Function

함 수

1.함수의 정의

- 함수(Function)
  - 함수는 여러 개의 문장을 하나로 묶은 단위입니다.
  - 여러 개의 문장을 하나의 이름으로 묶어서 호출해서 사용하기 위한 개념입니다.
  - 반복적으로 호출해서 실행이 가능하며 코드의 일정 부분을 별도의 논리적인 개념으로 분리하기 위해서 사용하기도 합니다.
  - 함수는 위처럼 코드를 재사용할 수 있게 해주고 프로그램을 논리적으로 구성할 수 있도록 해줍니다.
  - 파이썬의 함수는 전부 일급 함수(First Class Function)라서 함수를 다른 함수의 인수로 대입할 수 있고 함수의 반환값을 다른 함수에 전달할 수 있고 변수에 대입하는 것도 가능

1.함수의 정의

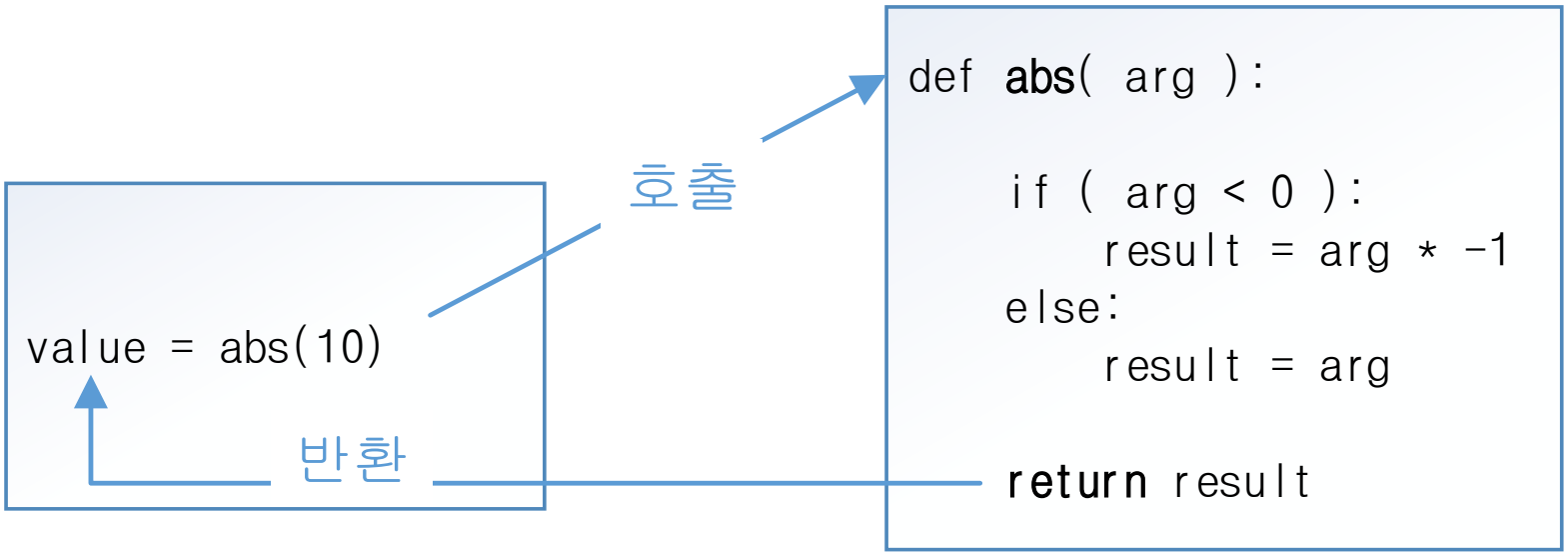
- 정의
  - 정의(Definition)란, 어떤 이름을 가진 코드가 구체적으로 어떻게 동작하는지를 "구체적으로 기술"하는 것
- 파이썬에서는 함수나 메소드를 정의할 때 definition(정의)를 줄인 키워드인 **def**를 사용
- def 다음에 함수이름과 인수들을 나열하고 :
- 함수의 몸체는 그 다음 줄에 들여쓰기를 하고 시작해야 하며 파이썬은 어떤 형식의 데이터도 인수로 전달할 수 있기 때문에 인수의 자료형은 기재하지 않습니다.
- return 은 결과를 돌려주고자 할 때 결과를 함수를 호출한 곳으로 돌려줄 때 사용

- def 키워드를 이용한 함수 정의

```
def 함수이름(인수들):
  문장을 나열
  return <값>
```

1.함수의 정의

- 호출(Call)
  - 모든 함수는 이름을 갖고 있으며, 이 이름을 불러주면 파이썬은 그 이름 아래 정의되어 있는 코드를 실행.
- 반환(Return)
  - 함수가 자신의 코드를 실행하고 나면 결과가 나오는데, 그 결과를 자신의 이름을 부른 코드에게 돌려줌.



1.함수의 정의

- return 문장 다음에 아무러 값이 없으면 파이썬은 None 객체를 반환합니다.
- return 문장이 없어도 none 객체를 반환한 것으로 간주합니다.
- return을 할 때 여러 개의 값을 return 하면 튜플을 만들어서 리턴하게 됩니다.

```
def nothing():
    return
print(nothing())
def swap(a, b):
    return b, a
a = 10
b = 20
print(swap(a,b))
```

=>첫번째 출력문은 return 뒤에 데이터가 없으므로 None
=>두번째 출력문은 2개의 값을 리턴했으므로 튜플로 만들어져서 (20, 10)

2.매개변수 전달

- 파이썬에서는 매개변수(argument)에 데이터를 넘겨주면 객체의 참조를 넘겨줍니다.
- 변경이 불가능한 데이터를 넘겨주면 복제를 해서 넘겨주고 변경이 가능한 객체를 넘겨주면 참조를 넘겨줍니다.
- 매개변수가 있는 함수를 호출할 때 매개변수를 대입하지 않으면 에러 발생

```
def f(t):
    t = 10
a = 20
f(a)
print(a)
```

=>a는 정수 상수를 대입했기 때문에 변경이 불가능하므로 20

2.매개변수 전달

```
def f(t):
    t[0] = 100
li = [1,2,3]
f(li)
print(li[0])
```

=>list가 저장하고 있는 주소를 넘겨주었기 때문에 함수 내에서 리스트의 내용을 변경하면 원본의 데이터도 변경됩니다.

=>따라서 li[0]의 값은 100

2.매개변수 전달

- 매개변수의 기본값: 함수의 매개변수가 있는 경우 매개변수를 넘겨주지 않으면 에러가 발생합니다.
- 파이썬에서는 매개변수에 기본값을 할당할 수 있는데 매개변수이름=값 의 형식을 취하게 되면 매개변에 값을 대입하지 않을 때 기본적으로 값이 대입되면 매개변수에 값이 있으면 그 값을 취하게 됩니다.
- 기본값이 있는 매개변수 뒤에는 기본값이 없는 매개변수가 올 수 없습니다.

```
def f(n, step=1):
    return n+step;
a = 1
a = f(a,10)
print(a)
b = 1
b = f(b)
print(b)
```

=>첫번째의 경우는 매개변수가 있으므로 step에 10이 대입되서 11
=>두번째의 경우는 매개변수가 생략되어 있으므로 step에 1이 대입되서 2

2.매개변수 전달

- 매개변수를 대입할 때는 기본적으로 순서대로 대입되지만 파이썬에서는 키워드를 이용해서 순서와 상관없이 대입이 가능합니다.
- 매개변수를 대입할 때 매개변수의 이름과 함께 대입하면 됩니다.

```
def f(height, weight):
    print("키는", format(height, 'd'))
    print("몸무게는 ", format(weight, 'd'))

f(180, 67)
f(weight=80, height=190)
```

2.매개변수 전달

- 가변 매개변수(Arbitrary Argument List)
  - 입력 개수가 달라질 수 있는 매개변수
  - *를 이용하여 정의된 가변 매개변수는 튜플이고 **이용하여 정의된 매개변수는 dic

```
def 함수이름(*매개변수):
    코드블록
```

매개변수 앞에 *를 붙이면 해당매개변수는 가변으로 지정됩니다.

- 실습 (가변 매개변수)

```
def merge_string(*text_list):
        result = ''
        for s in text_list:
                result = result + " " + s
        return result
print(merge_string('안녕하세요', '반갑습니다'))
print(merge_string('아버지가', '방에', '들어가신다.'))
```

2.매개변수 전달

- 실습 (가변 매개변수)

```
def f(width, height, **kw):
    print(width, height)
    print(kw)
f(width=10, height=5, depth=20, dimension=40)
```

2.매개변수 전달

- 일반 매개변수와 함께 사용하는 가변매개변수

```
>>> def print_args(argc, *argv):
        for i in range(argc):
            print(argv[i])


>>> print_args(3, "argv1", "argv2", "argv3")
argv1
argv2
argv3

>>> print_args(argc=3, "argv1", "argv2", "argv3")
SyntaxError: non-keyword arg after keyword arg
```

가변 매개변수 앞에 정의된 일반 매개변수는 키워드 매개변수로 호출할 수 없습니다.

2.매개변수 전달

- 가변 매개변수와 함께 사용하는 일반 매개변수

```
>>> def print_args(*argv, argc):
        for i in range(argc):
            print(argv[i])

>>> print_args("argv1", "argv2", "argv3", argc=3)
argv1
argv2
argv3

>>> print_args("argv1", "argv2", "argv3", 3)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    print_args("argv1", "argv2", "argv3", 3)
TypeError: print_args() missing 1 required keyword-only
argument: 'argc'
```

가변 매개변수 뒤에 정의된 일반 매개변수는 반드시 키워드 매개변수로 호출해야 합니다.

3.유효범위

- "함수 밖에서 변수 a를 정의하여 0을 대입하고, 함수 안에서 변수 a를 또 정의하여 1을 대입했다. 이 함수를 실행(호출)하고 나면 함수 밖에서 정의된 변수 a의 값은 얼마일까?"
  - 답 : 0
  - 함수 밖에 있는 a와 안에 있는 a는 이름은 같지만 사실은 완전히 별개의 변수

- 실습 1

```
>>> def scope_test():
        a = 1
        print('a:{0}'.format(a))


>>> a = 0
>>> scope_test()
a:1
>>> print('a:{0}'.format(a))
a:0
```

함수를 정의하는 시점에서는 a가 메모리에 생성되지 않습니다. 함수를 호출하면 그제서야 함수의 코드가 실행되면서 a가 메모리에 생성됩니다.

함수 밖에서 a를 정의하고 0으로 초기화 합니다.

scope_test()가 호출되면 함수 내부에서 a를 정의하고 1로 초기화합니다.

하지만 함수 밖에서 정의한 a를 출력해보면 여전히 0을 갖고 있음을 확인할 수 있습니다.

3.유효범위

- 변수는 자신이 생성된 범위(코드블록) 안에서만 유효
- 함수 안에서 만든 변수는 함수 안에서만 살아있다가 함수 코드의 실행이 종료되면 그 생명이 다함 ➔ 이것을 지역변수(Local Variable)라고 함
- 이와는 반대로 함수 외부에서 만든 변수는 프로그램이 살아있는 동안에는 함께 살아있다가 프로그램이 종료될 때 같이 소멸됨.
  ➔이렇게 프로그램 전체를 유효 범위로 가지는 변수를 전역 변수(**Global** Variable) 라 함.
- 파이썬은 함수 안에서 사용되는 모든 변수를 지역변수로 간주
- 전역 변수를 사용하기 위해서는 **global** 키워드를 이용
- 함수 내에서 자신을 포함하는 함수의 변수를 사용할 때는 nonlocal 키워드를 사용

3.유효범위

- 변수의 유효범위

```
def outer():
        a = 1
        def inner():
            nonlocal a
            print(a)
        inner()
a = 0
outer()
```

nonlocal 키워드는 지정한 변수의 유효범위가 함수를 포함하고 있는 함수임을 알리고 지역변수의 생성을 막습니다. 따라서 결과는 1

4.재귀함수(Recursive Function)

- 재귀함수(Recursive Function)는 자기 스스로를 호출하는 함수
- 함수가 자기 자신을 부르는 것을 재귀호출(Recursive Call)이라 함.

- 재귀 함수의 예

```
def some_func(count):
    if count > 0:
        some_func(count-1)
    print(count)
```

4.재귀함수(Recursive Function)

- 팩토리얼

```
>>> def factorial(n):
        if n == 0:
                return 1
        elif n > 0:
                return factorial(n-1)*n


>>> factorial(5)
120
>>> factorial(10)
3628800
>>> factorial(100)
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

4.재귀함수(Recursive Function)

- 재귀 호출의 단계가 깊어질 수록 메모리를 추가적으로 사용하기 때문에 재귀 함수가 종료될 조건을 분명하게 만들어야 함.

```
>>> def no_idea():
        print("나는 아무 생각이 없다.")
        print("왜냐하면")
        no_idea()

>>> no_idea()
나는 아무 생각이 없다.
왜냐하면
나는 아무 생각이 없다.
왜냐하면…
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    no_idea()
  File "<pyshell#8>", line 4, in no_idea
    no_idea()
…
File "<pyshell#8>", line 2, in no_idea
    print("나는 아무 생각이 없다.")
  File "C:₩Python34₩lib₩idlelib₩PyShell.py", line 1342, in write
    return self.shell.write(s, self.tags)
RuntimeError: maximum recursion depth exceeded while calling a Python object
```

종료할 조건도 지정해주지 않은 채 무조건 재귀호출을 수행하면 스택 오버플로우가 발생합니다.

스택 오버 플로우가 발생하면 파이썬에서 지정해놓은 최대 재귀 단계를 초과했다는 에러가 출력됩니다.

5. 함수를 변수에 담아 사용

```
>>> def print_something(a):
        print(a)


>>> p = print_something
>>> p(123)
123
>>> p('abc')
abc
```

() 없이 함수의 이름만을 변수에 저장합니다.

변수의 이름 뒤에 ()를 붙여 함수처럼 호출하면 됩니다.

```
>>> def plus(a, b):
        return a+b

>>> def minus(a, b):
        return a-b

>>> flist = [plus, minus]
>>> flist[0](1, 2)
3
>>> flist[1](1, 2)
-1
```

plus() 함수와 minus() 함수를 리스트의 요소로 집어넣습니다.

flist[0]은 plus() 함수를 담고 있습니다. 따라서 이 요소 뒤에 괄호를 열고 매개변수를 입력하여 호출하면 plus() 함수가 호출됩니다.

flist[1]는 minus()를 담고 있으므로 이 코드는 minus(1, 2)와 같습니다.

5. 함수를 변수에 담아 사용

- 함수를 변수에 담을 수 있는 이유?
  - 파이썬이 함수를 일급 객체(First **Class** Object)로 다루고 있기 때문
  - 일급 객체란 프로그래밍 언어 설계에서 매개변수로 넘길 수 있고 함수가 반환할 수도 있으며 변수에 할당이 가능한 개체를 가리키는 용어
  - 파이썬에서는 함수를 "매개변수"로도 사용할 수 있고 함수의 결과로 "반환"하는 것도 가능

```
>>> def hello_korean():
        print('안녕하세요.')

>>> def hello_english():
        print('Hello.')

>>> def greet(hello):
        hello()

>>> greet(hello_korean)
안녕하세요.
>>> greet(hello_english)
Hello.
```

5. 함수를 변수에 담아 사용

❖ 함수를 결과로써 반환하기

```
>>> def hello_korean():
        print('안녕하세요.')

>>> def hello_english():
        print('Hello.')

>>> def get_greeting(where):
        if where == 'K':
            return hello_korean
        else:
            return hello_english

>>> hello = get_greeting('K')
>>> hello()
안녕하세요.
>>> hello = get_greeting('E')
>>> hello()
Hello.
```

매개변수 where가 'K'인 경우 hello_korean() 함수를 반환합니다.

그 외의 경우 hello_english() 함수를 반환합니다.

get_greeting() 함수가 반환하는 결과를 변수 hello 에 담아 "호출"합니다.

6. 중첩함수

- 중첩 함수(Nested Function) : 함수 안에 정의된 함수
  - 중첩 함수는 자신이 소속되어 있는 함수의 매개변수에 접근할 수 있음.

```
>>> import math
>>> def stddev(*args):
        def mean():
                return sum(args)/len(args)
        def variance(m):
                total = 0
                for arg in args:
                        total += (arg - m ) ** 2
                return total/(len(args)-1)
        v = variance(mean())
        return math.sqrt(v)

>>> stddev(2.3, 1.7, 1.4, 0.7, 1.9)
0.6
```

중첩 함수

중첩 함수

6. 중첩함수

- 중첩함수의 자신이 소속되어 있는 함수 외부에서는 보이지 않음.

```
>>> mean()
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    mean()
NameError: name 'mean' is not defined
```

7.Lambda

- lamdba
  - 이름이 없는 한 줄 짜리 함수
  - 작성 방법

    Lambda <인수 나열>:<반환할 내용>
  - 인수가 없으면 생략 가능

    항상 1을 리턴하는 람다 함수

    f = lambda:1

    2개의 인수를 받아서 합을 구해주는 람다 함수

    f = lambda x , y : x+y
  - 람다 함수도 매개변수의 기본값을 가질 수 있으며 정의되지 않은 키워드 인수도 사용이 가능

8. 함수적 프로그래밍

- map 내장 함수
  - 컬렉션과 함수를 매개변수로 받아서 컬렉션의 모든 데이터를 함수의 매개변수로 대입해서 결과를 리턴하는 함수

```
def f(x):
    return x*x


li = [1,2,3,4,5]
print('반복문을 이용한 실행')
for imsi in li:
    print(f(imsi), end=" ")
print('₩nmap을 이용한 실행')
result = list(map(f,li))
print(result)
print('lambda와 map을 이용한 실행')
result = list(map(lambda x : x*x,li))
print(result)
```

8. 함수적 프로그래밍

- filter 내장 함수
  - 컬렉션과 함수를 매개변수로 받아서 컬렉션의 모든 데이터를 함수의 매개변수로 대입해서 결과가 참인 경우만 리턴하는 함수

```
li = [1,2,3,4,5]
print('lambda와 filter을 이용한 실행')
result = list(filter(lambda x : x%2==0, li))
print(result)
```