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 ပရိုဂရမ်မာလေ့ကား ပထမ တစ်ထစ် တက်လှမ်းအောင်မြင်ပြီ ပြောနိုင်ပါတယ်။ ဒီသဘောတရားတွေကို ဘီဂင်နာတွေ အရိုးရှင်းဆုံးနည်းနဲ့ နားလည်အောင်၊ လေ့ကျင့်လို့ရအောင် ကားရဲလ်က ထောက်ကူပေးတာပါ။ စက်ရုပ်လေး ကားရဲလ်ကို နှုတ်ဆက်ခဲ့ပြီး အခု ဆက်လက်လေ့လာကြမှာကတော့ ဒေတာ၊ အိပ်စ်ပရက်ရှင်းနဲ့ ဗေရီရေဘဲလ်တွေ အကြောင်းပါ။

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

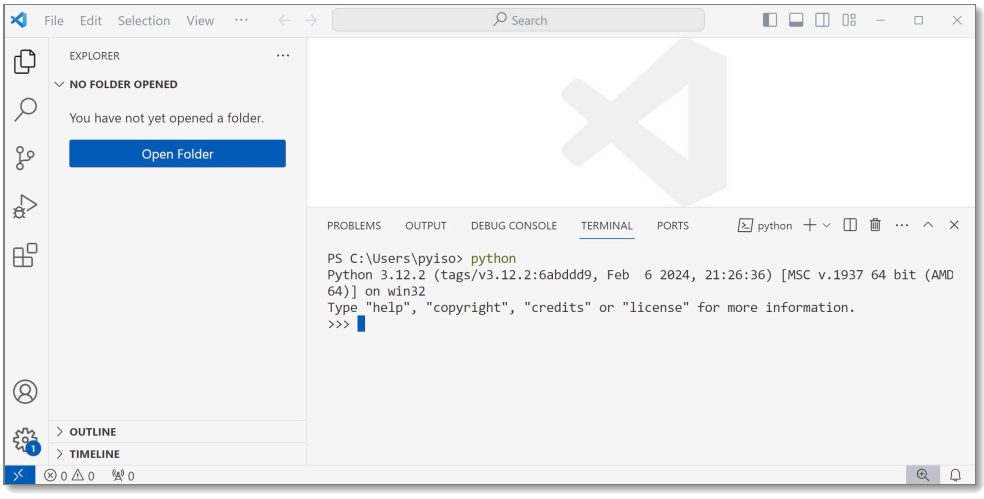
ဒေတာ အမျိုးအစား အများအပြားရှိပေမဲ့ အခြေခံအကျဆုံးက ကိန်းဂဏန်းတွေပါ။ ကွန်ပျူတာတွေကို စတင်တီထွင်ဖို့ ကြိုးစားလာကြတဲ့ အဓိက အကြောင်းအရင်းကလည်း ဂဏန်းသချိ်ာ အတွက်အချက်တွေကို လုပ်ဆောင်ရာမှာ လူတွေကို အကူအညီ ပေးဖို့အတွက်ပဲလို့ ဆိုနိုင်ပါတယ်။ ဒါ့အပြင် စာသား၊ ရုပ်သံ စတဲ့ အခြားဒေတာ အမျိုးအစားတွေကို ကွန်ပျူတာထဲမှာ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

၁.၁ ကိန်းဂဏန်းများ

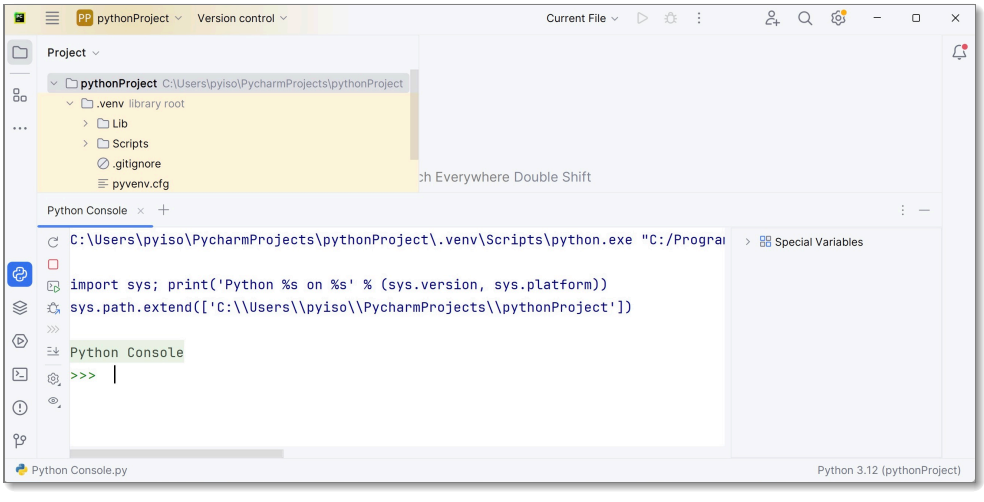
Python အင်တာပရက်တာနဲ့ Python ကုဒ်တွေကို run လို့ရတဲ့နည်း နှစ်ခုရှိပါတယ်။ တစ်နည်းက ကုဒ်တွေကို **.py** ဖိုင်နဲ့သိမ်းပြီး **python** ကွန်မန်းနဲ့ run တာပါ။ ဒါက ကားရဲလ် ပရိုဂရမ်တွေမှာ သုံးခဲ့တဲ့နည်း။ နောက်တစ်နည်းကတော့ ကုဒ်တစ်ကြောင်းချင်း ရိုက်ထည့်ပြီး run တဲ့နည်းပါ။ ဒီနည်းလမ်းက interactive mode နဲ့ အင်တာပရက်တာကို 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** + `` ` `` ရော့ကတ်ကီးနဲ့ ဖွင့်လို့လည်းရတယ်) ပြီး **python** ကွန်းမန်း 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 အနှုတ်ကိန်းပြည့် တန်ဖိုးကို ကန့်သတ်ထားတာ မရှိဘူး။ သီအိုရီအရ ကြိုက်သလောက် ကြီးလို့/ငယ်လို့ ရပါတယ်။ လက်တွေ့

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

```
>>> 900000000000000000000000000000 + 5000000000
900000000000000000005000000000
>>> -900000000000000000000000000000 + 1
-899999999999999999999999999999
```

`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
```

ဒဿမကိန်းကို e အမှတ်အသားနဲ့ရေးထားတာပါ။ အကွာရာ `e` နောက် လိုက်တဲ့ ကိန်းပြည့်ဂဏန်းကို 10 ထပ်ကိန်းလို့ မှတ်ယူရပါတယ်။ `e3` က 1×10^3, `e-3` က 1×10^{-3} ပါ။ အကွာရာ 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၊ မော်ဒျူလို နှင့် ထပ်ကိန်းတင်ခြင်း

အကယ်၍ ဒဿမကိန်းမထွက်ဘဲ 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 မော်ဒျူလို (modulo) အော်ပရိတ်တာလို့ ခေါ်တယ်။ remainder အော်ပရိတ်တာလို့လည်း ခေါ်တယ်။

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

အင်တီဂျာ ဒီဗီးရှင်းနဲ့ မော်ဒျူလိုကို 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 သူ့ထက်ပိုငယ်တဲ့ `-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 ဒဿမကိန်းတွေနဲ့လည်း 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 အခု လို တွက်ပါတယ်။

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

သခ်ၤာဖန္ရွင္မ်ား

ကိန်းဂဏန်းတွေအကြောင်း လေ့လာလက်စနဲ့ math လိုက်ဘရီ သခ်ၤာဖန်ရှင်တချို့ကိုလည်း တစ်ခါတည်း ဆက်ကြည့်လိုက်ရအောင်။ အဓိကက သခ်ၤာဖန်ရှင်ဆိုတာထက် ဖန်ရှင် အခြေခံအသုံးပြုပုံကို စပြီးလေ့လာ မှာပါ။ 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
```

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

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

အကြီးဆုံးဘုံဆခွဲကိန်းကို gcd ဖန်ရှင်နဲ့ ရှာတာပါ။ အင်တီဂျာ input တစ်ခုနဲ့အထက် လက်ခံတဲ့ ဖန်ရှင် ဖြစ်တယ်။ 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 ဥပမာ ပြထားတာ 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 ဒဿမကိန်းတွေကို 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 အချိန်ကာလပေါ်မူတည်ပြီး 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 စီတန်းဖွဲ့စည်းထားတယ်။ ကာရက်တာဆိုတာ အခြေခံ သတင်းအချက်အလက် 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။ ကာရက်တာ 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 ဒဿမ 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 ဗေရီရေဘဲလ်များ

ဗေရီရေဘဲလ်တွေကို 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။

ဖော်ပြဖို့ အသုံးပြုတယ်။ ပါဝင်တဲ့ အော်ပရေးရှင်းတွေကတော့ `and`, `or` နဲ့ `not` တို့ ဖြစ်ပါတယ်။ `and` အော်ပရိတ်တာက တန်ဖိုးနှစ်ခုလုံး မှန်ရင်

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

Escape Sequence	အဓိပ္ပါယ်

တေဘဲလ် ၁.၂ Python Escape Sequences

တန်ဖိုးတစ်ခုနဲ့ တစ်ခု နှိုင်းယှဉ်တဲ့အခါ `True` သို့မဟုတ် `False` ပြန်ပေးတယ်။

```
>>> 5 == 5
True
>>> 3 > 4
False
```

ကိန်းဂဏန်းနဲ့ စာသားတွေအကြောင်း အတန်အသင့်သိထားပြီး ဘူလီယန် (boolean) ကို ဆက်ကြည့်ရအောင်။ "နှစ်က သုံးထက်ငယ်တယ်"၊ "ဆယ့်သုံးဟာ သုံးနဲ့စားလို့ ပြတ်တယ်"၊ "$2+2$ ဟာ $5-1$ နဲ့ ညီတယ်" စတဲ့ ဖော်ပြချက်တွေက မှန်တာ (သို့) မှားတာ ဖြစ်နိုင်ပါတယ်။ ဒီဖော်ပြချက်တွေ မှန်သည်၊ မှားသည် ဒီလိုမျိုး စစ်ကြည့်နိုင်တယ်။

```
>>> 2 < 3
True
>>> 13 % 3 == 0
False
>>> 2 + 2 == 5 - 1
True
```

`<` (less than) က ငယ်သလား စစ်ပေးတာ။ `==` (equal to) က ညီသလားစစ်တဲ့ဟာ။ ညီမျှခြင်းသင်္ကေတနှစ်ခုဖြစ်ရမယ်။ (`=` တစ်ခုထဲက assignment operator ပါ။ `==` နဲ့ လုံးဝမတူဘူး။)

ဘူလီယန် မှာ `true` နဲ့ `false` တန်ဖိုးနှစ်မျိုးပဲရှိတယ်။ `true` နဲ့ `false` မှာ double quotes မပါလို့ စာသားလို့ မယူဆရပါဘူး။ ဂဏန်းလည်း မဟုတ်ဘူး။ ဘူလီယန်က ဘူလီယန်သတ်သတ်ပဲ။ `<`, `>`, `<=`, `>=`, `==`, `!=` တို့ဟာ comparison operator တွေ ဖြစ်တယ်။ Relational operator တွေလို့လည်း ခေါ်ကြတယ်။ အလယ်တန်းသချိုးမှာ သင်ခဲ့ရတဲ့ $<$, $>$, $\leq$, $\geq$, $=$, $\neq$ တို့နဲ့ အဓိပ္ပါယ်တူတူပါပဲ။

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 ကေသီဘဏ်အကောင့်၊ $\frac{7}{13}$ (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 ယူဆတာ ဟာလည်း သဘာဝကျတယ် ဆိုရမှာပါ။

အော့ဘ်ဂျက်တွေကို အက်ဘ်စရက်ရှင်း (*abstraction*) အနေနဲ့လည်း ရှုမြင်နိုင်တယ်။ ဘယ်လို ဖန်တီး 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 ခဏာရပ်၊ 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 (တနလ်ာက 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 စာသားအဖြစ် `'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 အော့ဂုမန့်တွေကို 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)
```

အော့ဂုမန့် 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

၃.၀ Python Arcade Library

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

```
pip install arcade
```

၃.၂ Arcade ဖြင့် ပုံဆွဲခြင်း

ပုံဆွဲလို့ရတဲ့ ဝင်းဒိုးတစ်ခုပေါ်လာအောင် လိုအပ်တဲ့ အဆင့်တစ်ဆင့်ချင်းကို အောက်ပါ Arcade ပရိုဂရမ် မှာ တွေ့ရပါမယ်။ ဝင်းဒိုးပေါ်က ရုပ်ပုံတွေကို အန်နီမေးရှင်း (animation) လုပ်တဲ့နည်းကိုလည်း မကြာ ခင်တွေ့ရမှာပါ။ ပရိုဂရမ် run ရင် ပုံ (၃.၁)

```
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()
```

ပုံ ၃.၁

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

အပေါ်ဆုံးက `arcade` လိုက်ဘရီ အင်ပို့လုပ်ထားတာပါ။ ရှေ့ပိုင်းမှာသုံးတဲ့ `from ... import ...` နဲ့ ကွာခြားတာက အခုနည်းနဲ့ အင်ပို့လုပ်ထားရင် လိုက်ဘရီမှာ ပါတဲ့ အစိတ်အပိုင်းတွေကို ဒေါ့ထ် အမှတ်အသားနဲ့ အသုံးပြုရပါမယ်။ ဥပမာ `open_window` ဖန်ရှင်ကို

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

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

အောက်ပါ `set_viewport` ဖန်ရှင်ကတော့ ဝင်းဒိုး ကိုဩဒိနိတ်စစ်စတမ် origin အမှတ်နဲ့ x, y ဒါရိုက်ရှင် သတ်မှတ်ပေးတာပါ။

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

ဝင်းဒိုး ဘယ်ဘက်စွန်း x ကိုဩဒိနိတ်ကို 0 နဲ့ ညာဘက်စွန်းကို 300 သတ်မှတ်ပြီး အပေါ်ဘက်စွန်း (တိုက်တယ်လ်ဘားမပါ) y ကိုဩဒိနိတ်ကို 0 နဲ့ အောက်ဘက်စွန်းကို 200 သတ်မှတ်ထားပါတယ်။ တစ်နည်းအားဖြင့် ဝင်းဒိုးရဲ့ ဘယ်ဘက်အပေါ်ထောင့်စွန်းကို origin အမှတ် $(0, 0)$ အဖြစ် သတ်မှတ်တာပါ။ ဘယ်ဘက်ကို သွားရင် x တန်ဖိုး တိုးသွားပြီး အောက်ဘက်ကို ဆင်းရင် y တန်ဖိုး တိုးသွားမှာဖြစ်တယ်။ ကိုယ်တိုင် အခုလို မသတ်မှတ်ပေးဘဲ အရှိအတိုင်းဆိုရင် 'အောက်ခြေ' ဘယ်ဘက်စွန်းကို origin အမှတ် $(0, 0)$ အဖြစ် Arcade က သတ်မှတ်တယ်။ `set_viewport` ကို ဒီလိုခေါ်ထားတာနဲ့ တူမယ်

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

ဒီလိုနည်းက အခြား ဂိမ်း လိုက်ဘရီတွေနဲ့ မတူဘဲဖြစ်နေတဲ့အတွက် လိုက်ဘရီပြောင်းသုံးရင် အခက်အခဲရှိနိုင်တယ်။ ဒါကြောင့် အခြားလိုက်ဘရီတွေ နည်းတူဖြစ်အောင် အရှိအတိုင်းမသုံးဘဲ ကိုယ်ပိုင် သတ်မှတ်ပေးရတာပါ။

အောက်ပါ ဖန်ရှင်ကတော့ ဝင်းဒိုးရဲ့ နောက်ခံရောင် သတ်မှတ်တာပါ။ လိုက်ဘရီရဲ့ `color` မော်ဒျူး (*module*) မှာ အရောင်တန်ဖိုးတွေ အဆင်သင့် သတ်မှတ်ပေးထားတယ်။ (မော်ဒျူးဆိုတာ 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 ခေါ်ပေးရပါမယ်။ ပြီးရင်လည်း finish_render ခေါ်ဖို့လိုတယ်။ ၎င်းတို့နှစ်ခုကြားမှာ Arcade နဲ့ ပုံဆွဲတဲ့ ဖန်ရှင်တွေကို ခေါ်ရမှာပါ။

```
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()
```

ဒါကို run လိုက်ရင် ပုံ (၃.၂) မှာလို တွေ့ရမှာပါ။ အနက်ရောင် အနားသတ်နဲ့ အပြာရောင်ဖြည့်ထားတဲ့ အပေါ်ပုံကို ဒီနှစ်ခုနဲ့

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

ဆွဲထားတာပါ။ ပါရာမီတာတွေက $x, y, width, height, color$ အစဉ်အတိုင်းဖြစ်တယ်။ $x = 5$ ဖြစ်လို့ ဘယ်ဘက် နည်းနည်းရောက်နေတာပါ။ $y = 5$ ဖြစ်လို့ အောက်ဘက် နည်းနည်းရောက်နေတာပါ။ သုည ထားပြီး စမ်းကြည့်ပါ။ အခြားတန်ဖိုးတွေ ပြောင်းလဲပြီးလည်း စမ်းကြည့်ပါ။ ပိုပြီးသဘောပေါက် လာပါလိမ့်မယ်။ အောက်ဘက် ထောင့်မှန်စတုဂံကို ဒီနှစ်ခုနဲ့

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

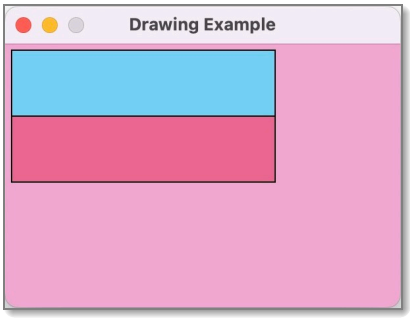
ဆွဲထားတာပါ။ ဘယ်ဘက်ခွာထားတဲ့ အကွာအဝေး အပေါ်နဲ့ တူပါမယ် $(x = 5)$။ အပေါ်ပုံရဲ့ အောက်ခြေ အနားနဲ့ ကပ်နေအောင် $y = 5 + 50 = 55$ ထားရပါမယ်။

```
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

def draw_xywh_rectangle_outline(bottom_left_x: float,
                                bottom_left_y: float,
                                width: float,
                                height: float,
                                color: tuple[int, int, int]
                                    | list[int]
                                    | tuple[int, int, int, int],
                                border_width: float = 1) -> None
```

Arcade ဝင်းဒိုး origin က သူနဂိုအတိုင်းဆိုရင် အောက်ခြေဘယ်ဘက်စွန်းလို့ ရှေ့မှာ ပြောခဲ့ပါတယ်။ `set_viewport` နဲ့ အပေါ်ဘယ်ဘက်စွန်းကို origin အဖြစ်ပြောင်းလဲ သတ်မှတ်ထားတယ်။ ဒီအတွက်ကြောင့် အထက်အောက် ပြောင်းပြန်ဖြစ်သွားပါတယ်။

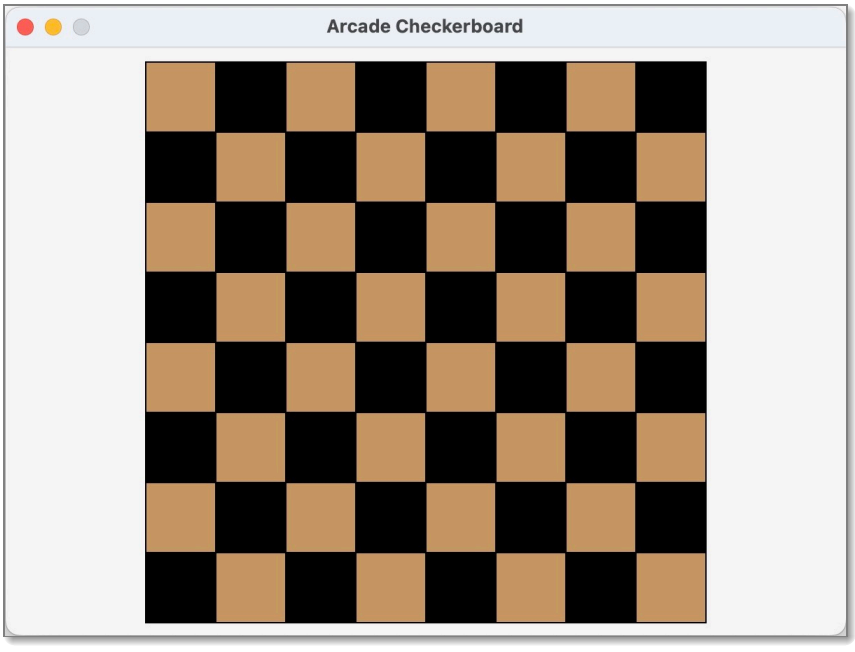
```
import arcade
from arcade.color import *

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,
```

JC



ộ ২.J

```
                    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()
```



চিত্র ২.২

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 အခုလိုမက် သဒ်တွေ 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 အာ့ဂုမန့် (*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 ရှိတယ်လို 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 ပြောင်းရွေ့ ရောက်ရှိသွားပါမယ်။ ဖန်ရှင်ထဲက စတိတ်မန့်တွေ အစဉ်အတိုင်း စတင် လုပ်ဆောင်တယ်။ ဖန်ရှင်က return လုပ်တဲ့အခါ လုပ်ဆောင်မှုက ဖန်ရှင်ဘလောက်ထဲကနေ ခေါ်ခဲ့တဲ့ နေရာကို တဖန်ပြန်၍ ပြောင်းရွေ့သွားတယ်။ return ပြန်လိုက်တဲ့ တန်ဖိုးကို ဖန်ရှင်ခေါ်တဲ့နေရာမှာ ရရှိ ပြီး 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 ပြောင်းရွေ့သွားတာကို ပြတယ်။ မြှားအနီက return ပြန်တဲ့အခါ ခေါ်ခဲ့တဲ့နေရာ ပြန်ရောက်သွားတာကို ပြတာပါ။

ဆက်လက်ပြီး ပါရာမီတာ တစ်ခုထက်ပိုတဲ့ ဖန်ရှင်တချို့ကို ကြည့်ပါမယ်။ ပါရာမီတာဆိုတာ ဖန်ရှင် အတွက် လိုအပ်တဲ့ input ကို လက်ခံတဲ့ မေရီရောဲလ်ပါပဲ။ 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။

```
# 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။

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

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

def can_by_cig(dob):
```

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

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

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


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

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။

```
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

```
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
```