

Day 7 - User defined functions

기본 틀:

```
def 함수명(매개변수1, 매개변수2, ...):  
    # 함수 내용  
    return 결과값
```

```
temp = func1      # 함수의 주소값을 변수에 저장  
print(temp)       # 함수의 주소값을 프린트  
temp()            # 함수의 주소값을 이용해서 함수를 실행
```

```
<function func1 at 0x7c305a3bb060>  
처음만든 함수
```

- 함수를 실행하지 않고 temp 안에 func1 을 넣게 될 경우, 주소값을 복사할 수 있다.
- 따라서 temp()를 할 경우 함수를 실행시켜준다.
 - ()가 달리면 함수로 인식한다. (같은 주소값을 가지고 있음으로 함수로 인식한다.)

반환 값이 있는 함수:

```
def func4():  
    return '📁' # 이모지를 리턴
```

```
[47] print(func4()) # 함수실행
```

```
📁
```

```
[ ] temp = func4()    # 함수의 반환값을 temp에 저장  
    print(f'temp에 저장된 값: {temp}') #프린트
```

- 반환값을 저장해서 리턴해준다.

기본값이 설정된 매개변수:

```
[24] # 기본값으로 num1 = 0, num2 = 0 설정
def func6(num1 = 0, num2 = 0):
    sum = num1 + num2
    return sum
```

```
print(func6())
print(func6(10))
print(func6(10, 3))
# print(func6(, 3))
# print(func6(None, 3))
print(func6(num2 = 3))
```

- 만약 num2만 넣고싶을때는 num2 = 3 이런식으로 하면된다.

가변 매개변수

- 함수를 호출할 때 *를 사용하면 시퀀스(리스트, 튜플 등)의 요소를 개별적인 위치 인자로 풀어서 전달할 수 있습니다.

```
def func7(*args):
    return args

print(func7())
print(func7(10))
print(func7(10, 30, 50))

()
```

- *지정시 함수에 요소를 몇개를 넣든 상관없다.
- * (리스트 또는 튜플을 지정)
- 콤마로 나눠줘야 한다.

ex)

```
(10, 30, 50)
[45] def func8(a, b, c):
      return a + b + c

numbers = [1, 2, 3]
print(func8(*numbers))
```


- 함수에다가 지정하지 않아도 값을 넣을때 지정해도 된다.
- 하나씩 펼쳐서 넣어준다. (언패킹)

키워드 매개변수

- 키워드 매개변수는 일반적으로 기본값이 설정된 매개변수와 함께 사용됩니다. 함수의 매개변수에 기본값을 설정하면, 함수를 호출할 때 해당 매개변수를 생략할 수 있습니다.


ex)

```
[37] def func9(id, name, age):  
      print(f'아이디: {id}')  
      print(f'이름: {name}')  
      print(f'나이: {age}')
```

 func9('apple', '김사과', 20)


func9('김사과', 20, 'apple') # 순서가 다르면 다른 값이 나온다


func9(age=30, id='orange', name='오렌지') # 각 요소를 하나씩 지정해주면 된다.

 아이디: apple
이름: 김사과
나이: 20
아이디: 김사과
이름: 20
나이: apple
아이디: orange
이름: 오렌지
나이: 30


- 순서대로 값을 넣어야 원하는대로 나온다.
- 따라서 순서가 다른 경우, 요소를 하나씩 지정해줘야 한다.

ex2) **

```
 # 매개변수명과 딕셔너리의 키가 같아야함  
# 딕셔너리의 키는 반드시 문자열 형태  
dic1 = {'age':25, 'id':'banana', 'name':'반하나'}  
func9(**dic1) # ** 두개일 경우 딕셔너리로 보낸다는 뜻
```

 아이디: banana
이름: 반하나
나이: 25

```
[42] # *의 데이터를 보낼 경우 키가 저장  
func9(*dic1)
```

 아이디: age
이름: id
나이: name

- ** 두개 사용시 dictionary 형태로 함수에 전달한다.
- 매개변수명과 딕셔너리의 키가 같아야 하며, 딕셔너리의 키는 반드시 문자열 형태여야 한다.
- * 한개 형태로 보낼 경우 키가 저장된다.

여러개의 반환값

8. 여러개의 반환 값

```
def func10(num1=0, num2=0):  
    return num1 + num2, num1 - num2, num1 * num2, num1 / num2 # 여러개의 반환값
```

```
[45] result = func10(10, 3)  
     print(result) #튜플형태로 나온다.
```

```
(13, 7, 30, 3.3333333333333335)
```

```
[46] # 언패킹  
     result1, result2, result3, result4 = func10(10, 3)  
     print(f'덧셈:{result1}')  
     print(f'뺄셈:{result2}')  
     print(f'곱셈:{result3}')  
     print(f'나눗셈:{result4}')
```

```
덧셈:13  
뺄셈:7  
곱셈:30  
나눗셈:3.3333333333333335
```

```
[51] # 하나의 리턴값만 뽑아야 할 경우:  
     _, _, result3, _ = func10(10, 3)  
     print(f'곱셈: {result3}')
```

```
곱셈: 30
```

- 여러개의 반환값이 필요 할 경우, 튜플로 만들어서 하나씩 언패킹을 해서 각 변수를 사용할 수 있다.

None의 특징

- None은 파이썬에서 특별한 값으로, 아무런 값이 없음을 표현하는 데 사용됩니다. 다른 언어에서의 null 또는 nil과 유사한 개념입니다. None은 파이썬의 내장 상수이며, 그 자체로 데이터 타입이 `NoneType`입니다. 모든 None은 동일하므로, 두 개의 None 값을 비교할 때 항상 `True`를 반환합니다.

자세한건 코드참조