Objects in Python

Python တွင် class များ တည်ဆောက်ခြင်း နှင့် ထို class တွင် ပါဝင်သော object များ ပြုလုပ်ခြင်း (instantiate objects) အကြောင်းကို ရှင်းပြထားသည်။ `Class` keyword ကို အသုံးပြု၍ Python class တစ်ခု တည်ဆောက်နိုင်သည်။ ဥပမာ-

```
class MyFirstClass:
    pass
```

`class MyFirstClass:` သည် class statement ဖြစ်သည်။ Class statement ကို execute လုပ်လိုက်သည့်အချိန်တွင် class တစ်ခု တည်ဆောက်ပြီး ဖြစ်သည်။ ထည့်ပေးထားသည့် နာမည်(class name)နှင့် တည်ဆောက်လိုက်သည့် class ကို bind လုပ်လိုက်သည်။

MyFirstClass ဆိုသည့် Python class တစ်ခု တည်ဆောက်(create)လိုက်ပြီး MyFirstClass ဟုသည့် နာမည်ပေးထားသည်။ MyFirstClass ဆိုသည့် နာမည် assign လုပ်သည်ဟု သုံးသည်။ Class ထဲတွင် ဘာမှ မထည့်ရသေးသောကြောင့် empty class တစ်ခုသာ ဖြစ်နေသေးသည်။ Python တွင် class သည်လည်း object တစ်ခု ဖြစ်သည်။

MyFirstClass တည်ရှိသွားပြီ ဖြစ်သောကြောင့် ၄င်းနှင့် သက်ဆိုင်သည့် အချက်အလက်များကို **print()** ဖြင့် ဖတ်နိုင်သည်။ သိနိုင်သည်။

```
class MyFirstClass:
    pass

print("memory loation/address:", id(MyFirstClass))
print("directory", dir(MyFirstClass))
```

```
memory loation/address: 2901261525480
directory ['__class__', '__delattr__', '__dict__', '__dir__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
 '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '_
_module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__r
epr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__']
```

Python code များထဲတွင် Memory address များကို အသုံးပြုလေ့ မရှိပါ။ Class တည်ဆောက်ပြီးကြောင့် သက်သေပြရန်အတွက်သာ ဖော်ပြခြင်း ဖြစ်သည်။

Class တစ်ခုထဲတွင် instance များ ပြုလုပ်ခြင်း(Instantiation)

တည်ဆောက် ပြီးသည့် MyFirstClass ဆိုသည့် Python class ထဲတွင် ရှိနေမည့်၊ ပါဝင်မည့် object များ ထည်နိုင်ပါပြီ။ a နှင့် b ဟု အမည်ပေးထားသည့် Python object နှစ်ခု တည်ဆောက်မည်။ instance object a နှင့် b တည်ဆောက်လိုက်ခြင်းကို `Instantiation` ဟုခေါ်သည်။

Python အတွင် instance object များသည် first class object များ ဖြစ်ကြသည်။ first class object များ ဖြစ်ခြင်းကြောင့် instance object များကို list, tuple, dictionary နှင့် set စသည်တို့၏ element

များအဖြစ် ထည့်သွင်းနိုင်သည်။ Instance object များကို တခြားသော function ထဲသို့ argument အဖြစ် ထည့်ပေး(pass လုပ်) နိုင်သည်။

Class object များသည်လည်း first class object များ ဖြစ်ကြသည်။

```
a = MyFirstClass()
b = MyFirstClass()
```

ထို object a နှင့် b တို့သည် MyFirstClass ထဲတွင်သာ တည်ရှိသည်။ ထို object a နှင့် b တို့၏ အချက်အလက်များကို **print(), dir(), id()** ဖြင့် ဖတ်နိုင်သည်။ သိနိုင်သည်။

```
print("Id of a", a)
print("\n Type of a", type(a))
print("\n Dir of a", dir(a))
```

```
Id of a <__main__.MyFirstClass object at 0x000002A380F81F88>

 Type of a <class '__main__.MyFirstClass'>

 Dir of a ['__class__', '__delattr__', '__dict__', '__dir__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
 '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '_
_module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__r
epr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__']
```

```
print("Id of Obejct b", b)
print("\n Type of a", type(b))
print("\n Dir of Obejct b", dir(b))
```

```
Id of Obejct b <__main__.MyFirstClass object at 0x000002A380F81F48>

 Type of a <class '__main__.MyFirstClass'>

 Dir of Obejct b ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '_
_init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__n
ew__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__']
```

ထိုသို့ MyFirstClass ထဲတွင် ပါဝင်နေမည့် object a နှင့် b တို့ ပြုလုပ်လိုက်ခြင်းကို instantiate လုပ်သည်ဟု ပြောလေ့ရှိသည်။ Instance object a နှင့် b တို့ သည် MyFirstClass ၏ instance များ ဖြစ်ကြသည်။ Instance များ ဖြစ်အောင် ပြုလုပ်ခြင်းကြောင့် instantiate လုပ်သည်ဟု သုံးနှုန်းခြင်း ဖြစ်သည်။

Object a နှင့် b ကို ပို၍ တိကျစွာ ရည်ညွှန်းပြောဆိုလိုပါက class instance ဟု သုံးသည်။ Instance object ဟုလည်း သုံးနှုန်းလေ့ရှိသည်။

Class ၏ object တစ်ခု ပြုလုပ်ရန် object နာမည်ပေးပြီး ညီမျှခြင်း လက္ခဏာနှင့် class နာမည် နောက်တွင် လက်သည်းကွင်း() ထည့်ရေးခြင်း ဖြစ်သည်။

Function တစ်ခုကို call လုပ်သည့် ပုံစံဖြင့် ရေးခြင်း ဖြစ်သည်။ သို့သော် ထိုသို့ရေးလျှင် Python က function ကို "call" လုပ်ခြင်း မဟုတ်ဘဲ class ကို "call" လုပ်ခြင်း ဖြစ်ကြောင်း သိသည်။

Object a နှင့် b တို့ သည် MyFirstClass ထဲတွင် တည်ရှိနေသည့် empty obejct များ ဖြစ်သောကြောင့် အသုံးချ၍ မရသေးပါ။

Attribute များ ထည့်ခြင်း(Adding Attributes)

Instance object ထဲသို့ data များ(Adding Attributes)ထည့်မည်။ **dot notation** ကို အသုံးပြု၍ အောက်ပါအတိုင်း ရေးပြီး instance object ထဲသို့ ဒေတာများ ထည့်နိုင်သည်။ ဥပမာအဖြစ် အောက်တွင် Point ဆိုသည့် class တစ်ခု တည်ဆောက်(define)ပြီး သရုပ်ပြထားသည်။

```
class Point:
    pass
```

Point class ထဲတွင် မည်သည့် data(attributes) နှင့် behaviors (methods) တို့မျှ မရှိသေးပါ။ p1 နှင့် p2 ဆိုသည့် class instance များ တည်ဆောက်(create)သည်။

```
p1 = Point()
p2 = Point()
```

p1 ထဲသို့ attribute အဖြစ် x တန်ဖိုးနှင့် y တန်ဖိုး ထည့်ပေးသည်။

```
p1.x = 5
p1.y = 4
```

p2 ထဲသို့ attribute အဖြစ် x တန်ဖိုးနှင့် y တန်ဖိုး ထည့်ပေးသည်။

```
p2.x = 3
p2.y = 6
```

ထည့်ပေးလိုက်သည့် attribute value များ ရှိ၊ မရှိကို ပြန်ဖတ် ကြည့်သည်။

```
print(p1.x, p1.y)
print(p2.x, p2.y)
```

Class instance များ ဖြစ်ကြသည့် p1 နှင့် p2 တို့ထဲသို့ value များ ထည့်ရန်(assign လုပ်ရန်) အတွက် **dot notation** ကို အသုံးပြုသည်။ dot notation ကို ရေးသည့်ပုံစံ(syntax)မှာ

```
<object>.<attribute> = <value>
```

Attribute value သည် Python ဒေတာများ ဖြစ်နိုင်သလို function တစ်ခုခု သို့မဟုတ် တခြား class လည်း ဖြစ်နိုင်သည်။ တစ်နည်းအားဖြင့် တခြား class များ, function များကို instance object ၏ attribute value များ အဖြစ် assign လုပ်နိုင်သည်။ ထည့်ပေးနိုင်သည်။

Instance Variable

Instance object ၏ variable များကို Instance Variable ဟု ခေါ်သည်။ Instance object တွင် တွဲမိသွားသည့်(attached ဖြစ်သွားသည့်) Variable ကို Instance Variable ဟု ခေါ်သည်။

Python တွင် variable ကို assigment လုပ်သည့်နည်းဖြင့် create လုပ်ကြသည်။ ကြိုတင် declare လုပ်ရန် မလိုပါ။ တန်ဖိုးတစ်ခုခုကို assign လုပ်လိုက်ရုံဖြင့် variable တည်ရှိသွားပြီ ဖြစ်သည်။ ထို့နည်းတူ Instance variable များကိုလည်း assigment လုပ်သည့်နည်းဖြင့် create လုပ်နိုင်သည်။

အောက်တွင် instance variable x နှင့် y တို့သည် p1 ဆိုသည့် Instance object တွင် တွဲမိသွားသည်။ Attached ဖြစ်သွားသည်။ ဥပမာ-

```
p1.x = 5
p1.y = 4
```

5 နှင့် 4 သည် p1 ၏ instance variable များ ဖြစ်သောကြောင့် p1 နှင့်သာ သက်ဆိုင်သည်။ p2 နှင့် မသက်ဆိုင်ပါ။ p2 အတွက်လည်း သူ့ကိုယ်ပိုင် instance variable တန်ဖိုးများ ရှိကြသည်။

Instance variable များကို class အတွင်း၌သာ attach လုပ်နိုင်သည်။ တည်ရှိနိုင်သည်။ Instance variable များသည် class အပြင်ဘက်တွင် မတည်ရှိနိုင်ပါ။

Class တစ်ခုအတွင်းရှိ Instance object များ အားလုံးတွင် same set of instance variable များရှိသည်။ Class တစ်ခုအတွင်း၌ instance variable များကို access လုပ်လိုပါက self. နောက်တွင် variable_name ထည့်ရေး၍ (`self.variable_name`) ဖြင့် access လုပ်နိုင်သည်။

```
class Point: # Point class ကို define လုပ်သည်။
    pass

p1 = Point() # Instance object p1 ကို define လုပ်သည်။
p2 = Point() # Instance object p2 ကို define လုပ်သည်။

p1.x = 5  # p1 ၏ Instance variable x ကို assign လုပ်သည်။
p1.y = 4  # p1 ၏ Instance variable y ကို assign လုပ်သည်။

p2.x = 3  # p2 ၏ Instance variable x ကို assign လုပ်သည်။
p2.y = 6  # p2 ၏ Instance variable y ကို assign လုပ်သည်။

print(p1.x, p1.y) # p1 ၏ attribute (ဒေတာ)ကို print ထုတ်ကြည့်သည်။
print(p2.x, p2.y) # p2 ၏ attribute (ဒေတာ)ကို print ထုတ်ကြည့်သည်။
```

```
5 4
3 6
```

Class instance p1 နှင့် p2 ထဲသို့ attribute value များ(data)များ ထည့်နိုင်ပြီ ဖြစ်သည်။ သို့သော် မည်သည့် အလုပ်မျိုးမှ မလုပ်နိုင်သေးပါ။ Class instance များကို မိမိ အလိုရှိသည့် အလုပ်တစ်ခုခု လုပ်ပေး စေလိုလျှင် class ထဲတွင် method သို့မဟုတ် behavior များ ရှိရမည်။

Car class တွင် ကားမောင်းခြင်း(drive)၊ ရပ်ခြင်း(stop) စသည်တို့ သည် Car class ၏ method သို့မဟုတ် behavior ဖြစ်သည်။ Dog class တွင် ခွေးဟောင်ခြင်း၊ ပြေးခြင်း စသည်တို့သည် Dog class ၏ method သို့မဟုတ် behavior ဖြစ်သည်။

အပေါ်က Point class တွင် pint တစ်ခုကို မူလ(origin) နေရာသို့ ပြန်သွားစေခြင်း၊ ရွှေ့ပေး လိုက်ခြင်းသည် method သို့မဟုတ် behavior ဖြစ်သည်။ (moves the point to the origin)။ Method ၏ နာမည်ကို reset ဟု ပေးမည်။

Python တွင် class ၏ method သည် function ပင် ဖြစ်သည်။ (A method in Python is identical to defining a function.) အဓိက ကွာခြားချက်မှာ self ဆိုသည့် argument မဖြစ်မနေ ထည့်ပေး ရခြင်း ဖြစ်သည်။

self ဆိုသည့် argument အပြင် တခြားသော parameters (argument)များ ထည့်ပေးရသည့် method များ ရှိနိုင်သလို parameters (argument)မထည့်ပေးရသည့် method များလည်း ရှိနိုင်သည်။ အောက်တွင် reset method ကို define လုပ်ပြီး သရုပ်ပြထားသည်။

Method တွင် self ဆိုသည့် argument ထည့်ပေးရခြင်းသည် သက်ဆိုင်သည့် object ကို ရည်ညွှန်းရန် (reference လုပ်ရန်) အတွက် ဖြစ်သည်။ ဥပမာ p1 ဆိုသည့် instance obejct အတွက် ဖြစ်လျှင် self argument များ အားလုံးသည် ထို p1 ကို ရည်ညွှန်းသည်။ Reference လုပ်သည်။

`reset()` method သည် ၎င်းထဲသို့ self argument မှ လွဲ၍ မည်သည့် argument မျှ ထည့်ပေးရန် မလိုသည့် method မျိုးဖြစ်သည်။

```
class Point:
    def reset(self):
        self.x = 0
        self.y = 0

p = Point()
p.reset()
print(p.x, p.y)
```

```
0 0
```

အထက်က ဥပမာတွင် p.reset() method ကို call လုပ်သည့်အခါ self argument ကို ထည့်ရေး ပေးရန် မလိုပါ။ **Python က self argument ကို အလိုအလျောက် ထည့်ပေးသည်။**

p.reset() method ကို call လုပ်သည့်အခါ p object ကို ရည်ညွှန်းမှန်း သိထားသည့်အတွက် Python က self argument ကို အလိုအလျောက် ထည့်ပေးပြီး ဖြစ်သည်။

Method တစ်ခုသည် function တစ်ခု ဖြစ်သည်။ သို့သော် class တစ်ခုထဲတွင် ပါဝင်နေသည့် function ဖြစ်သည်။ ထို့ကြောင့် object တစ်ခုအတွက် method ကို call လုပ်သည့်အခါ အောက်ပါအတိုင်း အရှည် ဖြန့်ရေးနိုင်သည်။ Class ထဲမှ function ကို invoke လုပ်သကဲ့သို့ အရှည် ဖြန့်ရေးနိုင်သည်။

```
p = Point()
Point.reset(p)  # အရှည် ဖြန့်ရေးသည်နည်း
print(p.x, p.y)
```

`Point.reset(p)` သည် Point class ထဲမှာ method ဆိုသည့် `reset()` function ကိုခေါ်ပြီး `instance object p` ကို argument အဖြစ် ထည့်ပေးသည်။ `p.reset()` နှင့်အတူတူပင် ဖြစ်သည်။

Point class ကို define လုပ်စဉ် self argument ကို ထည့်မပေးပါက မည်သို့ ဖြစ်မည်နည်း။

```
class Point:
    def reset():  # ဥပမာ အဖြစ် သရုပ်ပြရန် self argument ကို ချန်ထားသည်။
        pass

p = Point()
p.reset()
```

```
TypeError         Traceback (most recent call last)
<ipython-input-7-0ec399eeff4a> in <module>
      4
      5 p = Point()
----> 6 p.reset()

TypeError: reset() takes 0 positional arguments but 1 was given
```

အဓိပ္ပာယ်မှာ `reset()` ထဲသို့ မည်သည့် argument ထည့်ပေးရန် မလိုပါ။ သို့သော် **p.reset()** တွင် argument တစ်ခု ထည့်ပေးထားသည်။ မည်သူ့ ထည့်ပေးသည့် မည်သည့် argument နည်း?

Python က အလိုအလျောက် ထည့်ပေးသည့် self argument ဖြစ်သည်။

Class တစ်ခုအတွင်းရှိ method များကို define လုပ်သည့်အခါ self argument ထည့်ပေးရမည်။ Method တိုင်းတွင် self argument ထည့်ပေးရန် မဖြစ်မနေ လိုအပ်သည်။

self ဟုသည့် စာလုံး မသုံးဘဲ မိမိ နှစ်သက်သည့် နာမည် ပေးနိုင်သည်။ Python programmer အားလုံးနီးပါးက self ဟုသာ သုံးကြသည်။

```
class Point:
    def reset(self):
```

```
        self.x = 0
        self.y = 0
        print("print self: ", id(self))

p = Point()
p.reset()
print("print self: ", id(p))
```

```
print self:  2901267355912
print self:  2901267355912
```

အထက်က ဥပမာတွင် `reset()` method သည် instance object p ရည်ညွှန်းနေသည် ဖြစ်သောကြောင့် `reset()` method အတွင်းရှိ self ၏ Id နှင့် p ၏ Id တို့မှာ အတူတူဖြစ်သည်။ ဆိုလိုသည်မှာ object p နှင့် self တို့သည် အတူတူပင် ဖြစ်သည်။ p ကို method အတွင်း၌ self ဖြင့် refer လုပ်သည်။

Pass multiple arguments to a method

Method တစ်ခုထဲသို့ argument များ ထည့်ပေးခြင်း

```
import math
class Point:
    def move(self, x, y):
        self.x = x
        self.y = y

    def reset(self):
        self.move(0, 0)

    def calculate_distance(self, other_point):
        return math.sqrt((self.x - other_point.x)**2 +
                         (self.y - other_point.y)**2)
# how to use it:
point1 = Point()

point2 = Point()
point1.reset()
point2.move(5,0)

print(point2.calculate_distance(point1))

assert (point2.calculate_distance(point1) ==
point1.calculate_distance(point2))

point1.move(3,4)
```

```
print(point1.calculate_distance(point2))
print(point1.calculate_distance(point1))
```

```
5.0
4.47213595499958
0.0
```

Instance Variabel နှင့် Local Variable

```
class Person:
    def set_details(self, name, age):
        self.name = name
        self.age = age

p1 = Person()
p2 = Person()
p1.set_details("Aye", 49)
p2.set_details("Lin", 12)

print(p1.name, p1.age)
print(p2.name, p2.age)
```

```
Aye 49
Lin 12
```

အထက်က ဥပမာတွင် `self.name` နှင့် `self.age` တို့သည် instance variable များ ဖြစ်ကြသည်။ name နှင့် age တို့သည် `set_details()` ၏ parameter များ ဖြစ်ကြသည်။ name နှင့် age တို့သည် `set_details()` method အတွင်းရှိ local parameter(local variable) များ ဖြစ်ကြသည်။

name နှင့် age ကို `set_details()` method အတွင်း၌သာ အသုံးပြုနိုင်သည်။ **self.name** နှင့် self.age တို့ကို class အတွင်း မည်သည့်နေရာတွင် မဆို အသုံးပြုနိုင်သည်။ Instance variabel များသည် instance object များကို attach လုပ်ထားသောကြောင့် ဖြစ်သည်။ Instance object များ တည်ရှိ နေသမျှ ကာလပတ်လုံး `self.name` နှင့် `self.age` စသည့် instance variabel များ တည်ရှိနေလိမ့်မည်။

Method များကို class အတွင်း၌ def statement ကိုသုံး၍ define လုပ်နိုင်သည်။ Method definition တွင် မရှိမဖြစ် ပါရမည့် ပထမဆုံး parameter မှာ self ဖြစ်သည်။

Class အတွင်းမှ method ကို call လုပ်သည့်အချိန်တွင် self argument ကို python က အလိုအလျောက် ထည့်ပေးသည်။

Class အတွင်းမှ method အတွင်း၌ မည်သည့် instance variable ကို မဆို အသုံးပြုလိုပါက instance variable name ၏ အရှေ့(prefix)တွင် **self.** ကို ထည့်ပေးရသည်။ ဥပမာ- **self.name** instance variable အရှေ့တွင် `self.` ထည့်ရန် မေ့ကျန်ခဲ့ပါက ထို instance variable သည် update မလုပ်ခြင်းမျိုး ကြုံတွေ့ရနိုင်သည်။

Class အတွင်းမှ method တစ်ခုကို တခြားသော method တစ်ခုမှ လှမ်း၍ call လုပ်လျှင် ထိုအခေါ်ခံရသည့် method ၏ နာမည်ရှေ့တွင် **self.** ကို ထည့်ပေးရသည်။ မေ့ကျန်ခဲ့ပါက ထို call လုပ်သည့် method ကို define မလုပ်ရသေးပါဟု error message ပေးလိမ့်မည်။ Method ကို အမှန်တကယ် define လုပ်ထားခဲ့ပြီး ဖြစ်သော်လည်း method နာမည်အရှေ့အတွင် **self.** ထည့်ရန် မေ့ကျန်ခဲ့သောကြောင့် error message ပေးခြင်း ဖြစ်သည်။

Class ၏ အပြင်ဘက်တွင် method ၏ နာမည်ရှေ့တွင် instance object name ထည့်ရေးရသည်။ ဥပမာ- `p1.reset()` Class ၏ အပြင်ဘက်တွင် ဖြစ်သောကြောင့် reset() ဆိုသည့် method name ၏ အရှေ့တွင် instance object name ရှိသည်။

```
import math

class Point:
    def move(self, x, y):
        self.x = x
        self.y = y

    def reset(self):
        self.move(0, 0)

    def calculate_distance(self, other_point):
        return math.sqrt(
        (self.x - other_point.x)**2 +
        (self.y - other_point.y)**2)
# how to use it:
point1 = Point()

point2 = Point()
point1.reset()
point2.move(5,0)
```

__init__() method

`__init__()` method သည် object create လုပ်ပြီးသည်နှင့် တစ်ပြိုက်နက် execute လုပ်သည်။ `__init__()` method ကို သီးသန့် call လုပ်စရာ မလိုအောင် python က အလိုအလျောက် လုပ်ပေးသည်။ တနည်းအားဖြင့် object များ တည်ဆောက်ပြီးသည်အခါ `__init__()` method ကို အလိုအလျောက် call လုပ်စရာ မလိုဘဲ python က အလိုအလျောက် လုပ်ပေးသည်။

ထိုသို့ python က အလိုအလျောက် လုပ်ပေးသည့် အလုပ်ကို initialization work ဟုခေါ်သည်။ ဤနည်းဖြင့် instance variabel များကို create လုပ်ပြီး အလိုအလျောက် initialize လုပ်နိုင်သည်။

`__init__()` method ဖြင့် တခြားသော မိမိလုပ်လိုသည့် အလုပ်များကို အလိုအလျောက် လုပ်နိုင်သည်။ ဖိုင်ဖွင့်ခြင်း(opening file), network ချိတ်ခြင်း(setting up network connection) စသည့် အလုပ်များကို အလိုအလျောက် လုပ်ပေးနိုင်သည်။

`init()` method ၏ ပထမဆုံး parameter သည် self ဖြစ်သည်။ self နောက်တွင် လိုအပ်သည့် တခြား parameter များပို ထည့်ပေးနိုင်သည်။

Java သို့မဟုတ် C++ စသည့် OOP language များမှ constructor နှင့် တူသည်ဟု မထင်စေလိုပါ။ `__init__()` method သည် constructor မဖြစ်နိုင်ပါ။ Python မှ `__init__()` method ကို call လုပ်သည့်အချိန်တွင် obejct ကို တည်ဆောက်ပြီး ဖြစ်နေသည်။ (Obejct is already constructed , by the time `__init__()` method is called)

Python မှ `__init__()` method သည် initialization method သာ ဖြစ်သည်။ အသစ် တည်ဆောက်ပြီးခါစ instance obeject တစ်ခုအတွက် ပထမဆုံးနှင့် အရင်ဆုံး call လုပ်သည့် method မှာ __init__() method ဖြစ်သည်။ `__init__()` method သည် တည်ဆောက်ပြီးသည့် instance obeject ကို initialze လုပ်ပေးခြင်းသာ ဖြစ်သည်။

Python တွင် class တစ်ခုအတွင်း၌ initialization method တစ်ခုသာ ရှိနိုင်သည်။ Python တွင် function overloading သဘောတရား(concept) မရှိပါ။

Data Hiding

OOP တွင် data(attribute) နှင့် method() များကို private နှင့် public ဟူ၍ အမျိုးအစား နှစ်မျိုး ခွဲခြားထားသည်။ Private data နှင့် Private method များကို class အတွင်း၌သာ အသုံးပြုနိုင်သည်။ (can be used only inside the class)။ Public data နှင့် public method များကို class အပြင်ဘက်မှ လှမ်း၍ access လုပ်နိုင်သည်။ Python တွင် private နှင့် public သဘောတရား(concept)ကို အသုံးမပြုပါ။

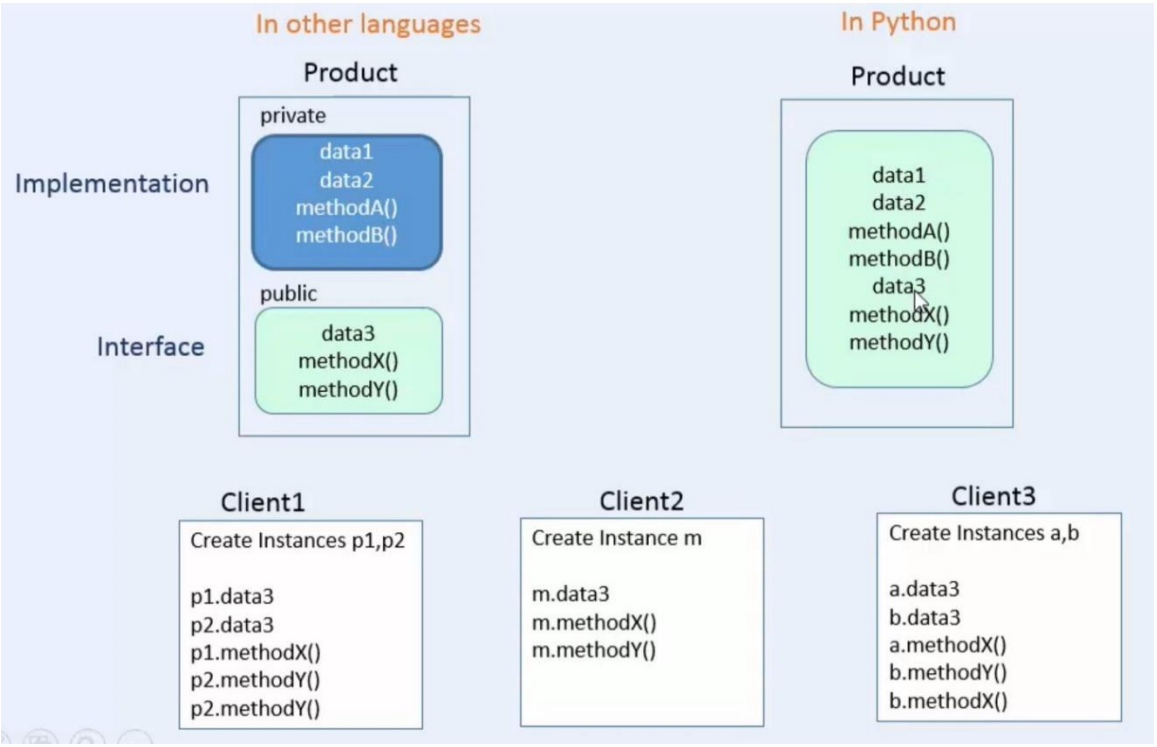
Class များ တည်ဆောက်ပြီး အသုံးချရန်အတွက် class များ အကြောင်းကို လေ့လာသည့်အခါ private နှင့် public သဘောတရား (concept)ကို သိထားရန် လိုသည်။

Class ကို အသုံးပြုသည့်သူများ(users)ကို client ဟုခေါ်သည်။ Client သို့မဟုတ် class ကို အသုံးပြုသည့်သူများ(users)သည် object များ တည်ဆောက်(create)ခြင်းဖြင့် အသုံးပြုကြသည်။ (uses the class by instantiating)

Private data နှင့် private method များကို client နှင့် user တို့က အသုံးပြုခွင့် မရကြပါ။ ဥပမာအားဖြင့် ကားတစ်စီးတွင် စတီယာတိုင်နှင့် ဘရိတ်တို့ကိုသာ မောင်းသူက အသုံးပြုခွင့် ရှိသည်။ ကား၏ Electronic Fuel Injection System မှ setting များကို ကားမောင်းသူက သိရန် မလိုအပ်သလို ပြောင်းနိုင် ခွင့်လည်း မရှိပါ။

Python ကို သုံးသူများ အားလုံးသည် တာဝန်ယူနိုင်သူများ ဖြစ်ကြသည်ဟု ယူဆထားသောကြောင့် private နှင့် public ဟူ၍ တင်းတင်းကြပ်ကြပ် ခွဲခြား တားမြစ်ထားခြင်း မရှိပါ။ Data များနှင့် method များ အားလုံးသည် public သာ ဖြစ်သည်။ Debugging လုပ်သည့်အခါ အဆင်ပြေစေရန်အတွက် ရည်ရွယ်၍

private များကို တင်းကြပ်စွာ ကန့်သတ် မထားခြင်းလည်း ဖြစ်သည်။ တစ်ခါတစ်ရံ debugging လုပ်ရန် အတွက် private attributes(data များ) နှင့် method များကို access လုပ်ခွင့် ရရန်လိုအပ်သည်။



Non Public

Attributes(data များ) နှင့် method များသည် non public များ ဖြစ်ကြောင်းသိသာစေရန် single underscore ကို variable name ၏ အရှေ့တွင် ထားရေးလေ့ရှိသည်။ ဥပမာ- _age

Variable name ၏ အရှေ့တွင် single underscore ထည့်ရေးလျှင် ၄င်းတို့သည် class အတွင်း၌သာ အသုံးပြုရန် ဖြစ်သည်။ Class အပြင်ဘက်၌ အသုံးမပြုသင့်ပါ။

Variable name ၏ အရှေ့တွင် double underscores ထည့်ရေးလျှင် class ၏ အပြင်ဘက်မှ တိုက်ရိုက် မမြင်တွေ့နိုင်ပါ။(will not be directly visible from outside the class)

double underscores ဖြင့် ရေးထားလျှင် Name mangling ပြုလုပ် ထားသည်။ ဥပမာ __value သည် _MyClass__value ဖြစ်သည်။ တိုက်ရိုက် access မလုပ်နိုင်သော်လည်း တဆင့်ခံနည်းဖြင့်(indirectly) access လုပ်နိုင်သည်။

Variable name ၏ အရှေ့တွင် double underscores နှင့် အဆုံးတွင် double underscores ထည့်ရေးလျှင် ၄င်း variable သည် Python internal use အတွက် ဖြစ်သည်။ ၄င်းတို့ကို ပြင်ဆင်ရေးသားခွင့် မရှိပါ။

Variable name ၏ အဆုံးတွင် single underscore ထည့်ရေးလျှင် ၄င်းတို့သည် Python built in name များနှင့် မရောထွေးစေရန်၊ name clashe မဖြစ်စေရန်အတွက် ဖြစ်သည်။

သင့် ပရိုဂရမ်ထဲတွင် class နှင့် range ကို variable name အဖြစ် သုံးလိုပါက class နှင့် range ၏ နောက်တွင် single underscore ထည့်ရေးနိုင်သည်။ ဥပမာ- class_ နှင့် range_

```
class Product:
    def __init__(self):
        self.data1 = 10
        self._data2 = 20

    def methodA(self):
        pass

    def _methodB(self):
        pass
p = Product()
p.data1
```

```
10
```

```
p.methodA()
p._data2
```

```
20
```

```
p._methodB()
class Product:
    def __init__(self):
        self.data1 = 10
        self.__data2 = 20 # double underscores အဖြစ် ပြောင်းသည်။

    def methodA(self):
        pass

    def __methodB(self): # double underscores အဖြစ် ပြောင်းသည်။
        pass
p = Product()
p.data1
```

```
10
```

```
p._methodB()
```

```
AttributeError                Traceback (most recent call last)
<ipython-input-6-61dd07b67969> in <module>
----> 1 p._methodB()

AttributeError: 'Product' object has no attribute '_methodB'
```

methodB ကို double underscores အဖြစ်ပြောင်းလိုက်သောကြောင့် call လုပ်၍ မရတော့ပါ။

```
print(dir(p))
```

```
['_Product__data2', '_Product__methodB', '__class__', '__delattr__', '__dict__'
, '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '
__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__mo
dule__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__set
attr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'data1', '
methodA']
```

Attribute နှစ်ခု ဖြစ်သည့် **'_Product__data2'** နှင့် **'_Product__methodB'** တို့ကို mangled လုပ်ထားသည်။ __data2 ၏ အရှေ့တွင် classname နှင့် underscore တစ်ခု ထည့်ထားသည်။

ထို့ကြောင့် data2 နှင့် methodB ကို access လုပ်လိုပါက '_Productdata2' နှင့် '_ProductmethodB' အဖြစ် ခေါ်ရသည်။

```
p._Product__methodB()
p._Product__data2
```

```
20
```

Variable name ၏ အရှေ့တွင် double underscores ထည့်ရေးလိုက်သောကြောင့် class အပြင်ဘက်တွင် တိုက်ရိုက် access လုပ်ခွင့် မရှိတော့ပါ။ သို့သော် class အတွင်း၌ တိုက်ရိုက် အသုံးပြု နိုင်သည်။

Property ကို သုံး၍ Validation Code လုပ်ခြင်း

အောက်က code သည် သာမာန် class တစ်ခုသာ ဖြစ်သည်။ အသက်(age)နှင့် နာမည်(name) ကို လက်ခံသည် Person class တစ်ခု ဖြစ်သည်။

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(self.name,self.age)

if __name__ == '__main__':
    p = Person('Raj', 30)

p.age = 100
p.display()
```

```
Raj 100
```

အသက်(age)ထည့်ပေးသည့်အခါ မည်သည့်ကိန်းကို မဆို လက်ခံသည်။ age ထည့်ပေးသည့်အခါ (assign လုပ်သည့်အခါ) value သည် 20 မှ 80 အတွင်းသာ လက်ခံရန် ကန့်သတ်ထားသင့်သည်။ ထိုသို့

ကန့်သတ်ရန် နည်းတစ်ခုမှာ age ကို private variable အဖြစ် သတ်မှတ်ပြီး validation လုပ်သည့် code ကို ထည့်ပေးနိုင်သည်။ Validation code မှာ အောက်ပါအတိုင်း ဖြစ်သည်။

```
def set_age(self, new_age):
    if 20 <new_age< 80:
        self._age = new_age
    else:
        raise ValueError('Age must be between 20 and 80')
```

အသက် (၂၀)မှ (၈၀) အတွင်းဖြစ်လျှင်(**20 <new_age< 80**) လက်ခံသည်။ ထိုသို့မဟုတ်လျှင် ValueError ထုတ်ပေးမည်။

```
self._age = age
```

Private variable ဖြစ်အောင် ပြုလုပ်ပြီး data validation လုပ်သည့် code ထည့်မည်။ age ဆိုသည့် variabel name အရှေ့တွင် underscore ထည့်ပြီး private variable ဖြစ်အောင် ပြုလုပ်နိုင်သည်။

set_age method ကို define လုပ်မည်။ age ထည့်ပေးသည့် အခါ(assign လုပ်သည့်အခါ) value သည် 20 မှ 80 အတွင်း မဟုတ်ခဲ့လျှင် ValueError ထုတ်ပေးမည်။

_age သည် private variable ဖြစ်သောကြောင့် client သည် တိုက်ရိုက် access မလုပ်သင့်ပါ။ **get_age** method ကိုလည်း define လုပ်မည်။ User က age ပြောင်းလိုလျှင် **set_age** method ကို အသုံးပြုလိမ့်မည်။ ထိုအခါ data validation လုပ်နိုင်သည်။

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self._age = age

    def display(self):
        print(self.name, self._age)

    def set_age(self, new_age):
        if 20 <new_age< 80:
            self._age= new_age
        else:
            raise ValueError('Age must be between 20 and 80')

if __name__ == '__main__':
    p = Person('Raj', 30)
```

အောက်တွင် အသက်ကို နှစ်(၁၀၀) ဟုထည့်၍ ValueError ထုတ်ပေး မပေး စစ်ကြည့်မည်။

```
p.set_age(100)
```

```
ValueError                          Traceback (most recent call last)
<ipython-input-4-d6b15ca28bbd> in <module>
----> 1 p.set_age(100)
<ipython-input-2-c0b1013d1d95> in set_age(self, new_age)
     11             self._age= new_age
     12         else:
---> 13         raise ValueError('Age must be between 20 and 80')
     14
     15 if __name__ == '__main__':

ValueError: Age must be between 20 and 80
```

အောက်တွင် အသက်ကို (၁၀)နှစ် ဟုထည့်၍ ValueError ထုတ်ပေး မပေး စစ်ကြည့်မည်။

```
p.set_age(10)
```

```
ValueError                          Traceback (most recent call last)
<ipython-input-5-a73f8f80f7d0> in <module>
----> 1 p.set_age(10)

<ipython-input-2-c0b1013d1d95> in set_age(self, new_age)
     11             self._age= new_age
     12         else:
---> 13         raise ValueError('Age must be between 20 and 80')
     14
     15 if __name__ == '__main__':

ValueError: Age must be between 20 and 80
```

အောက်တွင် အသက်ကို (၂၅)နှစ် ဟုထည့်၍ ထည့်ပေးပြီး ValueError ထုတ်ပေး မပေး စစ်ကြည့်မည်။ သတ်မှတ်ထားသည့် နှစ်အတွင်း ဖြစ်သောကြောင့် လက်ခံသည်။

```
p.set_age(10)
p.display()
```

```
Raj 25
```

အကယ်၍ user က အသက်ကို တစ်နှစ်တိုးမည်ဟု ပြောင်းလိုလျှင် current age ကို ရရန်အတွက် **get_age** method ကို သုံးရမည်။ Current age ကို (၁)နှစ်တိုးပြီး set_age method ဖြင့် re-assign လုပ်ရမည်။

```
# p.get_age()+1  # current age ဖတ်ယူပြီး (၁) တိုးသည်။
p.set_age(p.get_age()+1) # set_age method ဖြင့် re-assign လုပ်သည်။
```

```
p.display()
```

```
AttributeError                            Traceback (most recent call last)
<ipython-input-7-1da61f8e87c3> in <module>
      1 # p.get_age()+1  # current age ဖတ်ယူပြီး ၁ တိုးသည်။
----> 2 p.set_age(p.get_age()+1) #set_age method ဖြင့် reassign လုပ်သည်။
      3 p.display()

AttributeError: 'Person' object has no attribute 'get_age'
```

```
p.set_age(p.get_age()+1)
```

အပေါ်က current age ကို (၁) နှစ်တိုးသည့် code သည် အူကြောင်ကြောင် နိုင်လှသည်။ User က object အသစ်တစ်ခုကို create လုပ်ပြီး ကြိုက်သည့် အသက်ထည့်ပေးနိုင်သည်။ object အသစ်တစ်ခုကို create လုပ်သည့်အခါ အသက်ကို ကြိုက်သည့် ကိန်းထည့်နိုင်သေးသည်။ ထိုသို့ ဖြစ်ရခြင်းမှာ initializer ၌ မည်သည့် data validation မှ မပြုလုပ်ထားသောကြောင့် ဖြစ်သည်။ ဥပမာ-

```
# New object ပြုလုပ်ချိန်တွင် ကြိုက်သည့်အသက် ထည့်ပေးနိုင်သေးသည်။
p11 = Person("Aye", 300)
p11.display()
```

```
Aye 300
```

Object အသစ်တစ်ခု ပြုလုပ်ပြီး initializer တွင် data validation code ထည့်မည်။

```
class Person:
    def __init__(self, name, age):
        self.name = name
        if 20 < age < 80:
            self._age = age
        else:
            raise ValueError('Age must be between 20 and 80')

    def display(self):
        print(self.name,self._age)

    def set_age(self, new_age):
        if 20 <new_age< 80:
            self._age = new_age
        else:
            raise ValueError('Age must be between 20 and 80')

    def get_age(self):
        return self._age
```

```
# New object ပြုလုပ်ချိန်တွင် သတ်မှတ်ထားသည့် အသက်ကိုသာ လက်ခံလိမ့်မည်။
p1 = Person('Dev', 200)
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-10-88dd09e83c82> in <module>
----> 1 p1 = Person('Dev', 200)
<ipython-input-9-9c9fefd73b7f> in __init__(self, name, age)
      5             self._age = age
      6         else:
----> 7             raise ValueError('Age must be between 20and 80')
      8
      9     def display(self):

ValueError: Age must be between 20 and 80
```

Data validation code အလုပ် လုပ်သောကြောင့် အသက်ကို နှစ်(၂၀၀)ဟု ထည့်သည့်အခါ ValueError ထုတ်ပေးသည်။

Code သည် ကောင်းစွာ အလုပ်လုပ်သော်လည်း python စတိုင်မဟုတ်ပါ။(not Pythonic) တူညီသည့် code များကို နှစ်ကြိမ် နှစ်ခါ ရေးထားသည်။ နောက်ထပ် ပြဿနာ တစ်ခုမှာ it breaks the existing client code ဖြစ်သည်။

Attribute age ကို တိုက်ရိုက်(directly)access လုပ်ထားသည့် client code များ စွာရှိနေနိုင်သည်။ ထို့ client code များကို လိုက်ပြောင်း ရန် လိုအပ်သည်။

တိုက်ရိုက်(directly)access လုပ်ထားသည့် client code အားလုံး အလုပ်လိမ့်မည် မဟုတ်တော့ပေ။ instance variable သည် ယခင်က age မှ ယခု **_age** အဖြစ် ပြောင်းသွားသည်။ ထို့ကြောင့် client များက ယခင်က p.age ဟု ရေးထားသည့်နေရာအားလုံးတွင် **p.set_age()** ဟု ပြောင်းရေးရတော့မည်။ (new update is not backward compatible)

အောက်တွင် ဖော်ပြထားသည့် client code သည် အလုပ်လုပ်လိမ့်မည် မဟုတ်ပေ။

```
from person import Person
p = Person("Peter", 30)
p.age = 25
print(p.age)
```

p.age ဟု assign လုပ်ရန် ရေးထားသည့် နေရာများအားလုံးကို p.set_age ဟု လိုက်ပြောင်းပေး ရလိမ့်မည်။ print(p.age)ဟု ရေးထာသည့် နေရာများတွင် print(get_age()) ဟု လိုက်ပြောင်းပေး ရလိမ့်မည်။

Validation code ထည့်ရန် update လုပ်လိုက်ခြင်းကြောင့် backward compatible မဖြစ်တော့ပါ။ Interface ကို break လုပ်ခြင်း ဖြစ်သည်။ အကောင်းဆုံး ဖြေရှင်းနိုင်သည့်နည်းမှာ property တစ်ခု create လုပ်ခြင်း ဖြစ်သည်။

Property ကို အသုံးပြုသည့်နည်း

Attribute age တွင် data validation လုပ်ရန်အတွက် property ကို အသုံးပြုသည့်နည်းဖြစ်သည်။ **set_age** နှင့် **get_age** methods ကို သုံးမည့်အစား property ကို အသုံးပြုသည့်နည်း ဖြစ်သည်။ setter method ထပ်ထည့်မည်။ setter method တွင် data validation လုပ်ပါမည်။

ထိုအခါ age ကို instance variable အဖြစ် အလွယ်တကူ access လုပ်နိုင်ပြီး data validation လည်း လုပ်နိုင်သည်။

```
class Person:
    def __init__(self, name, age):
        self.name = name
        if 20 < age < 80:
            self._age = age
        else:
            raise ValueError('Age must be between 20 and 80')

    def display(self):
        print(self.name,self.age)

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, new_age):
        if 20< new_age<80:
            self._age = new_age
        else:
            raise ValueError('Age must be between 20 and 80')

if __name__ == '__main__':
    p = Person('Raj', 30)

print(p.age)
```

```
30
```

```
p.age = 34   # ၃၄ နှစ်ကို assign လုပ်သည်။
print(p.age)
```

```
34
```

```
p.age = 200
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-14-92f6dc1e610f> in <module>
----> 1 p.age =  200
<ipython-input-11-d0f63a773dbd> in age(self, new_age)
     19             self._age = new_age
     20         else:
---> 21             raise ValueError('Age must be between 20 and80')
     22
     23 if __name__ == '__main__':
ValueError: Age must be between 20 and 80
```

အသက် တစ်နှစ်တိုးလိုပါက အလွယ်တကူ လုပ်နိုင်သည်။

```
p.age += 1
print(p.age)
```

```
35
```

set_age and **get_age** method ကို အသုံးပြုခြင်းထက် property ကို အသုံးပြုခြင်းကြောင့် ပိုပြီး တိတိကျကျ(concise)နှင့် ရှင်းလင်းစွာ ရေးနိုင်သည်။ Existing client code ကိုလည်း ဘာမျှ ပြောင်းရန် မလိုတော့ပါ။ Backward compatible လည်း ဖြစ်သည်။

```
from person import Person
p = Person('Peter', 30)
p.age=27
print(p.age)
```

Data validation လည်းလုပ်နိုင်သည်။ Existing instance attribute ကို property အဖြစ် ပြောင်းလဲနိုင်သည်။

Read only or Write only Property

Property ကို အသုံးပြု၍ instance variable တစ်ခုကို read only သို့မဟုတ် write only အဖြစ် ကြိုက်သလို လုပ်နိုင်သည်။ setter method မပါဘဲ getter method ကိုသာသုံးလျှင် property သည် read only property သာ ဖြစ်သည်။ ဥပမာ တစ်ခုဖြင့် ရှင်းပြပါမည်။

```
class Employee:
    def __init__(self, name, password, salary):
        self._name = name
        self._password = password
        self._salary = salary

    @property
```

```
    def name(self):
        return self._name

    @property
    def password(self):
        raise AttributeError('password not readable')

    @password.setter
    def password(self, new_password):
        self._password = new_password

    @property
    def salary(self):
        return self._salary

    @salary.setter
    def salary(self, new_salary):
        self._password = new_salary
```

name, password နှင့် salary စသည်ဖြင့် property (၃)ခု ပါဝင်သည့် Employee class ဖြစ်သည်။ name ဆိုသည့် property တွင် getter method သာ ထည့်ထားသည်။ setter method မပါဝင်ပါ။ ထို့ကြောင့် name property သည် read only property သာ ဖြစ်သည်။ Attribute name သည် read only သာ ဖြစ်သည်။

Attribute password တွင် getter method သာ ရှိသောကြောင့် write only သာ ဖြစ်သည်။ read လုပ်လျှင် AttributeError ပေးလိမ့်မည်။

Property တစ်ခုတွင် getter နှင့် setter methods နှစ်ခုစလုံး ရှိလျှင် ထို property သည် read/write property ဖြစ်သည်။ attribute salary ကို referenced လည်းလုပ်နိုင်သည်။(read property) assign လည်း လုပ်နိုင်သည်။(write property)

```
ep1 = Employee("Lin", "pwdd21", 50000)
ep1.name  # .name သည် read only property ဖြစ်သည်။
```

```
'Lin'
```

```
ep1.name = "KoKo"
# assign လုပ်လျှင် သို့မဟုတ် write လုပ်လျှင် AttributeError ပေးလိမ့်မည်။
```

```
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-18-2a83174e5404> in <module>
```

```
----> 1 ep1.name = "KoKo"  # assign လုပ်လျှင် သို့မဟုတ် write လုပ်လျှင် Attri
buteError ပေးလိမ့်မည်။
```

```
AttributeError: can't set attribute
```

password attribute သည် readable property တစ်ခု မဟုတ်ပါ။ password not readable ဆိုသည့် error ပေးလိမ့်မည်။

```
ep1.password   # password ကို ဖတ်သည်။
```

```
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-19-1535e94f8324> in <module>
----> 1 ep1.password   # password ကို ဖတ်သည်။

<ipython-input-16-5df8eba9bbb3> in password(self)
     11     @property
     12     def password(self):
---> 13         raise AttributeError('password not readable')
     14
     15     @password.setter

AttributeError: password not readable
```

assign လုပ်နိုင်သည်။ write လုပ်နိုင်သည်။

```
ep1.password = "123qwer"
```

salary attribute သည် readable property ဖြစ်သလို writable property လည်း ဖြစ်သည်။ Property ကို အသုံးပြု၍ attribute များကို readable property ဖြစ်အောင်သော်လည်းကောင်း writable property ဖြစ်အောင် သော်လည်းကောင်း လုပ်နိုင်သည်။ နောက်ထပ် property နှင့် သက်ဆိုင်သည့် ဥပမာ တစ်ခုကို လေ့လာကြည့်ရအောင်

```
class Rectangle():
    def __init__(self,length,breadth):
        self.length = length
        self.breadth = breadth
        self.diagonal = (self.length*self.length + self.breadth * self.bre
adth)**0.5

    def area(self):
        return self.length * self.breadth
```

```
    def perimeter(self):
        return 2*(self.length + self.breadth)
```

Rectangle class တွင် length, breadth နှင့် diagonal စသည့် instance variable (၃)ခု ပါဝင်သည်။ Area method နှင့် perimeter method နှစ်ခု ပါသည်။ Diagonal သည် instance variable ဖြစ်သည်။ Diagonal value သည် instance variable ဖြစ်သည့် length နှင့် breadth တို့မှ တွက်ယူသည်။

```
# 2 နှင့် 5 သည် obejct အသစ်လုပ်စဉ် initialize လုပ်ပြီး ထည့်သည့် တန်ဖိုးဖြစ်သည်။
r = Rectangle(2,5)
print("Diagonal : ", r.diagonal)
print("Area : ", r.area())
print("Perimeter : ", r.perimeter())
```

```
Diagonal :  5.385164807134504
Area :  10
Perimeter :  14
```

```
# 10 သည် ရှိပြီးသား obejct ၏ length တန်ဖိုးကို ပြောင်းခြင်း ဖြစ်သည်။
r.length =  10
print("Diagonal : ", r.diagonal)
print("Area : ", r.area())
print("Perimeter : ", r.perimeter())
```

```
Diagonal :  5.385164807134504
Area :  50
Perimeter :  30
```

Length ၏ တန်ဖိုးကို ပြောင်းလိုက်သည့်အခါ Diagonal တန်ဖိုးသည် မပြောင်းသွားပါ။ Area နှင့် perimeter တို့၏ တန်ဖိုး ပြောင်းလဲသွားသည်။ area နှင့် perimeter တို့၏ တန်ဖိုးကို method ဖြင့် တွက်ယူသောကြောင့် ပြောင်းလဲသွားခြင်း ဖြစ်သည်။ Diagonal တန်ဖိုး မပြောင်းလဲပါ။

```
r.length =  10
```

အထက်က ရေးထားသည့် rectangle class တွင် length သို့မဟုတ် breadth တန်ဖိုးကို (r.length = 10 ဖြင့်)assign လုပ်လျှင် diagonal မပြောင်းပါ။

r.length ဖြင့် 10 ကို assign လုပ်သည့်အခါ iniatializer ဆီကို မရောက်ဘဲ တန်ဖိုး 10 သည် method များဆီကိုသာ တိုက်ရိုက်ရောက်သည်။ ပြောင်းလိုက်သည့် instance variable တန်ဖိုးသည် diagonal တန်ဖိုးဆီသို့ မရောက်သောကြောင့် အလိုအလျောက် update မလုပ်ခြင်း ဖြစ်သည်။

ထိုပြဿနာကို ဖြေရှင်းနိုင်သည့်နည်းတစ်ခုမှာ diagonal တန်ဖိုးကို method နည်းဖြင့် တွက်ယူခြင်း ဖြစ်သည်။ Method နည်းအဖြစ် ပြုလုပ်လိုက်လျှင် instance variable အဖြစ် access လုပ်ခွင့် မရှိတော့ပါ။ မလုပ်နိုင်တော့ပါ။ Method ဖြင့် access လုပ်လျှင် parentheses() ထည့်ပေးရသည်။

```
class Rectangle():
    def __init__(self,length,breadth):
```

```
        self.length = length
        self.breadth = breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2*(self.length + self.breadth)

    @property
    def diagonal(self):
        return (self.length*self.length+self.breadth * self.breadth)**0.5

r = Rectangle(2,5)
r.length =  10   # 10 သည် ရှိပြီးသား obejct ၏ length တန်ဖိုးကို ပြောင်းခြင်း ဖြစ်သည်။
print("Diagonal : ", r.diagonal)
print("Area : ", r.area())
print("Perimeter : ", r.perimeter())
```

```
Diagonal :  11.180339887498949
Area :  50
Perimeter :  30
```

Diagonal ကို instance variable အဖြစ် access လုပ်နေသည့် existing client code များလည်း break ဖြစ်လိမ့်မည်။ အကောင်းဆုံးနည်းမှာ property အဖြစ် ပြောင်းလိုက်ခြင်း ဖြစ်သည်။

Change it to a Property

Diagonal သည် property အဖြစ် ပြောင်းပေးသည့် code ဖြစ်သည်။

```
@property
    def diagonal(self):
        return (self.length*self.length+self.breadth * self.breadth)**0.5
```

Diagonal ကို property ဖြစ်အောင် ပြုလုပ်ပြီးနောက် instance variable အဖြစ် access လုပ်နိုင်သည်။ အလိုအလျောက် update လုပ်ပြီးသား တန်ဖိုးကို access လုပ်နိုင်သည်။

Diagonal ကို user က တိုက်ရိုက် ပြောင်းမည် မဟုတ်သောကြောင့် diagonal property အတွက် setter method ကို define လုပ်ရန် မလိုအပ်ပါ။

Deleter Method

Property တစ်ခုတွင် getter method နှင့် setter method တို့အပြင် deleter method ကိုလည်း define လုပ်နိုင်သည်။ Deleter method သည် property တစ်ခုခုကို delete လုပ်လိုက်ချိန်တွင် execute လုပ်ရမည့် အလုပ်များကို define လုပ်သည်။

```
@value.deleter
```

```
    def value(self):
        print('value deleted')
```

Deleter method ပြုလုပ်လိုပါက property နာမည်အတိုင်း method တစ်ခုကို define လုပ်ရပါမည်။ ဥပမာ value ဆိုသည့် property ကို delete လုပ်ရန် value ဆိုသည့် method နာမည်နောက်တွင် decorator **@value.deleter** ထည့်ပေးရပါမည်။ Delete လုပ်သည့်အချိန်တွင် ထို deleter method ကို execute လုပ်မည်။

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    @property
    def value(self):
        return self._x

    @value.setter
    def value(self, val):
        self._x = val

    @value.deleter
    def value(self):
        print('value deleted')

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, val):
        self._y = val

p = Product(12,24)
del p.value
```

```
value deleted
```

Summary of Property

Method တစ်ခုခုကို property အဖြစ် define လုပ်ပြီးသည်အခါ method syntax ကို မသုံးဘဲ instance variable အဖြစ် access လုပ်နိုင်သည်။

Instance variable တစ်ခုကဲ့သို့ reference လုပ်နိုင်သည့် assigne လုပ်နိုင်သည် သို့မဟုတ် delete လုပ်နိုင်သည့် method များကို property syntax ကို အသုံးပြီး define လုပ်နိုင်သည်။

Property တစ်ခုကို get လုပ်ရန်အတွက်

@property = to be replaced before the getter method

Property တစ်ခုကို set လုပ်ရန်အတွက်

@property.setter = to be replaced before the setter method

Property တစ်ခုကို delete လုပ်ရန်အတွက်

@property.deleter = to be replaced before the deleter method

Property အဖြစ် define လုပ်သည့်အခါ method များ အားလုံး၏ နာမည်သည် အတူတူ ဖြစ်သည်။ self ကို first argument အဖြစ် ထည့်ပေးရသည်။ setter method သည် set လုပ်ရန်အတွက် (assign လုပ်မည့်) argument တစ်ခု ထည့်ပေးရသည်။

Property ကို attribute တစ်ခုခု၏ အမျိုးအစား(type)ကို စစ်ရန်(check)အတွက်နှင့် validation လုပ်ရန်အတွက် အသုံးပြုသည်။

Property ကို define လုပ်သည့်အခါ read only သို့မဟုတ် write only စသည်ဖြင့် လိုသလို define လုပ်နိုင်သည်။

Property ကို သုံး၍ instance variable ကို dynamically calculated attribute အဖြစ် ပြောင်းလဲနိုင်သည်။

Property ကို သုံး၍ existing client code များကို ဘာမျှ ပြောင်းလဲရန် မလိုဘဲ instance variable များတွင် behaviour အသစ်များ ထပ်ထည့်နိုင်သည်။

Property ကို သုံး၍ existing instance variable များကို functionality အသစ်များ ထပ်ထည့်ပေးနိုင်သည်။

Class တစ်ခုတွင် instance variable များကို public အဖြစ် ရေးသားပြီး နောင်တစ်ချိန်တွင် control လုပ်လိုပါက private attribute အဖြစ် ပြောင်းလဲနိုင်သည်။ Property အဖြစ် define လုပ်ပြီး ထို instance variable များကို access လုပ်နိုင်သည်။ Existing client code များကို လိုက်ပြင်ရေး ရန် မလိုအပ်ပါ။

Attribute များကို Referencing လုပ်ရန် getter method ကို သော်လည်းကောင်း assigning လုပ်ရန် setter methodကို သော်လည်းကောင်း property အဖြစ် define လုပ်ပြီး သုံးနိုင်သည်။

အလွန်ရှုပ်ထွေးပြီး အချိန်ကြာမြင့်နိုင်သည့်(very complex and time consuming) အခြေအနေမျိုးတွင် property ကို အသုံးမပြုဘဲ normal method နည်းကို အသုံးပြုသင့်သည်။

Property

Class တွင် property တစ်ခုကို define လုပ်ပုံနှင့် ဘာကြောင့် property လိုအပ်သလဲဆိုတာကို ရှင်းပြထားသည်။ အောက်တွင် Point ဆိုသည့် class တစ်ခုတည်ဆောက်သည်။

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)
```

p ဆိုသည့် instance တစ်ခုကို define လုပ်သည်။ display method ဖြင့် instance variable နှစ်ခုကို ဖော်ပြခိုင်းသည်။

```
p = Product(12,24)
p.display()
```

```
12 24
```

Property တစ်ခုကို define လုပ်ရန်အတွက် class တွင် attribute အသစ်တစ်ခု ထပ်ထည့်သည်။ Instance variable _x ကို return ပြန်ပေးသည့် value ဆိုသည့် method ကို define လုပ်သည်။

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    def value(self):
        return self._x

p = Product(12,24)
p.value()
```

```
12
```

အောက်တွင် @property ဖြင့် value method ကို property ထည့်မည်။

```
@property
def value(self):
```

```
    return self._x
```

@property ထည့်ပြီးနောက် value ဆိုသည့် method သည် value ဆိုသည့် property ဖြစ်သွားသည်။

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    @property
    def value(self):
        return self._x
```

Value ဆိုသည့် method သည် value ဆိုသည့် property ဖြစ်သွားသောကြောင့် instance variable ကဲ့သို့ access လုပ်နိုင်သည်။ တနည်းအားဖြင့် **p.value()** ဟု မရေးဘဲ p.value အဖြစ်သာ ရေး၍ access လုပ်နိုင်သည်။

```
p = Product(12,24)
p.value
```

```
12
```

```
type(Product.value) # value အမျိုးအစားသည် property ဖြစ်သည်။
```

```
property
```

ယခု variable _x ၏ value ကို access လုပ်ခွင့်ရသွားပြီ ဖြစ်သည်။ Method ၏ code များ execute လုပ်ခိုင်းရန်အတွက် parentheses () ထည့်ပေးရန် မလိုတော့ပါ။

Method တစ်ခုကို property ဖြစ်အောင် define လုပ်ခြင်း ဖြင့် instance variable တစ်ခု ကဲ့သို့ access လုပ်နိုင်သည်။ @property သည် decorator တစ်ခုဖြစ်သည်။ decorator များအကြောင်းကို ရှင်းပြပြီး ဖြစ်သည်။

http://www.acmv.org/python/python_intro/Chapter_15_Decorators.pdf

Method တစ်ခုကို decorate လုပ်လိုက်ခြင်းဖြင့် ထို method ကို instance variable တစ်ခု ကဲ့သို့ access လုပ်နိုင်သည်။

```
p = Product(12,24)
p.value + 5
```

```
17
```

Property ဖြစ်သောကြောင့် p.value တွင် 5 ထည့်ပေါင်းနိုင်သည်။ **p.value** သည် instance variable တစ်ခု ဖြစ်သောကြောင့် ဖြစ်သည်။ **Method ဖြစ်သည့် p.value() တွင် 5 ထည့်ပေါင်းခြင်းကို တိုက်ရိုက် မပြုလုပ်နိုင်ပါ။**

```
p = Product(12,24)
p.value() + 5
```

```
TypeError                       Traceback (most recent call last)
<ipython-input-9-04f300e40f11> in <module>
      1 p = Product(12,24)
----> 2 p.value() + 5

TypeError: 'int' object is not callable
```

dir() function ကို သုံး၍ အသစ်ဖြစ်ပေါ်လာသည့် attribute အသစ်၏ အချက်အလက်များကို ကြည့်နိုင်သည်။

```
print(dir(Product))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__for
mat__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init
_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce_
_', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__sub
classhook__', '__weakref__', 'display', 'value']
```

Value ဆိုသည့် attribute ကို တန်ဖိုး တစ်ခုခု assign လုပ်ကြည့်လျှင် attribute error ပေးလိမ့်မည်။ 10 ကို p.value တွင် assign လုပ်ကြည့်သည်။

```
p.value = 10
```

```
AttributeError                  Traceback (most recent call last)
<ipython-input-11-86bbee8037b6> in <module>
----> 1 p.value = 10

AttributeError: can't set attribute
```

Value ဆိုသည့် attribute အသစ်တစ်ခု ဖြစ်လာသော်လည်း reference သာလုပ်နိုင်သည် assign မလုပ်နိုင်ပါ။ ထို့ကြောင့် attribute value တစ်ခု assign လုပ်ရန်အတွက် method တစ်ခု ထပ်ရေးရပါမည်။

Method နာမည်ကို value ဟုသာ ပေးပါသည်။ (ပြီးခဲ့ property ဖြစ်ခဲ့သည့် method နာမည်နှင့် တူနေလိမ့်မည်။) self ဆိုသည့် parameter တစ်ခု ထည့်ပေးရသည်။

@value.setter ဆိုသည့် ထည့်သည်။ Attribute value ကို တစ်ခုခု assign လုပ်လျှင် အောက်က code များ executer လုပ်သည်။

```
@value.setter
def value(self, val):
self._x = val
```

Assign လုပ်ရန် ထည့်ပေးလိုက်သည့် value သည် argument တစ်ခု အဖြစ် ရောက်သွားသည်။ method ၏ second parameter အဖြစ် ရောက်သွားသည်။ First parameter သည် self ဖြစ်သည်။

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    @property
    def value(self):
        return self._x

    @value.setter
    def value(self, val):
        self._x = val

p = Product(12,24)
p.value = 10
p.value
```

```
10
```

Value ဟုနာမည် ပေးထားသည့် attribute ကို reference လုပ်လျှင် အောက်က code (၃)ကြောင်း execute လုပ်သည်။ Getter method ဟုခေါ်သည်။

```
@property
def value(self):
    return self._x
```

value ဟုနာမည် ပေးထားသည့် attribute ကို တန်ဖိုးတစ်ခုခု assign လုပ်လျှင် အောက်က code (၃)ကြောင်း execute လုပ်သည်။ Setter method ဟုခေါ်သည်။ Setter method သည် assign လုပ်မည့် တန်ဖိုး တစ်ခုခုကို argument အဖြစ် လက်ခံပေးရသည်။

```
@value.setter
def value(self, val):
self._x = val
```

နောက်ထပ် y အတွက် property နှစ်ခုထပ်ပြီး define လုပ်ကြရအောင်။ y အတွက် setter method

```
@property
def y(self):
```

```
return self._y
```

y အတွက် setter method

```
@y.setter
def y(self, val):
self._y = val
```

Variable **_x** အတွက် getter method ဖြင့် referencing လုပ်သည်။ Setter method ဖြင့် assign လုပ်သည်။ ထို့အတူ variable **_y** အတွက် getter method ဖြင့် referen လုပ်သည်။ Setter method ဖြင့် assign လုပ်သည်။

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    @property
    def value(self):
        return self._x

    @value.setter
    def value(self, val):
        self._x = val

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, val):
        self._y = val
```

Class Variable

Instance variable create လုပ်ပုံနှင့် အသုံးပြုပုံကို ရှင်းပြခဲ့ပြီး ဖြစ်သည်။ မတူညီသည့် instance object အမျိုးမျိုးတွင် variable များ ပါရှိနေသည်။ Instance object ၏ variable တန်ဖိုးများ အမျိုးမျိုး ကွဲပြားနေနိုင်သည်။

Method အတွင်းတွင် self parameter ဖြင့် instance variable များကို create လုပ်နိုင်သည်။ reference လုပ်နိုင်သည်။ အခု class variable အကြောင်း ရှင်းပြမည်။

Class level တွင် assign လုပ်ထားသည့် variable သည် Class Variable ဖြစ်သည်။ Class အတွင်း၌ ရှိပြီး method များ၏ အပြင်ဘက်တွင် ရှိလျှင် class variable သို့မဟုတ် class attribute ဖြစ်သည်။

အောက်က ဥပမာတွင် species သည် class variable ဖြစ်သည်။

```
class Person:
    species = 'Homo sapiens'

    def __init__(self,name,age):
        self.name = name
        self.age = age

    def display(self):
        print(f'{self.name} is {self.age} years old')
```

Person class တွင် species ဆိုသည့် class variable ကို define လုပ်သည်။ species ဟု နာမည်ပေးထားသည့် variable တစ်ခုသည် class အတွင်းနှင့် method များအားလုံး၏ အပြင်ဘက်တွင် define လုပ်သည်။ ထို့ကြောင့် species သည် class variable တစ်ခု ဖြစ်သည်။

Class variable ကို class အတွင်းရှိ instance များ အားလုံးက အသုံးပြုနိုင်သည်။ Class level တွင် ရှိသောကြောင့် class အတွင်းရှိ instance များအားလုံးက သုံးနိုင်သည်။

Class variable ၏ data ကို class ထဲတွင်သာ သိမ်းဆည်းထားသည်။ Instance variable တစ်ခုချင်းစီ၏ data ကို instance object တစ်ခုချင်းစီ၌ သိမ်းဆည်းထားသည်။

Person object များအားလုံးအတွက် species တွင် assign လုပ်ထားသည့် တန်ဖိုးသည် အတူတူပင်ဖြစ်သည်။ Instance တစ်ခု ချင်းစီအတွက် သီးသန် မတူညီသည့် တခြားတန်ဖိုး တစ်ခုခု မရှိပါ။ အားလုံးအတွက် ဖြစ်သောကြောင့် class variable အဖြစ် define လုပ်ခြင်း ဖြစ်သည်။

Class variable များကို class definition ၏ အပေါ်ဆုံး class header ၏ အောက်နား၌ define လုပ်ပြီး ရေးသားလေ့ရှိသည်။

Class variable ကို class name ဖြင့်သာမက instance name ဖြင့်ပါ နှစ်မျိုးလုံးနှင့် access လုပ်နိုင်သည်။

```
class Person:
    species = 'Homo sapiens'

    def __init__(self,name,age):
        self.name = name
        self.age = age
```

```
    def display(self):
        print(f'{self.name} is {self.age} years old')

p1 = Person('John',20)
p2 = Person('Jack',34)
p1.display()
p2.display()
```

```
John is 20 years old
Jack is 34 years old
```

```
print(Person.species) # class name ဖြင့် access လုပ်သည်။
print(p1.species)  # instance name ဖြင့် access လုပ်သည်။
print(p2.species)  # instance name ဖြင့် access လုပ်သည်။
```

```
Homo sapiens
Homo sapiens
Homo sapiens
```

species သည် အားလုံးအတွက် (class အတွက်သာမက instance များအားလုံးအတွက်) ဖြစ်သောကြောင့် class variable ကို class name ဖြင့်လည်း access လုပ်နိုင်သည်။ instance name ဖြင့်လည်း access လုပ်နိုင်သည်။

Class name ကို အသုံးပြုပြီး instance variable များကို access လုပ်ခွင့် မရှိပါ။ ဥပမာ- Person.name ဖြင့် access လုပ်ခွင့် မရှိပါ။

```
# Person class တွင် name attribute မရှိသောကြောင့် access လုပ်မရနိုင်ပါ။
print(Person.name)
```

```
AttributeError                            Traceback (most recent call last)
<ipython-input-14-95ffcc93b984> in <module>
      1
----> 2 print(Person.name)

AttributeError: type object 'Person' has no attribute 'name'
```

id() function ဖြင့် စစ်ကြည့်လျှင် variable တစ်ခုတည်းကို ရည်ညွှန်းထားသည်(reference လုပ်ထားသည်) ကို တွေ့ရလိမ့်မည်။ Class object တွင် သိမ်းထားသည့် class variable တစ်ခုတည်းကို reference လုပ်ထားခြင်း ဖြစ်သည်။

```
print(id(Person.species))
print(id(p1.species))
```

```
print(id(p2.species))
```

```
1895587312560
1895587312560
1895587312560
```

Instance object ကိုမှ မတည်ဆောက်ရသေးသည့်အချိန်တွင် class variable ကို access လုပ်နိုင်သည်။ (class variables can be accessed even before any instance object is created)

Class ထဲမှ method များ အတွင်း၌ class variable ကို class name သို့မဟုတ် self ဖြင့် access လုပ်နိုင်သည်။

Inside the method

```
d(Person.species)
print(f'{self.name} is {self.age} years old {Person.species}')
```

```
NameError                 Traceback (most recent call last)
<ipython-input-4-78aa51557d98> in <module>
----> 1 d(Person.species)
      2 print(f'{self.name}is {self.age} years old{Person.species}')
NameError: name 'd' is not defined
```

Class variable ကို အသုံးပြု၍ create လုပ်လိုက်သည့် instance object များကို ရေတွက်နိုင်သည်။ ထိုသို့ ရေတွက်ရန်အတွက် class variable တစ်ခု ထပ်ထည့်မည်။

Count ဟု နာမည်ပေးထားသည့် class variable တစ်ခု ပြုလုပ်ပြီး သုညဖြင့် စမည်။ (initialized it to 0) create လုပ်လိုက်သည့် Person instance object များ၏ အရေအတွက်ကို count ဖြင့် သိမ်းဆည်းမည်။ Count ကို initializer method ထဲတွင် ထည့်ထားမည်။ Instance object အသစ်တစ်ခု ပြုလုပ်လိုက်တိုင်း count ကို (၁)တိုးမည်။ (increment count by 1.)

```
class Person:
    species = 'Homo sapiens'
    count = 0

    def __init__(self,name,age):
        self.name = name
        self.age = age
        Person.count+=1

    def display(self):
        print(f'{self.name} is {self.age} years old')

p1 = Person('John',20)
p2 = Person('Jack',34)
p1.display()
```

```
p2.display()
Person.count
```

```
John is 20 years old
Jack is 34 years old
2
```

Object (၂)ခု ပြုလုပ်လိုက်သောကြောင့် value of this variable count သည် (၂) ဖြစ်သွားသည်။ နောက်ထပ် object (၂)ခု ပြုလုပ်၍ count အရေအတွက်ကို ထပ်စစ်မည်။

```
p3=Person('Jill', 40)
p4=Person('Jane', 35)
Person.count
```

```
4
```

Count variable ၏ တန်ဖိုး(value)သည် (၄) ဖြစ်သွားသည်။ Class တစ်ခုလုံးနှင့် သက်ဆိုင်သည့် အချက်အလက်များဖြစ်လျှင် (instance object နှင့် မသက်ဆိုင်သည့် အချက်အလက်) ဖြစ်လျှင် class variable အဖြစ် define လုပ်နိုင်သည်။

ဥပမာအဖြစ် အောက်တွင် bank account class တည်ဆောက်ထားသည်။ account number , owner name နှင့် balance တို့သည် instance variable များ ဖြစ်ကြသည်။

```
class BankAccount:
    rate_of_interest = 5
    min_balance = 100
    min_balance_fees = 10

    def __init__(self,account_number, owner_name, balance):
        self.account_number = account_number
        self.owner_name = owner_name
        self.balance = balance

    def withdraw(self,amount):
        self.balance -= amount

    def deposit(self,amount):
        self.balance += amount

account1 = BankAccount('7348', 'Tom', 50)
account2 = BankAccount('6378', 'Bob', 400)
```

```
<__main__.BankAccount at 0x1b959cef2c8>
```

အတိုးနှုန်းသည်(rate of interest)သည် account တစ်ခုချင်းစီ(instance)အတွက် အတူတူပင် ဖြစ်သည်။ ထို့ကြောင့် class variable အဖြစ် define လုပ်သည်။

လက်ကျန်ငွေ(balance)သည် သတ်မှတ်ထားသည့် minimum amount ထက် ပိုနည်းသွားပါက charge some fees လုပ်သည်။ ထို့ကြောင့် minimum balance နှင့် minimum balance_fees ကို class variable အဖြစ် define လုပ်သည်။ ထိုအချက်များသည် account များ အားလုံးအတွက် အတူတူပင် ဖြစ်သည်။ Instance များ အားလုံးအတွက် အတူတူ ဖြစ်သောကြောင့် class level တွင် define လုပ်သည်။ (တစ်နည်းအားဖြင့် class variable အဖြစ် define လုပ်သည်။)

Class variable များကို class အတွင်းရှိ method များ အားလုံးအတွက် အသုံးပြုနိုင်သည်။ Class variable နှင့် instance variable တို့ တွင် နာမည်တူ variable ရှိနေလျှင် class variable ကို မယူဘဲ instance variable ကို ယူရသည်။ Instance variable နှင့် class variable တို့သည် variable နာမည်တူ ဖြစ်နေလျှင် instance variable ကို ယူရသည်။

အောက်က ဥပမာတွင် class variable x နှင့် instance variable x ဟူ၍ နာမည်တူနေသည့် variable x ရှိနေသည်။

```
class Book():
    x = 5 # class variable
    def __init__(self):
        self.x = 100

    def display(self):
        print("Instance variable : ", self.x)
        print("Class variable : ", Book.x)
        b = Book()

book1= Book()
book1.display()
```

```
Instance variable :  100
Class variable :  5
```

```
book1= Book()
book2= Book()
print("Class variable x:", Book.x)
print("instance variable x:", book1.x)
```

```
Class variable x: 5
instance variable x: 100
```

Instance object တစ်ခုတွင် variable name တစ်ခုကို ထည့်လိုက်ပါက Python သည် ထို instance အတွင်း၌ထည့်ပေးသည့် variable name ရှိ မရှိ အရင်စစ်သည်။ ရှိလျှင် execute လုပ်သည်။ မရှိလျှင် class အတွင်း၌ နာမည်တူ class variable ရှိ မရှိ စစ်သည်။

display method အတွင်း၌ self.x သည် instance variable x ဖြစ်သည်။ Instance မှ တစ်ဆင့် access လုပ်သောကြောင့် ဖြစ်သည်။ Class variable x အဖြစ် access လုပ်လိုလျှင် x အရှေ့တွင် class name ကို ထည့်ရေးပေးရသည်။

ဘယ်လို ခွဲခြားပြီး သုံးရမလဲ?

self.x သည် instance variable x ဖြစ်ပြီး **class_name.x** ဆိုလျှင် class variable ဖြစ်သည်။ ထို့ကြောင့် instance variable အဖြစ် access လုပ်လိုလျှင် self.x ဖြစ်ပြီး class variable အဖြစ် access လုပ်လိုလျှင် self.x ဖြစ်ပြီး class_name.x ဖြစ်သည်။ Class variable ကို class name ဖြင့်ကော instance name ဖြင့်ပါ access လုပ်နိုင်သည်။

သို့သော် lass variable တစ်ခု၏ တန်ဖိုး(value)ကို instance name ဖြင့် တန်ဖိုး တစ်ခုခု ပြောင်းလိုပါက ထို့ instance အတွက်သာ ပြောင်းပေး(update လုပ်ပေး)သည်။ နောက်ထပ် ဥပမာ တစ်ခုမှာ

```
class Account:
    rate = 5

a1 = Account() # a1 ဆိုသည့် instance တစ်ခု ပြုလုပ်သည်။
a2 = Account() # a2 ဆိုသည့် instance တစ်ခု ပြုလုပ်သည်။

print("Account.rate : ", Account.rate) #  class name ဖြင့် print လုပ်သည်။
print("a1.rate : ", a1.rate) #  instance name ဖြင့် print လုပ်သည်။
print("a2.rate : ", a2.rate) #  instance name ဖြင့် print လုပ်သည်။
```

```
Account.rate :  5
a1.rate :  5
a2.rate :  5
```

```
Account.rate = 6 # class variable ကို  class name ဖြင့် 6 ဟု re-assign လုပ်သည်။

print("Account.rate : ", Account.rate) #  class name ဖြင့် print လုပ်သည်။
print("a1.rate : ", a1.rate) #  instance name ဖြင့် print လုပ်သည်။
print("a2.rate : ", a2.rate) #  instance name ဖြင့် print လုပ်သည်။
```

```
Account.rate :  6
a1.rate :  6
a2.rate :  6
```

```
# class variable ကို  instance name "a1" ဖြင့် 7 ဟု re assign လုပ်သည်။
a1.rate = 7
```

```
print("Account.rate : ", Account.rate) #  class name ဖြင့် print လုပ်သည်။
print("a1.rate : ", a1.rate) #  instance name ဖြင့် print လုပ်သည်။
print("a2.rate : ", a2.rate) #  instance name ဖြင့် print လုပ်သည်။
```

```
Account.rate :  6
a1.rate :  7
a2.rate :  6
```

Class name ဖြင့် 6 ဟု re-assign လုပ်သည့်အခါ re-assign တန်ဖိုး အသစ်သည် အားလုံးအတွက် အကျုံးဝင်ပြီး ပြောင်းလဲသွားသည်။ Instance name ဖြင့် re-assign လုပ်သည့်အခါ ထို instance (a1) အတွက် အကျုံးဝင်သည်။

```
print("id of Account.rate : ", id(Account.rate))
print("id of a1.rate : ", id(a1.rate))
print("id of a2.rate : ", id(a2.rate))
```

```
id of Account.rate :  140711325442608
id of a1.rate :  140711325442640
id of a2.rate :  140711325442608
```

```
class Account():
    rate = 5 # class variable
    def some_method(self):
        print(self.rate, Account.rate, id(self.rate), id(Account.rate))

        # method အတွင်း၌ instant variable rate ကို 10 ဟု re-assign လုပ်သည်။
        self.rate = 10
        print(self.rate, Account.rate, id(self.rate),id(Account.rate))

a1 = Account()
a2 = Account()
a1.some_method()
```

```
5 5 140711325442576 140711325442576
10 5 140711325442736 140711325442576
```

Class variable ၏ တန်ဖိုးမှာ အားလုံးအတွက် အတူတူပင် ဖြစ်သည်။

Ref: ***Python OOP: Object Oriented Programming in Python" by Deepali Srivastava***