZE,
                                              SQ_SIZE,
                                              WOOD_BROWN)
        else:
            arcade.draw_xywh_rectangle_filled(x,
                                              y,
                                              SQ_SIZE,
                                              SQ_SIZE,
                                              BLACK)
        arcade.draw_xywh_rectangle_outline(x,
                                           y,
                                           SQ_SIZE,
                                           SQ_SIZE,
                                           BLACK)

arcade.finish_render()
arcade.run()
```

၃.၄ while loop

while loop ဟာ *indefinite loop* ဖြစ်ပါတယ်။ Loop ကို စတင် လုပ်ဆောင်တဲ့အချိန်မှာ ဘယ်နှစ်ကြိမ် ပြန်ကျော့မလဲ အတိအကျ ကြို 'မသိ' ရင် indefinite loop လို့ သတ်မှတ်တယ်။ တစ်နဲ့ တစ်ဆယ်ကြား (အပါအဝင်) ကိန်းပြည့်တစ်လုံးကို ကျပန်း (random) ထုတ်ထားမယ်။ မမှန်မချင်း အဲဒီဂဏန်းကို ခန့်မှန်းပေးရမယ်။ ခန့်မှန်းတာ မှန်သွားရင် ဘယ်နှစ်ခါ ခန့်မှန်းရလဲနဲ့ မှန်တဲ့ဂဏန်းကို ပြပေးတဲ့ ပရိုဂရမ်မှာ while loop အသုံးပြု ရေးထားတာ တွေ့ရပါမယ်။

```
from random import *
```



ပုံ ၃.၃

```
num = randint(1, 10)

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

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

`guess != num` (ဘူလီယန် အိပ်စ်ပရက်ရှင်) မမှားမချင်း loop က ပြန်ကျော့ပေးနေမှာပါ။ မှားတဲ့အခါ ထပ်မကျော့တော့ဘဲ ရပ်သွားမှာဖြစ်တယ်။ ကျပန်းဂဏန်းက ခုနစ်ကျတယ် ဆိုပါစို့။ တစ်၊ ကိုး၊ သုံး၊ တစ်ဆယ်နဲ့ ခုနစ် တို့ကို အစဉ်အတိုင်း ခန့်မှန်း ထည့်ပေးမယ်ဆိုရင် တစ်ကျော့ချင်း လိုက်ကြည့်တဲ့အခါ အခုလို တွေ့ရမှာပါ။

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

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

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

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

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

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

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

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

Sentinel Controlled Loop

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

Sentinel controlled loop မှာ loop ကနေ ထွက်ဖို့အတွက် အသုံးပြုတဲ့ သီးသန့်တန်ဖိုးတစ်ခု ရှိရပါမယ်။ ဒီတန်ဖိုးကို *sentinel value* လို့ ခေါ်တယ်။ ဂဏန်းတွေပေါင်းတဲ့ ကိစ္စအတွက် *sentinel value* ရွေးချယ်သတ်မှတ်တဲ့အခါ ပေါင်းရမဲ့ဂဏန်း မဖြစ်နိုင်တဲ့ တစ်ခုကို ရွေးရပါမယ်။ သုညဟာ အပေါင်းထပ်တူရ ဂုဏ်သတ္တိရှိတဲ့အတွက် ၎င်းကို *sentinel value* အဖြစ် ထားနိုင်တယ်။

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

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

print(f'Total is {total}')
```

ထည့်ပေးတဲ့ ဂဏန်းဟာ သုညမဟုတ်မချင်း `total` မှာ ပေါင်းထည့်ပြီး နောက်ဂဏန်းတစ်ခု ထည့်ပေးဖို့ input တောင်းမှာပါ။ Sentinel တန်ဖိုး သုည ထည့်လိုက်ရင် loop က ထွက်သွားမယ်။ ပြီးတဲ့အခါ

total ကို ပြပေးမှာပါ။ စစချင်းပဲ သုည ထည့်လိုက်ရင် loop က တစ်ခါမှ မကျော့ဘဲ ထွက်သွားမယ်။ total လည်း သုညပဲ ထွက်ပါမယ်။ အခုပရိုဂရမ်ကို နောက်တစ်နည်း ရေးလို့ရပါတယ်။

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

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

print(f'Total is {total}')
```

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

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

Result Controlled Loop

တစ်နှစ်ငါးရာခိုင်နှုန်း အတိုးနဲ့ ဘဏ်မှာ ဒေါ်လာတစ်ထောင် စုထားတယ်။ တစ်နှစ်ပြည့်တိုင်း အတိုးအရင်းပေါင်းပြီး နောက်နှစ်အတွက် အတိုးတွက်တယ်။ နှစ်ထပ်တိုး (yearly compound interest) လို့ခေါ်ပါတယ်။ ဒီနည်းလမ်းနဲ့ အနည်းဆုံး ဒေါ်လာတစ်သန်း စုမိဖို့ (မီလျံနာတစ်ယောက်ဖြစ်ဖို့) ဘယ်နှစ်နှစ်စောင့်ရမလဲ။ သုံးနှစ်ပြည့်ရင် စုမိမဲ့ ငွေပမာဏ တွက်ထားတာကို ကြည့်ပါ။

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

ဇယား ၃.၂ သုံးနှစ်စာ နှစ်ထပ်တိုး

ဒေါ်လာတစ်သန်း အနည်းဆုံးရဖို့ နှစ်ဘယ်လောက်ကြာမလဲ အခုလို ရှာနိုင်ပါတယ်

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

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

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

`while` loop ထဲမှာ တစ်နှစ်ကုန်တဲ့အခါ ရမဲ့ အတိုးနဲ့ အတိုးအရင်းပေါင်း တွက်ထားပြီး `yr` ကိုလည်း တစ်နှစ် တိုးပေးတယ်။ `yr`, `interest`, `balance` ထုတ်ပြပေးတယ် (မပြလည်း ပြဿနာမရှိပါ။ တစ်နှစ်စာ တွက်ထားတာ စစ်ကြည့်လို့ရအောင် ထုတ်ကြည့်တာပါ)။ တစ်သန်းမပြည့်မချင်း ပြန်ကျော့ပေးအောင် loop ကွန်ဒီရှင်ကို `balance < TARGET` နဲ့ စစ်ထားတယ်။ တစ်သန်းနဲ့ညီရင် (သို့) တစ်သန်းကျော်သွားတာနဲ့ ကွန်ဒီရှင် `False` ဖြစ်သွားပြီး ထပ်မကျော့တော့ဘူး။ Loop တစ်ခါကျော့တိုင်း တစ်နှစ်စာ အတိုးအရင်းပေါင်း `balance` ကို တွက်ချက်ပြီး loop ကနေ ဘယ်အချိန် ထွက်မလဲကလည်း အဲဒီရလဒ်အပေါ် မူတည်နေတာ တွေ့ရပါမယ်။ ဒီလို သဘောတရားရှိတဲ့ loop မျိုးကို result controlled loop လို့ ခေါ်ပါတယ်။ ကျော့မဲ့ အကြိမ်အရေအတွက်က loop ထဲမှာ တွက်ချက်ထားတဲ့ ရလဒ်ပေါ် မူတည်တယ်။

ပရိုဂရမ် output ကို အခုလို ကော်လံတစ်ခုစီကို အကျယ် တစ်သမတ်တည်းဖြစ်၊ စာသားတွေ ညာဘက်ကပ်ပြီး ညီနေအောင် f-strings ကို format spec လို့ခေါ်တဲ့ ဖော့မတ်သတ်မှတ်တဲ့ နည်းစနစ်ကို သုံးထားတာပါ။

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

Format spec နဲ့ ပါတ်သက်ပြီး အကျယ်တဝင့် မဖော်ပြတော့ဘဲ အခုလို ဖော့မတ်ရအောင် ဘယ်လိုလုပ်ထားလဲကိုပဲ ရှင်းပြပါမယ်။

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

ကော်လံ ခေါင်းစည်း အတွက် f-string ပါ။ ဖော့မတ်လုပ်ရမဲ့ တန်ဖိုး/ဗေရီရေဘဲလ်/အိပ်စ်ပရက်ရှင်က `:` (ကော်လံ) ဘယ်ဘက်မှာ ရှိမယ်။ ညာဘက်မှာ format spec ရှိမယ်။ **`>4s`** က `'yr'` အတွက် format spec ဖြစ်တယ်။ **`>`** က ညာဘက်ကပ်ဖို့၊ **`4`** က ကော်လံအကျယ်ကို ကာရက်တာ လေးလုံးသတ်မှတ်တာ။ **`s`** ကတော့ တန်ဖိုးကို string အနေနဲ့ ပြပေးပါလို ဆိုလိုတယ်။ ကျန်တဲ့ ကော်လံနှစ်ခု အတွက် format spec ကို **`>10s`** သတ်မှတ်တယ်။ ကာရက်တာ ဆယ်လုံး ကော်လံအကျယ် သတ်မှတ်တယ်။

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

ဒါကတော့ `yr`, `interest`, `balance` တန်ဖိုးတွေကို ပြပေးဖို့ပါ။ ကော်လံအကျယ် `4`, `10`, `10` သတ်မှတ်တယ်။ `yr` ကို ကိန်းပြည့်ဂဏန်းအနေနဲ့ ပြပေးအောင် `d` သုံးတယ်။ ကျန်တဲ့နှစ်ခုကို ဒဿမနဲ့ ပြဖို့ `f` သုံးတယ်။ `.2` ကတော့ ဒဿမနောက် ဂဏန်းနှစ်လုံး ပြပေးခိုင်းတာ။ ကိန်းဂဏန်းဆိုရင် သူ့နဂိုအတိုင်း ညာဘက်ကပ်ပေးတဲ့အတွက် `>` ထည့်ဖို့ မလိုဘူး။ ဘယ်ဘက်ကပ်ချင်ရင် `<` သုံးလို့ရတယ်။

၃.၅ လေ့ကျင့်ရန် ဥပမာများ

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

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

Internet Delicatessen

Online ကနေ အစားအသောက်တွေ delivery ပို့ပေးဖို့ အမှာလက်ခံတဲ့ ဆိုင်လေးတစ်ဆိုင်အတွက် ပရိုဂရမ်ရေးပေးရပါမယ်။ အော်ဒါမှာတဲ့အခါ မှာမဲ့ အစားအသောက် နံမည်၊ ဈေးနှုန်းနဲ့ အိပ်စ်ပရက်စ် delivery ယူမယူ ပရိုဂရမ်မှာ ထည့်ပေးရပါမယ်။ ပရိုဂရမ်က အော်ဒါနဲ့ စုစုပေါင်းကျသင့်ငွေကို ထုတ်ပေးရပါမယ်။ တစ်သောင်းနဲ့အထက်မှာရင် delivery ဖိုး ပေးစရာမလိုပါ။ တစ်သောင်းအောက်ဆိုရင်တော့ နှစ်ရာပေးရပါမယ်။ အိပ်စ်ပရက်စ် delivery ယူမယ်ဆိုရင် သုံးရာအပို ထပ်ပေးရပါမယ်။

Enter the item: Tuna Salad
Enter the price: 4500
Express delivery (0==no, 1==yes): 1

Invoice:
Tuna Salad   4500
delivery   500
total   5000

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

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

tot_cost = price + tot_deli_fee

print("Invoice: ")
```

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

အိပ်စ်ပရက်စ် delivery ယူ/မယူ တစ် (သို့) သုည ထည့်ပေးရမှာပါ။ တစ်/သုည မဟုတ်တဲ့ ဂဏန်းတစ်ခု မှားထည့်မိရင် if...elif ကွန်ဒီရှင်တွေ တစ်ခုမှ True ဖြစ်မှာ မဟုတ်ပါဘူး။ မှားထည့်ထားတယ်လို့ သတိပေးဖို့ else အပိုင်းမှာ လုပ်ထားတယ်။ မှန်/မမှန် စိစစ်တဲ့အခါ အောက်ပါဇယားမှ ဖြစ်နိုင်ခြေအားလုံး ခြုံငုံမိအောင် စစ်သင့်တယ်။ အိပ်စ်ပရက်စ် delivery အတွက် input ဂဏန်း မှားထည့်ရင် သတိပေးတာကိုလည်း စစ်သင့်တယ်။ တစ်သောင်းဖိုး ဝယ်တာကို သုံးမျိုးစစ်ထားတာ လေ့လာကြည့်ပါ။ တစ်သောင်း မပြည့်တာနဲ့ တစ်သောင်းကျော်ဖိုး အတွက်လည်း အလားတူ စစ်ကြည့်လို့ အားလုံးမှန်တယ်ဆိုရင် ဒီပရိုဂရမ်မှာ bug ပါနိုင်ခြေ လုံးဝမရှိသလောက် ဖြစ်သွားပါပြီ။

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

တေဘဲလ် ၃.၃ သုံးနှစ်စာ နှစ်ထပ်တိုး

Test Output:

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

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

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

```
delivery       0.00
total      10000.00
```

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

ပြီးခဲ့တဲ့ ပရိုဂရမ်မှာ ပြင်မယ်ဆိုရင် 10_000 ကို လေးနေရာ၊ ပုံမှန်ပို့ခ 200 နဲ့ အိပ်စ်ပရက်စ်အပိုကြေး 300 စတာတွေကို နှစ်နေရာစီ လိုက်ပြင်ရပါမယ်။ အလုပ်ရှုပ်တဲ့အပြင် ပြင်ဖို့ကျန်ခဲ့တာလို့ မှားတာလည်း ဖြစ်နိုင်တယ်။ “Find and Replace” လုပ်မှာပေါ့လို့ စောဒကတက်စရာ ရှိပါတယ်။ အခုလို ကုဒ်လိုင်းနည်းနည်းပဲရှိတဲ့ ပရိုဂရမ်အသေးလေးမှာ အဆင်ပြေနိုင်ပေမဲ့ ပိုရှုပ်ထွေးပြီး ကုဒ်လိုင်းတွေများတဲ့ ပရိုဂရမ်မျိုးတွေမှာ “Find and Replace” လုပ်ရင် မပြင်သင့်တာတွေကိုပါ မရည်ရွယ်ပဲ ပြင်မိသွားတာဖြစ်တတ်ပါတယ်။ ပရိုဂရမ်ရေးတဲ့အခါ ကျင့်သုံးရမဲ့ အလေ့အထကောင်းတစ်ခုက ကုဒ်ထဲမှာ ဒီတိုင်းချရေးထားတဲ့ တန်ဖိုးတွေ (literal constants) တွေကို နာမည်ပေးထားတာပါ။ အပေါ်က ပရိုဂရမ်မှာ literal constants တွေချည်း သုံးထားတယ်။ နံမည်ပေးထားတဲ့ constants (named constants) တွေအဖြစ် ပြောင်းရေးသင့်တယ်။

```python
FREE_DELI_AMT = 15_000
DELI_FEE = 350
EXP_DELI_FEE = 400

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

tot_deli_fee = 0

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

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

```
print(f'%-10s %8.2f' % ('delivery', tot_deli_fee))
print(f'%-10s %8.2f' % ('total', tot_cost))
```

ဒီပရိုဂရမ်ကို cascading if မသုံးဘဲ ရိုးရိုး if နဲ့ ရေးလို့လည်း ရတယ်။

```
FREE_DELI_AMT = 15_000
DELI_FEE = 350
EXP_DELI_FEE = 400

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

tot_deli_fee = 0

if price < FREE_DELI_AMT:
    tot_deli_fee += DELI_FEE

if is_exp_deli == 1:
    tot_deli_fee += EXP_DELI_FEE

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

tot_cost = price + tot_deli_fee

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

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

ဒီပရိုဂရမ်နဲ့ ဒီ့မတိုင်ခင် သူ့ရှေ့က ပရိုဂရမ်က ဖြစ်နိုင်ခြေအားလုံးအတွက်တော့ ရလဒ် တူတူမထွက်ပါဘူး။ အခြေအနေ တစ်ခုကလွဲလို့ ကျန်တာတွေအတွက်တော့ ရလဒ်တူပါတယ်။ တစ်သောင်းငါးထောင် ထက်ငယ်တဲ့ တန်ဖိုးနဲ့ အိပ်စ်ပရက်စ် delivery အတွက် ဂဏန်း လွဲထည့်ကြည့်ပါ။ ဥပမာ

```
Item name: salad
Price: 12000
Express delivery (0==no, 1==yes): 2
You may have wrong value for express deli.
Invoice:
salad      12000.00
delivery       0.00
```

```
total       12000.00

Item name: salad
Price: 12000
Express delivery (0==no, 1==yes): 2
You may have wrong value for express deli.
Invoice:
salad       12000.00
delivery      350.00
total       12350.00
```

နောက်ဆုံး ပရိုဂရမ်က အိပ်စ်ပရက်စ် delivery အတွက် မပေါင်းထည့်ပေမဲ့ ပုံမှန် delivery ခ သုံးရာ ငါးဆယ်ကိုတော့ ထည့်ပေါင်းသွားတာ တွေ့ရမယ်။ မှားထည့်တာကိုပဲ သတိပေးစာသားပြပေးတာ၊ ထည့်မတွက်သွားတာက ပိုပြီး သဘာဝကျတယ်လို့ ယူဆရမှာပါ။

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

တာရာ ပရက်ရှာ

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

ရှေ့တာယာနှစ်လုံး ပရက်ရှာအနီးစပ်ဆုံးတူသင့်ပြီး 3 psi အထိ ကွာဟလို့ရတယ်။ ကွာဟချက်က 3 psi ထက်တော့ မများသင့်ဘူး။ နောက်တာယာနှစ်လုံးလည်း ထိုနည်းတူစွာပဲ ဖြစ်တယ်။ ကားမော်ဒယ် အလိုက် recommended pressure ကွာခြားပေမဲ့ အိမ်စီးကားအများစုအတွက် 35 psi ဖြစ်တယ်လို့ ယူဆပြီး တာယာပရက်ရှာ အိုကေမကေ စစ်ပေးတဲ့ ပရိုဂရမ် ရေးပေးရပါမယ်။ Input အနေနဲ့ တာယာ တစ်ခုချင်းအတွက် ပရက်ရှာ psi တန်ဖိုး ထည့်ပေးရမှာပါ။ ထည့်ပေးလိုက်တဲ့ တာယာပရက်ရှာ 31.5 psi ထက်နည်းနေရင် သို့မဟုတ် 44 psi ထက်များနေတာနဲ့ သတ်မှတ်ဘောင် မဝင် (out of range) ဖြစ်နေကြောင်း သတိပေးရပါမယ်။ တာယာအားလုံးရဲ့ ပရက်ရှာတွေ သတ်မှတ်ဘောင်အတွင်း ဝင်တယ်၊ ရှေ့တာယာနှစ်လုံး ကွာဟချက်၊ နောက်တာယာနှစ်လုံး ကွာဟချက်တွေ ခွင့်ပြုလို့ရတာထက် မပိုဘူးဆိုရင် လေထိုးထားတာ အိုကေတယ်။ တာယာတစ်လုံး out of range ဖြစ်နေတာနဲ့ လေထိုးထားတာ မအိုကေဘူး ပြပေးရပါမယ်။

```
MIN_ALLOWABLE = 31.5
MAX_ALLOWABLE = 44.0
WARNING = 'Waring: Pressure is out of range!'
LEFT_RIGHT_DIFF_ALLOWABLE = 3.0

is_out_of_range = False
```

```
front_left = float(input("Front left pressure: "))
if not (MIN_ALLOWABLE <= front_left <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

front_right = float(input("Front right pressure: "))
if not (MIN_ALLOWABLE <= front_right <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

rear_left = float(input("Rear left pressure: "))
if not (MIN_ALLOWABLE <= rear_left <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

rear_right = float(input("Rear right pressure: "))
if not (MIN_ALLOWABLE <= rear_right <= MAX_ALLOWABLE):
    is_out_of_range = True
    print(WARNING)

front_diff = abs(front_left - front_right)
front_diff = abs(rear_left - rear_right)
if (front_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
        and rear_diff <= LEFT_RIGHT_DIFF_ALLOWABLE
        and not is_out_of_range):
    print("Inflation is OK.")
else:
    print("Inflation is not OK!")
```

is_out_of_range ဘူလီယန် သုံးထားတာ နည်းနည်း ရှင်းပြဖို့ လိုပါမယ်။ စစချင်း False တန်ဖိုး ထည့်ထားတာ တွေ့ရမှာပါ။ if စတိတ်မန့်တွေက တာယာတစ်ခုချင်းကို out of range ဖြစ်နေလား စစ် ထားတာတွေ့ရတယ်။ တာယာတစ်ခု out of range ဖြစ်တာနဲ့ is_out_of_range က True ဖြစ်သွား မှာပါ။

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

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

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

အခန်း ၄

ဖန်ရှင်များ

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

ခြုံငုံနားလည်အောင် ပြောရရင် မက်သဒ်ဆိုတာ ကိစ္စတစ်ခုခု လုပ်ဆောင်ပေးဖို့အတွက် နံမည်ပေးထားတဲ့ စတိတ်မန့်တွေပါပဲ။ နံမည်တစ်ခု (အဓိပ္ပါယ်ပေါ်လွင်တဲ့) ဟာ အရေးကြီးပါတယ်။ မှန်မှန်ကန်ကန် ရွေးချယ်ထားတဲ့ နံမည်တစ်ခုဟာ မက်သဒ်ရဲ့ လုပ်ဆောင်ချက်ကို ပေါ်လွင်စေပြီး နားလည်ရလွယ်ကူစေတယ်။ `cleanStreet`, `cleanCorner`, `turnNorth` စတဲ့ ပရိုဂရမ်ရဲ့ ဇာတ်လမ်းနဲ့ ကိုက်ညီမှုရှိတဲ့ နံမည်တွေဟာ ပရိုဂရမ်ကုဒ်ကို ဖတ်ရင် နားလည်ရလွယ်ကူစေတယ်။ တကယ့်လက်တွေ့အသုံးချ ပရိုဂရမ်တွေမှာ ဒီအချက်ဟာ ပိုလို့တောင် အရေးပါတယ်။ ကုမ္ပဏီတစ်ခုအသုံးပြုတဲ့ ပရိုဂရမ်တစ်ခုမှာ အခုလိုမက်သဒ်တွေ ပါကောင်းပါနိုင်ပါတယ်။

၄.၁ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်များ

အခန်း (၅) မှာ ဖော်ပြခဲ့တဲ့ နှစ်ထပ်ကိန်းရှာတဲ့ `square` ဖန်ရှင်ကိုပဲ အသေးစိတ် တစ်ခါထပ်ကြည့်ရအောင်။ ဒီလောက် ရှင်းရှင်းလေးကို အကျယ်ချဲ့နေတယ်လို့ ထင်ကောင်း ထင်ပါလိမ့်မယ်။ နည်းနည်းတော့ စိတ်ရှည်သည်းခံ ပေးရပါမယ်။ အခြေခံကျတဲ့ သဘောတရားတွေ ကျေညက်ထားမှ ရှေ့ဆက်တဲ့အခါ လွယ်ကူမှာ မို့လို့ပါ။

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

ဝိုက်ကွင်းထဲက ဗေရီရေဘဲလ် `x` က ဖန်ရှင် ပါရာမီတာ (*parameter*) ဖြစ်ပြီး ဖန်ရှင်ခေါ်တဲ့အခါ ထည့်ပေးမဲ့ အာဂုမန့် (*argument*) တန်ဖိုးကို ကိုယ်စားပြုတယ်။ `return` စတိတ်မန့်က ဖန်ရှင်ခေါ်တဲ့နေရာကို တန်ဖိုးပြန်ပေးတဲ့ စတိတ်မန့်ပါ။

ဖန်ရှင်အသုံးပြုတာကို *function call* လုပ်တယ်လို့ သိထားပြီးပါပြီ။ မြန်မာလိုတော့ 'ဖန်ရှင်ခေါ်တယ်' သို့မဟုတ် 'ဖန်ရှင်ကောလ်တယ်' လို့ အပြောများတယ်။ ဖန်ရှင်ခေါ်တဲ့ ပုံစံက ဒီလိုပါ

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

အခု ဖန်ရှင်ကောလ် အတွက် ပါရာမီတာ x ရဲ့ တန်ဖိုးက 2.5 ဖြစ်မှာပါ။ (ဖန်ရှင်ခေါ်တဲ့အခါ ပါရာမီတာ ဗေရီရေဘဲလ် x ကို အာ့ဂုမန့်နဲ့ အဆိုင်းမန့်လုပ်ပေးတယ်လို့ ယူဆနိုင်တယ်။ ဒီကိစ္စအတွက် အာ့ဂုမန့်က 2.5 ဖြစ်တယ်)။ အားလုံးသိပြီး ဖြစ်တဲ့အတိုင်း ဖန်ရှင်ခေါ်ရင် ဖန်ရှင်ဘလောက်ကို လုပ်ဆောင်ပေးမှာပါ။ ဖန်ရှင်ဘလောက်ထဲက return စတိတ်မန့် လုပ်ဆောင်တဲ့အခါ အိပ်စ်ပရက်ရှင် x ** 2 ကို တန်ဖိုးအရင်ရှာတယ်။ 6.25 ရတယ်။ ဒီတန်ဖိုးကို ဖန်ရှင်ခေါ်ထားတဲ့ နေရာကို return က ပြန်ပို့ပေးလိုက်တာပါ။ အောက်ပါ ဖန်ရှင်ကောလ်မှာလည်း ဒီဖြစ်စဉ် သဘောအတိုင်း တစ်ခါထပ်ဖြစ်မှာ ဖြစ်တယ်။

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

အခုတစ်ခါ ပါရာမီတာ x ဟာ အာ့ဂုမန့် a ရဲ့ တန်ဖိုး ဖြစ်တယ် (x = a အဆိုင်းမန့် လုပ်တဲ့သဘောပဲ)။ x ** 2 က ရလာတဲ့ 1048576 ကို ဖန်ရှင်ခေါ်တဲ့ နေရာက ပြန်ရတယ်။ နောက်ဆုံးတော့ ဒီတန်ဖိုးကို result မှာ အဆိုင်းမန့်လုပ်တယ်။ ဖြစ်စဉ်အရ ရိုးရှင်းပါတယ်။

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

ဒီလိုဆိုရင်ရော ဘယ်လို ဖြစ်မလဲ။ နည်းနည်းထူးခြားတာက အာ့ဂုမန့်နဲ့ ပါရာမီတာ နံမည်တူနေတာ။ ပါရာမီတာရဲ့ စကုပ်ဟာ ဖန်ရှင်သတ်မှတ်ချက် အတွင်းမှာပဲ ရှိတယ်လို့ ယူဆရမှာပါ။ ဒါကြောင့် အာ့ဂုမန့် x နဲ့ ပါရာမီတာ x နဲ့က သီးခြား ဗေရီရေဘဲလ်တွေ။

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

အာ့ဂုမန့်က အိပ်စ်ပရက်ရှင် ဖြစ်နေရင် တန်ဖိုးအရင်ရှာပြီး ရလဒ်ကို ပါရာမီတာနဲ့ အဆိုင်းမန့် လုပ်ပါတယ် (x = u + 2*t)။

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

ဒုတိယ ဖန်ရှင်ခေါ်တဲ့နေရာမှာ အိပ်စ်ပရက်ရှင်ကို z နဲ့ အဆိုင်းမန့် မလုပ်တော့ဘဲ တစ်ခါတည်း အာ့ဂုမန့် အနေနဲ့ ထည့်လိုက်တာပါ။ သဘောတရား တူတူပါပဲ။

ဖန်ရှင် return လုပ်တဲ့ သဘောကို နားလည်ထားဖို့လည်း အရေးကြီးတယ်။ return စတိတ်မန့်ဟာ ဖန်ရှင်ကနေ တန်ဖိုးတစ်ခုကို ဖန်ရှင်ခေါ်တဲ့ဆီကို ပြန်ပေးတယ်လို့ သိထားပြီးပါပြီ။ ဖန်ရှင်ထဲကနေ return လုပ်လိုက်တာနဲ့ ခေါ်ထားတဲ့နေရာကို ချက်ချင်း ပြန်ရောက်သွားတာ။

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

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

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

အခုအိပ်စ်ပရက်ရှင်ရဲ့ တန်ဖိုးရှာဖို့ get_sign(10) ခေါ်လိုက်တဲ့အခါ လက်ရှိနေရာကနေ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ဆီ ပြောင်းရွှေ့ ရောက်ရှိသွားပါမယ်။ ဖန်ရှင်ထဲက စတိတ်မန့်တွေ အစဉ်အတိုင်း စတင်လုပ်ဆောင်တယ်။ ဖန်ရှင်က return လုပ်တဲ့အခါ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ထဲကနေ ခေါ်ခဲ့တဲ့နေရာကို တဖန်ပြန်၍ ပြောင်းရွှေ့သွားတယ်။ return ပြန်လိုက်တဲ့ တန်ဖိုးကို ဖန်ရှင်ခေါ်တဲ့နေရာမှာ ရရှိပြီး လုပ်လက်စ အိပ်စ်ပရက်ရှင်ကို ဆက်လုပ်ပါတယ်။ ဒီလိုမြင်ကြည့်ပါ . . .

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

'10 is ' + get_sign(10)
```

မြှားအနက်က ဖန်ရှင်ခေါ်လိုက်တဲ့အခါ လုပ်ဆောင်မှု ပြောင်းရွှေ့သွားတာကို ပြတယ်။ မြှားအနီက return ပြန်တဲ့အခါ ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားတာကို ပြတာပါ။

ဆက်လက်ပြီး ပါရာမီတာ တစ်ခုထက်ပိုတဲ့ ဖန်ရှင်တချို့ကို ကြည့်ပါမယ်။ ပါရာမီတာဆိုတာ ဖန်ရှင်အတွက် လိုအပ်တဲ့ input ကို လက်ခံတဲ့ ဗေရီရေဘဲလ်ပါပဲ။ ထောင့်မှန်စတုဂံရဲ့ အလျားနဲ့ အနံကနေ ဧရိယာရှာပေးတဲ့ ဖန်ရှင်က ဒီလိုပါ

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

ဖန်ရှင်တစ်ခုကို အခြေခံ အုပ်ချုပ်သဖွယ် အသုံးပြု၍ အခြားဖန်ရှင်တွေ တည်ဆောက်ယူနိုင်တယ်။ rect_area ကို box_vol မှာ သုံးထားတာပါ

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

ဒီဖန်ရှင်ကို ခေါ်ရင် ဘယ်လိုဖြစ်မလဲ ကြည့်တတ်သင့်တယ်။ အခုလို ခေါ်မယ် ဆိုပါစို့

```
>>> box_vol(10, 5, 3)
```

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

```
rect_area(w, l) * h
```

rect_area ဖန်ရှင်ခေါ်တယ်။ wid=w, len=l ဖြစ်မယ်။ အခုကိစ္စအတွက် ပါရာမီတာနှစ်ခုရဲ့ တန်ဖိုးက 10 နဲ့ 5 အသီးသီး ဖြစ်မှာပါ။ 50 ရပါမယ်။ 50 * h ကို တန်ဖိုးဆက်ရှာပြီး ရလာတဲ့ 150 ကို box_vol ခေါ်ထားတဲ့နေရာကို return ပြန်ပေးမှာ ဖြစ်တယ်။ အခြေခံသဘောတရားတွေ သိပြီးတဲ့အခါ အတန်အသင့်ရှုပ်ထွေးတဲ့ ဖန်ရှင်တချို့ကို ကြည့်ပါမယ်။

ဖန်ရှင်များနှင့် အက်ဘ်စရက်ရှင်းလုပ်ခြင်း

မွေးသက္ကရာဇ် (date of birth) ကနေ အသက် တွက်ပေးတဲ့ ဖန်ရှင်ကို လေ့လာကြည့်ပါ။ အသက်တွက်တဲ့ လော့ဂျစ်ကို မရှင်းပြတော့ဘူး။ လေ့ကျင့်ခန်းအနေနဲ့ မိမိဖာသာ နားလည်အောင်ကြည့်ပါ။

```python
# File: age_today.py
from datetime import *

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

print(age_today(date(1990, 4, 2)))
```

ဖန်ရှင်အတွင်းပိုင်း လော့ဂျစ်တွေ ဘယ်လိုပဲ ရှုပ်ထွေးပါစေ၊ အသုံးပြုရတာကတော့ မခက်ပါဘူး။ ဖန်ရှင်ခေါ်တဲ့အခါ ဘယ်လို တည်ဆောက်ထားလဲ အတွင်းပိုင်း အယ်လ်ဂိုရစ်သမ်တွေ၊ လော့ဂျစ်တွေ သိစရာမလိုဘဲ သုံးရတာပါ။ ဖန်ရှင်က ၎င်းရဲ့ အတွင်းပိုင်း ကုဒ်တွေကို အက်ဘ်စရက်ရှင်း (*abstraction*) လုပ်ပေးလိုက်တာ ဖြစ်တယ်။ ဒါဟာ ဖန်ရှင်ရဲ့ အရေးပါဆုံး ဂုဏ်သတ္တိလို့ ဆိုရင်လည်း မမှားဘူး။

age_today ဖန်ရှင်ဟာ ပိုကြီးတဲ့ ပရိုဂရမ်တစ်ခုရဲ့ တစ်စိတ်တစ်ပိုင်း ဖြစ်လာနိုင်ပါတယ်။ ပရိုဂရမ်အသေးစားလေးတစ်ခုမှာ အသုံးပြုထားတာကို လေ့လာကြည့်ပါ။ နိုင်ငံအများစုမှာ (၁၈) နှစ် မပြည့်သေးတဲ့သူကို ဆေးလိပ်ရောင်းခွင့် မရှိဘူး။ ဥပဒေရှိပါတယ်။ စားသုံးသူရဲ့ မွေးသက္ကရာဇ် ထည့်ပေးလိုက်တာနဲ့ ရောင်းလို့ ရ/မရ ပြပေးတဲ့ ပရိုဂရမ်လေးပါ။

```python
# File: sell_cigarette.py
from datetime import *

def age_today(dob):
    today = date.today()
    this_bd = dob.replace(year=today.year)
    if today - dob >= this_bd - dob:
        return today.year - dob.year
    else:
        return today.year - dob.year - 1

def can_by_cig(dob):
```

```python
    age = age_today(dob)
    return True if age >= 18 else False

def main():
    """
    Given date of birth, this program tells whether the customer
    is eligible to buy cigarette or not.

    Enter 'exit' to quit the program.
    """
    print("Please enter 'quit' to exit this program.")
    while True:
        dobstr = input('Enter date of birth (yyyy-mm-dd): ')
        if dobstr == 'quit': break
        dob = date.fromisoformat(dobstr)
        print(dob)
        if can_by_cig(dob):
            print("Okay!")
        else:
            print('Too young to sell cigarette!')
    print('Program exited...')


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

၄.၂ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်များ

ဖန်ရှင်အားလုံးတော့ တန်ဖိုးပြန်ပေးတဲ့ ဖန်ရှင်တွေ မဟုတ်ကြပါဘူး။ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေလည်း ရှိတယ်။ ဥပမာ output ထုတ်တဲ့ `print` ဖန်ရှင်ဟာ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်မျိုးပါ။ အောက်ပါ `print_sign` ဖန်ရှင်ဟာ `get_sign` နဲ့ ဆင်တူပေမဲ့ တန်ဖိုး `return` ပြန်မပေးပါဘူး။

```python
def print_sign(r):
    if r > 0:
        print('positive')
    elif r < 0:
        print('negative')
    else:
        print('zero/nosign')
```

ဒီဖန်ရှင်မှာ `return` မပါတာ တွေ့ရပါမယ်။ ကားရဲလ်ဖန်ရှင်တွေမှာလည်း `return` မသုံးခဲ့တာ ပြန်အမှတ်ရမှာပါ။ `append_n_times` ကို လေ့လာကြည့်ပါ

```python
def append_n_times(lst, itm, n):
    for i in range(n):
        lst.append(itm)
```

```
lst = []
append_n_times(lst, 'hello', 10)
print(lst)
```

အိုက်တမ်တစ်ခုကို သတ်မှတ်ထားတဲ့ အရေအတွက်ပြည့်အောင် list တစ်ခုနောက်ကနေ ဆက်ပေးတယ်။ နဂို list မှာ အိုက်တမ်တွေ တိုးသွားပြီး စတိတ်အပြောင်းအလဲ ဖြစ်စေတယ်။

Output ထုတ်တဲ့ ဖန်ရှင်တွေဟာ တန်ဖိုးပြန်ပေးလေ့မရှိဘူး။ စခရင်မှာ စာသား (သို့) ရုပ်ပုံ ပြပေးတာဟာ output ဖြစ်တယ်။ ဖိုင်တစ်ခုမှာ ရေးတာလည်း output ပဲ (ဥပမာ Python ကုဒ်ဖိုင်ကို ပြင်ပြီး save လုပ်တာ) ။ အော့ဘ်ဂျက် စတိတ်ကို ပြောင်းလဲစေတဲ့ ဖန်ရှင်တွေဟာလည်း တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေ ဖြစ်လေ့ရှိတယ် (ဥပမာ list ရဲ့ append နဲ့ insert ဖန်ရှင်)။ စတိတ်အပြောင်းအလဲ ဖြစ်စေတဲ့ ဖန်ရှင်အားလုံး တန်ဖိုးပြန်မပေးတာတော့ မဟုတ်ဘူး။ ဥပမာ pop ဟာ တန်ဖိုးပြန်ပေးပါတယ်။ စတိတ်အပြောင်းအလဲလည်း ဖြစ်စေတယ်။

တန်ဖိုးပြန်တဲ့ ဖန်ရှင်ပဲ return ပြန်လို့ရတာ မဟုတ်ပါဘူး။ တန်ဖိုးပြန်မပေးတဲ့ ဖန်ရှင်တွေမှာလည်း return ပါနိုင်ပါတယ်။ print_sign ကို ဒီလိုရေးလို့လည်း ရပါတယ်

```
def print_sign2(r):
    if r > 0:
        print('positive')
        return
    elif r < 0:
        print('negative')
        return
    else:
        print('zero/nosign')
        return
```

တန်ဖိုးပြန်မပေးတဲ့အတွက် return ပဲဖြစ်ရပါမယ်။ တန်ဖိုး/အိပ်စ်ပရက်ရှင် တွဲပြီး ပါလို့မရပါဘူး။ ဖန်ရှင်ဘလောက် ပြီးတဲ့အခါ ခေါ်တဲ့နေရာကို ပြန်ရောက်သွားရမှာ ဖြစ်တဲ့အတွက် return မပါတဲ့ ဖန်ရှင်တွေရဲ့ ဘလောက်အဆုံးမှာ return ရှိတယ်လို့ ယူဆနိုင်တယ်။ ဥပမာ return မပါတဲ့ print_sign ကို အခုလို ယူဆနိုင်တယ်

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

၄.၃ ခွဲခြမ်းစိတ်ဖြာခြင်းနှင့် ပရိုဂရမ် ဒီဇိုင်း

Insurance Premium

နှစ် နှစ်ဆယ် သက်တမ်းကာလ $500,000 အသက်အာမခံထားရင် ကျန်းမာတဲ့ အသက် ၃၀ အမျိုးသမီး တစ်ယောက်အတွက် နှစ်စဉ်ပျမ်းမျှ အာမခံကြေး (premium) $229 ယူအက်စ် ဒေါ်လာ ကုန်ကျတယ်။ ရွယ်တူ အမျိုးသားတစ်ယောက် ဆိုရင်တော့ ဒီအာမခံအတွက်ကိုပဲ တစ်နှစ် ပျမ်းမျှ $373 ဒေါ်လာ ကုန်ကျပါမယ်။ ဒါကယေဘုယျ သဘောကိုပြောတာ။ တကယ်တမ်း အသက်အာမခံထားရင် သက်တမ်းကာလ၊ အကျိုးခံစားနိုင်မည့် ငွေပမာဏ (coverage)၊ ကျန်းမာရေး၊ အသက် စတဲ့ အချက်တွေပေါ် မူတည်ပြီး တွက်ချက်တာပါ။ လူတစ်ယောက်နဲ့ တစ်ယောက် ပရီမီယံ မတူဘူး။

Coverage Amount	Age 30	Age 40	Age 50	Age 60
$250,000	$142	$193	$392	$989
$500,000	$205	$307	$685	$1,781
$1 million	$325	$526	$1,227	$3,375
$2 million	$593	$984	$2,388	$6,758

တေဘဲလ် ၄.၁ နှစ်နှစ်ဆယ် အသက်အာမခံကြေး (မ)

Coverage Amount	Age 30	Age 40	Age 50	Age 60
$250,000	$162	$224	$499	$1,375
$500,000	$251	$360	$891	$2,567
$1 million	$408	$628	$1,681	$4,952
$2 million	$749	$1,190	$3,267	$9,660

တေဘဲလ် ၄.၂ နှစ်နှစ်ဆယ် အသက်အာမခံကြေး (ကျား)

မွေးသက္ကရာဇ်၊ အကျိုးခံစားလိုသည့် ပမာဏ (coverage amount)၊ ကျား/မ အလိုက် တစ်နှစ် ပျမ်းမျှ ပရီမီယံကြေး ကုန်ကျငွေ ပြပေးတဲ့ ပရိုဂရမ်အတွက် အာမခံ ကုမ္ပဏီတစ်ခုက သင့်ထံ ပရောဂျက် လာအပ်တယ် ဆိုပါစို့။ ဒီအတွက် ပရိုဂရမ် တစ်ခုကို ဒီဇိုင်းလုပ် ရေးသားပုံအဆင့်ဆင့်ကို ဆက်လက်ဖော်ပြပါမယ်။

အာမခံသက်တမ်း နှစ်ဆယ်နှစ်အတွက်ပဲ စဉ်းစားပါမယ်။ အသက် ၃၀ အောက် အနီးစပ်ဆုံး ယူပါမယ်။ ၃၁ မှ ၄၀၊ ၄၁ မှ ၅၀၊ ၅၁ မှ ၆၀ တို့ကို ၄၀၊ ၅၀၊ ၆၀ စသည်ဖြင့် အနီးစပ်ဆုံးယူပါမယ်။ အခြေခံအဆင့် လေ့လာသူအတွက် အလွန်အမင်း မရှုပ်ထွေးစေဘဲ သဘောတရားပိုင်းဆိုင်ရာကို အဓိက မြင်သာအောင် ပရိုဂရမ်ရဲ့ လိုအပ်ချက် (requirements) ကို ရိုးရှင်းအောင် လုပ်ထားတာပါ။

Input သုံးခုထည့်ပေးရမယ်။ မွေးသက္ကရာဇ် (date of birth) , coverage amount နဲ့ ကျား/မ (gender) တို့ဖြစ်တယ်။ အသုံးပြုသူအနေနဲ့ အဆင်ပြေဆုံး၊ အလွယ်ကူဆုံးဖြစ်အောင်၊ အမှားအယွင်း နည်းနိုင်သမျှနည်းအောင် စဉ်းစားသင့်တယ်။

မွေးသက္ကရာဇ်ကို ကြည့်ရအောင်။ ရက်စွဲကို နိုင်ငံနဲ့ နေရာဒေသ ပေါ်မူတည်ပြီး ဖော့မတ်အမျိုးမျိုးနဲ့ ရေးကြတယ်။ 04/05/2020, 04-05-2020, Apr-4-2020 စသည်ဖြင့်။ ခုနှစ်ကို ဂဏန်း နှစ်လုံးပဲ ရေးတာလည်း ရှိတယ်။ ရက်နဲ့လကို ရှေ့မှာ သုညမပါဘဲလည်း ရေးတယ်။ ဥပမာ 4/5/2020, 4/5/20, Apr-4-2020 ။ ဖော့မတ်တစ်မျိုးကိုပဲ သတ်သတ်မှတ်မှတ် သုံးသင့်တာ ဖြစ်ပေမဲ့ ဆော့ဖ်ဝဲအပ်သူက ဖော့မတ် သုံးမျိုးနဲ့ ထည့်လို့ရအောင် လုပ်ပေးဖို့ တောင်းဆိုထားတယ်။

01-01-2024
01/01/2024
Jan-1-2024

မွေးသက္ကရာဇ်ကို ဒီဖော့မတ်သုံးမျိုးနဲ့ ပရိုဂရမ်က လက်ခံရပါမယ်။ input ဖန်ရှင်က ကီးဘုဒ်ကထည့်ပေးတာကို string အနေနဲ့ ပြန်ပေးပါတယ်။

မွေးသက္ကရာဇ်ကနေ အသက်ကို တွက်ယူရမှာပါ။ ဒီအတွက် ဖန်ရှင် (ဥပမာ calc_age) သတ်မှတ်နိုင်တယ်။ နေ့ရက်၊ အချိန်နဲ့ သက်ဆိုင်တဲ့ အတွက်အချက်တွေ အတွက် စာသားကို အသုံးမပြုသင့်ဘူး။ date, datetime စတာတွေ သုံးသင့်တယ်။ ဒါကြောင့် စာသားကနေ မွေးသက္ကရာဇ်ကို ဖော်ပြတဲ့ date (သို့) datetime အော့ဘ်ဂျက် ပြောင်းဖို့ လိုပါမယ်။ '01-01-2024' ကနေ 1, 1, 2024 ဂဏန်းတွေ ရအောင် စာသားကို တစ်ဖြတ်ချင်း ဖြတ်ပြီး ပြောင်းမယ်။

```
>>> egstr.split('abc')
['123', '456', '789']
>>> dt1 = '01-01-2024'
>>> parts1 = dt1.split('-')
>>> parts1
['01', '01', '2024']
>>> dt2 = '01/01/2024'
>>> parts2 = dt2.split('/')
>>> parts2
['01', '01', '2024']
>>> dt3 = 'Feb-02-2024'
>>> parts3 = dt3.split('-')
>>> parts3
['Feb', '02', '2024']
```

split ဖန်ရှင်က string တစ်ခုကို အပိုင်းတွေ ပိုင်းပေးတာပါ။ အပိုင်းတွေ အားလုံးကို list အနေနဲ့ ပြန်ပေးတယ်။ ဘယ် ကာရက်တာ (သို့) string နဲ့ ခြားထားရင် ပိုင်းဖြတ်ချင်လဲ ထည့်ပေးလို့ရတယ်။ အပေါ်မှာ 'abc', '-', '/' အသီးသီးနဲ့ ပိုင်းဖြတ်ထားတယ်။

```
>>> dt1 = '01-01-2024'
>>> parts1 = dt1.split('-')
>>> date(int(parts1[2]), int(parts1[1]), int(parts1[0]))
datetime.date(2024, 1, 1)
```

'Feb-02-2024' ဖော့မတ်က နည်းနည်း ပိုရှုပ်မယ်။ လက 'Jan', 'Feb' စသည်ဖြင့် စာသားဖြစ်နေတယ်။ အခုလို dictionary တစ်ခု ရှိရင် လနံမည်ကနေ သက်ဆိုင်တဲ့ ဂဏန်းကို အလွယ်တကူ ရနိုင်ပါတယ်

```
>>> mths = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
...         'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
...         'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
>>> mths['Jan']
1
>>> mths['Dec']
12
```

ဒီလိုဆိုရင် date အော့ဘ်ဂျက် ရဖို့အတွက်လည်း သိပ်မခက်တော့ဘူး။ ဥပမာ

```
>>> dt4 = 'Dec-26-2024'
>>> parts4 = dt4.split('-')
>>> date(int(parts4[2]), mths[parts4[0]], int(parts4[1]))
datetime.date(2024, 12, 26)
```

အခုဖော်ပြခဲ့တဲ့ နည်းအပြင် အခြားနည်းလမ်းတွေလည်း ရှိပါတယ်။ datetime ကလပ်စ်မှာ နေ့ရက်ကို စာသားကနေ datetime ပြောင်းပေးတဲ့ strptime ဖန်ရှင် ရှိပါတယ်။

```
>>> dtstr1 = 'Jan-01-2024'
>>> datetime.strptime(dtstr1, "%b-%d-%Y")
datetime.datetime(2024, 1, 1, 0, 0)
>>> dtstr2 = '30-12-2024'
>>> datetime.strptime(dtstr2, "%d-%m-%Y")
datetime.datetime(2024, 12, 30, 0, 0)
>>> dtstr3 = '30/Dec/2024'
>>> datetime.strptime(dtstr3, "%d/%b/%Y")
datetime.datetime(2024, 12, 30, 0, 0)
```

ဒီဖန်ရှင်က စာသားနဲ့ ဖော်ပြထားတဲ့ အချိန်နေ့ရက်ကို datetime အော့ဘ်ဂျက် ပြောင်းပေးဖို့ ဖော့မတ်ကုဒ် (format code) လို့ခေါ်တဲ့ % နဲ့ စတဲ့ ကာရက်တာတွေကို လက်ခံတယ်။ **%d** က ရက်၊ **%m** က လကို ဂဏန်း တစ်လုံး (သို့) နှစ်လုံးနဲ့ ဆိုတဲ့ အဓိပ္ပါယ် (ဥပမာ ကိုးရက်နေ့ကို 09 သို့မဟုတ် 9၊ ငါးလပိုင်းကို 05 သို့မဟုတ် 5)။ **%Y** က ခုနှစ်ကို ဂဏန်းလေးလုံးနဲ့ ရေးတယ်လို့ ဆိုလိုတာပါ။ **%b** ကတော့ လရဲ့ နံမည်ကို Jan, Feb, Mar စသည်ဖြင့် အတိုကောက်ရေးတယ်လို့ ဆိုလိုတယ်။ 'Jan-01-2024' ရဲ့ ဖော့မတ်က '%b-%d-%Y'