3BeforeDjango_PythonCheck

함수와 클래스

- 함수: 반복되는 작업을 비효율적으로 매번 수행하는 것이 아니라 기계가 대신할 수 있도록 만들어서 작업을 단순화 시키는 것처럼, 기계와 같다.

```
def 함수 이름 (매개변수1, 매개변수2, ..)
  <실행되는 코드1>
  return 반환 값
```

입력값, 출력값 모두 있을 수 있고, 없을 수 있다.
return도 있을 수 있고 없을 수 있다.

return이 없으면 print(함수이름(a,b))할 경우 None 값이 나옴.
return이 없어도 함수 안에 결과값이 내장되어 있으면 그 값은 보여짐.

- 클래스: 하나의 클래스에 다양한 함수를 지닐 수 있음.

```
>>> class Square:
...     def setValue(self, underLine, height):
...         self.underLine = underLine
...         self.height = height
...     def getArea(self):
...         return self.underLine * self.height
```

- class → 객채 생성
- 객채 = class의 함수를 가짐
- self = class 자기자신을 가리킴

```
>>> square1 = Square()
>>> square1.setValue(10, 5)
>>> square1.getArea()
50
```

- 객체를 통해 함수를 호출했기에, 함수의 self 매개 변수에 직접 객체 정보를 주지 않아도 된다.
- underLine과 height를 class 안에서 사용되는 self.underLine과 self.height에 할당

```
>>> square2 = Square()
>>> Square.setValue(square2, 3, 5)
>>> Square.getArea(square2)
15
```

- 하지만 클래스를 통해 함수를 호출할 때는 함수에 객체 정보를 직접 self변수에 전달해주면 된다.

Q. 굳이 왜 이렇게 만들지? 뭔가 다른 방법이 있을 거 같은데...

- __init__: 클래스에서 특별하게 사용되는 함수 중 하나.

* 메직메소드, 특별메소드, Dunder method(Double UNDERscore method)

* class로 객체가 생성된 후에 자동으로 호출되는 함수이다.

* 내부 멤버 변수의 초깃값을 설정할 수 있음.

```
class Account:
    def __init__(self, name):
        print("[init start]")
        self.name = name
        self.money = 10000
        print("[init end]")
        print("객체가 생성되었습니다.")
    def deposit(self, money):
        self.money += money
        print("%d원을 저축하여 총 잔액은 %d원 입니다." %(money, self.money))
    def withdraw(self, money):
        self.money -= money
        print("인출 후 잔액: %d" %self.money)
    def printBalance(self):
        print("잔고 금액입니다.")
        return self.money
    def printOwner(self):
        print("소유자: ", self.name)
        #return self.name
```

라이브러리 또는 모듈

```
import datetime
print("datetime을 import했습니다")
current = datetime.datetime.today().strftime("%Y-%m-%d")
print(current)
print("datetime 라이브러리 사용 성공!")
```

- 파이썬 자체의 내부라이브러리
- 외부에 공개된 외부라이브러리 ex. numpy(데이터분석, 수치계산)

  * 라이브러리 설치는 특정 사이트에서 하는 것이 아니라 명령 프롬프트에서 설치를 진행한다.

  * 명령프롬프트에서 다음과 같이 명령어를 실행. 이때 어떤 경로에 위치하든 상관없다. `C:\> pip install numpy`

실전예제

1-1)

```
print("당신에 대해 알려주세요.")
name = input("이름:")
age = input("나이:")
```

```
name = 'Serena'
age = 23
dream = '취업성공'
```

```
dream = input("꿈:")

print("당신의 이름은 %s이며, \n"
      "나이는 %s살. 그리고 %s 라는 꿈을 갖고 있군요. \n 멋집니다."
      %(name, age, dream))
```

```
print("저는", name+"입니다.", end='')
print('그리고 꿈은', dream+'입니다')
print("저는 \n다양한 꿈이 있지만 \n'"+dream+"'에 집중하는
      +dream, end='')
print("!!")
print("그럼 취업성공을 향해서 다시 집중해보겠습니다.")
```

- 실전예제 1-2부터는 시간이 오래 걸릴 거 같아서, 여기서는 잠시 Pass.