Modules and Packages :

ماژول کردن یعنی یک مسئله ی بزرگ را به ماژول‌های کوچکتر بشکنید که هر کدام از این‌ها را به راحتی مدیریت کنید و به تنهایی یک سری از کارها را انجام دهند و از هم مستقل باشند. این ماژول‌های مستقل، تشکیل پروژه‌ی اصلی را می دهند. در تابع جاهایی که قرار است چند بار استفاده شود و منطق مشخصی دارد و بسته‌ای از عملیات انجام می‌دهد را در یک تابع می‌بریم ولی ماژول‌ها، scope بزرگتری هستند و بعد به package ها می‌رسیم و از این بزرگتر programing ی است که این طور نگاه می‌کند.

ماژول کردن چهار فایده اصلی دارد.

۱) ساده تر می‌شود و به بخش های کوچکتر شکسته می‌شود و آن ماژول قرار است یک کار کوچک تر انجام دهد محدوده مشخص دارد، توسعه‌ی آن راحت‌تر است و بهتر می‌شود debug کرد و مطمئن می‌شویم debug ندارد.

۲) می‌توان برای هر محدوده از کد اصلی یک ماژول نوشت، مثلاً این بخش از کد فقط قرار است event ها را هندل کند و بخشی دیگر یوزرها که بعدا اگر بخواهیم چیزی اضافه کنیم راحت تر است چون نگهداری و تغییر آن روی کل پروژه تاثیر گذار نیست.

۳) موضوع دیگر reusable بودن است که می‌شود یک ماژول را برای کد های دیگر هم استفاده کرد مثلاً لاگین و لازم نیست دوباره نوشت.

۴) هر کدام از ماژول ها فضای خودشان را دارند و باعث شده مشکلات مختلف که موقع کار کردن پیش می‌آید در اسکوپ های مختلف کمتر شود.

ماژول‌های csv, logging و از این قبیل ماژول های آماده پایتون هستند و با import کردن به برنامه وارد می شود و با ctrl+b می توان آن ماژول را باز کرده و دید.

ساخت دایرکتوری:

File > New > Directory > modules (for example) > R click > New > python file > modules_exp.py (for example)

بعد وقتی می‌خواهیم استفاده‌کنیم با همین نام استفاده می‌کنیم.

Example:

modules_exp.py

```
class Test:
    pass
def greeting(name):
    print(f'hello{name}')
user = "meimanat"
```

در این ماژول یک کلاس یک تابع و یک متغیر تعریف شده است.

حالا برای استفاده داخل main.py به صورت زیر import می‌کنیم:

```
import modules_exp
```

اگر بخواهیم با آنچه داخل ماژول است کار کنیم، در main به طور مثال به صورت زیر می باشد:

در اینجا به یوزر دسترسی دارد:

```
print(modules_exp.user)
```

```
test_class = modules_exp.Test( )
```

فانکشن داخل ماژول را صدا کرده:

```
modules_exp.greeting("Mim'')
```

output:
meimanat
hello Mim

باید اسم ماژول را نوشته و بعد اسم کلاس یا تابع یا ....

غیر از روش گفته شده دو روش دیگر برای Import کردن وجود دارد:

```
import name module(for example: modules_expname) as my_module(for example)
print(my_module.user)
test_class = my_module.Test( )
my_module.greeting(“Mim’’)
```

output:
meimanat
hello Mim

حالت دیگر اینکه از ماژولی که داریم ممکن است همه آن ماژول را نخواهیم و یا بخواهیم منحصراً اشاره کنیم به چیزی که داریم import میکنیم:

```
from name module (for example: modules_exp) import Test,greeting
test_class = Test( )
greeting (“Mim”)
```

* که اینجا با همان Test و greeting می شناسد و اگر به جای اینکه تک تک را بیاوریم همه را بخواهیم بیاوریم گذاشته که کلاس و متغیر و غیره قابل دسترسی است ولی پیشنهاد نمی شود چون ممکن است با اسم گذاری main یکی باشد و به conflict بخوریم.

از این روش می توان برای فراخوانی بخشی از کلاس ماژول هم استفاده کرد:

```
from name module (for example: modules_exp) import Test as class Test
```

بستگی به استفاده می توان چند کلاس را تبدیل به یک ماژول کرد.

تا الان ماژولی که استفاده می کردیم در یک دایرکتوری با main بودند ولی ممکن است همیشه این طور نباشد شاید لازم باشد از جای دیگری import کرد.

برای اینکه interpreter پایتون بیاید کد را بخواند و ماژول‌ها را که خواستیم پیدا کند چند مسیر را به طور پیش فرض چک می کند؛ ابتدا current directory و بعد لیستی از دایرکتوری های که در یک ... environment به اسم python path ذخیره شده که باید set شود که بستگی به os شما دارد و برای os های مختلف فرمتش فرق دارد. مسیری که پایتون چک می کند، وقتی چیزی را میخواهیم اضافه کنیم در python path چک می‌کند و می‌بیند که کجا بود پیدا می‌کند، برای اینکه ببینیم python path کجاست یک ماژولی هست به نام sys که مربوط به کارهای سیستمی است.

```
import sys
print(sys.path)
```

این، یک لیست آدرس می دهد که مربوط سیستم می باشد.

اولین مسیر current directory است که ممکن است خالی گذاشته باشد و بعدی ها بقیه مسیرهاست، اینها دایرکتوری هایی است که وقتی import می‌کنیم از آنها می خواند و از این آدرس‌ها ماژول را پیدا می‌کند. اگر از ماژولی استفاده می‌کنیم که در مسیر دیگری قرار گرفته، می‌توان به طور دائم sys.path را تغییر داده و به

environment variable اضافه کرد و یک مسیری به آن اضافه کرد یا می‌توان همینجا در کد تغییراتی داد و به لیست sys.path یک مسیر دیگر را append کرد:

sys.path.append("/home/meimanat/")

پایتون برای خواندن متغیرها از قاعده ای پیروی می‌کند که local inclosing global building گفته می شود، این ترتیبی است که استفاده می کند که کد را بخواند و متغیرهایی که تعریف کردیم را بشناسداول به طور local نگاه می‌کند ببیند همین جا دارد یا نه بعد inclosing است یعنی scope بالایی، مثلا اگر دو تابع تو در تو بود اگر در تابع داخلی نبود یه level بالاتر را نگاه می‌کند، اگر نبود در حوزه global را نگاه می‌کند، اگر در local inclosing global نبود سراغ building که هنوز تعریف نکردیم می‌رود. نه تنها هر متغیر، هر کلید واژه را این طور می شناسد.

main.py

```
from    name module (for example: modules_exp)    import   *
greeting ("Mim")
a="out1"
def test( ):
      a = "in_fune"
      print(a)

print(a)
test( )
```

output:

```
hello Mim
out1
in_fune
```

اینجا هم که ماژول ها را تعریف می می‌کنیم هر ماژول ای space و scope خودش را دارد و بهتر این است که اسمی که در ماژول ها تعریف می شود با اسم در main برنامه متفاوت باشد و مشابه نباشد، که ممکن است طبق خواست ما اجرا نکند پس برای هر scope اسم متفاوت انتخاب کرده که دچار مشکل نشویم. وقتی اسم یک ماژول را refactor می کنیم در میان برنامه هم تغییر می‌کند.

greeting.py

```
class User:
     def _ _init_ _(self,name):
5          print ("Just Testing the class {name}.")
          self.name = name
 def greeting (name):
1          print (f ' hello{name}')
2          my_user = User(name)
*
user = "meimanat"
greeting(user)
```

output:

```
hello meimanat
Just Testing the class meimanat.
```

main.py

```
   import   greeting   as   my_module
3  test_class = my_module.User(“Mim”)
4  my_module.greeting(“Mim”)
```

output:

```
1  hello meimanat
2  Just Testing the class meimanat.
3  Just Testing the class Mim.
4  hello Mim
5  Just Testing the class Mim.
```

(ترتیب اجرا با شماره گذاری مشخص شده است.)

وقتی greeting را import می‌کنیم ابتدا greeting اجرا شده و بعد از آن استفاده کرده، ولی ما می‌خواهیم مستقل از خودش در خود برنامه اش استفاده کنیم و از کلاس ها و فانکشن هایش هر جا لازم باشد استفاده کنیم. برای این موضوع از ماژول به صورت اسکریپت استفاده می‌شود.

به مجموعه‌ای از دستورات و کامند ها که قرار است یک taskی را انجام دهند گویند. در گذشته یک سری زبان‌ها ابتدا کل کد کامپایل می‌شد و بعد اجرا می‌شد و یک سری از زبان‌ها مفسری بودند و خط به خط اجرا می شدند. پایتون به این قدرت رسید که خوب عمل کند و نخواهد اسکریپتی کار کند و برنامه‌های بزرگ با آن بنویسند که آن اسکریپت می‌توانست اشیا پایتون باشد ولی هنوز این اصطلاح وجود دارد وقتی از یک سری چیزی که تعریف شده چیزهایی که هست یک خروجی بگیرند به صورت ترکیبی از کامند ها و آن را اجرا کند به این اسکریپت گویند.

حالا مثلاً یک ماژولی داریم می‌خواهیم از اسکریپت اجرای آن را ببینیم برای این موضوع باید تمایز وجود داشته باشد وقتی هم اینجا میخواهیم اجرا کنیم یا از main برنامه می خواهیم از آن استفاده کنیم و آن جا import کنیم. اگر بخواهیم مثلاً ماژول greeting در همان اسکریپت خودش اجرا کنیم از _ _name_ _ استفاده می‌کنیم و شرط را در (❊) می‌گذاریم:

```
if _ _name_ _ = = ‘_ _main_ _’:
```

و الان اگر این ماژول را اجرا کنیم:

Output:

```
hello meimanat
Just Testing the class meimanat.
```

و اگر main برنامه را اجرا کنیم:

```
Just Testing the class Mim.
hello Mim
Just Testing the class Mim.
```

و دیگر از بخش if به بعد را اجرا نکرد.

اگر در ماژول greeting قبل از if ، print(_ _name_ _) را بزنیم و ماژول را ران کنیم:

Output: _ _main _ _

ولی اگر main را ران کنیم:

Output: greeting

برابر با اسم ماژول می‌شود .

اگر فقط بخشی از ماژول را هم بخواهیم اجرا کنیم باز این مشکل وجود دارد و باید حتماً _ _name_ _ if را در ماژول قرار دهیم .
package یک scope بالاتر است و شامل یک سری ماژول هاست و شاید خیلی بزرگ و آن package هم می تواند sub package نیز داشته باشد. برای اضافه کردن پایتون پکیج با راست کلیک روی اسم پروژه می‌توان ایجاد کرد، تفاوتی که وجود دارد این است که در اینجا یک فایل هم init هم اضافه کرده (_ _init_ _.py) که آن پکیج را باید initialize کنیم.

فایلinit تا یک زمانی اجباری بوده ولی از احتمالاً ورژن ۳.۳ به طور ضمنی خودش می شناسد، برای اینکه ماژول هایی که داخل یک پکیج گذاشت تا بتواند بشناسد و تشخیص دهد لازم بود در فایل init این ماژول‌ها را تعریف می کردیم ولی الان لازم نیست که داخل فایل init تعریف کنیم و باز هم آن متد را می‌شناسد و به صورت پیش فرض init ساخته می‌شود.
برای ماژول بندی یک برنامه برای هر کلاسی یک فایل پایتون ساخته و هر کلاس را در فایل پایتون خودش گذاشته.
فرض کنید ۲فانکشن داشته باشید مثلاً فانکشن A و فانکشن B که در فانکشن A فانکشن B کال شود و وسط اینکه فانکشن B را اجرا کنید فانکشن A را کال کنید که در این صورت اینها وارد یک لوپ شده و وابستگی ایجاد شده که مشکل به وجود می‌آید حالا در کلاسها هم چنین چیزی ممکن است پیش بیاد مدام ماژول های مختلف import می کنیم و مثلاً به قبلی برمی‌گردیم که این مشکل به دلیل طراحی بد است و در چنین مواردی یا باید همه را در یک ماژول نگهداشت یا از اول طور دیگری طراحی کرد و کلاسها مستقل باشد و جدا باشد و وقتی کلاس ها را تعریف می کنیم به این صورت باشد که یک طرفه import شده باشد و مثلا از کلاس قبلی نخواهید import کنیم که وارد لوپ شود و مثلاً ماژول A ماژول B را import کند و بالعکس و ارتباط دو طرفه نباید باشد.
اگر بخواهیم ماژول را در یک مجموعه نگهداری کنید در یک package نگهداری می‌کنیم.
فرمت import پکیج : import package. module name

سوال درباره venv: وقتی میخواهیم کتابخانه ها را اضافه کنیم تک تک نصب نمی‌کنیم pip را نصب می کنیم تمام package ها نصب می‌شود. اگر جایی از برنامه پکیجی import کردیم و نمی شناسد یا ارور می دهد همان جایی که ارور می دهد باید یه چیزی را import کرد که می‌توان با زدن Alt + Enter چیزهایی که نیاز است خودش import کند.

کلاس استاد جلیلیان