W5 함수

content	함수
check	☐
date	@2023년 3월 19일
상태	To edit

#1. 함수 기초

여러 명이 프로그램을 함께 개발할 때, 코드를 어떻게 작성하면 좋을까

- 필요한 부분 나누어 작성한 후 합치기

함수의 개념과 장점

▼ **함수(function)?**

- 어떤 일 수행하기 위한 코드 덩어리 or 묶음
  - ex. 도형 넓이 구하는 프로그램 → 함수화하여 필요할 때마다 호출

▼ **함수의 장점**

- 필요할 때마다 호출 가능
  - 반복적 수행 업무할 때 유용
- 논리적 단위로 분할 가능
  - ex. 도형 계산 프로그램
    - 곱셈하는 / 덧셈하는 / 나눗셈하는 …
- 코드의 캡슐화
  - 인터페이스를 잘 정의함 → 타인이 이용가능
    - 인터페이스 정의? → 코드의 인풋과 아웃풋 명확히 함

함수의 선언

▼ 선언 방법

```
def 함수 이름 (매개변수 1 ....):
    명령문 1
```

```
    명령문 2
      return <반환값>
```

- def : ‘definition’ → 함수 정의 시작!
- 함수 이름 : 자유롭게 지정 , but 규칙 존재
  - 소문자
  - 띄어쓰기할 때 언더바(_) 사용
  - 동사, 명사 함께 사용
    - ex. find_number
  - 외부 공개용 → 줄임말 사용 X, 명료한 이름 사용
- 매개변수(parameter): 함수에서 입력값으로 사용하는 변수, 1개 이상 값 적기 가능
- 명령문 : 반드시 들여쓰기 후 코드 입력, 수행할 코드는 지금가지 배운 코드와 동일

▼ 함수 선언 예시

```
def calculate_circle_area(r, pi):
    return r ** 2 * pi
```

- 함수 이름 : calculate_circle_area
- 매개변수 : r, pi
- 값 반환(return) : r**2*pi의 결과를 반환
- +) 반환?
  - ex. f(x) = x + 1 → f(1) = 2
  - 함수 f(x)에서 x에 1이 들어가면 2가 반환되는 것
  - 결과값이라고 생각하면 됨

함수의 실행 순서

- 선언 후, 호출할 차례
- 코드

```
def calculate_rectanlge_area(x,y):
    return x * y

rectangle_x = 10
rectangle_y = 20
print("사각형 x의 길이:", rectangle_x)

print("사각형 y의 길이:", rectangle_y)

#넓이를구하는함수 호출
print("사각형의 넓이:", calculate_rectangle_area(rectangle_x, rectangle_y))
```

- 실행 화면

```
사각형 x의 길이: 10
사각형 y의 길이: 20
사각형의 넓이: 200
```

- 설명
  - 함수 정의된 부분은 실행되지 않음
  - 해당 코드를 메모리에 업로드 → 다른 코드 호출 사용 가능하도록 준비
  - 따라서 가장 먼저 함수 선언 코드 입력
  - 각 변수에 할당된 x,y 값이 매개변수가 됨

프로그래밍의 함수와 수학의 함수

- f(x) = x + 1
- in 수학

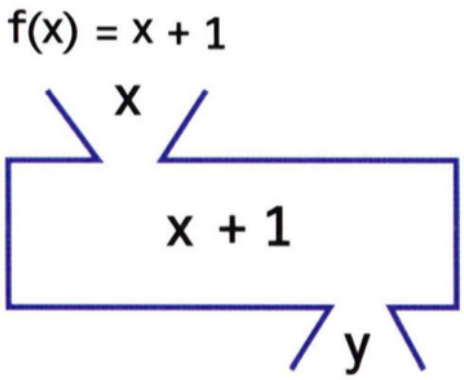


그림 5-1 수학에서 함수의 형태

- 문제

f(x) = 2x + 7, g(x) = x^2이고 x = 2일 때
f(x) + g(x) + f(g(x)) + g(f(x))의 값은?

f(2) = 11, g(2) = 4, f(g(x)) = 15, g(f(x)) = 121
∴ 11 + 4 + 15 + 121 = 151

- f(x) 와 g(x) 코드로 표현

```
def f(x):
  return 2 * x +7
def g(x):
    return x ** 2

x =2
print(f(x)+ g(x) + f(g(x)) + g(f(x)))
#151
```

- f(x) → 11
- g(x) → 4
- f(g(x)) → 15
- g(f(x)) → 121

▼ **+) 매개변수와 인수**

- 매개변수(parameter)
  - 어떤 변수 사용하는지 정의
  - 설계도
- 인수(argument)
  - 실제 매개변수에 대입되는 값
  - 그 설계도로 지은 건물

```
def f(x):
    return 2*x + 7

print(f(2))
#11
```

- def f(x):
  - ‘x’ → 매개변수
  - 어떤 변수 사용하는지 정의
- f(2)
  - ‘2’ → 인수
- 구분없이 사용하기도 함!

함수의 형태

매개변수와 반환값(returnvalue)의 유무에 따라 네가지 형태 분류

표 5-1 함수의 형태

매개변수 유무 / 반환값 유무	매개변수 없음	매개변수 있음
반환값 없음	함수 내부 명령문만 수행	매개변수를 사용하여 명령문만 수행
반환값 있음	매개변수 없이 명령문을 수행한 후 결과값 반환	매개변수를 사용하여 명령문을 수행한 후 결과값 반환

▼ 매개변수 X & 반환값 X

- 함수 내부 명령문만 수행
  - return 명령어 사용 X, 매개변수 입력 X

```
def a_rectangle_area():
    print(5 * 7)

a_rectangle_area()
#35
```

- 입력값 X & 출력값 X
- 함수 자체는 none 값을 가짐
  - 내부 함수인 print()로 인해 값 출력됨

▼ 매개변수 O & 반환값 X

- 매개변수 활용 → 명령문만 수행

```
def b_rectangle_area(x,y):
    print(x * y)

b_rectangle_area(5,7)
#35
```

- 마찬가지로 함수는 none으로 치환됨
  - 내부 함수 print()가 실행된 것

▼ 매개변수 X & 반환값 O

- 매개변수 활용하지 않고, 명령문 수행 후 결과값 반환

```
def c_rectangle_area():
    return(5 * 7)
```

```
print(c_rectangle_area())
#35
```

- 입력값 X & 출력값 O
- 함수 **호출할 때 print()** 사용
  - 내부 함수 수행 시키는 것 X
  - return 문으로 35로 치환됨

▼ **매개변수 O & 반환값 O**

- 매개변수 활용 , 명령문 수행 후 결과값 반환

```
def d_rectangle_area(x, y):
    return(x * y)

print(d_rectangle_area(3, 5))
#35
```

- return 있는 경우(반환값 있는 경우)
  - 해당 함수 호출하는 곳에 함수 반환값을 변수에 할당시켜 사용

```
result = d_rectangle_area(3, 5)
```

#2. 함수 심화

함수의 호출 방식

- 변수 호출

```
def f(x):
  y=x
  x=5
  return x*y

x=3
print(f(x))
#9
print(x)
#3
```

→ 내부의 x와 밖의 x는 다른 변수

- 함수가 변수 호출하는 방식

표 5-2 함수가 변수를 호출하는 방식

종류	설명
값에 의한 호출 (call by value)	• 함수에 인수를 넘길 때 값만 넘김 • 함수 내부의 인수값 변경 시 호출된 변수에 영향을 주지 않음
참조 호출 (call by reference)	• 함수에 인수를 넘길 때 메모리 주소를 넘김 • 함수 내부의 인수값 변경 시 호출된 변수값도 변경됨

변수의 사용 범위

- 변수가 코드에서 사용되는 범위
- 함수 내부 or 프로그램 전체
- 고려할 변수 두 가지
  - 지역 변수(local variable): 함수 내부
  - 전역 변수(global variable): 프로그램 전체

▼ **변수 사용 방법**

- 잘못 사용한 경우

```
def test(t):
    print(x)
    t = 20
    print('in function :',t)

x = 3
test(x)
#10
#in function : 20

print('in main :',x)
#10

print('in main :',t)
#error
```

- 신경 쓸 변수 → x, t
  - test(x)의 x
    - 10
    - 전역 변수
  - 내부의 t
    - 지역 변수
    - error 발생

```
Traceback (most recent call last):
    File "scoping_rule.py", line 9, in <module>
        print("In Main:", t)
NameError: name 't' is not defined
```

- **지역 변수(local variable)**

```
def f():
    s = 'I love London'
    print(s)

s = 'I love Paris'
f()
#I love London

print(s)
#I love Paris
```

  - 동일한 s가 아닌 다른 메모리 주소를 가진 서로 다른 변수
    - ‘I love Paris’로 s값을 변경해도 내부의 ‘I love London’이 출력됨!

- **전역 변수(global variable)**

```
#함수 정의
def add(x,y):
    total = x + y #total -> 지역변수
    print('in function')
    print('a:',str(a), "b:", str(b), "a + b:", str(a + b), "total:", str(total))
    return total

#전역 변수 선언 a, b, total
a = 3
b = 5
total = 0

#정의 함수 사용 X
print('in program -1')
print('a:',str(a), "b:", str(b), "a + b:", str(a + b))

#정의한 함수 사용
sum = add(a,b)
print("After Calculation")
printC'Total', str(total), " Sum:", str(sum))
```

- 실행 화면

```
In Program - 1
a: 5 b: 7 a + b: 12
In Function
a: 5 b: 7 a + b: 12 total: 12
After Calculation
Total : 0  Sum: 12
```

재귀 함수

- 자기 자신 다시 호출
- 수학의 점화식과 같은 형태
  - n!
  - n! = n x (n-1)!
- 코드로 구현?

```
def factorial(n):
    if n == 1:
      return 1
    else:
      return n * factorial(n - 1) #재귀

print(factorial(int(input("Input Number for Factorial Calculation: "))))
```

- 실행 화면

```
Input Number for Factorial Calculation: 5      ← 사용자 입력
120                                            ← 화면 출력
```

- 실행 과정

```
5 * factorial(5 - 1)
= 5 * 4 * factorial(4 - 1)
= 5 * 4 * 3 * factorial(3 - 1)
= 5 * 4 * 3 * 2 * factorial(2 - 1)
= 5 * 4 * 3 * 2 * 1
```

#3. 함수의 인수

- 인수
  - 함수의 입력으로 들어가는 변수

표 5-3 파이썬에서 인수를 사용하는 방법

종류	내용
키워드 인수	함수의 인터페이스에서 지정한 변수명을 사용하여 함수의 인수를 지정하는 방법
디폴트 인수	별도의 인수값이 입력되지 않을 때 인터페이스 선언에서 지정한 초깃값을 사용하는 방법
가변 인수	함수의 인터페이스에서 지정한 변수 이외의 추가 변수를 함수에 입력할 수 있도록 지원하는 방법
키워드 가변 인수	매개변수의 이름을 따로 지정하지 않고 입력하는 방법

키워드 인수

- 함수에 입력되는 매개변수의 변수명 사용 → 함수 인수 지정

```
def print_name(my_name,your_name):
    print("Hello {0}, My name is {1}".format(your_name,my_name))

print_name('kyuree', "TEAMLAB")
#Hello TEAMLAB, My name is kyuree

print_name(your_name = 'TEAMLAB',my_name = 'kyuree')
#Hello TEAMLAB, My name is kyuree
```

- print_name('kyuree', "TEAMLAB")
  - 'kyuree' → my_name
  - 'TEAMLAB' → your_name
- print_name(your_name = 'TEAMLAB',my_name = 'kyuree')
  - 변수명 명시 → 해당 변수에 원하는 값 할당
  - 실행 결과 동일

디폴트 인수

- 매개변수에 기본값 지정하여 사용
- 아무런 값도 입력되지 않을 대 기본값 사용

```
def print_name_2(my_name, your_name = "TEAMLAB"):
    print('Hello {0}, My name is {l}'.format(your_name, my_name))

print_name_2("kyuree", "TEAMLAB")
#Hello TEAMLAB, My name is kyuree
```

```
print_name_2("kyuree")
#Hello TEAMLAB, My name is kyuree
```

- def print_name_2(my_name, your_name = "TEAMLAB"):
  - your_name 매개변수에 기본값으로 ‘TEAMLAB’ 지정
  - 별도의 값 할당하지 않아도 작동
- 디폴트 인수 사용할 때 초깃값 입력해주기도 함
  - ex. 초깃값 : ‘0’
    - ‘init =0 ‘
    - ‘init = None’

가변 인수

- 매개변수 개수 정하지 않을 때
  - 고객이 몇 개 물건 구매할지 모르는 마트 계산기에서 합산할 때
- 가변인수 → *(asterisk)로 표현
- *
  - 곱셈
  - 제곱 연산
  - *변수 묶어주는 가변 인수 만들 때*

```
def asterisk_test(a,b,*args):
    return a + b +sum(args)

print(asterisk_test(1,2,3,4,5))
#15
```

- a,b 는 기본적으로 존재
  - 1, 2
- 나머지를 *args로 넘겨 받음
  - 3,4,5
- *args → 가변 인수
  - 가장 마지막으로 선언되어야 함

키워드 가변 인수

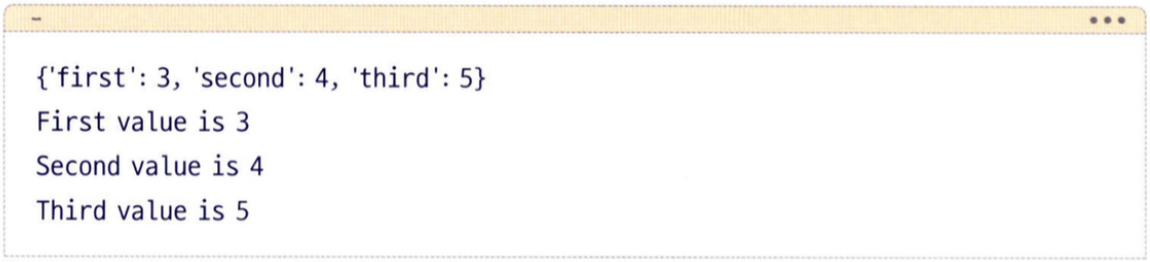
- 가변 인수
  - 변수 순서대로 튜플 형태로 저장
  - 변수 이름 지정할 수 없다는 단점 존재

- 단점 보완한 것이 키워드 가변 인수!
- *args → **kwargs

```
def kwargs_test(**kwargs):
    print(kwargs)
    print("First value is {first}".format(**kwargs))
    print("Second value is {second}".format(**kwargs))
    print("Third value is {third}".format(**kwargs))

kwargs_test(first =3, second =4, third = 5)
```

- 실행 화면



- print(kwargs)
  - 키워드 인수 지정
    - kwargs_test(first =3, second =4, third = 5)
  - {'first' :3, 'second ': 4, 'third:': 5}
    - 딕셔너리 자료형 → 키와 값이 일대일 대응