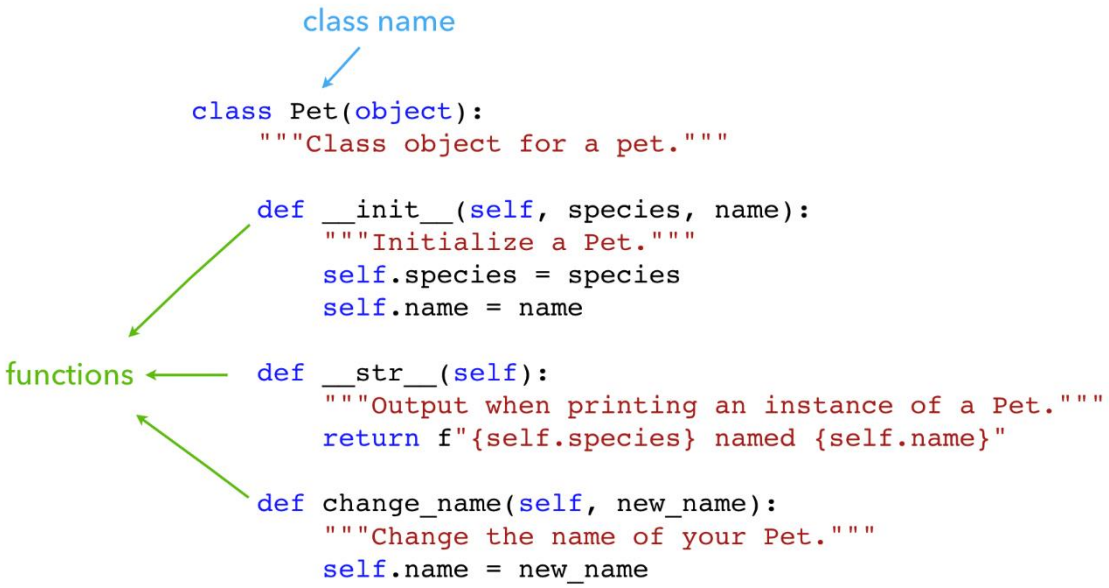
Contents

Chapter – 14 Python Classes

Class များသည် object-oriented programming ၏ အစိတ်အပိုင်းတစ်ခု ဖြစ်သည်။ Object-oriented programming ကို OOP ဟု အတိုကောက်အားဖြင့် ခေါ်သည်။ Class ဟုခေါ်သည့် ထပ်ခါထပ်ခါ အသုံးပြုနိုင်သည့် ကုဒ်အစုအဝေးများကို အထူးလေးပေးပြီး တည်ဆောက်ကြသည်။(focuses on building reusable blocks of code called classes.)။ Class မှ object များကို ပြုလုပ်ပေးနိုင်သည်။ OOP တွင် အသုံးပြုသော ဝေါဟာရအချို့ကို နားလည်ရန် လိုသည်။



14.1 Class နှင့် သက်ဆိုင်သည့်အခေါ်အဝေါ်များ

Class ဆိုသည်မှာ object များ၏ attributes နှင့် behavior ကို define လုပ်နိုင်သော (တည်ဆောက်ပေးနိုင်သော) ကုဒ်အစုအဝေး ဖြစ်သည်။ (A class is a body of code that defines the attributes and behaviors of object.)

Attribute ဆိုသည်များ object များ၏ အချက်အလက် ဖြစ်သည်။ ဥပမာ Car class တွင် my_car သည် object ဖြစ်သည်။ ထိုကား၏ အရောင်(အနီရောင်) သည် attribute ဖြစ်သည်။

Behavior ဆိုသည်မှာ object များက လုပ်ဆောင်ပေးနိုင်သည့် အရာ သို့မဟုတ် object များ၏ ပြုမူဆောင်ရွက်ချက် ဖြစ်သည်။ ထိုအချက်များကို class အတွင်း၌ ဖော်ပြထားမှသာ(define လုပ်ထား မှသာ) အကျုံးဝင်သည်။ Behavior တွင် method များပါဝင်သည်။ Method ဆိုသည်မှာ class တွင်း၌ ရေးထားသည် function များ ဖြစ်သည်။ တစ်နည်းအားဖြင်း object တစ်ခု၏ function များ သို့မဟုတ် method များကို ၎င်း၏ behavior ဟု ခေါ်သည်။ တစ်နည်းအားဖြင့် object များ လုပ်ဆောင်နိုင်သည့် အပြု အမူများသည် method ဖြစ်သည်။

Python သည် object oriented programming language တစ်ခု ဖြစ်သည်။ Python တွင် ရှိသမျှ အရာ အားလုံးသည် object များသာ ဖြစ်သည်။ Object များ အားလုံးတွင် ၎င်းတို့နှင့် သက်ဆိုင်သည့် ဂုဏ်သတ္တိများ (properties) ရှိသည်။

Class များသည် object များကို တည်ဆောက်ပေးသည့် **object constructor** များ ဖြစ်ကြသည်။ သို့မဟုတ် object များကို တည်ဆောက်နိုင်သည့် "blueprint" များ ဖြစ်ကြသည်။ တစ်နည်းအားဖြင့် Class များသည် object ထုတ်လုပ်ပေးနိုင်သည့် မို(mold) များဖြစ်သည်။ ဥပမာ-ပလတ်စတစ်ဘဲရုပ်လေးများသည် object များဖြစ်လျှင် ထိုပလတ်စတစ် ဘဲရုပ်လေးများ ပြုလုပ်သည့် မို(mold)သည် class ဖြစ်သည်။

သင်ကြားခဲ့ပြီးသမျှ list များ၊ string များ၊ integer များ၊ function များ စသည် တို့အားလုံးသည် object များ ဖြစ်ကြသည်။ ထို့ကြောင့် မည်သည့် object ဖြစ်ကြောင်းကို type function ကို အသုံးပြု၍ စစ်နိုင်သည်။ Type function သည် built in function တစ်ခု ဖြစ်သည်။

```
type([1, 2, 3])
```

<class 'list'>

List တစ်ခု သည် list class ၏ instance တစ်ခုဖြစ်သည်။

```
type({"a": 1, "b": 2})
```

<class 'dict'>

Dictionary တစ်ခု သည် dict class ၏ instance တစ်ခုဖြစ်သည်။ ထို့အတူ function တစ်ခု သည် function class ၏ instance တစ်ခုဖြစ်သည်။ object နှင့် instance တို့၏ သဘောတူညီသော တစ်ခါတစ်ရံ ဖလှယ်၍ အသုံးပြုသည်။

Object ဆိုတာ မိမိလုပ်ချင်တဲ့ ကိစ္စတွေ လုပ်လို့အောင် methodတွေကို အသုံးပြုနိုင်တဲ့ data သို့မဟုတ် variable အစုအဝေးပါ။ Object များအား တစ်စုံတစ်ခု ပြုလုပ်နိုင်သည့် နည်းများကို method ဟုခေါ်သည်။ ဥပမာ string object များ သို့မဟုတ် list object များကို append လုပ်ခြင်း remove လုပ်ခြင်း စသည့် နည်းများကို method ဟုခေါ်သည်။ 1,2,3,4,5 ကိန်းအစဉ်အတန်းတစ်ခု၏ စုစုပေါင်း တန်ဖိုးကို လိုချင်လျှင် တစ်ခုချင်းစီ လိုက်ပေါင်း ရသည်။ ထို 1,2,3,4,5 ကို list object တစ်ခု ဖြစ်အောင် ပြုလုပ်လိုက်လျှင် **sun()** method ဖြင့် စုစုပေါင်းတန်ဖိုး ရှာနိုင်သည်။ **max()** ဖြင့် အကြီးဆုံးကိန်း ရှာနိုင်သည်။ **min()** ဖြင့် method အငယ်ဆုံးကိန်း ရှာနိုင်သည်။

14.2 Python class ၏ self နှင့် __init__ method

Object

Object ဆိုသည်မှာ class တစ်ခုထဲတွင် ပါဝင်နေသည့် instance တစ်ခုဖြစ်သည်။ ဥပမာ Car class ထဲတွင် my_car သည် object တစ်ခု ဖြစ်သည်။ your_car သည် object တစ်ခု ဖြစ်သည်။

self

"self" သည် magic keyword ဖြစ်သည်။ **"self"** သည် class ၏ instance တစ်ခုခုကို ကိုယ်စား ပြုသည်။ **"self"** သည် attributes နှင့် ထည့်ပေးလိုက်သည့် argument တို့ကို ပေါင်းစပ်ပေးသည့် စကားလုံး ဖြစ်သည်။ Class ထဲတွင်ရှိသည့် methods နှင့် attributes တို့ကို access လုပ်ရန် "self" ကို အသုံးပြုသည်။

"self" သည် class ထဲရှိ object တစ်ခုခု သို့မဟုတ် instance တစ်ခုခုကို ရည်ညွှန်းသည်။

Object(instance of class) တစ်ခု၏ method ကို ခေါ်သုံးလိုသည့်အခါ "self" ကို parameter အဖြစ် အသုံးပြုသည်။ argument တွေကို self တစ်ဆင့် method ထဲသို့ရောက်ထည့်ပေး(pass လုပ်ပေး)သည်။

__init__

Python class တွင် "__init__" သည် reserved method ဖြစ်သည်။ Object များ တည်ဆောက်ပေးသည် constructor ဖြစ်သည်။ Object များကို initialize လုပ်ရန်အတွက် arguments များကို **"__init__" method** ထဲသို့ ထည့်ပေးသည်

Attributes

Car တစ်စီး၏ attribute များ အရောင် "color" ၊ ထုတ်လုပ်သည့်ကုမ္ပဏီ"company" ၊ ကန့်သတ် ထားသည့်မြန်နှုန်း("speed_limit") စသည်တို့ဖြစ်သည်။ Attribute များသည် ကားတစ်စီး၏ ပြုမူနိုင်သည် ဆောင်ရွက်ချက်များနှင့် မသက်ဆိုင်ကြပေ။

Methods

ကားတစ်စီး၏ ပြုမူနိုင်သည့် ဆောင်ရွက်ချက်များ(behaviour)သည် method ဖြစ်သည်။ ဥပမာ- "change_gear", "start", "accelerate", "move" စသည့် လုပ်ဆောင်ချက်များသည် ထိုကား၏ method များ ဖြစ်ကြသည်။

14.3 The __init__() Method

Class ၏ အဓိပ္ပာယ်ကို နားလည် သဘောပေါက်ရန် built-in **__init__()** function အကြောင်းကို သိရန် လိုသည်။ Class တိုင်းတွင် __init__() function ပါဝင်သည်။ Class ကို initiate လုပ်သည့်အခါတိုင်း **__init__()** function ကို အရင်ဆုံး initiate လုပ်သည်။

__init__() method ကို constructor ဟူ၍လည်း ခေါ်သည်။ Constructor ဆိုသည်မှာ class အတွင်းရှိ object များကို တည်ဆောက်ပေးသောကြောင့် ဖြစ်သည်။ Constructor သည် object များကို တည်ဆောက် ပေးခြင်းသည် initialize လုပ်ခြင်း ဖြစ်သည်။ Object တွင်ရှိမည့် value များကို assign လုပ်ခြင်း ဖြစ်သည်။

Constructor အားလုံး၏ နာမည်သည် အတူတူဖြစ်သည်။ Constructor နာမည်မှာ __init__() ဖြစ်သည်။ Constructor များသည် class အတွင်းရှိ object များ တည်ဆောက်ရန်အတွက် argument များကို လက်ခံ ရသည်။ Constructor မပါဘဲ class တစ်ခုကို တည်ဆောက်လိုက်လျှင် Python သည် အလိုအလျောက် default constructor ကို ထည့်ပေးသည်။ Class တိုင်းတွင် object များ တည်ဆောက်ရန်အတွက် constructor ပါရှိရန် မဖြစ်မနေလိုအပ်သည်။

__init__() Method Syntax

```
def __init__(self):
    # body of the constructor
```

14.4 Self keyword

__init__( ) method တွင် parameter များကို ထည့်ပေးရသည်။ “self” ကို မဖြစ်မနေ ထည့်ပေးရသည်။ တခြား parameter နှင့် အတူထည့်သွင်းရသည့်အခါ self ကို အရင်ဆုံးထည့်ပေးရသည်။ Class တစ်ခုအတွင်း၌

parameter များအားလုံး၏ ရှေ့တွင် self. ထည့်ရေးပေးရသည်။

“self” သည် class ထဲတွင် instance တစ်ခုခုကို ရည်ညွှန်းသည်။ တစ်နည်းအားဖြင့် object တစ်ခုကို class အတွင်း၌ ရည်ညွှန်းလိုသည်အခါ self ကို သုံးသည်။ မိမိဆိုလိုသည့် object မဟုတ်ဘဲ တခြား object ကို ရည်ညွှန်းလိုသည်အခါ **other** ကို သုံးသည်။

“self” keyword ကို အသုံးပြု၍ class ၏ attributes နှင့် method များကို access လုပ်နိုင်သည်။

self သည် attributes နှင့် argument များကို ချိတ်ဆက်(bind လုပ်)ပေးသည်။ တွဲပေးသည်။

Python တွင် @ syntax ကို အသုံးမပြုဘဲ self ကို အသုံးပြုသည်။

Object property ထဲသို့ value များကို assign လုပ်ရန် __init__() function ကို သုံးသည်။ Object တစ်ခု ဆောက်ပြီး(being created)သည့်အခါ အလုပ် တစ်ခုခု လုပ်စေလိုသည့်အခါ __init__() function ကို သုံးသည်။

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

p1 = Person("John", 36) # p1 ဆိုသည့် object တစ်ခု ပြုလုပ်သည်။

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

```
John
36
```

အပေါ်က ဥပမာတွင် `def __init__(self, name, age):` တွင် `name` နှင့် `age` သည် attribute များ ဖြစ်ကြသည်။ `self.name = name` ဟု assign လုပ်ပြီးနောက်ပိုင်း `self.name` ကို class အတွင်း မည်သည့်နေရာတွင် မဆို အသုံးပြုနိုင်သည်။

`p1 = Person("John", 36)` သည် p1 ဆိုသည့် class object တစ်ခု ပြုလုပ်လိုက်သည့် ကုဒ် ဖြစ်သည်။ ထိုအချိန်တွင် __init__ method သည် p1 ဆိုသည့် class object တစ်ခု တည်ဆောက်သည့် အလုပ်ကို စတင်လုပ်သည်။ "John" အရှေ့ဆုံးက ထည့်ပေးလိုက်သောကြောင့် name ဖြစ်သည်။ `self.name = name` ရောက်ပြီးနောက် p1 ဆိုသည့် object တွင် နာမည်သည် "John" ဖြစ်ပြီး အသက်သည် 36 ဖြစ်သွားသည်။

နောက်ထပ် `p2 = Person("Aung Aung", 16)` ကုဒ်ကို run ပါက p2 ဆိုသည့် object တစ်ခုကို အထက်က ကဲ့သို့ပင် __init__ method က တည်ဆောက်ပေးလိမ့်မည်။

`p1 = Person("John", 36)` နှင့်

`p2 = Person("Aung Aung", 16)` တို့သည် Person function ခေါ်သုံးသည့် (call လုပ်သည့်) ကုဒ်များ ဖြစ်သည်။

Argument များ လက်ခံရသည့် constructor နှင့် argument များ လက်ခံစရာမလိုသည့် constructor ဟူ၍ နှစ်မျိုးရှိသည်။

- Default constructor သည် argument များ လက်ခံစရာမလိုပေ။

- Parameterized constructor သည် argument များကို လက်ခံပေးသည်။

14.4.1 Adding Attributes to an Object

Attribute ဆိုသည်မှာ object တစ်ခု၏ specific characteristic တစ်ခု ဖြစ်သည်။ Python တွင် object တစ်ခုအဖြစ် တည်ဆောက်ပြီးပါက attribute များကို မကြေငြာဘဲ(dynamically) ထည့်နိုင်သည်။ Attribute ထည့်နည်းမှာ object name ၏ နောက်တွင် dot (.) နှင့် attribute name ကို ထည့်ပေးရုံပင် ဖြစ်သည်။

ဥပမာ- person1 ဆိုသည့် object တွင် သူ၏ အသက် (၃၇)နှစ်ကို ထည့်လိုပါက `person1.age = 37` ဖြစ်သည်။

ထိုကဲ့သို့ ထည့်ခြင်းသည် အကျင့်ဆိုး ဖြစ်သည်။ လုပ်၍ ရသော်လည်း ထိုသို့ ပြုလုပ်ရန် မတိုက်တွန်းပါ။ Debug လုပ်သည့်အခါ ပိုခက်ခဲစေသည်။ သင့်လျော်သည့် attribute ထည့်နည်းမှာ အောက်တွင် ဖော်ပြထားသည်။

14.5 Class Example-1

ပထမဦးစွာ User ဆိုသည့် class တစ်ခု တည်ဆောက်သည်။ ဤဥပမာ၏ ရည်ရွယ်ချက်မှာ class သဘော တရားများထက် object သဘောတရားများကို ပို၍ အလေးထား ရှင်းပြ လိုသောကြောင့် သာမန် class တစ်ခု အဖြစ်သာ တည်ဆောက်သည်။ အင်တာနက် အသုံးပြုသူများ၏ အချက်အလက်များကို သိမ်းဆည်းသည့် class ဖြစ်သည်။

```
class User:
    pass
```

တည်ဆောက်ပြီးသည့် class ကို ပြန်စစ်ကြည့်သည်။

```
User
```

__main__.User

user1 ဆိုသည် အသုံးပြုသူတစ်ယောက်ကို User class ၏ အဖွဲ့ဝင်ဖြစ်အောင် ထည့်သွင်းသည်။

```
user1 = User()
```

user1 သည် User class ၏ အဖွဲ့ဝင်ဖြစ်အောင် ဖြစ်ကြောင်း စစ်ဆေးသည်။

```
user1
```

<__main__.User at 0x241e8f03288>

user1 သည် User class ၏ အဖွဲ့ဝင်ဖြစ်သွားသောကြောင့် user1 သည် object တစ်ခုအဖြစ် တည်ရှိသွားပြီ ဖြစ်သည်။ ထိုအတူ user1 ကို User class ၏ instance တစ်ခုလည်း ဖြစ်သည်။ ("instance" of User class)

user1 သည် object တစ်ခု ဖြစ်သွားသောကြောင့် object နှင့် သက်ဆိုင်သည့် ဂုဏ်သတ္တိများ ရှိသွားသည်။ ထို့ကြောင့် user1 ၏ အမည် ပထမအလုံး(first_name)၊ နောက်ဆုံးအလုံး(last_name) နှင့် အသက် (age) စသည်တို့ ထည့်ပေးလို့ ရပြီ ဖြစ်သည်။ first_name , last_name, age စသည်တို့ကို field ဟုခေါ်သည်။

```
user1.first_name = "Myo"
```

```
user1.last_name = "Win"
user1.age = 37
```

`first_name, last_name, age` စသည်တို့က user1 ဆိုသည့် object နှင့် တွဲလိုက်သည်။ Attach သို့မဟုတ် bind လုပ်လိုက်သည် ကွန်ပျူတာစကားဖြင့် ပြောသည်။ user1 ၏ အချက်အလက်များကို အောက်ပါအတိုင်း ပြန်ဖတ် ယူနိုင်သည်။

```
print(user1.first_name, user1.last_name)
print(user1.age)
```

```
Myo Win
37
```

ထိုအတူ user2 ကိုလည်း ပြုလုပ်နိုင်သည်။ Class တစ်ခု အတွင်း၌ object အရေအတွက် အကန့်အသတ်မရှိ ပြုလုပ်ထားနိုင်သည်။

```
user2 = User
user2.first_name = "Ko"
user2.last_name = "Aung"
user2.hobby = "Reading"
```

user2 ၏ အချက်အလက်များကို အောက်ပါအတိုင်း ပြန်ဖတ်ယူနိုင်သသည်။

```
print(user2.first_name, user2.last_name)
print(user2.hobby)
```

```
Ko Aung
Reading
```

user1 ၏ အချက်အလက် ထဲတွင် age ပါဝင်သည်။ user2 ၏ အချက်အလက်ထဲတွင် hobby ပါဝင်သည်။ အကယ်၍ user2 ၏ age ကို print လုပ်ပါက AttributeError ပေးလိမ့်မည်။

```
print(user2.age)
```

AttributeError Traceback (most recent call last)

AttributeError: type object 'User' has no attribute 'age'

Object တွင် မရှိသည့် field တစ်ခုခုကို တောင်းယူပါက AttributeError ပေးလိမ့်မည်။

အထက်တွင် ပြထားသည့် Attribute ထည့်နည်းမှာ သင့်လျော်မှန်ကန်သည့်နည်း မဟုတ်ပါ။ ထိုကဲ့သို့ ထည့်ခြင်းသည် အကျင့်ဆိုး ဖြစ်သည်။ လုပ်၍ ရသော်လည်း ထိုသို့ ပြုလုပ်ရန် မတိုက်တွန်းပါ။ Debug လုပ်သည့် အခါ ပိုခက်ခဲစေသည်။ သင့်လျော်သည့် attribute ထည့်နည်းမှာ အောက်တွင် ဖော်ပြထားသည်။

14.5.1 Class Feature

Class ၏ feature များ ဖြစ်သည့် Methods, Initialization, help text အကြောင်းကို နိဒါန်းအဖြစ် အကျဉ်းချုံးရှင်းပြထားသည်။ ပြီးခဲ့သည့် ဥပမာတွင် တည်ဆောက်ခဲ့သည့် class တွင် pass တစ်ကြောင်းသာ ပါသည်။ ယခု class တွင် **__init__ method** ကို အသုံးပြုထားသည်။ နောက်ပိုင်းတွင် **__init__ method** ကို အသေးစိတ် ရှင်းပြထားသည်။

```
class User:
    def __init__(self, full_name, birthday):
        self.full_name = full_name
        self.birthday = birthday                    #yyyymmdd
```

Class တည်ဆောက်ပြီး၍ user များ၏ အချက်အလက်များကို ထည့်သည်။

```
user1 = User("Myo Win", 19700812)
```

ထို့နောက် အချက်အလက်များကို print ထုတ်သည်။

```
print(user1.full_name)
print(user1.birthday)
```

```
Myo Win
19700812
```

```
help(text)
```

help(text) ထည့်သည့်နည်းကို ဖော်ပြပါမည်။ ပြီးခဲ့သည့် class code တွင် docstring အဖြစ် """ User profile. Storing name and birthday. Birthday format is yyymmdd. """ ကို ထည့်သည်။ ဤကဲ့သို့ class code အတွင်း၌ docstring များ ထည့်သွင်း ရေးသား၍ ရှင်းပြခြင်းသည် အလေ့ အကျင့်ကောင်း ဖြစ်သည်။

```
class User:
    """ User profile. Storing name and birthday.
    Birthday format is yyymmdd. """
    def __init__(self, full_name, birthday):
        self.full_name = full_name
        self.birthday = birthday          #yyyymmdd
```

Class တစ်ခု၏ အကြောင်းကို သိလိုပါက help(class name) ဖြင့် ဖတ်ရှုနိုင်သည်။

```
help(User)
```

```
Help on class User in module __main__:
class User(builtins.object)
 |  User(full_name, birthday)
 |
 |  User profile. Storing name and birthday.
 |  Birthday format is yyymmdd.
 |
 |  Methods defined here:
 |
 |  __init__(self, full_name, birthday)
 |      Initialize self.  See help(type(self)) for accurate signature.
```

မိမိတည်ဆောက်လိုက်သည့် User class ၏ အချက်များကို ဖော်ပြပေးသည်။

14.6 Class Example-2 Polygon class ဥပမာ

အနားများစွာပါသည့် ဗဟုဂံ(polygon)များတွက် တည်ဆောက်ထားသည့် class ဖြစ်သည်။ အနား အရေအတွက်ကို လိုက်၍ တြိဂံ(triangle)၊ စတုဂံ(rectangle)၊ ပဉ္စဂံ(pentagon)စသည်ဖြင့် ကွဲပြားသွားသည်။

class Polygon: ဖြင့် Polygon ဟု အမည်ပေးထားသည့် class တစ်ခုကို တည်ဆောက်သည်။ __init__ method ဖြင့် အနား အရေအတွက် နှင့် နာမည်ကို ထည့်ပေးသည်။

```
class Polygon:
    def __init__(self, sides, name):
        self.sides = sides
        self.name = name

square = Polygon(4,"Square")
print(square.sides)
print(square.name)
```

```
4
Square
```

```
pentagon = Polygon(5,"Pentagon ")
print(pentagon.sides)
print(pentagon.name)
```

```
5
Pentagon
```

အထက်က Polygon class ကို အတွင်းထောင့် တစ်ခုချင်းစီ၏ ဒီဂရီ(angle)နှင့် အတွင်းထောင့် အားလုံး ပေါင်းသည့် ဒီဂရီ(interior_angles)ကို ထပ်ထည့်သည်။

```
class Polygon:
    def __init__(self, sides, name):
        self.sides = sides
        self.name = name
        self.interior_angles = (self.sides-2)*180
        self.angle = self.interior_angles / self.sides

triangle = Polygon(3,"Triangle")
square = Polygon(4,"Square")
pentagon = Polygon(5,"Pentagon ")
hexagon = Polygon(6,"Hexagon ")
```

Triangle, square, pentagon, hexagon တို့ကို list တစ်ခု ပြုလုပ်မည်။ ထို list ကို polygon_list ဟု နာမည်ပေးပြီး for loop ဖြင့် class instance များ၏ အချက်အလက်များကို print ထုတ်မည်။

```
polygon_list= [triangle, square, pentagon, hexagon]
for i in polygon_list:
    print(f'{i.name} has {i.sides} sides.')
```

```
    print(f'{i.sides} sides x {i.angle} Deg={i.interior_angles}Deg.\n')
```

```
Triangle has 3 sides.
3 sides x 60.0 Deg = 180 Deg.

Square has 4 sides.
4 sides x 90.0 Deg = 360 Deg.

Pentagon  has 5 sides.
5 sides x 108.0 Deg = 540 Deg.

Hexagon  has 6 sides.
6 sides x 120.0 Deg = 720 Deg.
```

14.7 Class Example-3 Car Class

Car class တစ်ခုကို နမူနာအဖြစ် ပြုလုပ်ရင်း class တစ်ခု တည်ဆောက်ပုံအကြောင်း ရှင်းပြပါမည်။

14.7.1 class အလုပ်လုပ်ပုံ

Car class အလုပ်လုပ်ပုံမှာ ကားတစ်စီးကို ဓာတ်ဆီထည့်ပေးလျှင် ကားရွေ့လျားနိုင်သည်။ သို့မဟုတ် တွန်းလျှင်လည်း ကားရွေ့လျားနိုင်သည်။ ကားရွေ့ရောက်သွားမည့် အကွာအဝေးကို ဖော်ပြပေးရန် ဖြစ်သည်။ ဓာတ်ဆီ(၁)လီတာ ထည့်လျှင် (၁၂)ကီလိုမီတာ သွားနိုင်သည်။ တစ်ခါတွန်းလျှင် (၀.၁)ကီလိုမီတာ (မီတာ ၁၀၀)သာ သွားနိုင်သည်။

14.7.2 Class method

ကားသည် နည်း(၂)နည်းဖြင့် ရွေ့သွားမည်ဖြစ်သောကြောင့် method (2)ခု လိုအပ်ပါသည်။ ဆီထည့်ပေး၍ ဆီဖြင့်သွားသည့် method နှင့် တွန်းသည့် method တို့ဖြစ်သည်။

14.7.3 Car class တည်ဆောက်ခြင်း

```
class Car():
```

Car class တစ်ခု တည်ဆောက်သည်။

`"""Simulates a car travelled distance for a game, or a physics simulation."""`သည် docstring ဖြစ်သည်။ docstring ဆိုသည်မှာ classနှင့် သက်ဆိုင်သည့် သိသင့်သိထိုက်သည့် အချက်အလက်များကို ဖတ်သူများ class ကို အသုံးပြုသူများ အထောက်အကူ ပြုစေရန် ရည်ရွယ်ပြီး ရေးသားထားသည့် စာသားများ ဖြစ်သည်။

Docstring သည် documentation string ၏ အတိုကောက်ဖြစ်သည်။ Docstring ကို function, method, class, module သည်တို့၏ definition အပြီးတွင် ထည့်သွင်းရေးသားထားလေ့ရှိသည်။ Triple quotes ဖြင့် docstrings များကို ရေးသားထားလေ့ရှိသည်။

__init__ method

Car class အတွင်းရှိ object များ ပြုလုပ်ရန်အတွက် __init__ method ဖြင့် စသည်။ Car object တစ်ခု၏ အစဆုံးတည် နေရာကို (x, y)ဖြင့် ဖော်ပြပြီး မူလအမှတ်သည် (0, 0) ဖြစ်သည်။ စ,စချင်း ကားတည်ရှိ နေမည့် နေရာကို print လုပ်သည်။

```
def __init__(self):
```

```
        # Each car has an (x,y) position.
        self.x = 0
        self.y = 0
        print("Car travelled distance:", self.x)
```

14.7.4 ဆီထည့်မည့် fuel_add method

ဓာတ်ဆီထည့်ပေးမည့် method ဖြစ်သည်။ ဓာတ်ဆီ (၁)လီတာထည့်လျှင် (၁၂)ကီလိုမီတာ သွားနိုင်သည်။ ရောက်နေ့သည့်နေရာမှ ဆက်သွားလိမ့်မည်။ ဓာတ်ဆီမည်မျှ ထည့်လိုက်သည်။ ဓာတ်ဆီ မည်မျှ ကျန်သေးသည်ကို print ထုတ်သည်။ ထည့်ပေးသည့် ဓာတ်ဆီ ပမာဏအတိုင်း ရောက်မည့် နေရာကို တွက်၍ ဖော်ပြသည်။ ရောက်ပြီးသည့် နေရာမှ ဆီပမာဏ ရှိသလောက် ထပ်သွားသည်။

```
def fuel_add(self, liters):
        self.liters = liters
        print(f'\nYou have topped up {liters} liters')
        print(f'Fuel {liters} liters left\n')
        self.x += self.liters * 12
        print("Car travelled distance:", self.x,"meter")
```

14.7.5 ကားတွန်းမည့် push_up method

ဆီကုန်သွား၍ ကားရွေ့စေလိုလျှင် ကားကို တွန်း၍ ရွေ့နိုင်သည်။ တစ်ခါတွန်းလျှင် (၀.၁)ကီလိုမီတာ (မီတာ ၁၀၀)သာ သွားနိုင်သည်။ တွန်းပြီး၍ ရောက်သည့်နေရာကို ဖော်ပြရန် print ထုတ်သည်။

```
def push_up(self):
        # Increment the x-position of the car.
        self.x += 0.1
        print("Car travelled distance:", self.x, "meter")
```

အောက်တွင် code အပြည့်အစုံကို ဖော်ပြထားသည်။

```
class Car():
    """(ဓာတ်ဆီ (၁)လီတာထည့်လျှင် (၁၂)ကီလိုမီတာ သွားနိုင်သည်။
        တစ်ခါတွန်းလျှင် (၀.၁)ကီလိုမီတာ (မီတာ ၁၀၀)သာ သွားနိုင်သည်။)
        Simulates a car travelled distance for a game,
        or a physics simulation."""

    def __init__(self):
        # Each car has an (x,y) position.
        self.x = 0
        self.y = 0
        print("Car travelled distance:", self.x)

    def fuel_add(self, liters):
        # Top up fuel
        self.liters = liters
        print(f'\nYou have topped up {liters} liters')
        print(f'Fuel {liters} liters left\n')
```

```
        self.x += self.liters * 12
        print("Car travelled distance:", self.x,"meter")

    def push_up(self):
        # Increment the x-position of the car.
        self.x += 0.1
        print(f"Car travelled distance: { self.x:.1f} meter")

# my_car ဟုခေါ်သည့် Car object တစ်ခု စတင် တည်ဆောက်သည်။
my_car = Car()
print(type(Car))
print(type(my_car))
```

```
Car travelled distance: 0
<class 'type'>
<class '__main__.Car'>
```

```
# တစ်ခါ တွန်းသည်။ အစမှ 0.1 km ရောက်သည်။
my_car.push_up()
# ဆီ (၃) လီတာ ထည့်သည်။ အစမှ 36.1 km ရောက်သည်။
my_car.fuel_add(3)
# တစ်ခါ တွန်းသည်။ အစမှ 36.2 km ရောက်သည်။
my_car.push_up()
# ဆီ (၂) လီတာ ထည့်သည်။ အစမှ 60.2 km ရောက်သည်။
my_car.fuel_add(2)
```

```
You pushed the car for (0.1 km): 0.1 km

You have topped up 3 liters.
Fuel 3 liters left
Car travelled distance: 36.1 km

You pushed the car for (0.1 km): 36.2 km

You have topped up 2 liters.
Fuel 2 liters left
Car travelled distance: 60.2 km
```

ကားသည် စထွက်သည့်အချိန်မှာ စ၍ ရောက်သည့် နေရာမှ စ၍ သုံးသည့် method ကို လိုက်၍ ရွေ့သွားသည့် travelled distance ကို ထည့်ပေါင်းသည်။

```
# မိမိ ရေးခဲ့သည့် Car class ကို help() ဖြင့် ဖတ်ကြည့်ရန်

help(Car)
```

```
Help on class Car in module __main__:

class Car(builtins.object)
```

```
|   (ဓာတ်ဆီ (၁)လီတာထည့်လျှင် (၁၂)ကီလိုမီတာ သွားနိုင်သည်။
|   တစ်ခါတွန်းလျှင် (၀.၁)ကီလိုမီတာ (မီတာ ၁၀၀)သာ သွားနိုင်သည်။)
|   Simulates a car travelled distance for a game,
|   or a physics simulation.
|
|   Methods defined here:
|
|   __init__(self)
|       Initialize self. See help(type(self))for accurate signature.
|
|   fuel_add(self, liters)
|
|   push_up(self)
|   ----------------------------------------------------------------
```

14.7.6 Class Example-3a Hybrid Car Class

ပြီးခဲ့တဲ့ car class ကို နည်းနည်း ပိုကောင်းအောင် modify လုပ်ကြည့်ကြစို့။ ဓာတ်ဆီကားကို ဓာတ်ဆီ နဲ့ကော ဘက်ထရီ နှစ်မျိုးလုံးမောင်းလို့ရတဲ့ Hybrid car အဖြစ်ပြောင်းမှာပါ။ ဘက်ထရီကို အားသွင်း ရင်လည်း မောင်းလို့ရတဲ့ method (၁)ခုနှင့် ကားဘယ်လောက် ဝေးဝေး ခရီးသွားခဲ့သလဲဆိုတာကို သိဘို့ ဂရုပ်ပုံဆွဲတဲ့ method (၁)ခု ထပ်ထည့်ပါမည်။ ဓာတ်ဆီနဲ့လည်းမောင်းလို့ရတယ်။ တွန်းရင်လည်း ကားရွေ့တယ်။

ကားသွားထားသမျှ ကီလိုမီတာတွေ အားလုံးကို စုပြီးတော့ အနှစ်ချုပ်အဖြစ် ဂရုပ်ပုံဆွဲပြီး ဘယ်လောက် မောင်းခဲ့တယ် ဆိုတာကို ပြဘို့ ဂရုပ်ဆွဲပါမည်။ ပုံဆွဲလိုကောင်းအောင်ကားသွားနိုင်သည့် ကီလိုမီတာ များကို အနည်းငယ် ပြောင်းထားသည်။

- ဓာတ်ဆီ (၁)လီတာထည့်လျှင် (၂)ကီလိုမီတာသာ သွားနိုင်သည်လို့ ပြင်လိုက်သည်။
- Hybrid car battery ကို (၁)နာရီ အားသွင်းလျှင် (၁)ကီလိုမီတာ သွားနိုင်သည်။
- တစ်ခါတွန်းလျှင် (၀.၃)ကီလိုမီတာ (မီတာ ၁၀၀)သာ သွားနိုင်သည်။

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 3)

class Car():
    """(ဓာတ်ဆီ (၁)လီတာထည့်လျှင် (၂)ကီလိုမီတာ သွားနိုင်သည်။
       Hybrid car battery ကို (၁)နာရီအားသွင်းလျှင် (၁)ကီလိုမီတာ သွားနိုင်သည်။
       တစ်ခါတွန်းလျှင် (၀.၃)ကီလိုမီတာ (မီတာ ၁၀၀)သာ သွားနိုင်သည်။)
       Simulates a car travelled distance for a game,
       or a physics simulation."""

    def __init__(self):
        # Each car has an (x,y) position.
        self.x = 0
        self.y = 0
        print("Car travelled distance:", self.x)
```

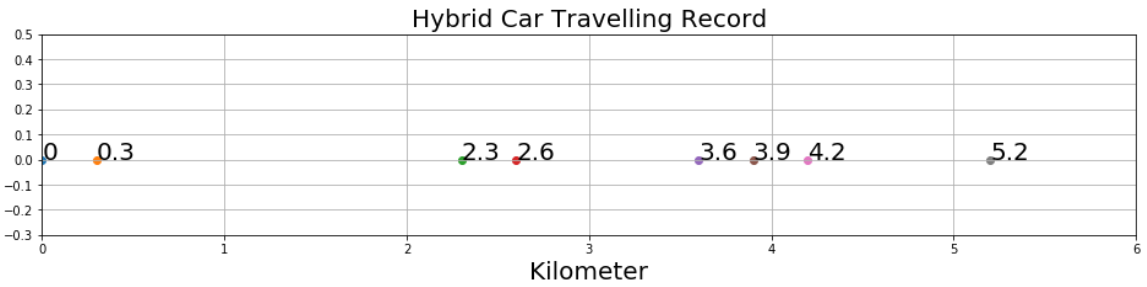
```
    def fuel_add(self, liters):
        # Top up fuel
        self.liters = liters
        print(f'\nYou have topped up {liters} liters.')
        print(f'Fuel {liters} liters left')
        self.x += self.liters * 2
        print(f"Car travelled distance: {self.x:.1f} km\n")

    def battery_charge(self, time_hr):
        # Charge Hybrid car battery
        self.time_hr = time_hr
        print(f'\n You have charged batter for {time_hr} hrs')
        print(f'Batter full for {time_hr * 1} km. ')
        self.x += self.liters * 1
        print(f"Car travelled distance: {self.x:.1f} km\n")

    def push_up(self):
        # Increment the x-position of the car.
        self.x += 0.3
        print(f"You pushed the car for (0.3 km): {self.x:.1f} km \n")

    def plot(self):
        plt.scatter(self.x, self.y )
        plt.text(self.x , self.y+0.002 , str(round(self.x,1)
        plt.grid(True)
        plt.ylim((-0.3,0.5))
        plt.xlim((0,6))
        plt.title('Hybrid Car Travelling Record')
        plt.xlabel('Kilometer')
```

Car() class တည်ဆောက်ပြီးပါပြီ။ Car() class ထဲက my_car ဆိုတဲ့ object တစ်ခုပြုလုပ်ပြီး စမောင်းကြစို့

```
# Create a Car object
my_car = Car()
my_car.plot()
# တစ်ခါ တွန်းသည်။ (0.3 km)
my_car.push_up()
my_car.plot()

# ဆီ (၁) လီတာ ထည့်သည်။ (0.3 km+ 2km)
my_car.fuel_add(1)
my_car.plot()

# တစ်ခါ တွန်းသည်။ (0.3 km+ 2km +0.3 km)
my_car.push_up()
```

```
my_car.plot()

# (၂) နာရီကြာ ဘတ္ထရီအားသွင်းသည်။(Charge 2 hr Hybrid car battery)
my_car.battery_charge(2) #(1 km)
my_car.plot()

# တစ်ခါ တွန်းသည်။ #(0.3 km)
my_car.push_up()
my_car.plot()

# တစ်ခါ တွန်းသည်။ #(0.3 km)
my_car.push_up()
my_car.plot()

# (၁) နာရီ ဘတ္ထရီအားသွင်းသည်။ #(1 km)
my_car.battery_charge(1)
my_car.plot()
```

```
Car travelled distance: 0
You pushed the car for (0.3 km): 0.3 km


You have topped up 1 liters.
Fuel 1 liters left
Car travelled distance: 2.3 km

You pushed the car for (0.3 km): 2.6 km


 You have charged batter for 2 hrs
Batter full for 2 km.
Car travelled distance: 3.6 km

You pushed the car for (0.3 km): 3.9 km

You pushed the car for (0.3 km): 4.2 km

You have charged batter for 1 hrs
Batter full for 1 km.
Car travelled distance: 5.2 km
```



14.8 Magic Methods

Magic method ဆိုသည်မှာ syntax တွင် အစနှင့် အဆုံး၌ underscore နှစ်ခု ပါသည့် special method များ ဖြစ်ကြသည်။ ထို့ကြောင့် double underscores method ဟုလည်း ခေါ်သည်။ Double underscore ကို dunder ဟု အတိုခေါက် ခေါ်ဆိုသောကြောင့် magic method များကို dunder method ဟူ၍လည်း ခေါ်ဆိုကြသည်။

14.8.1 The __str__() Method

Python object အားလုံးတွင် str__() method သည် အလိုအလျောက်(by default) ရှိပြီးသား ဖြစ်သည်။ print()ကို သုံးသည့်အခါတိုင်း str__() method ကို ခေါ်သုံးသည်။ ထို object အတွင်း၌ ရှိသည့် string များကို ရယူ(retrieve)ရန်အတွက် ဖြစ်သည်။

14.8.2 The __del__() Method

__del__() method သည် ရှိပြီးသာ object ကို ဖျက်ဖြစ်ရန်အတွက် ဖြစ်သည်။ object ကို ဖျက်လိုသည့်အခါ __del__() method ကို ခေါ်(call)သုံးသည်။ အောက်တွင် str__() method နှင့် __del__() method ကို ဥပမာဖြင့် ဖော်ပြမည်။

Point object နှစ်ခု ပြုလုပ်သည်။

```
u = Point(0,1)
v = Point(2,2)
print(u, v)
```

```
(0, 1) (2, 2)
```

```
str(v)
```

```
'(2, 2)'
```

```
v.__str__()
```

```
'(2, 2)'
```

```
del(v)   # delete v
v
```

```
Object deleted
NameError: name 'v' is not defined
```

14.8.3 The__repr__method

__repr__method သည် class အတွင်း၌ တည်ဆောက်လိုက်သည့် object များကို ဖော်ပြပေးရန် အတွက် ဖြစ်သည်။ (returns the object representation) ။ ဥပမာ အောက်တွင် vector class ကို အသုံးပြု၍ vector object တစ်ကိုတည်ဆောက်သည်။ ထို vector a ကို ကြည့်ရန် print(a)ကို သုံးသည်အခါ အောက်တွင် ဖော်ပြထားသည့် memory reference ကိုသာ မြင်ရသည်။

```
class Vec2D(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

a = Vec2D(4,5)
print(a)
```

```
<__main__.Vec2D object at 0x00000251EDC6CC08>
```

__repr__method ကို ထည့်ရေးလိုက်လျှင် vector a ဖော်ပြ ပေးသည်။

```
class Vec2D(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self): # returns the object representation
        return ("({0.x!r}, {0.y!r})".format(self))

a = Vec2D(4,5)
print(a)
```

```
(4, 5)
```

__repr__method ကို အောက်တွင် ဖော်ပြထားသည့်အတိုင်း (၂)မျိုး ရေးနိုင်သည်။

```
print(repr(a))        # vector a ဖော်ပြရန် ရေးနည်း
print(a.__repr__())   # vector a ဖော်ပြရန် ရေးနည်း
```

```
(4, 5)
(4, 5)
```

14.8.4 The __iter__ method

__iter__ method သည် iterate လုပ်၍ မရသည်အရာများကို iterable ဖြစ်အောင် ပြုလုပ်ပေးသည်။ **__iter__** သည် unpacking လုပ်သည်အခါတွင် သုံးသည်။ Unpacking ဥပမာတစ်ခုမှာ vector တစ်ခု၏ x တန်ဖိုးနှင့် y တန်ဖိုးကို ခွဲထုတ်ခြင်း ဖြစ်သည်။ အပေါ်တွင် ဖော်ပြခဲ့သည့် vector a ကို x တန်ဖိုးနှင့် y တန်ဖိုး သီးခြားဖြစ်အောင် unpack လုပ်လိုကြည့်လျှင်__iter__ method ထည့်မရေးထားသောကြောင့် **TypeError**: cannot unpack non-iterable Vec2D object ပေးလိမ့်မည်။ အောက်တွင် **__iter__** method ဖြင့် iterable ဖြစ်အောင် ပြုလုပ်ပေးသည်။

```
class Vec2D(object):
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self): # returns the object representation
        return ("({0.x!r}, {0.y!r})".format(self))

    def __iter__(self):   # __ iter__ method ကို define လုပ်သည်။
        return (i for i in (self.x, self.y))

a = Vec2D(4,5)
x, y = a
print(x)
print(y)
```

```
4
5
```

```
import math
class Vec2D(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):       # vector (၂)ခု ပေါင်းခြင်း
        return Vec2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):      # vector နုတ်ခြင်း
        return Vec2D(self.x - other.x, self.y - other.y)

    def __mul__(self, other):      # vector မြှောက်ခြင်း
        return self.x*other.x + self.y*other.y

    def __abs__(self):             # Return the magnitude of vector
        return math.sqrt(self.x**2 + self.y**2)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)

    def __ne__(self, other):
        return not self.__eq__(other)  # reuse __eq__

    def __repr__(self):
        return ("({0.x!r}, {0.y!r})".format(self))
```

class Vec2D အသုံးပြု၍ vector u, v, w တို့ကို ပြုလုပ်သည်။

```
u = Vec2D(2,5)
v = Vec2D(4,2)
w = Vec2D(1,3)
a = u + v                       # (x+x , y+y)
print(u)
print(v)
print(a)
```

```
(2, 5)
(4, 2)
(6, 7)
```

14.8.5 The __add__ method

The __add__ method ၏ operator သည် အပေါင်းလက္ခဏာ(+) ဖြစ်သည်။ ပေါင်းခြင်း ပြုလုပ်ရန် ကိန်းနှစ်ခု(self, other) လိုအပ်သည်။ __add__ သည် vector ၏ x တန်ဖိုးအချင်းချင်း y တန်ဖိုးအချင်းချင်း ပေါင်းခြင်းဖြစ်သည်။

`object.__add__(self, other)`

```
print(u + v)                # ( x+x , y+y )
print(u.__add__(v))
```

```
(6, 7)
(6, 7)
```

14.8.6 The __sub__ method

__sub__ method ၏ operator သည် အပေါင်းလက္ခဏာ(-) ဖြစ်သည်။ နုတ်ခြင်း ပြုလုပ်ရန် ကိန်းနှစ်ခု(self, other) လိုအပ်သည်။ __sub__ သည် vector ၏ x တန်ဖိုးအချင်းချင်း y တန်ဖိုးအချင်းချင်း နုတ်ခြင်း ဖြစ်သည်။

`object.__sub__(self, other)`

```
print(u - v)
print(u.__sub__(v))
```

```
(-2, 3)
(-2, 3)
```

14.8.7 The __mul__ method

__mul__ method ၏ operator သည် အမြှောက်လက္ခဏာ(*) ဖြစ်သည်။ မြှောက်ခြင်း ပြုလုပ်ရန် ကိန်းနှစ်ခု(self, other) လိုအပ်သည်။ __sub__ သည် vector ၏ x တန်ဖိုးအချင်းချင်း y တန်ဖိုးအချင်းချင်း မြောက်ခြင်း ဖြစ်သည်။

`object.__mul__(self, other)`

```
print(u * v)
print(u.__mul__(v))
```

```
18
18
```

14.8.8 The __abs__ method

__abs__ method ၏ operator သည် abs() ဖြစ်သည်။ abs() ပြုလုပ်ရန် ကိန်းတစ်ခု (self) လိုအပ်သည်။ abs() သည် vector ၏ x နှစ်ထပ်ကိန်းနှင့် vector ၏ y နှစ်ထပ်ကိန်းကို ပေါင်း၍ square root ယူထားခြင်း ဖြစ်သည်။

```
print(abs(u))
print(u.__abs__())
```

```
5.385164807134504
5.385164807134504
```

14.8.10 The __eq__ method

__eq__ method ၏ operator သည် ညီမျှခြင်းလက္ခဏာ နှစ်ခု(==) ဖြစ်သည်။ == ပြုလုပ်ရန် ကိန်းနှစ်ခု(self, other) လိုအပ်သည်။ __eq__ သည် vector ၏ x တန်ဖိုးအချင်းချင်း y တန်ဖိုးအချင်းချင်း နှိုင်းယှဉ်ခြင်း ဖြစ်သည်။ နှစ်ခုစလုံးညီလျှင် True ပေးသည်။

```
print(u.__eq__(v))
```

```
False
False
```

14.8.11 The __ne__ method

__ne__ method ၏ operator သည် not equal(!=) လက္ခဏာ ဖြစ်သည်။ != ပြုလုပ်ရန် ကိန်းနှစ်ခု(self, other) လိုအပ်သည်။ __ne__ သည် vector ၏ x တန်ဖိုးအချင်းချင်း y တန်ဖိုးအချင်းချင်း နှိုင်းယှဉ်ခြင်း ဖြစ်သည်။ x သို့မဟုတ် y တစ်ခုခု မညီလျှင် True ပေးသည်။

```
print(u != v)
print(u.__ne__(v))
```

```
True
True
```

14.9 Overview of Magic Methods

Binary OperatorsBinary Operators

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__truediv__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)

^	object.__xor__(self, other)
|	object.__or__(self, other)

Extended AssExtended Assignmentsignments

Operator	Method
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other)
**=	object.__ipow__(self, other[, modulo])
<<=	object.__ilshift__(self, other)
>>=	object.__irshift__(self, other)
&=	object.__iand__(self, other)
^=	object.__ixor__(self, other)
|=	object.__ior__(self, other)

Unary OUnary Operatorsperators

Operator	Method
-	object.__neg__(self)
+	object.__pos__(self)
abs()	object.__abs__(self)
~	object.__invert__(self)
complex()	object.__complex__(self)
int()	object.__int__(self)
long()	object.__long__(self)
float()	object.__float__(self)
oct()	object.__oct__(self)
hex()	object.__hex__(self

ComparisComparison Operatorson Operators

Operator	Method
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)

> object.__gt__(self, other)

14.10 Built-in class attributes

Python class တွင် class ၏ အချက်အလက်များကို ဖော်ပြပေးမည့် built-in class attribute တချို့ ပါဝင်သည်။

SN	Attribute	Description
1	__dict__	It provides the dictionary containing the information about the class namespace.
2	__doc__	It contains a string which has the class documentation
3	__name__	It is used to access the class name.
4	__module__	It is used to access the module in which, this class is defined.
5	__bases__	It contains a tuple including all base classes.

__dict__

__dict__ သည် class namespace နှင့် ပတ်သက်သည့် အချက်အလက်များကို ဖော်ပြပေးသည့် dictionary ဖြစ်သည်။

__doc__

__doc__သည် class နှင့် သက်ဆိုင်သည့် မှတ်တမ်းများ(documentation) ဖြစ်သည်။

__name__

__name__ သည် class ၏ နာမည်ကို access လုပ်ရန် ဖြစ်သည်။

__module__

__module__ သည် class ကို define လုပ်ထားသည့် module ကို access လုပ်ရန် ဖြစ်သည်။

__bases__

__bases__ သည် all base classes များအားလုံးပါဝင်သည့် tuple ဖြစ်သည်။

Built-in class attribute များကို ဥပမာဖြင့်ထားသည်။

```
class Operator:      # class တစ်ခု တည်ဆောက်သည်။
    def __init__(self, x, y):
    """This Operator docstring."""
        self.x = x
        self.y = y
```

```
    def add(self):
        return self.x + self.x

    def multiply(self):
        return self.x * self.x

my_operator = Operator(3,4) # class ၏ object တစ်ခု define လုပ်သည်။

print(my_operator.add())
print(my_operator.multiply())

print(my_operator.__doc__)
print(my_operator.__dict__)      # object dictionary
print(my_operator.__module__)
```

```
6
9
None
{'x': 3, 'y': 4}
__main__
```

```
Operator.__dict__   # class dictionary
```

```
mappingproxy({'__module__': '__main__',
              '__init__':<function __main__.Operator.__init(self,x,y)>,
              'add': <function __main__.Operator.add(self)>,
              'multiply': <function __main__.Operator.multiply(self)>,
              '__dict__': <attribute '__dict__' of 'Operator' objects>,
              '__doc__': None})
```

End