Chapter – 17 Modules and Packages

Contents

17.11 Functions, Modules and Packages (Modularization) 2
17.1.2 Python Modules: Overview 2
17.2 Module များ တည်ဆောက်ခြင်း(Defining Modules) 2
17.3 Packages 4
17.3.1 __init__.py 4
17.4 Import Modules 4
17.4.1 Module တစ်ခုလုံးကို Import လုပ်ခြင်း 4
17.4.2 Module ထဲမှ အလိုရှိသည့် အရာကိုသာ Import လုပ်ခြင်း 5
17.4.3 Import လုပ်ပြီး မိမိကြိုက်သည့် အမည်သို့ပြောင်းခြင်း 5
17.4.4 Asterisk (*) ဖြင့် ရှိသမျှ အကုန် Import လုပ်ခြင်း 5
17.4.5 Absolute Path Import 6
17.4.6 Relative Path Import 6
17.5 Import Example -1 6
17.6 When Classes and Objects versus Modules 7
17.7 ImportError 8
17.8 The Module Search Path 8

Chapter – 17 Modules and Packages

Python သည် modular programming တစ်ခုဖြစ်သည်။ Modular programming ဆိုသည်မှာ ကြီးမား သည့် ပရိဂရမ် သို့မဟုတ် လုပ်ဆောင်စရာများကို အလုပ်ငယ်(subtask)များ သို့မဟုတ် module များ ဖြစ်အောင် စိတ်ပိုင်း(break)ပြီး program လုပ်နိုင်သည့် programming language ဖြစ်သည်။

Code များကို မော်ဂျူလာဖြစ်အောင်(modularizing) ရေးနိုင်ခြင်းကြောင့် ရရှိနိုင်သည့် အကျိုးများ

- ရိုးရှင်း လွယ်ကူသည်။(Simplicity)
- ပြုပြင်ထိန်းသိမ်းရန်လွယ်ကူသည်။ (Maintainability)
- အကြိမ်များစွာ ပြန်သုံးပြုနိုင်သည်။ (Reusability)
- namespace များကို သီးခြား(separate) ဖြစ်အောင် ခွဲထုတ်နိုင်သည်။

17.1.1 Functions, Modules and Packages (Modularization)

Function များကို စု၍ module အဖြစ် ပြုလုပ်နိုင်သည်။

Module များကို စု၍ package အဖြစ် ပြုလုပ်နိုင်သည်။

17.1.2 Python Modules: Overview

Module များနှင့် ပတ်သက်၍ module များကို ပြုလုပ်ခြင်း(ရေးသားခြင်း)နှင့် module များကို ခေါ်သုံးခြင်း ဟူ၍ အပိုင်း(၂)ပိုင်း ရှိသည်။

Python တွင် နည်း(၃)နည်းဖြင့် module တစ်ခုဖြစ်အောင် define လုပ်နိုင်သည်။

၁။ Python ပရိဂရမ်ထဲတွင် ထည့်ရေးခြင်း
၂။ Module ကို C language ဖြင့်ရေး၍ run သည့်အခါ(run-time)မှ ဆွဲတင်ခြင်း(loaded dynamically)
၃။ Python interpreter ထဲတွင် အလိုအလျောက် ပါဝင်နေသည့် built-in module များ။ ဥပမာ itertools module

Module တစ်ခုသည် python(.py) ဖိုင်တစ်ခုသာ ဖြစ်သည်။ ယေဘူယျအားဖြင့် အလုပ်လုပ်ပုံ တူညီသည့် သို့မဟုတ် အမျိုးတူသည့် function များ သို့မဟုတ် class များကို module တစ်ခုအဖြစ် တည်ဆောက်ထား လေ့ရှိသည်။ Module များကို မိမိကြိုက်နှစ်သက်သည့် နာမည်ပေးနိုင်သော်လည်း Python က သတ်မှတ်ထားသည့် နာမည်ပေးသည့် စည်းမျဉ်များ(naming convention)ကို လိုက်နာရန် လိုသည်။

17.2 Module များ တည်ဆောက်ခြင်း(Defining Modules)

Python ကုဒ်များဖြင့် ရေးထားသည့် **.py** တစ်ခုသည် Python module တစ်ခုဖြစ်သည်။ (a valid Python module is any .py file containing valid Python code.)

အောက်တွင် my_module.py အမည်ဖြင့် module တစ်ခုကို တည်ဆောက်ထားသည်။ my_module ဟု အမည်ပေးထားသည့် module ထဲတွင် class (၂)ခု ပါဝင်သည်။ Class တစ်ခုစီတွင် function တစ်ခုစီ ပါဝင်သည်။ `my_module1.py` သည် module တစ်ခုဖြစ်သည်။ အောက်ကဖိုင်သည် `my_module1.py` ဖိုင် ဖြစ်သည်။

```
# my_module1.py       (\my_package\my_module1.py)
class M_class_a1:
    def a1():
        print("This is 'my_module1.py' file.
               This is function a1 from M_class_a1")


class M_class_b1:
    def b1():
        print("This is 'my_module1.py' file.
               This is function b1 form M_class_b1")
```

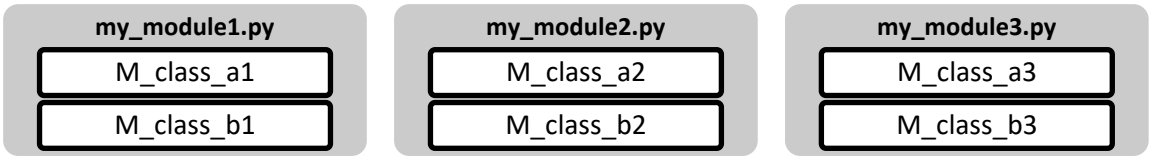
အောက်ကဖိုင်သည် **my_module2.py** ဖိုင် ဖြစ်သည်။

```
# my_module2.py               (\my_package\my_module1.py)
class M_class_a2:
    def a2():
        print("This is 'my_module2.py' file.
               This is function a2 from M_class_a2.")

class M_class_b2:
    def b2():
        print("This is 'my_module2.py' file.
               This is function b2 from M_class_b2.")
```

အောက်ကဖိုင်သည် **my_module2.py** ဖိုင် ဖြစ်သည်။

```
# my_module3.py               (\my_package\my_module1.py)
class M_class_a3:
    def a3():
        print("This is 'my_module3.py' file.
               This is function a3 from M_class_a3")

class M_class_b3:
    def b3():
        print("This is 'my_module3.py' file.
               This is function b3 from M_class_b3 ")
```

ထိုကဲ့သို့ module (၃) ခုတည်ဆောက်ထားသည်။



17.3 Packages

Module တွေ တဖြည်းဖြည်းဖြင့် များလာသည့်အခါ နောက်ထပ်အဆင့်တစ်ခုဖြစ်သည့် package များ ပြုလုပ်ကြသည်။ Package ဆိုတာ Module တွေ စုစည်းထားတဲ့ folder တစ်ခု ဖြစ်သည်။ (Package is a collection of modules in a folder.)။ Folder နာမည်သည် package နာမည်ဖြစ်သည်။ Package တစ်ခုသည် directory တစ်ခုဖြစ်သည်။ Package တစ်ခုထဲတွင် တခြားသော sub package များနှင့် module များ ပါဝင်နိုင်သည်။ သက်ဆိုင် module လေးများကို folder တစ်ခုအောက်တွင် တစ်စုတစ်ဝေး ထားခြင်းသည် package တစ်ခု ပြုလုပ်ခြင်း ဖြစ်သည်။

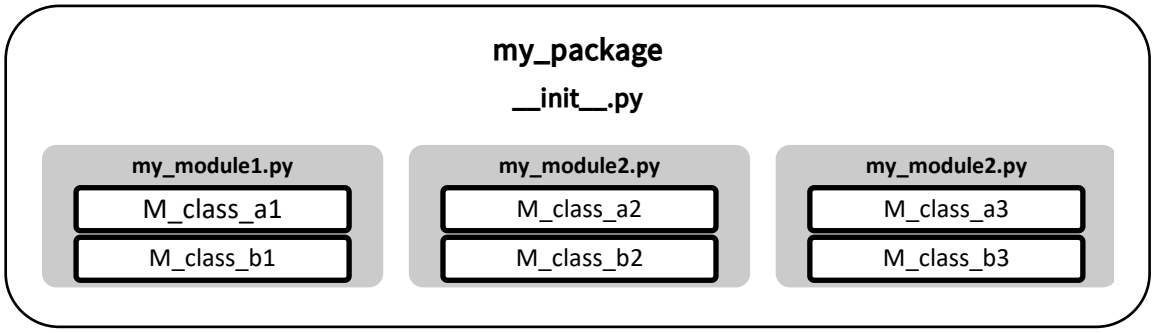
17.3.1 __init__.py

Folder တစ်သည် package တစ်ခုဖြစ်ကြောင်း မည်ကဲ့သို့ သိနိုင်မည်နည်း?

Package လုပ်မည့် folder ထဲတွင် **__init__.py** ဆိုသည့် python file ရှိနေရပါမည်။ **__init__.py** က ဘာမှမရှိသည့် empty file တစ်ခု လည်း ဖြစ်နိုင်သည်။ **__init__.py** မရှိရင် package ထဲက module တွေကို import လုပ်လို့ မရပါဘူး။

Folder တစ်ခု သို့မဟုတ် directory တစ်ခုထဲတွင် **__init__.py** ပါဝင်နေပါက ထို folder သို့မဟုတ် directory သည် Python package တစ်ခု ဖြစ်သည်။

Folder တစ်ခုထဲတွင် **__init__.py** ရှိနေခြင်းသည် Python interpreter အား ထို folder သည် package တစ်ခုဖြစ်ကြောင်း အသိပေးခြင်း ဖြစ်သည်။ ထို့အပြင် ထို folder ထဲရှိ ဖိုင်များသည် module များ ဖြစ်ကြောင်း ကိုလည်း register လုပ်ခြင်း ဖြစ်သည်။



အသုံးများသော package များမှာ

Numpy

Matplotlib

Scikit-learn

17.4 Import Modules

Import keyword ဖြင့် import လုပ်သည်။ Import လုပ်နိုင်သည့် နည်းများစွာ ရှိသည်။

17.4.1 Module တစ်ခုလုံးကို Import လုပ်ခြင်း

Import keyword ဖြင့် module နာမည်ကို ရေး၍ module တစ်ခုလုံးကို import လုပ်သည်။

```
import my_package.my_module1
```

17.4.2 Module ထဲမှ အလိုရှိသည့် အရာကိုသာ Import လုပ်ခြင်း

from . . . import ဖြင့် module တစ်ခုထဲမှ အလိုရှိသည့် အရာကိုသာ import လုပ်နိုင်သည်။

```
from my_package import my_module1
my_module1.M_class_a1
```

```
my_package.my_module1.M_class_a1
```

```
from my_package import my_module1
my_module1.M_class_a1.a1()
```

```
This is 'my_module1.py' file. This is function a1 from M_class_a1
```

17.4.3 Import လုပ်ပြီး မိမိကြိုက်သည့် အမည်သို့ပြောင်းခြင်း

Module ကို import လုပ်သည့်ပြီး သုံးသည့်အခါတိုင်း နာမည် အပြည့်အစုံ ရေးရသည်။ နာမည် အပြည့်အစုံ ရေးလျှင် ရှည်လွန်းလှသောကြောင့် နာမည်အတိုခေါက်ရေး ရန်တွက် **as** keyword ကို သုံး၍ နာမည် ပြောင်းနိုင်သည်။

```
from my_package import my_module2 as mymo2
mymo2.M_class_a2.a2
```

```
<function my_package.my_module2.M_class_a2.a2()>
```

```
from my_package import my_module2 as mymo2
mymo2.M_class_a2.a2()
```

```
This is 'my_module2.py' file. This is function a2 from M_class_a2
```

```
import my_package.my_module1 as mm
mm.M_class_a1.a1()
mm.M_class_b1.b1()
```

```
This is 'my_module1.py' file. This is function a1 from M_class_a1.
This is 'my_module1.py' file. This is function b1 form M_class_b1.
```

17.4.4 Asterisk (*) ဖြင့် ရှိသမျှ အကုန် Import လုပ်ခြင်း

Module တွင်းရှိ class များနှင့် function များအားလုံးကို current namespace အတွင်းသို့ import လုပ်သည်။ တခြားသော import လုပ်ပြီးသား class များ သို့မဟုတ် function များနှင့် clash ဖြစ်နိုင်သောကြောင့် သတိပြုသင့်သည်။

```
from my_package.my_module1 import *
print(M_class_a1.a1())
print(M_class_b1.b1())
```

```
This is 'my_module1.py' file. This is function a1 from M_class_a1.
None
```

```
This is 'my_module1.py' file. This is function b1 form M_class_b1.
None
```

17.4.5 Absolute Path Import

Absolute path import နည်းဖြင့် import လုပ်မည်ဆိုပါ path အပြည့်အစုံ ထည့်ရေးပေးရမည်။

my_package ထဲမှ my_module1 ထဲမှ M_class_a1 ကို import လုပ်သည်။

```
import my_package.my_module3
my_package.my_module3.M_class_a3.a3()
```

```
<my_package.my_module1.M_class_a1 at 0x1cf77364f48>
```

from ကို သုံးသည့်နည်း-၁ (module ထဲမှ class ကို import လုပ်သည်။)

my_package ထဲမှ **my_module2** ထဲမှ **M_class_b2()** ကို import လုပ်သည်။

```
from my_package.my_module2 import M_class_b2()
obj = M_class_b2
```

from import ကို သုံးသည့်နည်း-(၂) (package ထဲမှ class ကို import လုပ်သည်။ Class နာမည်တွင် module နာမည် ထည့်ရေးပေးရသည်။)

```
from my_package import my_module2.M_class_b2
obj = my_module2.M_class_b2()
```

မိမိကြိုက်နှစ်သက်သည့်နည်းကို သုံးနိုင်သည်။ လက်သည်းကွင်း() ထည့်ရန်၊ မရန် သတိပြုပါ။

ကော်မာခံရေး၍ module နှစ်ခုကို တစ်လိုင်းထဲတွင် import လုပ်နိုင်သည်။ ဥပမာ- my_package ထဲမှ my_module1, my_module2 နှင့် my_module3 တို့ကို ကော်မာခံရေး၍ တစ်လိုင်းထဲတွင် import လုပ်သည်။

```
from my_package import my_module1, my_module2, my_module3
```

17.4.6 Relative Path Import

Relative path import အကြောင်း မဖော်ပြပါ။

17.5 Import Example -1

ပထဦးစွာ calculator.py ဆိုသည့် module တစ်ခု ပြုလုပ်၍ import လုပ်နည်း ရေးသားပုံ အနည်းငယ် ကွဲပြားခြင်းကို ဖော်ပြမည်။

```
def add(x, y):
    return x + y
```

dot (.) notation ကို သုံးသည်။

Calculator module ကို **import** လုပ်သည်။ module အတွင်းရှိ **add()** function ကို **dot (.)** notation ဖြင့် ခေါ်သုံးသည်။

```
import calculator
calculator.add(8, 9)
```

```
17
```

from...import... syntax

Calculator module ထဲမှ **add()** function ကို import လုပ်သည်။ Package သို့မဟုတ် module အတွင်းမှ မိမိ အလိုရှိသည့် အရာကိုသာ ရွေး၍ import လုပ်ခြင်း ဖြစ်သည်။

```
from calculator import add
add(8, 9)
```

```
17
```

```
from calculator import *
```

Calculator module ထဲမှ ရှိသမျှ အားလုံးကို import လုပ်သည်။ * သုံးခြင်းကို အားမပေးပါ။

as keyword ကို သုံး၍ နာမည်ပြောင်းခြင်း

Calculator module ကို import လုပ်ပြီး module နာမည်ကို calc ဟု ပြောင်းသည်။

```
import calculator as calc
calc.add(8,9)
```

```
17
```

မှတ်ရန်အချက်မှာ မိမိ import လုပ်သည့် ပုံစံ(ကုဒ်ရေးသည့်ပုံစံ)ကို မူတည်၍ ထို module ကို သုံးသည့်အခါ ကုဒ်ရေးသည့် ပုံစံ အနည်းငယ်လိုက်ပြောင်းလဲ ရသည်။

17.6 When Classes and Objects versus Modules

Class အဖြစ် code ရေးမည်လား? module အဖြစ် code ရေးမည်လား? ခွဲခြားရန်အတွက် ညွှန်ကြားချက်များ

- မတူညီသည့် attributeများ(internal states)နှင့် တူညီသည့် behavior(methods) များရှိနေသည့် instance များအတွက် object များအဖြစ် ကုဒ်လုပ်ခြင်းသည် အကောင်းဆုံးဖြစ်သည်။
- Class များတွင်သာ inheritance ကို သုံးနိုင်သည်။ တစ်နည်းအားဖြင့် class များကသာ inheritance ကို support လုပ်သည်။ Module များက inheritance ကို support မလုပ်ပါ။
- သီးခြားလုပ်ဆောင် ပေးနိုင်သည့်အရာတစ်ခုအဖြစ် ရှိနေနိုင်ပါက module အဖြစ် ထားရှိသင့်သည်။ Program ထဲတွင် Python module import လုပ်သည့်ကုဒ် အကြိမ် မည်မျှ ရေးထားပါစေ တစ်ကြိမ်သည် load လုပ်သည်။

- Value များစွာ ပါဝင်သည့် variable များကို function များထဲသို့ argument အဖြစ် ထည့်ပေး(pass လုပ်ရန်) လိုအပ်ပါက class များ အဖြစ် define လုပ်သင့်သည်။ ဥပမာ

ပြဿနာများကို အရိုးရှင်းဆုံးနည်းဖြင့် ဖြေရှင်းပါ။ Dictionary, list, tupleစသည်တို့သည် class သို့မဟုတ် module ထက် ပို၍ သေးငယ်ပြီး လျှင်မြန်စွာ အဖြေထုတ်ပေးနိုင်သည်။ ဖြစ်နိုင်လျှင် dictionary, list, tuple စသည် တို့ကို တိုက်ရိုက်သုံးပါ။

17.7 ImportError

ImportError သည် module တစ်ခုခု import လုပ်၍ မရသည့်အခါတွင် ဖြစ်ပေါ်သည်။ ဥပမာ- မရှိသည့် (non-existent) module တစ်ခုကို import လုပ်မိသောကြောင့် ဖြစ်နိုင်သလို ၊ module နာမည်ကို စာလုံးပေါင်း မှားရိုက်မိသောကြောင့်လည်း ဖြစ်နိုင်သည်။ ModuleNotFoundError ထုတ်ပေး(raise)လိမ့်မည်။

ModuleNotFoundError သည် ImportError class ၏ subclass ဖြစ်သည်။

```
import nonexistentmodule
```

```
ModuleNotFoundError          Traceback (most recent call last)
----> 1 import nonexistentmodule
ModuleNotFoundError: No module named 'nonexistentmodule'
```

17.8 The Module Search Path

သင် module တစ်ခုကို import လုပ်လိုက်သည့်အခါ Python interpreter သည် built-in module တစ်ခု ဟုတ် မဟုတ် အရင် module name စစ်ဆေးသည်။ ဟုတ်လျှင် built-in module ကို ဆွဲတင်(load လုပ်)ပေးသည်။ Built-in module များထဲတွင် module လုပ်မည့် module name ကို မတွေ့လျှင် **sys.path** အတွင်းရှိ directories များထဲတွင် import လုပ်မည့် module name နှင့် တူသည် .py extension ဖိုင်ကို လိုက်ရှာသည်။

ထို့ကြောင့် module များ package များနှင့်ပတ်သက်လာလျှင် PYTHONPATH ကို နားလည်သဘောပေါက် ထားရန်လိုသည်။

```
import sys
sys.path
```

```
['C:\\ProgramData\\Anaconda3\\lib\\site-packages',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\IPython\\extension
s',
 'C:\\Users\\kaung\\.ipython']
```

ကွန်ပျူတာတစ်လုံးနှင့် တစ်လုံး၏ **sys.path** မတူနိုင်ပါ။

```
import sys
```

```
dir(sys)
```

- End -