

PYTHON

함수 객체

함수

수학에서 함수가 식이었다면,

**파이썬에서 함수란
같은 기능을 하는 코드 블록을 묶어
한 곳에 모아 둔 것을 뜻합니다.**

파이썬에서 함수의 기본구조는

def 함수명(**매개변수**):

수행할문장1

수행할문장2

...

return 반환값

입니다.

```
def add (a, b):  
    result = a + b  
    return result  
라면
```

함수명은 **add** 이고
매개변수는 **a, b** 두 개이며
수행할 문장은 **a와 b를 더해**
변수 **result** 에 저장하는 것이고
반환값은 **result** 인 것입니다.

하지만 기본구조를 지키지 않아도 괜찮습니다!

def 함수명():
수행할문장
return 반환값

매개변수가 생략된 함수

def 함수명(매개변수):
수행할문장

반환값이 생략된 함수

def 함수명():
수행할문장

...

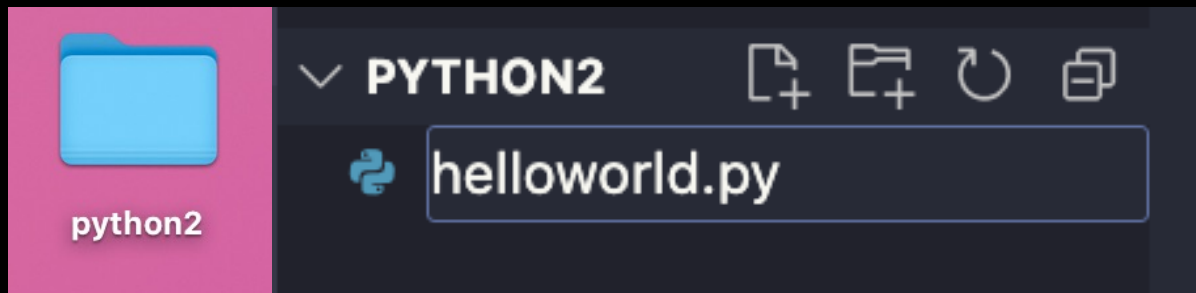
매개변수와 반환값이
생략된 함수

def 함수명():

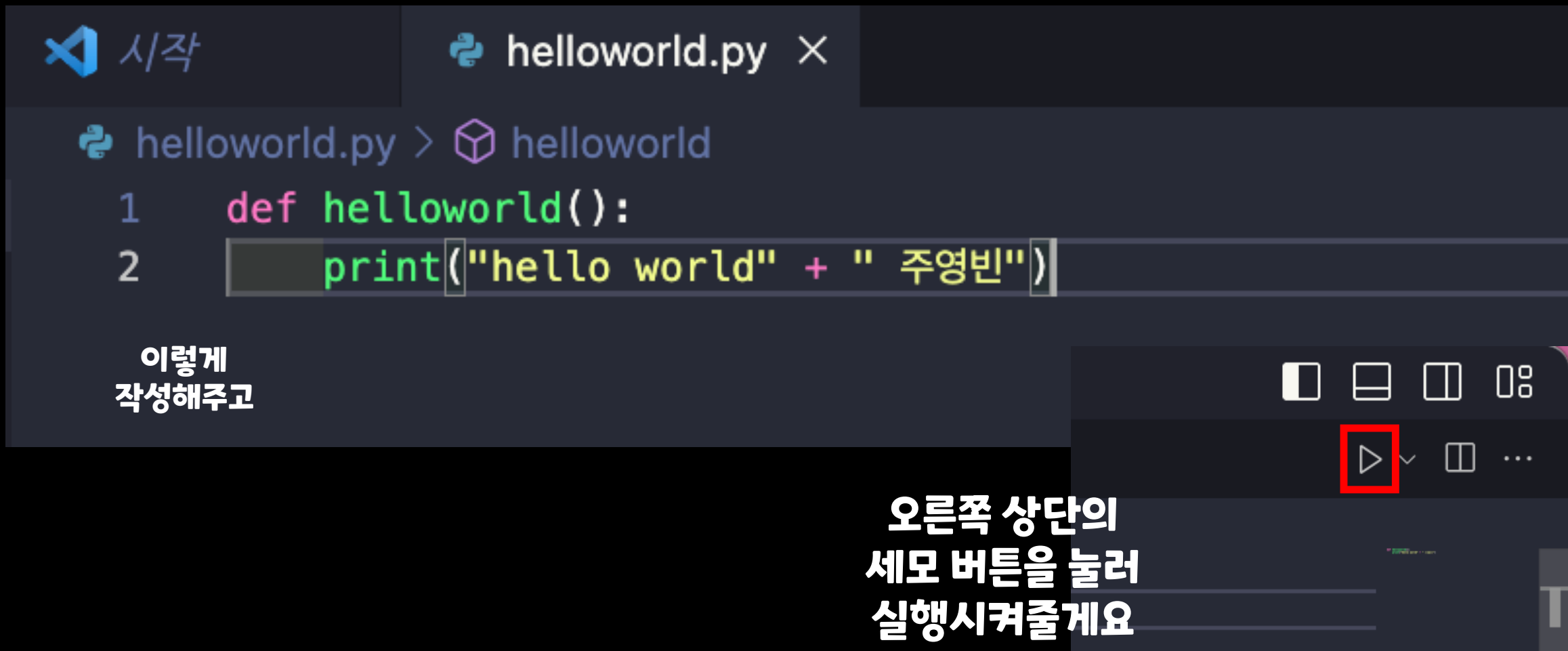
매개변수와 반환값
수행할문장
모두 생략된 함수

**정말 간단한 실습을 하면서
파이썬의 함수를 배워볼게요**

**혹시 어려운 부분이 있다면 바로바로
손 들고 물어보셔야 합니다**



바탕화면에 폴더를 만들고
사진과 같이 .py 파일을
만들어주세요




```
● ✕ youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/helloworld.py
○ youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 |
```

아무리 눌러도 안되는데요 ...

시작

helloworld.py ×

helloworld.py > helloworld

```
1 def helloworld():  
2     print("hello world" + " 주영빈")
```

안 되는 게 정상!
그 이유는 **함수를 호출** 해주는 부분이
없기 때문입니다.

우리의 코드는 함수를 만들기만 했을 뿐
이를 사용하는 부분이 없는 것과 같아요.

🔗 helloworld.py > ...

```
1  def helloworld():  
2      print("hello world" + " 주영빈")  
3  
4  helloworld()
```

이렇게 우리가 지정한 함수명() 을 통해
함수를 호출할 수 있어요
다시 실행해주면 이렇게 잘 뜨시죠 ?

```
● youngbeen@YoungBeenui-MacBookAir 🐻 ~/Desktop/python2 /Users/youngb  
hello world 주영빈  
○ youngbeen@YoungBeenui-MacBookAir 🐻 ~/Desktop/python2 []
```

helloworld.py > ...

```
1  def helloworld():  
2      print("hello world" + " 주영빈")  
3  
4  helloworld()
```

우리가 만들어준
helloworld 함수는

def 함수명():
수행할문장

...

매개변수와 반환값이
생략된 함수

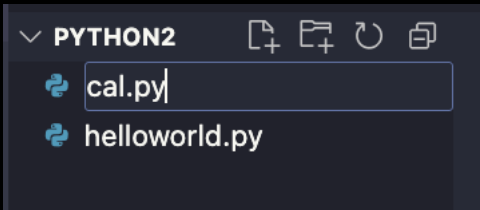
예요

하지만 이렇게 입력값에 관계없이 출력값이 동일한 함수는 많이 쓰이지 않아요.

우리는 이렇게 하는 동작이 고정되어있는 함수보다는, 우리의 **입력값에 따라 출력값이 달라지는 함수**가 필요해요.

이럴 때 사용되는 게 **매개변수**입니다.
값을 받으려면 함수 선언 시 () 안에 변수를 넣어주면 되는데,
이 변수를 **매개변수** 라고 합니다.

그러면 매개변수와 함께 반환값이 존재하는 함수를 만들어볼게요



cal.py를
만들고

```
cal.py > ...  
1  def cal(a, b):  
2      add = a + b  
3      return add  
4  
5  cal(1,2)
```

이렇게 작성해줄게요
코드 설명 뒤에
실행해볼게요!



cal.py >

함수명은 cal 매개변수가 a, b

```
1 def cal a, b:
```

```
2     add = a + b
```

```
3     return add
```

```
4
```

```
5 cal(1, 2)
```

함수 선언시에 입력받는
매개변수 a 와 b의 합을 저장하는 변수

변수 add를 반환,
즉 add가 반환값이 된다

a에는 1을, b에는 2를 저장하고
cal함수를 호출한다.

이제 실행해줄게요!


```
● ✖ youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
● youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/cal.py
○ youngbeen@YoungBeenui-MacBookAir 🐱 ~/Desktop/python2 |
```

**아까처럼 또 안되는데..
그 이유는 우리가 사용한
return에 있어요 !**



cal.py > ... 우리는 함수 선언시에 매개변수 a, b를
... 사용하여 입력을 받겠다고 했어요

```
1 def cal(a, b):
```

```
2     add = a + b
```

```
3     return add
```

```
4
```

```
5 cal(1, 2)
```

그래서 입력받은 값을 통한
합을 저장해줬죠

변수 add를 반환,
즉 add가 반환값이 되었어요.

우리가 함수에서 값을 반환하기 위해서는 **return 키워드**를 사용해요
return을 사용하면 값을 함수 바깥으로 반환해요.
이는 함수를 호출해준 바깥에
우리가 반환값으로 지정한 결과를 알려주기 위함이에요.

즉, 값을 바깥으로 반환하고 알려주기만 하지
이를 출력해주지는 않아요.

cal.py > ...

```
1 def cal(a, b):  
2     add = a + b  
3     return add
```

```
4  
5 cal(1,2)
```

즉 문제는 함수를 호출했을 뿐
반환값을 출력해주지 않았기 때문이군요!

cal.py > ...

```
1 def cal(a, b):  
2     add = a + b  
3     return add  
4  
5 print(cal(1,2))
```

이렇게 수정해주고
실행해주세요

● youngbeen@YoungBeenui-MacBookAir ~/Deskto
3

그럼 잘 뜨죠??

이 return 키워드는 값을 반환하는 기능 뿐만 아니라
함수를 중간에 바로 빠져나오는 기능도 수행해요.

반복문 강의를 수강하였다면 나오는 break를
기억하시나요?

break는 제어흐름을 중단하고 빠져나오는데,
return은 함수 실행 중간에 빠져나오고
아래 실행문은 수행하지 않아요.
주로 if문과 같이 사용합니다.

예시와 같이 학습해볼게요

 drink.py > ...

```
1  def drink(age):  
2      if (age < 20):  
3          return  
4      print(age, "살로 음주가 가능합니다.")  
5
```

drink.py를 만들고 코드를 작성해주세요.
나이에 따라
음주가 가능한지 판별하는 함수예요.
같이 작성해주세요.

```
drink(20)
print()
drink(23)
```

이렇게 작성해준다면

20 살로 음주가 가능합니다.

23 살로 음주가 가능합니다.

이렇게 실행될거예요
print()는 실행창 사이에 한 줄을 띄워주기 위해
사용했습니다

nk.py > ...

```
def drink(age):  
    if (age < 20):  
        return  
    print(age, "살로 음주가 가능합니다.")
```

Print문이 실행된 것을
알 수 있어요.

```
drink(18)
print()
drink(19)
```

이렇게 작성해준다면

```
youngbeen@YoungBeenui-MacBookAir ~/Desktop/python2 $ python2 /Us
youngbeen@YoungBeenui-MacBookAir ~/Desktop/python2 $
```

이렇게 아무것도 뜨지 않을 거예요.

즉 if 문에 걸리기 때문에
return의 기능으로

함수를 중간에 빠져나오게 된 것이고

```
nk.py > ...
def drink(age):
    if (age < 20):
        return
    print(age, "살로 음주가 가능합니다.")
```

Print문이 실행되지
않은 것을 알 수 있어요.

또한 **return**은 반환값을 여러개 반환 해줄 수도 있어요.
이러한 반환값은 튜플 자료형 으로,
언패킹 을 통해 각 변수에 할당됩니다.

이것도 간단히 실습해줄게요.
아까 실습한 **cal.py** 파일을 사용해줄게요.

cal.py > ...

```
1  def cal(a, b):  
2      return a + b, a - b, a * b, a / b  
3  
4  add, sub, mul, div = cal(6, 3)  
5  print(f'합은 {add} 차는 {sub} 곱은 {mul} 몫은 {div}')
```

이렇게 작성해줄게요

cal.py > ...

```
1  def cal(a, b):  
2      return a + b, a - b, a * b, a / b  
3  
4  add, sub, mul, div = cal(6, 3)  
5  print(f'합은 {add} 차는 {sub} 곱은 {mul} 몫은 {div}')
```

이렇게 잘 뜨나요?

● youngbeen@YoungBeenui-MacBookAir 🐼 ~/Desktop/python2
합은 9 차는 3 곱은 18 몫은 2.0

cal.py > ...

```
1  def cal(a, b):  
2      return a + b, a - b, a * b, a / b  
3  
4  add, sub, mul, div = cal(6, 3)  
5  print(f'합은 {add} 차는 {sub} 곱은 {mul} 몫은 {div}')
```

이렇게 반환값의 개수와
함수를 호출하며 반환값이 저장될 변수의 개수를 맞춰준다면
return에서 튜플 자료형으로 저장된 반환값들이 **언패킹** 됩니다

```
return a + b, a - b, a * b, a / b
```

→ $(a + b,$ $a - b,$ $a * b,$ $a / b)$

즉 이렇게 튜플 자료형으로 묶여 전달되는 반환값들이

add, sub, mul, div = cal(6, 3)

```
● youngbeen@YoungBeenui-MacBookAir 🐼 ~/Desktop/python2  
합은 9 차는 3 곱은 18 몫은 2.0
```

언패킹되어 각각 변수에 반환값들이 저장되는 것을 알 수 있습니다.

cal.py > ...

```
1  def cal(a, b):  
2      return a + b, a - b, a * b, a / b  
3  
4  add, sub = cal(6, 3)  
5  print(f'합은 {add} 차는 {sub}')
```

이렇게 반환값의 개수와 변수의 개수가 일치하지 않으면 오류가 뜹니다

Traceback (most recent call last):

File "/Users/youngbeen/Desktop/python2/cal.py", line 4, in <module>

add, sub = cal(6, 3)

ValueError: too many values to unpack (expected 2)

함수를 더 다채롭게 작성하기 위한 것들에 대해
더 알아보게요.

우선 우리가 지금까지 사용한 것처럼
함수에 인수를 순서대로 넣기 위해서
리스트나 튜플 자료형을 사용해줄 수 있어요.

이렇게 **인수를 순서대로 넣는 방식**을
위치 인수(positional argument)라고 해요.

intro.py

intro.py > ...

```
1 def intro(name, schoolNum, grade):  
2     print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```

```
4 intro('주영빈', 202114041, 3)
```

```
이름은 주영빈  
학번은 202114041  
학년은 3
```

이렇게 잘 되죠?

**intro.py를 만들어주고 이렇게 작성해주세요.
실행해줄게요!**

```
1  def intro(name, schoolNum, grade):  
2      print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```

```
3  
4  intro('주영빈', 202114041, 3)  
5  |
```

이런 함수가 있다고 할 때
이렇게 직접 인수를 넣어도 되지만,


```
def intro(name, schoolNum, grade):  
    print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```



```
youngbeen = ['주영빈', 202114041, 3]  
intro(*youngbeen)
```

이렇게 리스트를 사용할 수도 있어요.
실행해주면
아까와 동일하게 출력되는 것을 알 수 있어요.

 intro.py > ...

```
1  def intro(name, schoolNum, grade):  
2      print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```

```
3  
4  intro(*['주영빈', 202114041, 3])
```

**혹은 이렇게 리스트를 직접 넣어줘도 돼요.
이것도 동일하게 출력되죠?**

```
def intro(name, schoolNum, grade):  
    print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```



```
youngbeen = ['주영빈', 202114041, 3]  
intro(*youngbeen)
```

코드 설명을 해볼게요.

이름에 주영빈이 / 학번에 202114041이 / 학년에 3이 잘 들어간 것을 볼 수 있어요.
즉 **리스트 안의 요소들이 함수의 매개변수 값에 저장된 것**을 알 수 있어요.

이렇게 함수에 인수를 순서대로 넣을 때는 **리스트나 튜플**을 사용할 수 있어요.
그런데 이러한 리스트나 튜플을 사용하기 위해서는
리스트 또는 튜플 앞에 *(애스터리스크)를 붙여서 함수에 넣어주어야 해요.
이렇게 사용하는 것을 리스트(튜플) 언패킹이라고 해요

이것도 마찬가지로 매개변수의 개수와 리스트 혹은 튜플 요소의 개수가 다르면 오류가 나겠죠 ... ?

**그런데 지금까지는 함수 인수의 개수를 모두
함수를 만들어줄 때 제한해주었어요.**

**하지만 우리는 이렇게 인수의 개수가 제한된 것이 아니라
인수의 개수가 정해지지 않은 함수도 만들어보고 싶어요.**

**이렇게 함수의 인수의 개수가 정해지지 않은 것을
가변 인수(variable argument) 라고 해요.**

**이렇게 가변 인수를 사용하면 같은 함수에 인수를 1개를 넣을
수도, 5개를 넣을 수도, 넣지 않을 수도 있어요.**

가변 인수 함수는

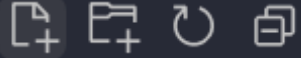
def 함수명(*매개변수):
코드부분

으로 이루어져 있습니다.

관례적으로 **arguments**를 줄여서 **args**로 사용해요.
특히 이 **args**는 튜플이라서 **for**로 반복할 수 있어요.

그럼 이제 숫자를 여러개 받아 출력하는 함수를 만들어볼게요

PYTHON2



print_numbers.py

cal.py

print_numbers.py를 만들고

```
1 def print_numbers(*args):  
2     for arg in args:  
3         print(arg)
```

이렇게 작성해주세요

 print_numbers.py > ...

```
1  def print_numbers(*args):  
2      for arg in args:  
3          print(arg)  
4  
5  print_numbers(1)  
6  print()  
7  print_numbers(10, 20, 30, 40, 50)
```

이렇게 작성해주고 실행하면

print_numbers.py > ...

```
1 def print_numbers(*args):
```

```
2     for arg in args:
```

```
3         print(arg)
```

```
4
```

```
5 print_numbers(1)
```

```
6 print()
```

```
7 print_numbers(10, 20, 30, 40, 50)
```

1

10

20

30

40

50

이렇게 출력돼요
이렇게 직접 인수를 넣어도 되고

print_numbers.py > ...

```
1  def print_numbers(*args):  
2      for arg in args:  
3          print(arg)  
4  
5  print_numbers(*["주"])  
6  print()  
7  
8  youngbeen = ["주", "영", "빈"]  
9  print_numbers(*youngbeen)
```

이제는 이렇게 수정하고 실행해줄게요

print_numbers.py > ...

```
1 def print_numbers(*args):  
2     for arg in args:  
3         print(arg)  
4  
5 print_numbers(*["주"])  
6 print()  
7  
8 youngbeen = ["주", "영", "빈"]  
9 print_numbers(*youngbeen)
```

주
주영빈

그럼 이렇게 실행되는 것을 알 수 있어요.
이렇게 아까 배운 리스트 (튜플) 언패킹을 사용할 수도 있어요.

그럼 리스트, 튜플 언패킹에 대해서도 배웠으니
딕셔너리 언패킹에 대해서도 배워볼게요.

그런데 딕셔너리 언패킹 전에 간단히 짚고
넘어가야 하는 개념이 바로
키워드 인수예요

**이건 어렵지 않으니 간단히 보고만
넘어갈게요**

```
1  def intro(name, schoolNum, grade):  
2      print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```

```
3  
4  intro('주영빈', 202114041, 3)  
5  |
```

아까 학습한 intro 함수예요.

**여기서 각 인수의 순서에 맞게 데이터를 넣어주었죠?
즉, name, schoolNum, grade 순서로 매개변수가 나열되어있으니
우리도 '주영빈', 202114041, 3 으로 해주었어요.**

하지만 키워드 인수를 활용하면
매개변수의 순서에 상관없이 인수를 매개변수에 저장할 수 있어요.

키워드 인수는 아래와 같이

```
1 def intro(name, schoolNum, grade):  
2     print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```



```
4 intro('주영빈', 202114041, 3)  
5 intro(schoolNum=123456789, name='김멋사', grade=1)
```

매개변수 = 인수 (저장할 값)
으로 사용할 수 있어요

그럼 이제 딕셔너리 언패킹에 대해 알아보게요.
딕셔너리 언패킹은 *을 두 개, 즉 ** 을 사용해요.

딕셔너리에 '키워드': 값 형식으로 인수를 저장해야하고,
딕셔너리의 키워드(키)는 반드시 문자열 형태 여야 합니다.

그럼 한 번 실습을 해볼게요.
아까 사용한 intro.py를 열어주세요.

```
1 def intro(name, schoolNum, grade):  
2     print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```

```
3  
4 youngbeen = {'name': '주영빈', 'schoolNum': 202114041, 'grade': 3}  
5 intro(**youngbeen)
```

**이렇게 작성해주고 실행해주면
아래와 같이 실행되죠?**

```
이름은 주영빈  
학번은 202114041  
학년은 3
```

```
1 def intro(name, schoolNum, grade):  
2     print(f'이름은 {name}\n학번은 {schoolNum}\n학년은 {grade}')
```

```
3  
4 intro(**{'name': '주영빈', 'schoolNum': 202114041, 'grade': 3})
```

**아까 리스트 언패킹에서와 동일하게
이렇게 작성해주고 실행해준다면
아까와 같이 같은 출력이 뜰 거예요**

```
이름은 주영빈  
학번은 202114041  
학년은 3
```


람다 표현식

람다 표현식은 함수와 비슷하지만,
함수보다 더 간단하게 사용할 수 있어요.

람다 표현식은 식 형태로 되어 있다고해서
람다 표현식(lambda expression) 이라고 하는데
이는 익명 함수로
다른 함수의 인수를 넣을 때 주로 사용해요.

우리가 오늘 배운 것처럼
입력값을 받아 5를 곱하여 반환하는 함수를 만든다면

```
def mul_five(num):  
    return num * 5
```

이렇게 만들지만
람다 표현식을 사용하면

```
lambda x: x * 5
```

이렇게
작성할 수 있어요.

```
def mul_five(num):  
    return num * 5
```

하지만 이 함수를 호출하고 싶다면
mul_five() 라고 호출하면 되텐데

```
lambda x: x * 5
```

람다 표현식은 특별한 함수명이 없는 것을 알 수 있어요.

이 때문에 람다 표현식을 이름이 없는 함수,
즉 익명 함수 (anonymous function) 라고 불러요.

그래서 람다 표현식을 호출해주기
위해서는

```
mul = lambda x: x * 10  
mul(5)
```

처럼 람다 표현식을 변수에 할당해주면 돼요.
여기서 x에 해당하는 값이 5이에요.

lambda x: x * 5는 매개변수 x 하나를 받고,
x에 5를 곱해 반환한다는 뜻입니다.
(그럼 위 값을 출력하고 싶다면 print()를
사용해야겠죠?)

혹은 변수에 할당하지 않고도 사용할 수도 있어요.

(lambda 매개변수들: 식) (인수들)
과 같은 구조로
아까 본 식을 변환하면

(lambda x: x * 5)(5)

처럼 사용할 수 있어요.

이렇게 함수를 간단히 표현할 수 있다는 점에서 장점이 있지만,
람다 표현식 안에 새로운 변수를 생성할 수 없다는 단점이 있어요.

```
(lambda x: y = 5: x * y)(5)
```

즉 이렇게 $y = 5$ 처럼 새 변수를 생성할 수 없습니다.

```
    y = 5  
(lambda x: x * y)(5)
```

하지만 이처럼
람다 표현식 외부에 있는 변수는 사용할 수 있어요.

전역변수
지역변수

함수에 있어 중요한 개념인
전역 변수와 지역변수를 짚고 넘어가볼게요.

함수를 포함한 파이썬 스크립트 전체에서
접근할 수 있는 변수를 전역 변수(global variable).

그리고 전역 변수에 접근할 수 있는 범위를
전역 범위(global scope)라고 해요.

아래와 같은 코드를 VSCode에
작성해주세요.

```
x = 10
def foo():
    print(x)

foo()
print(x)
```



```
globaltest.py > ...
1  x = 10
2  def foo():
3      print(x)
4  foo()
5  print(x)
6
```

globaltest.py

실행해주면
터미널에 출력되는 값이
동일할거예요

```
x = 10
```

```
def foo():  
    print(x)
```

```
foo()  
print(x)
```

**foo() 함수를 잘 보면
함수 외부에 있는 변수 x의 값을
출력해주는 코드임을 알 수 있어요.**

**우리가 foo() 함수 안에 따로
변수 x에 값을 할당해주지 않았는데도
외부의 변수 값을 출력하였어요.**

```
x = 10
```

```
def foo():  
    print(x)
```

```
foo()  
print(x)
```

즉 foo() 함수의 print문과
하위 print문은 동일한
전역변수 x를 출력함을 알 수 있어요.

전역 변수



```
x = 10
```

```
def foo():
```

```
    print(x)
```

```
foo()
```

```
print(x)
```

전역 범위

**따라서 전역 변수는 x이고
전역변수에 접근할 수 있는 범위인 전역범위는
그림과 같겠죠?**

```
def foo():  
    x = 10  
    print(x)
```

```
foo()  
print(x)
```

```
globaltest.py > ...  
1  def foo():  
2      x = 10  
3      print(x)  
4  
5  foo()  
6  print(x)  
7
```

**globaltest.py를 이렇게
수정해주고 실행하면**

```
youngbeen@YoungBeenui-MacBookAir ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/globaltest.py  
10  
Traceback (most recent call last):  
  File "/Users/youngbeen/Desktop/python2/globaltest.py", line 6, in <module>  
    print(x)  
NameError: name 'x' is not defined
```

이렇게 오류가 날거예요.

```
youngbeen@YoungBeenui-MacBookAir ~/Desktop/python2 /Users/youngbeen/.pyenv/versions/3.10.5/bin/python /Users/youngbeen/Desktop/python2/globaltest.py
10
Traceback (most recent call last):
  File "/Users/youngbeen/Desktop/python2/globaltest.py", line 6, in <module>
    print(x)
NameError: name 'x' is not defined
```

**에러를 잘 볼게요.
터미널을 보면
foo()는 잘 실행돼서 10이 잘 출력되었는데
print(x)에서
name 'x' is not defined
즉 x가 정의되지 않았다고 해요**

```
def foo():  
    x = 10  
    print(x)
```

```
foo()  
print(x)
```

코드를 잘 보면
아까와는 다르게
이번에는 변수 x가 함수 안에서 만들어진
것을 알 수 있어요.

따라서
변수 x가 포함된 foo() 함수는
x를 출력할 수 있지만
print()문은 출력할 수 없어요.

함수 내에서 선언된 변수로
일반적으로 함수 내에서만 사용할 수 있는 변수를
지역 변수(local variable)

지역 변수를 접근할 수 있는 범위를
지역 범위(local scope) 라고 해요.

지역 변수는 변수를 만든 함수 내부에서만 접근할 수 있고,
함수 외부에서는 접근할 수 없어요.

전역 변수

→ x = 10

```
def foo():  
    print(x)
```

```
foo()  
print(x)
```

전역 범위

지역 변수

→ x = 10

```
def foo():  
    print(x)
```

지역 범위

```
foo()  
print(x)
```

**이렇게 비교해보면
이해하기
쉬울거예요!**

객체

파이썬이 객체지향 언어라는 건 알고 계시죠 ?? 혹시 모르시는 분들을 위해 잠시 설명하고 넘어갈게요



절차지향 프로그램



객체지향 프로그램

절차지향 프로그래밍
위에서 아래로 절차에 따라
순차적으로 진행되는 프로그램

객체지향 프로그래밍
객체를 기능별로 묶어
객체들의 상호작용으로
프로그래밍이 진행되는 것

즉,
여러개의 독립적인 단위인 객체들간의
상호작용으로 이루어지는 것

클래스 (Class) 란

객체지향 프로그래밍에서
특정 개체를 생성하기 위해 변수와 메소드를 정의하는
일종의 틀과 같은 개념

객체 (Object) 란

클래스라는 틀에 의해 만들어진 것

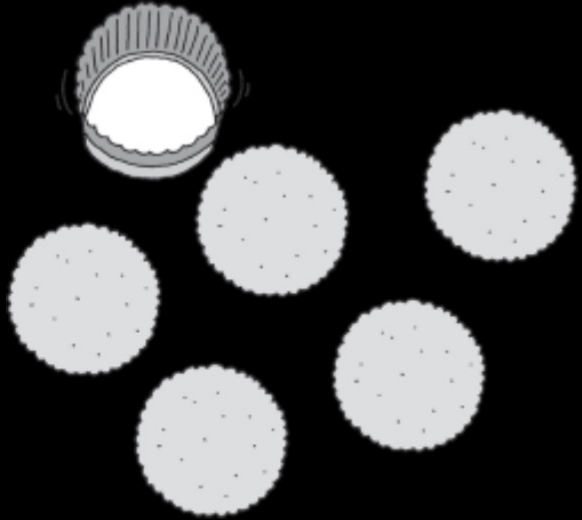
인스턴스 (Instance) 란

클래스로부터 만들어진 객체

- 특정 객체가 어떤 클래스의 객체인지
관계를 중점으로 표현할 때 사용합니다.

이러한 객체는

데이터와 같은 속성(attribute)
메서드(method) 로 이루어져 있어요.



class 쿠키 틀



라즈베리쿠키라는 새로운 객체 생성
= 쿠키 틀 클래스의 인스턴스



초코칩쿠키라는 새로운 객체 생성
= 쿠키 틀 클래스의 인스턴스

쿠키 틀 -> 클래스 (Class)
쿠키 틀에 의해 만들어진 과자 -> 객체 (Object)

클래스의 구조는

class 클래스명:
변수

...

메서드1
메서드2

로 이루어져 있습니다.

여기서 메서드란
클래스 내에 존재하는 함수를
메서드라고 해요.

객체는 매우 중요하니 실습을 통해 학습해볼게요.
classtest.py를 만들고

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade

    def print_name(self):
        print(f'이름은 {self.name} 입니다')

    def print_schoolnum(self):
        print(f'학번은 {self.schoolnum} 입니다')

    def print_grade(self):
        print(f'학년은 {self.grade} 입니다')
```

```
youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)
```

```
youngbeen.print_name()
youngbeen.print_schoolnum()
youngbeen.print_grade()
```

```
likelion.print_name()
likelion.print_schoolnum()
likelion.print_grade()
```

꼭 이어서 적어준 후 실행해주세요.


```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade

    def print_name(self):
        print(f'이름은 {self.name} 입니다')

    def print_schoolnum(self):
        print(f'학번은 {self.schoolnum} 입니다')

    def print_grade(self):
        print(f'학년은 {self.grade} 입니다')
```

**Likelion 클래스 에
name, schoolNum, grade
와 같은 속성 이 존재합니다.**

```
class Likelion:
```

```
def __init__(self, name, schoolnum, grade):  
    self.name = name  
    self.schoolnum = schoolnum  
    self.grade = grade
```

```
def print_name(self):  
    print(f'이름은 {self.name} 입니다')
```

```
def print_schoolnum(self):  
    print(f'학번은 {self.schoolnum} 입니다')
```

```
def print_grade(self):  
    print(f'학년은 {self.grade} 입니다')
```

속성을 만들때는
__init__() 메서드를 사용하여,
self.속성 = 값 의 형태로 사용해요.

__init__ 메서드는
인스턴스를 만들 때 호출되는 특별한
메서드입니다.
즉, 인스턴스(객체)를 초기화하는
기능을 해요.

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade

    def print_name(self):
        print(f'이름은 {self.name} 입니다')

    def print_schoolnum(self):
        print(f'학번은 {self.schoolnum} 입니다')

    def print_grade(self):
        print(f'학년은 {self.grade} 입니다')
```

여기서 계속 사용되는 **self** 는
인스턴스(객체) 자신 을 의미해요.
이 코드에서는 Likelion() 이 되겠죠?

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade

    def print_name(self):
        print(f'이름은 {self.name} 입니다')

    def print_schoolnum(self):
        print(f'학번은 {self.schoolnum} 입니다')

    def print_grade(self):
        print(f'학년은 {self.grade} 입니다')
```

**이 세 함수가
Likelion 클래스의 메서드가 되겠네요.**

**우리는
print_name :: 이름을 출력
Print_schoolNum :: 학번을 출력
print_grade :: 학년을 출력**

총 3개의 메서드를 정의했어요.

```
youngbeen = Likelion('주영빈', 202114041, 3)  
likelion = Likelion('likelion', 1234, 1)
```

```
youngbeen.print_name()  
youngbeen.print_schoolnum()  
youngbeen.print_grade()
```

```
likelion.print_name()  
likelion.print_schoolnum()  
likelion.print_grade()
```

**이 부분이 바로
Likelion 클래스로 찍어낸
객체 (인스턴스) 입니다.**

**youngbeen 이라는 객체 이자
Likelion 클래스의 인스턴스 인 것입니다.**

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade
```

```
youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)
```

```
youngbeen.print_name()
youngbeen.print_schoolnum()
youngbeen.print_grade()
```

```
likelion.print_name()
likelion.print_schoolnum()
likelion.print_grade()
```

그런데 아까
__init__ 메서드를 설명하며
인스턴스(객체)를
초기화하는 기능을 한다고 했었죠?

**객체 전반적 설명은 끝났으니
전체적인 코드를 보며
이해해볼게요**

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade

    def print_name(self):
        print(f'이름은 {self.name} 입니다')

    def print_schoolnum(self):
        print(f'학번은 {self.schoolnum} 입니다')

    def print_grade(self):
        print(f'학년은 {self.grade} 입니다')

youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)

youngbeen.print_name()
youngbeen.print_schoolnum()
youngbeen.print_grade()

likelion.print_name()
likelion.print_schoolnum()
likelion.print_grade()
```

**객체에 초기값을 설정하기 위해
생성자 를 구현하였어요.**

**생성자(Constructor)란
객체가 생성될 때
자동으로 호출되는 메소드 를 의미합니다.**

**__init__ 메서드가 생성자가 되겠죠.
객체를 만들며 동시에 실행되며
객체를 만드는 초기화를 해주는
메서드입니다.**


```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade
```

```
def print_name(self):
    print(f'이름은 {self.name} 입니다')

def print_schoolnum(self):
    print(f'학번은 {self.schoolnum} 입니다')

def print_grade(self):
    print(f'학년은 {self.grade} 입니다')
```

```
youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)
```

```
youngbeen.print_name()
youngbeen.print_schoolnum()
youngbeen.print_grade()
```

```
likelion.print_name()
likelion.print_schoolnum()
likelion.print_grade()
```


첫 번째 매개변수인 **self** 는
메소드를 호출한 객체가 자동으로 전달 됩니다.
즉 self 매개변수에 해당 인스턴스를 받아오기
때문입니다.

사용할 때는 **self**를 제외한 값을 넘겨줍니다.

관례적으로 Python 메서드의
첫 번째 매개변수 이름은 self를 사용합니다.
이는 객체를 호출할 때 호출한 객체 자신이
전달되기 때문입니다.

```
youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)
```

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade
```



“주영빈” “likelion” 문자열을 복사해 **name** 매개변수에 저장 하고
202114041 1234 를 복사해 **schoolNum** 매개변수에 저장 하고
1 3 을 복사해 **grade** 매개변수에 저장 합니다

```
youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)
```

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade
```

name 매개변수의 값을 각각

youngbeen.name / likelion.name 인스턴스 변수에 저장하고

schoolNum 매개변수의 값을 각각

youngbeen.schoolNum / likelion.schoolNum 인스턴스 변수에 저장하고

grade 매개변수의 값을 각각

youngbeen.grade / likelion.grade 인스턴스 변수에 저장해요.

```
class Likelion:
    def __init__(self, name, schoolnum, grade):
        self.name = name
        self.schoolnum = schoolnum
        self.grade = grade

    def print_name(self):
        print(f'이름은 {self.name} 입니다')

    def print_schoolnum(self):
        print(f'학번은 {self.schoolnum} 입니다')

    def print_grade(self):
        print(f'학년은 {self.grade} 입니다')

youngbeen = Likelion('주영빈', 202114041, 3)
likelion = Likelion('likelion', 1234, 1)
```


```
youngbeen.print_name()
youngbeen.print_schoolnum()
youngbeen.print_grade()
```

```
likelion.print_name()
likelion.print_schoolnum()
likelion.print_grade()
```

**클래스 내부의
인스턴스 변수나 메소드를 사용하려면**

**객체.인스턴스변수
객체.메소드이름**

으로 사용합니다.

강의 많이
어려우셨죠 ... 

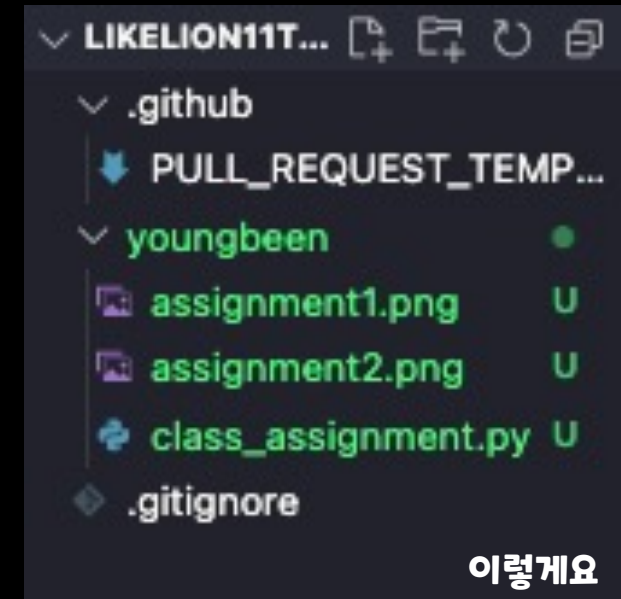
근데 과제 있어요 

과제 제출은
5월 18일 목요일 오후 11시 59분까지

과제 파일명은
class_assignment.py로 해주세요.

평소 제출하던 방식과 동일하게
본인영문이름폴더 속에
class_assignment.py와
과제 수행 사진 두 장 이상을 함께 넣어
제출해주세요.

사진 파일 명은 assignment숫자 로 해주세요
Ex) assignment1.png



과제 상세 가이드라인입니다 !

**클래스 명
Student 로
생성해주세요**

**속성은
name, schoolNum, semester, subject 로
생성해주세요**

클래스 메서드 상세 가이드라인입니다 !

1. 생성자를 구현해야 합니다.
(메서드 사용해주세요)
2. 메서드명 :: **print_name**
이름을 포함하여 출력합니다.
포매팅을 사용하여 출력합니다.

예시 :: 이름은 주영빈입니다.

3. 메서드명 :: **print_schoolNum**
학번을 포함한 문자열을 반환합니다.
문자열은 포매팅을 사용하여
반환합니다.

schoolNum을 슬라이싱한 결과를
통해 반환값을 달리해야 합니다.

xxxxxx1xxx :: 인문융합 xxxxxx3xxx :: 미디어융합
xxxxxx2xxx :: 사회융합 xxxxxx4xxx :: it융합
그 외의 숫자 “오류입니다” 를 반환해야 한다

예시 ::
학번은 202114041로 아이티융합자율학부 소속입니다.

클래스 메서드 상세 가이드라인입니다 !

4. 메서드명 :: print_semester
학기과 이름을 포함한 문자열을 출력합니다.
포매팅을 사용하여 출력합니다.
입력받은 semester에 따라 출력을
달리해야합니다.

~ 3학기 :: 전공선택 전
4학기 ~ :: 전공선택 후
그 외의 숫자 “오류입니다” 를 반환해야한다

예시 :: 주영빈은(는) 5학기차로 전공선택을 마쳤습니다.

5. 메서드명 :: print_subject
과목 리스트를 포함한 문자열을
반환합니다.

예시 ::
['백엔드프레임워크', '프론트엔드개발', '자료구조']를 수강합니다.

함수 상세 가이드라인입니다!

함수명 :: **subject_info**

딕셔너리 언패킹을 사용합니다.

첫 줄에 '자세한 수강목록입니다.'를 출력하세요.

포매팅을 사용하여 딕셔너리의 key, value를 출력합니다.

예시)

자세한 수강목록입니다.

과목명: 백엔드프레임워크 / 과목명의 길이: 8

과목명: 프론트엔드개발 / 과목명의 길이: 7

과목명: 자료구조 / 과목명의 길이: 4

터미널 입력 가이드라인입니다 !

1. **반복문(while)** 을 사용합니다.
2. 제일 먼저 객체 명을 저장할 **Class_name** 변수 를 받아주세요.
3. **Class_name** 변수를 입력받았다면
차례로 **name, schoolNum, semester, subject** 를 입력받아주세요.
4. 과목들을 담을 빈 **리스트** 와 **딕셔너리** 가 필요합니다.
5. **반복문(for)** 을 사용하여 최소 3개 이상의 과목을 입력 받아주세요.
6. 리스트에 입력받은 과목을 추가해주세요.
7. 딕셔너리에는 입력받은 과목을 key
입력받은 과목의 문자열 길이를 value로 해주세요.

터미널 출력 가이드라인입니다!

`Class_name` 변수를 입력 받을 때 “객체 명을 입력하시오. (단, 영문으로): “ 라고 뜨도록 해주세요.
단, 종료를 입력받았다면 반복문을 빠져나와주세요.

`name`을 입력 받을 때 “이름을 입력하시오. (단, 한글로): “라고 뜨도록 해주세요.

`schoolNum`을 입력 받을 때 “학번을 입력하시오: “라고 뜨도록 해주세요.

`schoolNum`은 무조건 9자리여야 합니다.

`semester`를 입력 받을 때 "학기를 입력하시오. (단, 숫자로): “ 라고 뜨도록 해주세요.

단, `semester`는 정수로 입력받아야 합니다.

`subject`를 입력 받을 때 "과목을 입력하시오. : " 라고 뜨도록 해주세요.

입력을 다 받으면 인스턴스를 생성해준 후 한 줄 띄어주세요 (가독성을 위함입니다)

메서드를 모두 출력해준 후 한 줄 띄어주세요

함수를 출력해주세요

While문의 마지막에 한 줄 띄어주세요

사진 파일 가이드라인입니다!

객체를 두 개 이상 만들어주신 후 이에 대한 터미널 출력 결과를
각각 사진 파일로 만들어주세요.
즉 과제를 잘 수행한다면
`class_assignment.py`와 두 사진파일을 제출하게 되겠죠

```
객체 명을 입력하시오. (단, 영문으로) : youngbeen
이름을 입력하시오. (단, 한글로): 주영빈
학번을 입력하시오: 202114041
학기를 입력하시오. (단, 숫자로): 5
과목을 입력하시오. : 백엔드프레임워크
과목을 입력하시오. : 프론트엔드개발
과목을 입력하시오. : 자료구조
```

```
이름은 주영빈입니다.
학번은 202114041로 아이티융합자율학부 소속입니다.
주영빈은(는) 5학기차로 전공선택을 마쳤습니다.
['백엔드프레임워크', '프론트엔드개발', '자료구조']를 수강합니다.
```

```
자세한 수강목록입니다.
과목명: 백엔드프레임워크 / 과목명의 길이: 8
과목명: 프론트엔드개발 / 과목명의 길이: 7
과목명: 자료구조 / 과목명의 길이: 4
```

assignment1.png

```
객체 명을 입력하시오. (단, 영문으로) : likelion
이름을 입력하시오. (단, 한글로): 김멋사
학번을 입력하시오: 202311111
학기를 입력하시오. (단, 숫자로): 1
과목을 입력하시오. : 말과글
과목을 입력하시오. : 대생세
과목을 입력하시오. : 채플
```

```
이름은 김멋사입니다.
학번은 202311111로 인문융합자율학부 소속입니다.
1학기차인 김멋사는 아직 전공선택 전입니다.
['말과글', '대생세', '채플']를 수강합니다.
```

```
자세한 수강목록입니다.
과목명: 말과글 / 과목명의 길이: 3
과목명: 대생세 / 과목명의 길이: 3
과목명: 채플 / 과목명의 길이: 2
```

assignment2.png

자세한 터미널 입력이에요

```
객체 명을 입력하시오. (단, 영문으로) :  
이름을 입력하시오. (단, 한글로):  
학번을 입력하시오:  
학기를 입력하시오. (단, 숫자로):  
과목을 입력하시오. :  
과목을 입력하시오. :  
과목을 입력하시오. :
```

```
객체 명을 입력하시오. (단, 영문으로) : 종료  
○ youngbeen@YoungBeenui-MacBookAir 🍌 ~/Desktop/python2 |
```

**객체명이 종료라면 이렇게
반복문을 빠져나와야 해요**

자세한 터미널 출력이에요

학번을 입력하시오: 202114041

학번은 202114041로 아이티융합자율학부 소속입니다.

학번을 입력하시오: 202311111

학번은 202311111로 인문융합자율학부 소속입니다.

**입력받은 학번 슬라이싱을 통해
출력을 달리해주세요**

학기를 입력하시오. (단, 숫자로): 5

주영빈은(는) 5학기차로 전공선택을 마쳤습니다.

학기를 입력하시오. (단, 숫자로): 1

1학기차인 김멋사은 아직 전공선택 전입니다.

**입력받은 학기를 통해
출력을 달리해주세요**

그럼 정말정말
수고하셨습니다

다음주 Django 수업에서 봐어요 ~