변수가 값을 보관하는 역할을 한다면, 함수(function)는 명령들을 보관하는 역할을 합니다. 동일한 내용을 반복하여 쓰고 있거나, 프로그램을 깔끔하게 정리하고 싶을 때 함수를 사용하면 되는거죠.

수학에서는 독립 변수 **x** 의 값(입력값)을 넣었을 때, 종속 변수 **y** 의 값(결과값)이 나오는 것을 함수라고 하죠? 파이썬 프로그램에서 함수는 더 다양한 방법으로 쓰입니다.

첫째로, 입력값이 있을 수도 있고 없을 수도 있습니다.

둘째로, 입력값과 결과값이 '수'로 한정되는 것이 아니라, 문자열, 리스트, 불린 등 어떤 자료형이든 될 수 있습니다. 심지어 입력값과 결과값이 또 다른 함수가 될 수도 있습니다.

기본 구조

```
1. def 함수이름(파라미터):
2.     실행할 문장 1
3.     실행할 문장 2
4.     실행할 문장 3
```

파이썬 함수의 구조는 위와 같습니다. **def** 는 define(정의)의 약자이며, 함수이름 괄호 안에 있는 파라미터(입력값)는 있을 수도 있고, 없을 수도 있습니다. 함수이름이 포함되는 첫 번째 줄의 끝에 콜론(**:**)이 들어가고, 함수내용이 있는 두 번째 줄부터는 **Tab** 키로 들여써야 한다는 걸 주의하세요. **(1)** 번 줄은 함수의 헤더(header)에 해당되고, **(2)** ~ **(4)** 번 줄은 함수의 바디(body)에 해당됩니다.

사례 1

우선 파라미터 없이 **"Hello, World!"** 와 **"Welcome to Codeit!"** 을 출력하는 **hello** 라는 함수를 만들려고 합니다.

```
def hello():
    print("Hello, world!")
    print("Welcome to Codeit!")
```

위의 프로그램은 오류 없이 실행됐지만, 콘솔에는 아무것도 나오지 않습니다. 함수는 잘 정의했는데, 함수를 호출하는 명령(function call)을 안 썼기 때문입니다.

```
def hello():
    print("Hello, world!")
    print("Welcome to Codeit!")

hello()
```

```
Hello, World!
Welcome to Codeit
```

파라미터가 없을 때는 함수이름() 의 형식으로 함수를 불러올 수 있습니다. 반면, 파라미터가 있을 때는 함수이름(파라미터1, 파라미터2, ...) 의 형식으로 함수를 불러올 수 있습니다.

사례 2

"Hello, world!" 가 아니라, "Hello, 유재석" , "Hello, 김태희" 등 함수를 쓸 때마다 조금씩 다른 문자열을 출력하게 하려면, 함수에게 매번 필요한 정보를 전달해줘야겠죠?

그 중 한가지 방법이 바로 파라미터(parameter)를 넘겨 주는 것입니다.

```
def hello(name):
    print("Hello, %s!" % (name))
    print("Welcome to Codeit!")

hello("유재석")
hello("김태희")
```

```
Hello, 유재석!
Welcome to Codeit!
Hello, 김태희!
Welcome to Codeit!
```

사례 3

```
def print_sum(a, b):
    print(a + b)

def print_product(a, b):
    print(a * b)

def print_residue(a, b):
    print(a % b)

def print_average(x, y, z):
    print((x + y + z) / 3)

print_sum(4, 2)
print_product(3, 4)
print_residue(13, 3)
print_average(5, 10, 15)
```

```
6
12
1
10.0
```

1. print_sum 은 파라미터 두 개를 받고, 두 값의 합을 출력하는 함수입니다. 따라서 print_sum(4, 2) 라는 함수가 불려왔을 때는 4 와 2 를 더한 6 이 출력됩니다.
2. print_product 는 파라미터 두 개를 받고, 두 값의 곱을 출력하는 함수입니다. 따라서 print_product(3, 4) 는 3 과 4 를 곱한 12 가 출력됩니다.

print_product(3, 4) 는 3 과 **4** 를 곱한 **12** 가 출력됩니다.

3. **print_residue** 는 파라미터 두 개를 받고, 첫번째 값을 두번째 값으로 나눈 나머지를 출력하는 함수입니다. **print_residue(13, 3)** 의 경우, **13** 을 **3** 으로 나누었을 때 몫이 **4** 이고 나머지가 **1** 이므로, **1** 이 출력됩니다.
4. **print_average** 는 파라미터 세 개를 받고, 세 수의 평균값(세 수를 모두 더해 3으로 나눈 값)을 출력하는 함수입니다. 따라서 **print_average(5, 10, 15)** 는 세 수의 평균값인 **10.0** 이 출력됩니다.

사례 4

```
1. # 프로필을 출력하는 함수
2. def print_profile(name, age):
3.     print(name)
4.     print(age)
5.
6. my_name = "김연우"
7. my_age = 45
8.
9. print_profile(my_name, my_age)
```

```
김연우
45
```

1. **(2)** ~ **(4)** 번 줄에는 함수 정의가 있으므로 건너뜁니다.
2. **(6)** ~ **(7)** 번 줄에서 **"김연우"** 라는 값이 **my_name** 이라는 변수에 지정되고, **45** 라는 값이 **my_age** 라는 변수에 지정됩니다.
3. **(9)** 번 줄에서 파라미터 **my_name** , **my_age** 와 함께 **print_profile** 함수가 호출되었으므로, 함수가 정의된 **(2)** 번 줄로 이동합니다.
4. 파라미터 **my_name** 에 해당하는 **"김연우"** 라는 값과 **my_age** 에 해당하는 **45** 라는 값이, **(2)** 번 줄에서 **print_profile** 함수 내의 **name** 과 **age** 에 각각 지정됩니다.
5. 함수 내에서 **(3)** ~ **(4)** 번 줄의 명령에 따라, **"김연우"** 와 **45** 가 출력됩니다.
6. **(4)** 번 줄 끝에서 함수의 실행이 종료되면서, 파이썬은 원래있던 **(9)** 번 줄의 끝으로 이동하고 프로그램은 종료됩니다.

들여쓰기 (Indentation)

파이썬에서 들여쓰기는 매우 중요한 역할을 차지합니다. 들여쓰기의 단계에 따라 논리의 층위가 나뉘기 때문입니다. 동일한 논리단계에 있어 들여쓰기의 정도가 같은 명령들의 집합을 블록(block)이라고 부릅니다.

잘못된 들여쓰기는 오류를 일으킵니다.

```
def next_number(n):
    m = n + 1
        print(m)

next_number(3)
```

```
File "untitled.py", line 3
  print(m)
  ^
IndentationError: unexpected indent
```

두번째 줄과 세번째 줄은 **next_number(n)** 의 함수에 대한 정의로서, 같은 논리의 층위에 있습니다. 그럼에도 불구하고 **m = n + 1** 과 **print(m)** 은 서로 다른 블록으로 나뉘어져 있습니다. 따라서 위처럼 오류가 발생한 것이지요. 이 문제를 해결하기 위해서는, 세번째 줄에 들여쓰기가 한 번 더 된 것을 지워주어야 합니다.

```
def next_number(n):
    m = n + 1
    print(m)

next_number(3)


4
```

파이썬에서는 일반적으로 들여쓰기를 위해 **Tab** 키 또는 공백 **4** 칸을 사용합니다. 들여쓰기 시 항상 같은 개수의 공백을 이용해야 한다는 것을 기억하시길 바랍니다.

☑ 수업을 완료하셨으면 체크해주세요.

수강생 Q&A 보기

(/questions?assignment_id=41&sort_by=popular)

질문하기

(/questions/new?assignment_id=41&op1=%ED%94%84%EB%A1%9C%EA%B7%B

이전 강의
파라미터 (/assignments/40)

다음 강의
파라미터 연습 (/assignments/42)