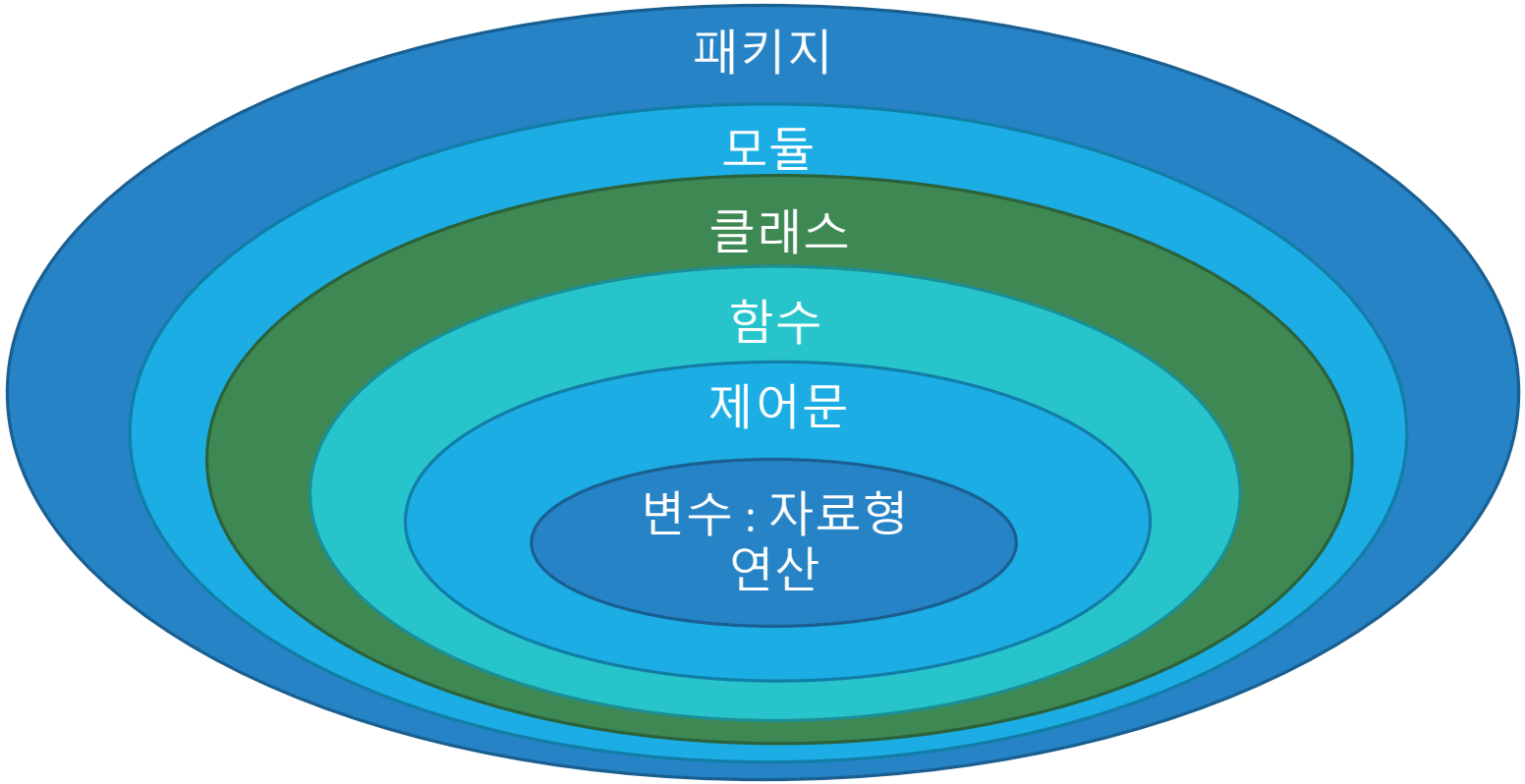
Python 4강

4강 – CLASS, MODULE, PACKEAGE, EXCEPTION HANDLING, (+GIT)


패키지
모듈
클래스
함수
제어문
변수 : 자료형
연산

클래스란

```
class Calculator :
        def __init__(self):
                self.result = 0
        def adder(self, num):
                self.result += num
                return self.result

cal1 = Calculator()
cal2 = Calculator()

print(cal1.adder(3))
print(cal1.adder(4))
print(cal2.adder(3))
print(cal2.adder(7))
```

Calculator 실행 결과 :

```
3
7
3
10
```

클래스란

클래스는 C 와 같은 절차지향형 언어와 다른 객체지향형 언어의 가장 큰 핵심 성질이라고 합니다. 클래스는 쉽게 말해, 변수와 함수를 같이 모아 놓는 묶음 정도로 생각하면 쉬울 것 같습니다.

자주 쓰이는 여러 기능들을 클래스에 모아 놓고 클래스의 피조물인 객체를 여러 개 생성해서, 변수의 선언, 함수의 생성까지 과정의 반복을 줄일 수 있습니다.

Calculator 클래스로 result 변수를 두개 만들 필요없이 객체를 두개 만들어 필요한 기능을 실행했습니다.

이야기 형식으로 클래스 기초 쌓기

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다."
```

```
>>> pey = Service()
```

```
>>> pey.secret
"영구는 배꼽이 두 개다."
```

*

```
>>>Service.secret = "영구는 배꼽이 없다."
```

Service 라는 클래스가 있는데, 이 Serice 클래스는 어떤 유용한 정보를 제공해 주는 한 ' 인터넷 서비스 제공 업체'라고 가정하자.

가입하는 방법은 다음과 같다. pey라는 아이디로 인터넷 서비스 업체인 Service 클래스를 이용할 수 있다.

이제, 서비스 업체가 제공하는 정보를 얻어내면, 다음과 같이 값이 나온다.

* 다음과 같이, 값을 바꿀 수도 있다.

클래스 함수

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다"     # 유용한 정보
...     def sum(self, a, b):                  # 더하기 서비스
...         result = a + b
...         print("%s + %s = %s입니다." % (a, b, result))
...
```

```
>>> pey = Service()
```

```
>>> pey.sum(1,1)
1 + 1 = 2입니다.
```

이 서비스는 정보 제공 서비스 이외에 더하기 서비스도 제공해 주기 위해 다음과 같이 클래스를 업그레이드 했다.

회원으로 가입되 있지 않다면 pey 라는 아이디로 가입을 한다. 다음에 더하기 서비스를 다음과 같이 이용할 수 있다.

self 살펴보기

```
... def sum(self, a, b):
...     result = a + b
...     print("%s + %s = %s입니다." % (a, b, result))
```

여기서 self는 신원확인 같은 일을 한다고 보면 된다.

클래스의 객체를 생성했을 때, 그 객체가 self로 자동으로 입력된다. 그래서, 앞의 예제에서 pey=Service(), pey.sum(3,4) 를 하여, 첫번째 인수를 무시하고 함수를 사용해도 문제가 없는 것이다.

클래서 함수는 모두 self를 첫 번째 인수로 가져야 하고, 이러한 성질은 파이썬의 특징이다. C++, Java에선 self를 할 필요가 없다.

객체 변수

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다"
...     def setname(self, name):
...         self.name = name
...     def sum(self, a, b):
...         result = a + b
...         print("%s님 %s + %s = %s입니다." % (self.nam
e, a, b, result))

>>> pey = Service()

>>> pey.setname("홍길동")

>>> pey.sum(1, 1)
홍길동님 1 + 1 = 2입니다.
```

다음과 같이 pey라는 아이디로 서비스에 가입을 하고, 이름에 '홍길동 ' 이라고 입력을 하면, 내 이름과 함께 더하기 서비스를 이용할 수 있습니다.

클래스에 setname 함수를 보시면, self.name 이라는 객체변수라는 게 있는데, 이 변수는 클래스 전체로 사용 범위가 늘어나는 함수입니다. setname에서 들어온 name은 sum에서 사용할 수 없지만, pey.name 에 name을 넣으면 사용이 가능합니다.

__init__이란 무엇일까?

```
>>> babo = Service()
>>> babo.sum(1, 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in sum
AttributeError: 'Service' object has no attribute 'nam
e'
```

서비스를 이용중, 유저들이 자신을 이름 입력하는 것을 계속 까먹어서 다음과 같은 에러가 발생합니다. 그래서, 서비스를 개선했습니다.

```
>>> class Service:
...     secret = "영구는 배꼽이 두 개다"
...     def __init__(self, name):
...         self.name = name
...     def sum(self, a, b):
...         result = a + b
...         print("%s님 %s + %s = %s입니다." % (self.nam
e, a, b, result))
```

```
>>> pey = Service("홍길동")
```

__init__ 이란 무엇일까?

__init__은 객체가 생성될 때 자동으로 실행되는 함수로 생성자라고 한다.

이제는 객체를 만들 때, 다음과 같이 name 값을 인수로 입력해 주어야 객체가 생성될 수 있따.

```
class 클래스이름[(상속 클래스명)]:
    <클래스 변수 1>
    <클래스 변수 2>
    ...
    def 클래스함수1(self[, 인수1, 인수2,,,]):
        <수행할 문장 1>
        <수행할 문장 2>
        ...
    def 클래스함수2(self[, 인수1, 인수2,,,]):
        <수행할 문장1>
        <수행할 문장2>
        ...
    ...
```

클래스 구조

클래스의 구조는 다음과 같다.

다른 클래스의 요소들을 불러올 수 있는 상속이란 개념이 있는데, class 이름 옆에 상속 클래스명을 지정하며 사용할 수 있다.

예제 – 사칙연산 클래스를 만들어보자.

```
class FourCal:
```

모듈이란?

모듈이란 함수나 변수 또는 클래스들을 모아 놓은 파일이다. 모듈은 다른 파이썬 프로그램에서 불러와 사용 할 수 있게끔 만들어진 파이썬 파일이라고도 할 수 있다.

우리는 파이썬으로 프로그래밍을 할 때 굉장히 많은 모듈을 사용한다. 예를 들어, 둘째주 과제에 사용했던, 무작위 함수를 생성하는 randint 함수를 기억하는가? 이때, 파일의 가장 위에 선언했던 것이 import random 이다. 파이썬 system path 에 지정되어있는 random 모듈을 불러와서 randint 함수를 사용한 것이다.

다른 사람들이 이미 만들어 놓은 모듈을 사용할 수도 있고 우리가 직접 만들어서 사용할 수도 있다.

*

Import random

from random import randint

random.randint(1,10) 이렇게도 쓸 수 있고, randint(1,10) 이런식으로 모듈이름을 생략할 수 있다.

If __name__ == “__main__”: 의 의미

```python
# mod1.py
def sum(a, b):
    return a+b

def safe_sum(a, b):
    if type(a) != type(b):
        print("더할수 있는 것이 아닙니다.")
        return
    else:
        result = sum(a, b)
    return result

print(safe_sum('a', 1))
print(safe_sum(1, 4))
print(sum(10, 10.4))
```

mod1.py 라는 파일이 있다. 두 함수가 선언 돼있고, 밑에 print 를 사용했다.

이때, 만약 다른 파일에서 mod1.py 를 import 하면 문제가 생기는데, import mod1을 하면, 다음과 같이 원하지 않는 값이 출력 될수도 있다.

```
>>> import mod1
더할 수 있는 것이 아닙니다.
None
5
20.4
```

If __name__ == “__main__”: 의 의미

```
if __name__ == "__main__":
    print(safe_sum('a', 1))
    print(safe_sum(1, 4))
    print(sum(10, 10.4))
```

위와 같은 문제에서, mod1.py 파일에서 프로그램을 실행시켰을 때, print 출력을 하면서, mod1의 sum과 safe_sum 을 다른 파일에서 사용하고 싶다면, 다음과 같은 if 문에 print 출력을 넣어주면 다른 파일에서 import 를 해도 출력함수가 실행이 안된다.

모듈의 디렉토리 위치

random 이나 turtle 같은 모듈들은 어느 디렉토리에 있던 불러오는게 가능하다. 하지만, 내가 작성한 모듈일 경우, 같은 디렉토리 (폴더) 안에 있지 않으면, 그 모듈을 찾아올 수 없다.

그렇다면, 두가지 의문점이 생긴다. random 이나 turtle 같은 모듈은 어떻게 아무 디렉토리에서나 불러올 수 있는 것이고, 다른 디렉토리에서 모듈을 불러오는 방법은 없을까?

1. 첫번째, 의문점에 대한 해답은 다음과 같은 과정을 통해 이해할 수 있다. 일단 cmd 창을 열고 python으로 들어가서 sys 모듈을 불러와서 sys.path 라는 코드를 실행시켜봐라 그럼 다음과 같은 결과가 나온다.

```
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python35.zip', 'c:\\Python35\\DLLs',
'c:\\Python35\\lib', 'c:\\Python35', 'c:\\Python35\\lib\\site-packages']
```

이것은 system에 설정 돼있는 경로 리스트이다. 이들 디렉토리에 있는 모듈들은 모두 어느 디렉토리에서 접근이 가능하다.

모듈의 디렉토리 위치

그렇다면, 두번째 문제에 대한 해결책은 이 sys.path에 내가 모듈을 불러오고 싶은 디렉토리를 추가하면 같은 디렉토리에 모듈이 없더라도 불러올 수 있다. 다음과 같이, 원하는 path를 sys.path에 추가해주면 된다.

```
>>> sys.path.append("C:/Python/Mymodules")
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python35.zip', 'c:\\P
ython35\\DLLs',
'c:\\Python35\\lib', 'c:\\Python35', 'c:\\Python35
\\lib\\site-packages',
'C:/Python/Mymodules']
>>>
```

다른 방법으로는 다음과 같이 set PYTHONPATH라는 함수를 통해 PATHONPATH 환경변수에 디렉토리를 추가해주는 방법도 있다.

C:\Users\home\set PYTHONPATH=C:\Python\Mymodules

패키지란?

패키지는 도트(.)를 이용하여 파이썬 모듈을 계층적(디렉터리 구조)으로 관리할 수 있게 해 준다. 예를 들어, 모듈명이 pygame.math인 경우, pygame 패키지에 있는 math 모듈이라는 것이다.

쉽게 말해, 앞에서 모듈을 관리할 때, path를 추가해줄 필요 없이, 폴더 계층을 짜주면, 도트(.)를 통해 쉽게 다른 폴더의 모듈을 관리할 수 있다.

* 패키지를 사용할 때, Pycharm 같은 IDE를 사용하면 보기가 편해진다 (for 원빈)

패키지 만들기

```
C:/Python/game/__init__.py
C:/Python/game/sound/__init__.py
C:/Python/game/sound/echo.py
C:/Python/game/graphic/__init__.py
C:/Python/game/graphic/render.py
```

각 디렉토리에 다음 파일들을 만들어서 저장하고 __init__.py 파일들은 만들어놓기만 하고 내용은 비워둔다.

```
# echo.py
def echo_test():
    print ("echo")
```

```
# render.py
def render_test():
    print ("render")
```

다음과 같이, 함수를 만들고, 우리가 만든 game 패키지를 참조 할 수 있도록, 도스창에서 set 명령을 이용하여 PYTHONPATH 환경변수에 C:/Python 디렉터리를 추가한다.

set PYTHONPATH=C:/Python

패키지 안의 함수 실행하기

이제, 도스창에서 다음과 같이 실행해보면, echo_test() 함수를 사용할 수 있다.

```
>>> import game.sound.echo
>>> game.sound.echo.echo_test()
echo
```

game 디렉터리 안에, sound 디렉터리 안에, echo 파일 안에 있는 echo_test()함수를 실행한 것이다.

__init__.py 의 용도

__init__.py 파일은 해당 디렉터리가 패키지의 일부임을 알려주는 역할을 한다. 만약 game, sound, graphics 등 패키지에 포함된 디렉터리에 __init__.py 파일이 없다면 패키지로 인식되지 않는다.

* python3.3 버전부터는 __init__.py 파일 없이도 패키지로 인식이 된다. 하지만 하위 버전 호환을 위해 __init__.py 파일을 생성하는 것이 안전한 방법이다.

```
>>> from game.sound import *
>>> echo.echo_test()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
NameError: name 'echo' is not defined
```

```
# C:/Python/game/sound/__init__.py
__all__ = ['echo']
```

* 이 all 을 뜻하는 걸 기억하는가? 코드에 아무 문제 없어 보이지만, 에러가 발생한다. 왜냐면 * 를 이용하여 import 할 때에는 다음과 같이 해당 디렉터리의 __init__.py 파일에 __all__ 이라는 변수를 설정하고 import 할 수 있는 모듈을 정의해 주어야 한다.

과제

- 1. 다음의 조건을 만족하는 Point라는 클래스를 작성하세요.
  - Point 클래스는 생성자를 통해 (x, y) 좌표를 입력받는다.
  - setx(x), sety(y) 메서드를 통해 x 좌표와 y 좌표를 따로 입력받을 수도 있다.
  - get() 메서드를 호출하면 튜플로 구성된 (x, y) 좌표를 반환한다.
  - move(dx, dy) 메서드는 현재 좌표를 dx, dy만큼 이동시킨다.

2. get_String 과 print_String 이라는 두 함수를 가진 클래스를 작성하세요.
  - get_String 은 유저로부터 문자열을 입력받습니다.
  - print_String은 입력받은 문자열을 대문자로 출력합니다.
  - 문자열을 대문자로 만드는 내장함수가 있습니다. 구글링으로 찾아보세요.

  - 클래스를 만든 후, 모든 함수들이 실행되는지 스크린 샷으로 찍어서 같이 보내주세요.

과제

3. pygame 라이브러리를 다운받고, 다음 링크에서 아무 게임이나 받아서 게임을 해보고 플레이한 모습을 스샷으로 찍으세요. 아무 게임이나 상관없습니다.

Pygame Projects