범위

작성일시	@2022년 7월 20일 오후 2:34
내용	문법 파이썬 함수
주차	1주차
자료	

Python의 범위(Scope)

- 함수는 코드 내부에 **local scope**를 생성하며, 그 외의 공간인 **global scope**로 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 수명주기(life cycle)

- 변수는 각자의 수명주기(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

이름 검색 규칙(Name Resolution)

- 파이썬에서 사용되는 이름(식별자)들은 이름공간(namespace)에 저장되어 있음
- 아래와 같은 순서로 이름을 찾아나가며, **LEGB Rule**이라 부름

- **Local** scope: 지역 범위(현재 작업 중인 범위)
- **Enclosed** scope: 지역 범위 한 단계 위 범위
- **Global** scope: 최상단에 위치한 범위
- **Built-in** scope: 모든 것을 담고 있는 범위(정의하지 않고 사용할 수 있는 것)

```
print(sum) # <bulit-in function sum>
print(sum(range(2))) # 1

sum = 5
print(sum) # 5
print(sum(range(2)) # TypeError: 'int' object is not callable
```

```
a = 0
b = 1
def enclosed():
    a = 10
    c = 3
    def local(c):
        print(a, b, c) # 10 1 300
    local(300)
    print(a, b, c) # 10 1 3
enclosed()
  print(a, b) # 0 1
```

함수의 범위(Scope)

Global 문

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

```
# 함수 내부에서 글로벌 변수 변경하기

a = 10
def func1():
    golbal a
    a = 3

print(a) # 10
```

```
func()
print(a) # 3
```

- global 관련 에러

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

print(3)
func1()
print(a)
#SyntaxError: name 'a' is used prior to global declaration

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

print(a)
func1(3)
print(a)
#SyntaxError: name 'a' is parameter and global
```

nonlocal

- global을 제외하고 가장 가까운 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 않음
- global과는 달리 **이미 존재하는 이름과의 연결만 가능**함!

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

func1()
print(x) # 0
```