

Python Module, Package, Library, and Framework.

Topic 2

[Module](#)

[Module Example](#)

[Packages](#)

[Library](#)

[Framework V.S. Library](#)

[모듈 및 라이브러리 다루기](#)

[모듈을 다룰 때 주의사항 1](#)

```
if __name__ == "__main__":
```

[모듈을 다룰 때 주의사항 2](#)

Topic 2

Python을 사용하다보면 Module(모듈), Package(패키지), Library(라이브러리) 그리고 Framework(프레임워크) 라는 단어를 자주 접하게 된다.

위 네개는 각각 무엇을 의미하며, 서로 차이점이 무엇인지 알아본다.

Module

기본적으로 우리가 작성하는 “.py” 확장명을 갖고있는 모든 파일은 하나의 파이썬 모듈 이라고 본다.

우리가 Python Interpreter(인터프리터) 만을 갖고 코드를 작성한다고 생각해보자. Python의 인터프리터는 대화형 인터프리터로 우리가 코드를 작성하면 해당 코드가 즉시 실행되며, 그에 상응하는 응답을 바로바로 받아볼 수 있다. 하지만 해당 인터프리터를 종료한 후 다시 인터프리터를 실행하게 되면 당연하게도 그동안 작성한 코드는 모두 날아가게 되며, 처음부터 다시 코드 작성을 해야한다.

이러한 번거로움을 없애고, 작성한 코드를 유지하기 위해서 우리는 “.py” 확장명의 파일을 생성하고 해당 파일에 Python 코드를 작성하게 된다. 해당 파일 안에 작성된 모든 변수, 함수, 클래스 등은 또 다른 Python 파일에서 손쉽게 갖고와서 사용할 수 있다.

Module Example

```
# func.py
def add_numbers(*args):
    return sum(args)

# main.py
from func import add_numbers

my_sum = add_numbers(1,2,3,4,5)
print(my_sum)  # 15

# sub.py
from func import add_numbers

my_sum = add_numbers(5,6,7,8,9)
print(my_sum)  # 35
```

위 예시를 살펴보면, "func.py" 라는 파일에 `add_numbers()` 라는 함수를 작성하였다.

여기에서 "func.py" 파일은 "func" 라는 이름의 Python 모듈이 되는 것이다.

이후, 해당 모듈 내의 `add_numbers()` 함수를 각 "main.py" 파일과 "sub.py" 파일에서 `import` 하여 갖고온 후 함수를 재 정의 하지 않고 바로 사용할 수 있었다.

이렇게 반복적인 코드 또는 유용한 기능을 모듈화 한다면, 필요할 때 언제든지 불러와 재활용 할 수 있으며, 전체적인 코드 베이스 길이도 짧아지게된다.

Packages

Package(패키지)란 모듈의 집합을 의미한다.

즉, 모듈이 하나의 Python 파일이었다면, 패키지는 이러한 여러 Python 파일을 모아놓은 하나의 폴더 (또는 디렉토리) 라고 생각할 수 있다.

예를들어 아래와 같은 예제를 살펴보자.

- Calculator
 - addition.py
 - subtraction.py
 - multiplication.py
 - division.py

Calculator (계산기) 라는 디렉토리 안에 각각의 계산기의 기능이 각 하나의 파이썬 파일로 작성되어있다.

이 경우, Calculator 패키지 라고 하며, 그 하위 각각의 기능을 addition (덧셈) 모듈, subtraction (뺄셈) 모듈, multiplication (곱셈) 모듈, division (나눗셈) 모듈 이라고 볼 수 있다.

Library

Library(라이브러리)란 패키지의 집합을 의미한다.

하나의 라이브러리 안에는 여러 패키지가 존재하며, 여러 패키지 안에는 각각의 기능을 수행하는 모듈이 존재하게된다.

아래 예제를 살펴보자.

- MultifunctionalCalculator
 - ScientificCalculator
 - exponential.py
 - modulus.py
 - sqrt.py
 - fraction.py
 - GraphingCalculator
 - tabulate.py
 - plot.py
 - trigonometry.py

- Calculator
 - addition.py
 - subtraction.py
 - multiplication.py
 - division.py

Multifunctional Calculator(다용도 계산기) 라이브러리의 예시이다.

해당 라이브러리 내에는 각 Scientific Calculator(공학용 계산기) , Graphing Calculator(그래프 계산기) 그리고 Calculator(일반 계산기) 라는 세개의 패키지가 존재한다.

그리고 각 패키지 하위에는 목적에 맞는 기능의 모듈이 존재한다.

Framework V.S. Library

Framework(프레임워크)는 종종 라이브러리와 혼용 되는 경우가 많다. 하지만 프레임워크와 라이브러리는 분명한 차이가 존재한다.

라이브러리는 개발자가 본인의 코드에 제 3자가 이미 만들어놓은 코드를 갖다 사용하는 개념이다. 마치 레고와 같이 필요한 모양의 부품이 있다면 해당 부품만 갖고와 끼워 맞추는 형식과 같다.

반면, 프레임워크란 정해진 틀 안에서 코드를 작성해야한다. 마치 퍼즐과 같이 특정 코드는 항상 특정 부분에 작성되어야 하는 등의 규칙을 따라야 한다.

프레임워크는 많은 강력한 기능을 이미 모두 내장하고 있는 경우가 많다. 따라서 크게 힘들이지 않고도 다양하고 강력한 기능들을 구현해낼 수 있다. 하지만, 프레임워크에 너무 의존하다 보면 해당 틀에 갇혀버릴 수 있으니 주의해야한다.

모듈 및 라이브러리 다루기

```
# my_func.py

def say_hello():
    print('hello')

def say_bye():
    print('bye')

def say_hi(name):
    print(f'hi {name}')
```

위와 같은 `my_func.py` 라는 모듈이 존재한다고 가정해보자.

위 모듈을 다른 파일에서 사용을 하기 위해서는 아래와 같이 `import` 구문을 사용할 수 있다.

```
# main.py

import my_func

my_func.say_hello()
# hello

my_func.say_bye()
# bye

my_func.say_hi('daniel')
# hi daniel
```

위와같이 `import`를 하게된다면 `my_func` 모듈 내 모든 함수를 `import` 하여 사용할 수 있다.

이렇게 `import`된 함수는 모두 메모리에 저장되게된다.

만약 해당 모듈에서 한개의 함수만 사용하고, 그 외 함수를 사용하지 않는다면 그만큼의 메모리 공간이 낭비되게된다. 따라서 아래 예시처럼 필요한 함수만 골라서 사용할 수도 있다.

```
# main.py

from my_func import say_hello

say_hello()
# hello
```

위와같이 특정한 함수만 import하게 된다면 그 외 나머지 함수는 import되지 않는다.

모듈을 다룰 때 주의사항 1

아래 코드를 보자.

```
# my_func.py

def say_hello():
    print('hello')

def say_bye():
    print('bye')

def say_hi(name):
    print(f'hi {name}')

say_hi('Monica')
print("end of my_func")
```

```
# main.py

from my_func import say_hello
```

```
print("module imported")
say_hello()
```

위 코드에서 `main.py` 프로그램을 실행했을 때 결과를 예상해보자.

만약 `my_func` 모듈 내의 코드가 먼저 실행되고, `main.py` 코드가 실행된다고 예상하였다면 정확하다.

```
# Result
"""
hi Monica
end of my_func
module imported
hello
"""
```

위와같이 실행된다. 즉, 모듈을 import하게 되면 해당 모듈 내 모든 함수를 메모리로 읽어옴과 동시에 작성된 코드를 모두 실행하게된다.

아마 이러한 코드 실행은 대부분의 경우에서 의도치 않은 결과를 초래하게 될것이다. (만약 이러한 실행 방식을 의도하였다고 하면.. 코드 디자인이 잘못되었을 경우가 높다)

그럼 다른 스크립트 (모듈) 내에 있는 함수를 import하여 쓰고싶지만 해당 파일에 작성된 코드는 실행시키고 싶지 않을때는 어떻게 해야할까? 이때 우리는 `if __name__ == "__main__":` 구문을 사용할 수 있다.

```
if __name__ == "__main__":
```

```
# my_func.py

def say_hello():
    print('hello')

def say_bye():
    print('bye')
```

```
def say_hi(name):
    print(f'hi {name}')
```

```
if __name__ == "__main__":
    say_hi('Monica')
    print("end of my_func")
```

```
# main.py

from my_func import say_hello

print("module imported")
say_hello()
```

위와같이 실행되는 코드를 `if __name__ == "__main__":` 구문 안으로 넣게되면 import 구문에 의하여 코드 실행을 방지할 수 있다. 즉, import 하고자 하는 함수만 메모리로 읽어들이며 그 외 실행 코드는 실행하지 않게 된다.

```
# Result
"""
module imported
hello
"""
```

위 코드 블록의 원리는 제어문을 통하여 코드의 실행을 제어하는 것이다. 만약 우리가 `python3 ./main.py` 와 같이 파이썬 파일을 실행하게 되면 Python의 global namespace 에는 자동으로 `__name__ = "__main__"` 이라는 값이 저장되게 된다. 만약 현재 파일이 아닌 그 외 파일에서 import 구문을 통하여 실행되게되면 해당 파일의 이름이 `__name__` 변수에 저장되게 된다.

```
print(__name__)
```

위와같이 직접 실행하여 확인해볼 수도 있다.

즉, 특정 파이썬 파일을 직접 실행하였을때만 해당 제어문을 통하여 코드가 실행되며, 그 외 경우에는 실행되지 않게 된다.

모듈을 다룰 때 주의사항 2

```
from my_func import *
```

간혹 코드를 작성할때 위 예제처럼 `*` 을 사용하는 경우가 있다. 해당 `*` 의 의미는 `my_func` 모듈 내에 모든 함수를 `import` 하겠다는 의미가 된다. 하지만 이러한 사용은 코드의 가독성을 해치며 상황에 따라 위험할 수 있기 때문에 사용을 권장하지 않는다.

아래 예시를 확인해보자

```
# my_module.py

def isdir(_path):
    print("This is custom isdir function and will always return: True")
    return True

def isfile(_path):
    print("This is custom isfile function and will always return: True")
    return True
```


```
# main.py

from os.path import isdir, isfile
from my_module import *

isdir("/home/user1")
isfile("/home/user1/testFile.txt")

# This is custom isdir function and will always returns True
# This is custom isfile function and will always returns True
```

위와같이 함수 이름이 겹치게 된다면 나중에 `import`한 구문이 그 전에 `import`된 함수를 `override` 하게 된다.

또한,  사용하여 import된 함수는 해당 함수가 어디에서 import 되었는지 특정하기 어렵기 때문에 디버깅 시 문제를 찾기가 매우 힘들어 진다.

따라서 위와같은 방식의 import는 사용하지 않는편이 좋다.