Python_03

- Decomposition(분해) : 기능을 분해하고 재사용 가능하게 만들기
- Abstraction(추상화) : 복잡한 내용을 모르더라도 사용할 수 있도록(스마트폰) 재사용성과 가독성, 생산성

함수 기초

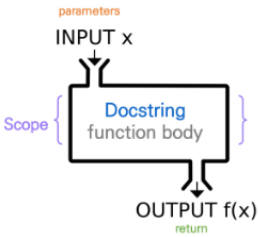
- 내장 함수
  - 파이썬에 기본적으로 포함된 함수
- 외장함수
  - import문을 통해 사용하며, 외부 라이브러리에서 제공하는 함수
- 사용자 정의 함수
  - 직접 사용자가 만드는 함수

함수의 정의

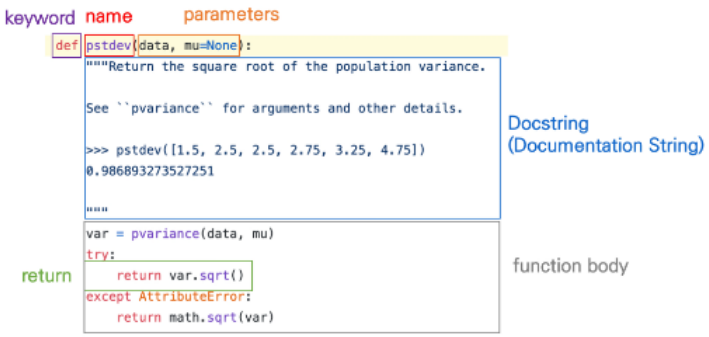
- 함수(Function)
  - 특정한 기능을 하는 코드의 조각(묶음)
  - 특정 코드를 매번 다시 작성하지 않고, 필요시에만 호출하여 간편히 사용

내장 함수

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	F	M	
ascii()	filter()	map()	S
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()



함수 기본 구조



- 선언과 호출(define & call)
- 입력(input)

- 문서화(docstring)
- 범위(scope)
- 결과값(output)

선언과 호출(define & call)

```
def function_name(parameter):
    #code block
    return returning_value
```

- 함수의 선언은 def 키워드를 활용함
- 들여쓰기를 통해 Function body(실행될 코드 블록)를 작성함
  - Docstring은 함수 body 앞에 선택적으로 작성 가능
    - 작성 시에는 반드시 첫 번째 문장에 문자열 “””
- 함수는 parameter를 넘겨줄 수 있음
- 함수는 동작 후에 return을 통해 결괏값을 전달함
- 함수는 함수명()으로 호출하여 사용
  - parameter가 있는 경우, 함수명(값1, 값2, …)으로 호출

함수의 결과값(Output)

값에 따른 함수의 종류

- void function
  - 명시적인 return 값이 없는 경우, None을 반환하고 종료
- value returning function
  - 함수 실행 후, return문을 통해 값을 반환
  - return을 하게 되면, 값 반환 후 함수가 바로 종료

> 주의 **: print VS return**
> - print 함수와 return의 차이점
> + print를 사용하면 호출 될 때마다 값이 출력됨(주로 테스트를 위해 사용)
> + 데이터 처리를 위해서는 return 사용
> + REPL(Read-Eval-Print Loop) 환경에서는 마지막으로 작성된 코드의 리턴 값을 보여주므로 같은 동작을 하는 것으로 착각 할 수 있음.

```
def print_function():
    print('야호')
def return_function():
    return '야호'

print_function()  # 야호
return_function() # 야호
```

함수 반환

- return X → None
- return O → 하나를 반환 ⇒ 여러 개를 원하면, Tuple 활용(혹은 리스트와 같은 컨테이너 활용)

함수의 입력(Input)

Parameter와 Argument

```
def function(ham):    # prarmeter : ham
    return ham

function('spam')      # Argument : 'spam'
# 함수 리턴값 : spam
```

- prarmeter : 함수를 정의할 때, 함수 내부에서 사용되는 변수
- Argument : 함수를 호출 할 때, 넣어주는 값
  - 함수 호출 시 함수의 parameter를 통해 전달 되는 값
  - Argument는 소괄호 안에 할당 func_name(argument)
    - 필수 Argument : 반드시 전달되어야 하는 argument
    - 선택 Argument : 값을 전달하지 않아도 되는 경우는 기본값이 전달

Positional Arguments

- 기본적으로 함수 호출 시 Argument는 위치에 따라 함수 내에 전달

```
def add(x, y):            ------------>         def add(x, y):
    return x + y            add(2,3)                 x = 2 y = 3
                          ------------>              return x + y
```

Keyword Arguments

- 직접 변수의 이름으로 특정 Argument를 전달할 수 있음
- keyword Argument 다음에 Positional Argument를 활용할 수 없음

```
def add(x, y):            add(x = 2, y = 5)
    return x + y          add(2, y = 5)
                          add(x = 2, 5) -> Error 발생!
```

Default Arguments Values

- 기본값을 지정하여 함수 호출 시 argument 값을 설정하지 않도록 함
  - 정의된 것 보다 더 적은 개수의 argument 들로 호출 될 수 있음

```
def add(x, y=0):          ------------>         def add(x, y=0):
    return x + y           add(2)                   x = 2
                          ------------>              return x + y
```

Python의 범위 (Scope)

Python의 범위 (Scope)

- 함수는 코드 내부에 local scope를 생성하며, 그 외의 공간인 global scope로 구분
- Scope
  - global scope : 코드 어디에서든 참조할 수 있는 공간
  - local scope : 함수가 만든 scope. 함수 내부에서만 참조 가능
- variable

- global variable : global scope에 정의된 변수
- local variable : local scope에 정의된 변수

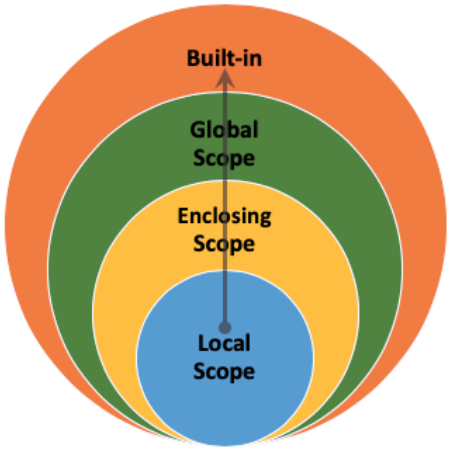
변수 수명 주기(lifecycle)

- 변수는 각자의 수명 주기(lifecycle)가 존재
  - built-in scope
    - 파이썬이 실행된 이후부터 영원히 유지
  - global scope
    - 모듈이 호출된 시점 이후 혹은 인터프리터가 끝날 때까지 유지
  - local scope
    - 함수가 호출될 때 생성되고, 함수가 종료될 때까지 유지

```
def func():
    a = 20
    print('local', a) # local 20

func()
print('global' a) # NameError :name 'a' is not defined
```

이름 검색 규칙(Name Resolution)



- 파이썬에서 사용되는 이름(식별자)들은 이름 공간(namespace)에 저장되어 있음
- 아래와 같은 순서로 이름을 찾아 나가며, LEGB Rule이라고 부름
  - **L**ocal scope : 지역 범위(현재 작업 중인 범위)
  - **E**nclosed scope : 지역 범위 한 단계 위 범위
  - **G**lobal scope : 최상단에 위치한 범위
  - **B**uilt-in scope : 모든 것을 담고 있는 범위(정의하지 않고 사용할 수 있는 모든 것)

    ex) print()
- 함수 내에서는 바깥 Scope의 변수에 접근 가능하나 수정은 할 수 없음

예시

예시 1)

Global scope 이름 공간의 sum 변수에 값 5가 할당

이후 global scope에서 sum은 LEGB에 의해 Built-in scope의 내장 함수보다 5가 먼저 탐색

```
print(sum) # <built-in function sum>
print(sum(range(2))) # 1
sum = 5
print(sum) # 5
print(sum(range(2))) # TypeError: 'int' object is not callable
```

예시 2)

LEGB rule에 따른 탐색

```
a = 0
b = 1
def enclosed():
    a = 10
    c = 3
    def local(c):
        print(a, b, c) # 10 1 300
        print(locals()) # {'c': 300}
        print(globals()) # {'a': 0, 'b': 1}
    local(300)
    print(a, b, c) # 10 1 3
enclosed()
print(a, b) # 0 1
```

global문

- 현재 코드 블록 전체에 적용되며, 나열된 식별자(이름)이 global variable임을 나타냄
  - global에 나열된 이름은 같은 코드 블록에서 global앞에 등장할 수 없음
  - global에 나열된 이름은 parameter, for 루프 대상, 클래스/함수 정의 등으로 정의되지 않아야 함

```
a = 10
def func1():
    global a
    a = 3

print(a) # 10
func1()
print(a) # 3
```

주의 사항

```
a = 10
def func1():
    print(a) # global a 선언 전에 사용
    global a
    a = 3

print(3)
func1()
print(a)

# SyntaxError: name 'a' is used prior to global declaration
```

```
a = 10
def func1():
    global a # parameter에 global 사용 불가
    a = 3

print(3)
func1(3)
print(a)

# SyntaxError: name 'a' is parameter and global
```

Nonlocal

- global을 제외하고 가장 가까운(둘러싸고 있는) scope의 변수를 연결하도록 함
  - nonlocal에 나열된 이름은 같은 코드 블록에서 nonlocal 앞에 등장할 수 없음
  - nonlocal에 나열된 이름은 parameter, for 루프 대상, 클래스/함수 정의 등으로 정의되지 않아야 함
- global과는 달리 이미 존재하는 이름과의 연결만 가능함

```
x = 0
def func1():
    x = 1
    def func2():
        nonlocal x
        x = 2
    func2()
    print(x)  # 2
func1()
print(x) # 0
```

함수의 범위 주의

- 기본적으로 함수에서 선언된 변수는 Local scope에 생성되며, 함수 종료 시 사라짐
- 해당 scope에 변수가 없는 경우 LEGB rule에 의해 이름을 검색함
  - 변수에 접근은 가능하지만, 해당 변수를 수정할 수는 없음
  - 값을 할당하는 경우 해당 scope의 이름공간에 새롭게 생성되기 때문
  - **단, 함수 내에서 필요한 상위 scope변수는 argument로 넘겨서 활용할 것**
- 상위 scope에 잇는 변수를 수정하고 싶다면 global, nonlocal키워드를 활용 가능
  - 단, 코드가 복잡해지면서 변수의 변경을 추적하기 어렵고 예기치 못한 오류 발생
  - 가급적 사용하지 않는 것을 권장하며, **함수로 값을 바꾸자면 항상 argument로 넘기고 리턴 값을 사용하는 것을 추천**