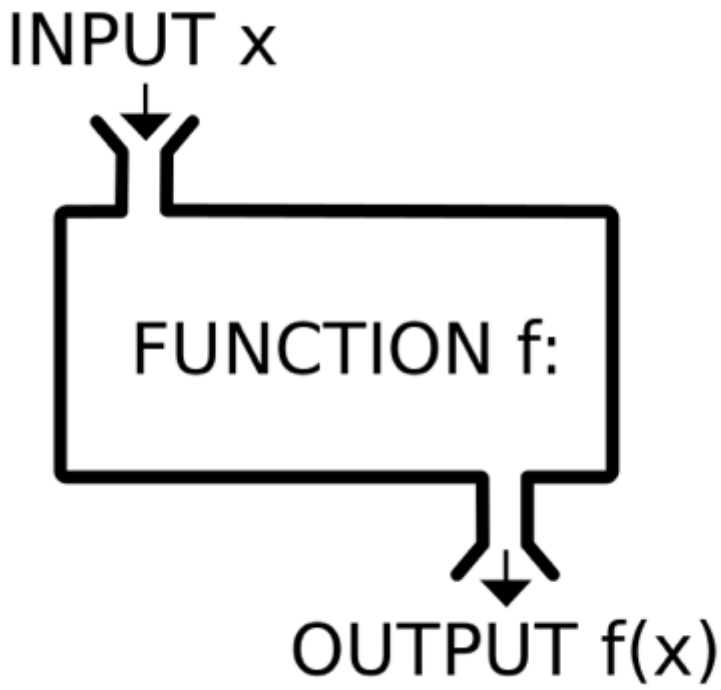
함수(function) I

반복적인 것을 재사용이 가능하게 만든 것이 함수이다.

함수(function) I

- 함수(function)?
- 함수의 Output
- 함수의 Input



들어가기전에

다음의 코드를 봅시다. 무엇을 하는 코드일까요? 표준편차!

In [ ]:

```
values = [100, 75, 85, 90, 65, 95, 90, 60, 85, 50, 90, 80]

total = 0
cnt = 0
# 값을 다 더하고, 갯수를 센다.
for value in values:
    total += value
    cnt += 1

mean = total / cnt

total_var = 0

for value in values:
    total_var += (value - mean) ** 2
```

```
sum_var = total_var / cnt

target = sum_var

count = 0

while True :
    count += 1
    root = 0.5 * (target + (sum_var / target))
    if (abs(root - target) < 0.0000000000000001) :
        break
    target = root

std_dev = target
print(std_dev)
```

> 이해하기 쉬운가요? 그리고 만약 다른 곳에서 동일한 작업을 다시해야할 경우 어떻게 해야 할까요?

In [ ]:
```
import math
values = [100, 75, 85, 90, 65, 95, 90, 60, 85, 50, 90, 80]
```

In [ ]:
```
# 함수 활용
# 갯수
cnt = len(values)
# 평균
mean = sum(values) / cnt
#
sum_var = sum(pow(value - mean, 2) for value in values) / cnt

std_dev = math.sqrt(sum_var)

print(std_dev)
```

> 한줄로도 가능할까요?

In [ ]:
```
import statistics  # 모듈 사용
values = [100, 75, 85, 90, 65, 95, 90, 60, 85, 50, 90, 80]
```

In [ ]:
```
statistics.pstdev(values)
```

함수(function)

특정한 기능(function)을 하는 코드의 묶음

함수를 쓰는 이유

- 높은 가독성: 짧아짐
- 재사용성:
- 유지보수: 코드의 기능별 분화

함수의 선언과 호출

- 함수 선언은 `def` 로 시작하여 `:` 으로 끝나고, 다음은 `4spaces` 들여쓰기 로 코드 블록을 만든다.
- 함수는 `매개변수(parameter)` 를 넘겨줄 수도 있다.
- 함수는 동작후에 `return` 을 통해 결과값을 전달 할 수도 있다. (`return` 값이 없으면, `None` 을 반환한다.)
- 함수는 호출을 `func()` / `func(val1, val2)` 와 같이 한다.

활용법

```
def <함수이름>(parameter1, parameter2):
    <코드 블럭>
    return value
```

[연습] 세제곱 함수

> 입력 받은 수를 세제곱하여 반환(return)하는 함수 `cube()` 을 작성해보세요.
> return이 있으니까 리턴 넣어서 코드 완성하기!

[입력 예시]

```
cube(2)
```

[출력 예시]

8

In [ ]:
```
# 아래에 코드를 작성하세요.
# 입력 받은 수 : Input이 하나 있구나!
# 함수 이름은 cube이구나!
```

In [ ]:
```
def cube(n):
    result = n ** 3
    return result
```

In [ ]:
```
cube(2) # 함수 호출
```

In [ ]:
```
# 아래 두 함수의 차이점은?
```

In [ ]:
```
def cube_return(n):
    result = n ** 3
    return result
# print문이 없기 때문에 vscode에 돌리면 출력 안됨.
# return의 이유는 함수의 내용 값을 저장하려고 , 즉, 변수에 저장하는 느낌쓰~
```

In [ ]:
```
def cube_print(n):
    result = n ** 3
    print(result)
```

In [ ]:
```
cube_return(2)   # out[6]의 의미는 개발하기 편하도록.. 마지막 줄의 내용을 출력해주는 것, 즉
```

In [ ]:
```
cube_return(2)
2 + 5
```

In [ ]:
```
cube_print(2)
```

In [ ]:
```
cube_print(2)    # 이거는 print
2 + 5            # 이거는 out으로 나와 .. 그냥 코드 결과값을 보여주는 것! 출력이 아니라
```

In [ ]:
```
name = '홍길동'
name
```

In [ ]:
```
# 함수 선언
def my_sum(a,b):
    return a+b
```

In [ ]:
```
# 함수 활용
number = my_sum(1,3)
print(number)
# 즉 리턴은 반환값을 후에 활용하고 싶을 때 쓰는것 , 변수에 저장 등등
```

In [ ]:
```
# print 함수는 None을 반환!
# print_number 변수에 None이 저장됨..
print_number = print(number)   # 얘는 number가 위에서 리턴된 값임!
print(print_number)   # 리턴값이 없으면 None을 반환함, print_number는 리턴값이 아니라서 그런
```

[실습] 사각형의 넓이를 구하는 함수

> 밑변(width)과 높이(height)를 입력받아 사각형의 넓이와 둘레를 반환(return)하는 함수 `rectangle()` 을 작성해보세요.

[입력 예시]

```
rectangle(30, 20)
```

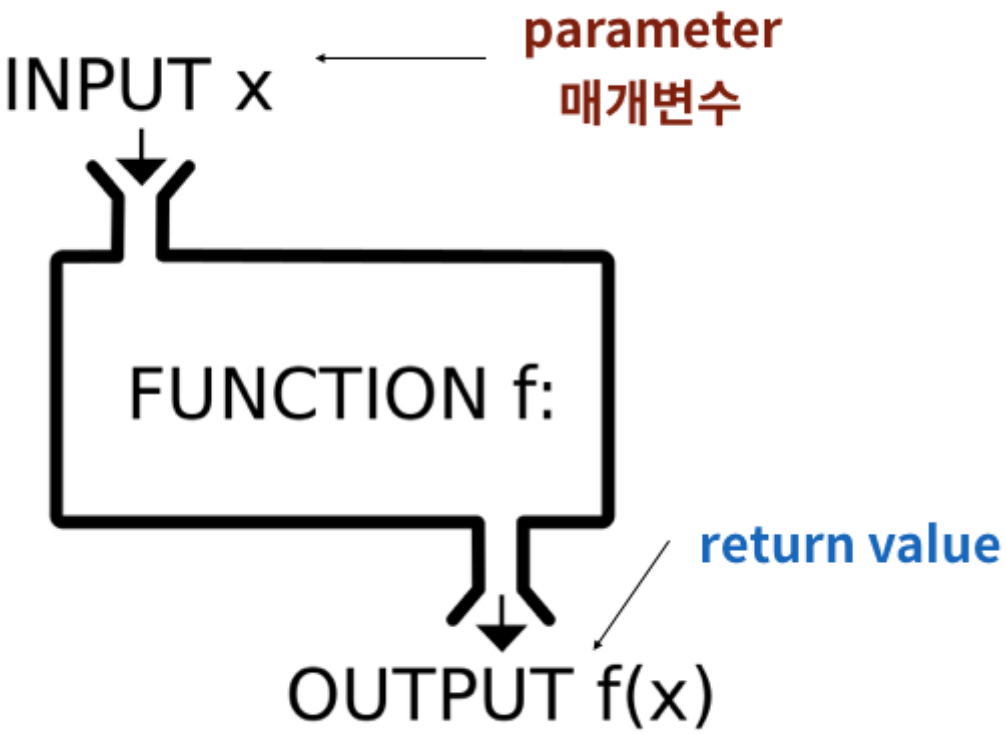
[출력 예시]

(600, 100)

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
def rectangle(w,h):
    return (w * h, (w + h) *2 )  # return을 이런식으로 할 수도 있구나!
```

In [ ]:
```
print(rectangle(30, 20))
print(rectangle(50, 70))
```



In [ ]:
```
# 우리가 활용하는 print문도 파이썬에 지정된 함수입니다.
# 아래에서 'hi'는 argument이고 출력을 하게 됩니다.
print('hi')
```

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

파이썬 문서

In [ ]:
```
# 내장함수 목록을 직접 볼 수도 있습니다.
dir(__builtins__)
```

In [ ]:
```
# 편하게 써왔던 random.sample() 함수의 내부도 확인해 볼까요?
# https://github.com/python/cpython/blob/master/Lib/random.py
```

[연습] 함수 만들기

> 아래의 코드와 동일한 `my_max` 함수를 만들어주세요.
>
> 정수를 두개 받아서, 큰 값을 반환합니다.

```
my_max(1, 5)
```

출력 예시)
5

In [ ]:
```
# 내장함수 max()를 확인해봅시다.
```

In [ ]:
```
max(1, 5)
```

In [ ]:
```
# 아래에 my_max 함수를 작성하고 호출하세요.
```

In [ ]:
```
def my_max(a,b):
    if a > b:
        return a
    else:
        return b

# 흠 둘이 같을 때는?
```

In [ ]:
```
# 해당 코드를 통해 올바른 결과가 나오는지 확인하세요.
my_max(1, 5)
```

함수의 Output

함수의 `return`

앞서 설명한 것과 마찬가지로 함수는 반환되는 값이 있으며, 이는 어떠한 종류(~~의 객체~~)라도 상관 없습니다.

단, **오직 한 개의 객체**만 반환됩니다.

함수가 return 되거나 종료되면, 함수를 호출한 곳으로 돌아갑니다.

- 아무것도 return 하지 않으면 none이 반환된다.

In [ ]:
```
def hello(name):
    print(f'hello, {name}')
```

In [ ]:
```
hello('혜인')
```

In [ ]:
```
a = hello('혜인')  # 함수에서 None이 return 되고 a에 저장..
# a 에는 그래서 none이 출력
print(a,type(a))    # return이 없으니까 none이 자동으로 반환됨. 이해완료!
```

- 프린트, 리턴 다 쓰면 어떻게 되나요?

In [ ]:
```
def hello_name(name):
    print(f'hello, {name}')
    return name
```

In [ ]:
```
a = hello_name('혜인')  # 프린트도 되고 리턴도 되는구나!
print(a,type(a))
```

- 여러개 반환되는데?

In [ ]:
```
def my_name(name):
    return 'hello', name
```

In [ ]:
```
name = my_name('홍길동')   # name에 함수 리턴값이 들어가있는 것!
```

In [ ]:
```
print(name)
```

In [ ]:
```
print(type(name))   # 사실은 튜플로 묶어서 하나로 보낸 것!!!
```

- 오직 하나 반환
- 만약, return이 없으면 => none을 반환
- 만약, 여러개를 , 로 이어서 리턴하면 => tuple로 묶어버린다.

In [ ]:
```
# sort()와 sorted()의 차이
```

In [ ]:
```
a = [5,1,3,2]
print(a)
```

In [ ]:
```
a = [5,1,3,2]
b = a.sort()    # 원본을 바꿔버리고, return이 값이 없음 ! b가 None
# 원본 리스트
print(a)
# a.sort() 결과
print(b)
```

In [ ]:
```
a = [5,1,3,2]
b = sorted(a)    # 원본은 no touch, return으로 정렬 리스트
print(a)
print(b)
```

In [ ]:
```
# 따라서 이렇게 활용하면 된다!!!!!
a = [5,1,3,2]
a.sort()
print(a)

a = [5,1,3,2]
a = sorted(a)
print(a)
```

[연습] 함수를 정의하고 값을 반환해봅시다.

> 리스트 두개를 받아 각각 더한 결과를 비교하여 값이 큰 리스트를 반환하는 함수를 만들어주세요.

```
my_list_max([10, 3], [5, 9])
```

예시 출력)
[5, 9]

In [ ]:
```
# 아래에 my_list_max 함수를 작성하고 호출하세요
# 입력에서 리스트 길이가 다른 테스트케이스들이 주어질 수 있다는 것을 알기!
```

In [ ]:
```
def my_list_max(a,b):
    # a.b리스트
    # 더한 값 준비
    sum_a = sum(a)
    sum_b = sum(b)
    result = []      # 비어있는 자료형을 미리 준비해야 하는 경우가 있음. 여기서는 특별한 의미
    if sum_a > sum_b:
        # a가 큰 상황
        result = a
```

```
    else:
        # b가 큰 상황
        result = b
    return result
```

In [ ]:
```
# 해당 코드를 통해 올바른 결과가 나오는지 확인하세요.
my_list_max([10, 3], [5, 9])
```

In [ ]:
```
def my_list_max(a,b):
    if sum(a) > sum(b):
        return a
    else:
        return b
```

In [ ]:
```
my_list_max([10, 3], [5, 9])
```

함수의 입력(Input)

매개변수(parameter) & 인자(argument)

(1) 매개변수(parameter)_ 함수 정의

```
def func(x):
      return x + 2
```

- x 는 매개변수(parameter)
- 입력을 받아 함수 내부에서 활용할 변수 라고 생각하면 된다.
- 함수의 정의 부분에서 볼 수 있다.

(2) 전달인자(argument)_ 함수 호출

```
func(2)
```

- 2 는 (전달)인자(argument)
- 실제로 전달되는 입력값 이라고 생각하면 된다.
- 함수를 호출하는 부분에서 볼 수 있다.

> 주로 혼용해서 사용하지만 엄밀하게 따지면 둘은 다르게 구분되어 사용됩니다. 개념적 구분보다 함수가 작동하는 원리를 이해하는게 더 중요합니다.

함수의 인자

함수는 입력값(input)으로 인자(argument) 를 넘겨줄 수 있습니다.

위치 인자 (Positional Arguments)

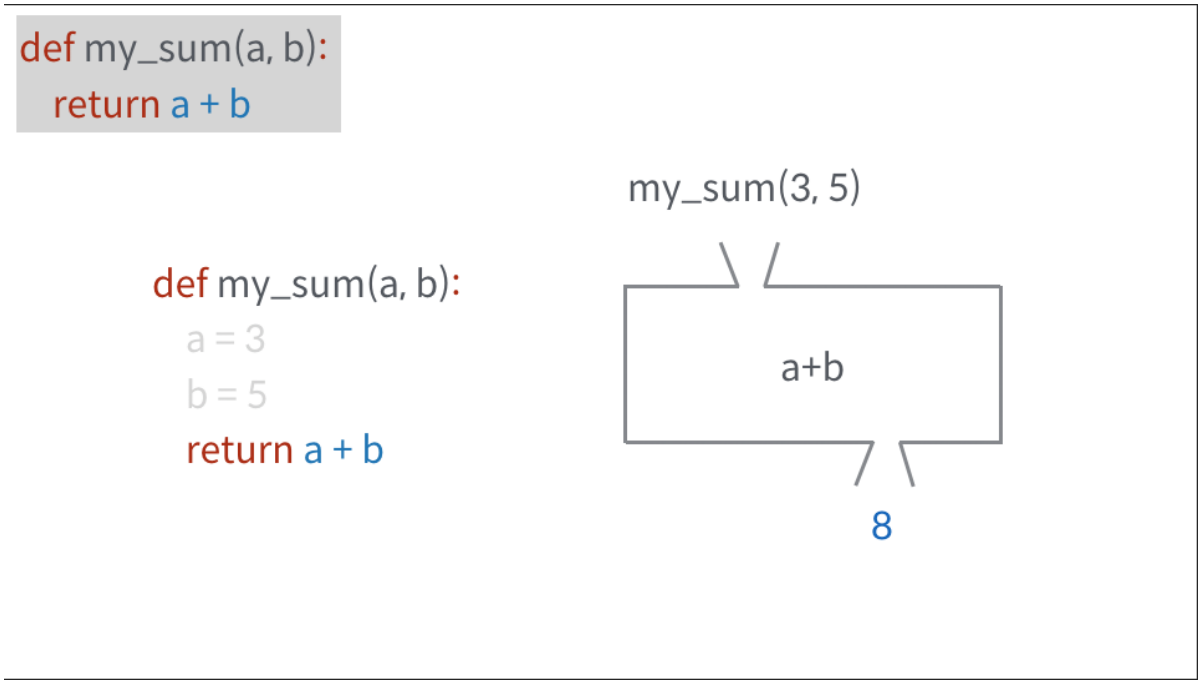
함수는 기본적으로 인자를 위치로 판단합니다.
즉, 순서대로 맵핑해서 쓴다!

[연습] 원기둥의 부피

> 원기둥의 반지름(r)과 높이(h)를 받아서 부피를 return하는 함수 `cylinder()` 를 작성하세요.
>
> 원기둥 부피 = 밑면의 넓이 높이*

In [ ]:
```
def cylinder(r,h):
    return r ** 2 * 3.14 * h
```

In [ ]:
```
print(cylinder(5,2))
print(cylinder(2,5)) # 순서를 바꾸면 다른 값이 나옵니다.
```



기본 인자 값 (Default Argument Values)

함수가 호출될 때, 인자를 지정하지 않아도 기본 값을 설정할 수 있습니다.

활용법

```
def func(p1=v1):
    return p1
```

[연습] 기본 인자 값 활용

> 이름을 받아서 다음과 같이 인사하는 함수 `greeting()` 을 작성하세요. 이름이 길동이면, "길동, 안녕?" 이름이 없으면 "익명, 안녕?" 으로 출력하세요. => return하라는 말이 없으므로 반환 넣지 않는다! 문제 잘보기

In [ ]:
```
# 아래에 greeting 함수를 작성하세요.
```

In [ ]:
```
def greeting(name):
    print(f'{name}, 안녕?')
```

In [ ]:
```
greeting('길동')
```

In [ ]:
```
greeting()    # 오류가 나온다.
```

따라서 아래와 같이 하면 입력값이 없을 때 출력되는 것을 지정할 수 있다.

In [ ]:
```
def greeting2(name='익명'):
    print(f'{name}, 안녕?')
```

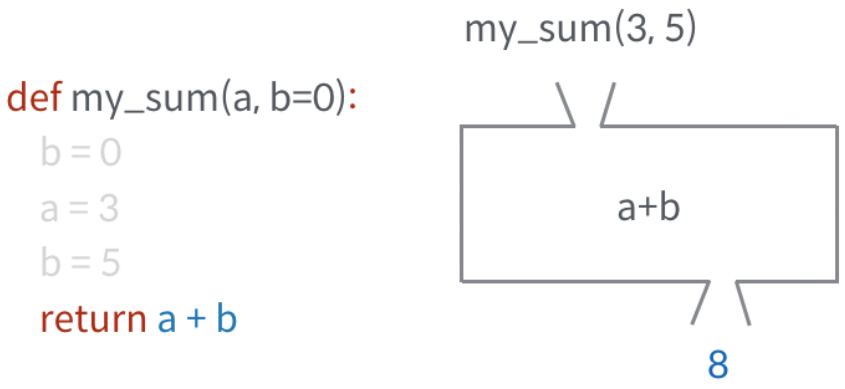
In [ ]:
```
greeting2('길동')
```

In [ ]:
```
greeting2()
```

In [ ]:
```
# 즉 위치인자가 가장 먼저
# 그러나 그 함수()이런식으로 할 때 안에꺼는 순서 관계없다
```

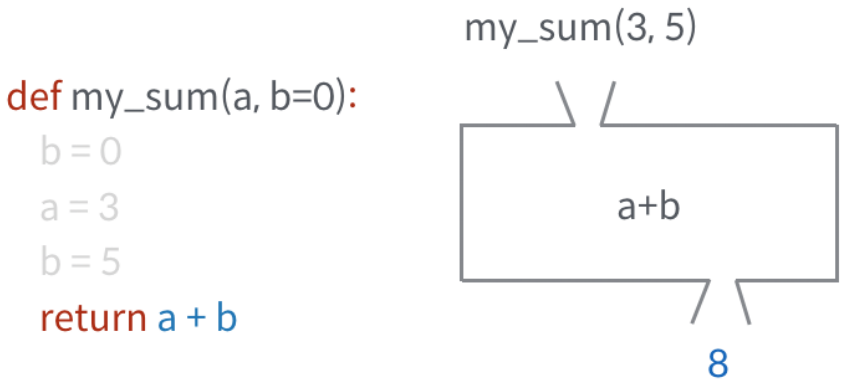
- 기본 인자 값이 설정되어 있더라도 기존의 함수와 동일하게 호출 가능하다.

```
def my_sum(a, b=0):
  return a + b
```



- 기본 인자 값이 설정되어 있더라도 기존의 함수와 동일하게 호출 가능하다.

```
def my_sum(a, b=0):
  return a + b
```



In [ ]:
```
# 아래 세가지 예시를 보자.
```

In [ ]:
```
def my_sum(a,b):
    return a + b
my_sum(3,5)
```

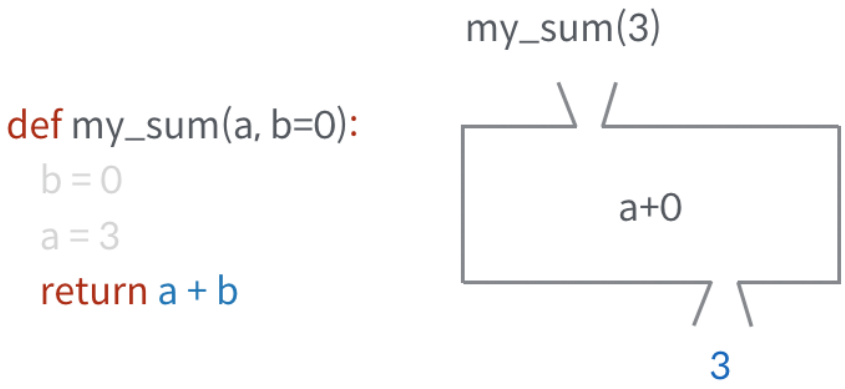
In [ ]:
```
def my_sum(a,b=0):
    return a + b
my_sum(3,5)
```

In [ ]:
```
def my_sum(a,b=0):
    return a + b
my_sum(3)
```

In [ ]:
```
def my_sum(a=0,b):    # 하지만 이거는 안돼다. 아래 예시 같이 보기
    return a + b
my_sum(5)
```

- 호출시 인자가 없으면 기본 인자 값이 활용된다.

```
def my_sum(a, b=0):
  return a + b
```



***주의* 단, 기본 인자값(Default Argument Value)을 가지는 인자 다음에 기본 값이 없는 인자를 사용할 수는 없습니다.**

In [ ]:
```
# 오류를 확인해봅시다.
```

In [ ]:
```
# SyntaxError: non-default argument follows default argument
# 기본 인자 뒤에 기본이 아닌 인자가 따라왔어
def greeting(name = '익명', age):
    print(f'안녕? 난 {name}, {age}살이야.')
```

In [ ]:
```
# 수정해 봅시다.
```

In [ ]:
```
def greeting(name, age='익명'):
    print(f'안녕? 난 {name}, {age}살이야.')
greeting(100)
greeting(1000,'단군')
```

키워드 인자 (Keyword Arguments)

키워드 인자는 직접 변수의 이름으로 특정 인자를 전달할 수 있습니다.

정리

기본인자값을 가지는 인자 다음에 기본 값이 없는 인자는 올 수 없음. 무조건 기본 인자값을 가지는 인자는 마지막에

키워드인자는 함수호출시 사용, 직접 변수의 이름으로 특정 인자 전달 가능 키워드 인자 다음에 위치 인자 불가 / 그냥 특정인자 전달하는거는 다음에 특정인자 없는거 못온다는 것 /

In [ ]:
```
# 정리
def greeting(age, name = 'john'):
    return f'{name}은 {age}살입니다.'
#가능
greeting(name = '철수', age = 24)
```

```
#가능
greeting(24)
 #불가 #키워드 인자 다음에 위치 인자 불가
greeting(age=24, '철수')
```

In [ ]:
```
# 키워드 인자 예시
```

In [ ]:
```
def greeting(age, name='익명'):
    print(f'안녕? 난 {name}, {age}살이야.')
```

In [ ]:
```
# 순서 바꾸기 가능하다
greeting(name='길동',age = 1000)
```

- **단 아래와 같이 키워드 인자 를 활용한 다음에 위치 인자 를 활용할 수는 없습니다.**

In [ ]:
```
# SyntaxError: positional argument follows keyword argument
# 위치인자가 키워드 인자 뒤에 옴
greeting(age = 3000, '곰')
```

In [ ]:
```
# 키워드 인자 예시
def greeting(age, name = 'john'):
    return f'{name}은 {age}살입니다.'
```

In [ ]:
```
greeting(name='철수', age=24)
```

In [ ]:
```
greeting(24)
```

In [ ]:
```
greeting(age = 24, '철수')
# 이 경우는 불가 # 키워드 인자 다음에 위치 인자 불가
```

정해지지 않은 여러 개의 인자 처리

우리가 주로 활용하는 `print()` 함수는 파이썬 표준 라이브러리의 내장함수 중 하나이며, 다음과 같이 구성되어있습니다.

> **print**(**objects*, *sep=' '*, *end='\n'*, *file=sys.stdout*, *flush=False*)
>
> *objects* 를 텍스트 스트림 *file* 로 인쇄하는데, *sep* 로 구분되고 *end* 를 뒤에 붙입니다. 있다면, *sep*, *end*, *file* 및 *flush* 는 반드시 키워드 인자로 제공해야 합니다.
>
> 모든 비 키워드 인자는 `str()` 이 하듯이 문자열로 변환된 후 스트림에 쓰이는데, *sep* 로 구분되고 *end* 를 뒤에 붙입니다. *sep* 과 *end* 는 모두 문자열이어야 합니다; `None` 일 수도 있는데, 기본값을 사용한다는 뜻입니다. *objects* 가 주어지지 않으면 `print()` 는 *end* 만 씁니다.
>
> *file* 인자는 `write(string)` 메서드를 가진 객체여야 합니다; 존재하지 않거나 `None` 이면, `sys.stdout` 이 사용됩니다. 인쇄된 인자는 텍스트 문자열로 변환되기 때문에, `print()` 는 바이너리 모드 파일 객체와 함께 사용할 수 없습니다. 이를 위해서는, 대신 `file.write(...)` 를 사용합니다.
>
> 출력의 버퍼링 여부는 일반적으로 *file* 에 의해 결정되지만, *flush* 키워드 인자가 참이면 스트림이 강제로 플러시 됩니다.
>
> *버전 3.3으로 변경:* *flush* 키워드 인자가 추가되었습니다.

In [ ]:
```
print('hi','hello' ,'guten tag')
```

In [ ]:
```
print('xx')
print('vv', end = '_')
print('bb','vv')
print('xd','ef','dfsf',sep='/',end='끝!')
```

In [ ]:
```
print('xx','ee','43',sep =',')
```

가변(임의) 인자 리스트(Arbitrary Argument Lists)

앞서 설명한 `print()` 처럼 개수가 정해지지 않은 임의의 인자를 받기 위해서는 가변 인자 리스트 `*args` 를 활용합니다.

가변 인자 리스트는 `tuple` 형태로 처리가 되며, 매개변수에 `*` 로 표현합니다.

활용법

```
def func(a, b, *args):
```

> `*args` : 임의의 개수의 위치인자를 받음을 의미
>
> 보통, 이 가변 인자 리스트는 매개변수 목록의 마지막에 옵니다.

In [ ]:
```
# 기본값
# 가변인자 f(3,5)   : 투플로 관리
# 가변 키워드 인자 f(a=3, b=5)  : 키워드 밸류 이렇게 관리

# 가변인자
def func(a, b, *args):
# *args는 임이의 개수의 위치 인자, 매개변수 목록의 가장 마지막
def students(*args,prof):
    for student in args:
        print(student)
    print(f'존경하는 교수님 {prof}')

students('희은','태영',prof = '탁희')   # 가능

def students(prof,*args):
    for student in args:
        print(student)
    print(f'존경하는 교수님 {prof}')

students('교수','태영','희은')   # 가능

# 가변 키워드 인자
def func(**kwargs):      # **kwargs   :  임의의 개수의 키워드 인자를 받음을 의미
```

In [ ]:
```
# 가변 인자 예시
# print문은 *obejcts를 통해 임의의 숫자의 인자를 모두 처리합니다.
```

In [ ]:
```
print('hi','hello')
```

In [ ]:
```
# args는 tuple입니다.
```

In [ ]:
```
def students(*args):   # 몇 개를 받을지 모를 때는 별을 붙여서 해주면 된다.
    print(args)
    print(type(args))
```

In [ ]:
```
students('희은','대영','태성')    # 여러개 묶어서 보내면 튜플로 묶어서 리턴
```

In [ ]:
```
# 전달받은 모든 학생을 출력하고 싶다..
def students(*args):
    for student in args:
        print(student)
```

In [ ]:
```
students('희은','대영','태성')
print('---')
students('희은','대영','태성','상진','국현')
```

In [ ]:
```
def students(*args,prof):
    for student in args:
        print(student)
    print(f'존경하는 교수님 {prof}')
```

In [ ]:
```
# 마지막이 prof...?? 자동으로 되지 않는다!
students('희은','태영','탁희')
```

In [ ]:
```
# 가변 이후의 변수는 직접 키워드 인자로 활용한다!
students('희은','태영',prof = '탁희')
```

In [ ]:
```
def students(prof,*args):
    for student in args:
        print(student)
    print(f'존경하는 교수님 {prof}')
```

In [ ]:
```
students('교수','태영','희은')   # 앞에 하나는 확실하니까 가능하다!
```

[연습] 가변 인자 리스트를 사용해봅시다.

> 정수를 여러 개 받아서 가장 큰 값을 반환(return)하는 함수 `my_max()` 를 작성하세요.

```
my_max(10, 20, 30, 50)
```

예시출력)
50

In [ ]:
```
max(1, 2, 3, 4)
# 아래에 코드를 작성하세요.
```

In [ ]:
```
def my_max(*args):
    result = 0
```

```
    for idx,val in enumerate(args):
        if idx == 0:
            result = val
        else:
            if val > result:
                result = val
    return result
```

In [ ]:
```
# 해당 코드를 통해 올바른 결과가 나오는지 확인하세요.
my_max(-1, -2, -3, -4)
```

가변(임의) 키워드 인자(Arbitrary Keyword Arguments)

정해지지 않은 키워드 인자들은 **dict** 형태로 처리가 되며, ** 로 표현합니다.

보통 kwagrs 라는 이름을 사용하며, **kwargs 를 통해 인자를 받아 처리할 수 있습니다.

활용법

```
def func(**kwargs):
```

> **kwargs : 임의의 개수의 키워드 인자를 받음을 의미

우리가 dictionary를 만들 때 사용할 수 있는 dict() 함수는 파이썬 표준 라이브러리의 내장함수 중 하나이며, 다음과 같이 구성되어 있습니다.

> *class* **dict**(***kwarg*)
> *class* **dict**(*mapping*, ***kwarg*)
> *class* **dict**(*iterable*, ***kwarg*)
> 새 딕셔너리를 만듭니다. dict 객체는 딕셔너리 클래스입니다. 이 클래스에 대한 도큐멘테이션은 dict 및 매핑 형 --- dict 을 보세요.

In [ ]:
```
# 딕셔너리 생성 함수 예시(가변 키워드 인자)
```

In [ ]:
```
dict(name = '홍길동', age = '1000')
```

In [ ]:
```
# 주의사항
# 식별자는 숫자만으로는 이루어질 수가 없다.(키워드인자로 넘기면 함수 안에서 식별자로 쓰이기
dict(1='1',2='2')   # 불가
```

In [ ]:
```
# 위의 경우 다음과 같이 사용해야 한다.
dict(((1,1),(2,1)))
```

In [ ]:
```
# 아래의 코드를 실행시켜 내부 구조를 살펴봅시다.
def my_dict(**kwargs):
    print(kwargs)
    print(type(kwargs))
    return kwargs

my_dict(한국어='안녕', 영어='hi', 독일어='Guten Tag')
```

In [ ]:
```
my_dict('안녕','hi')    # 불가
```

In [ ]:

[실습] URL 생성기

> my_url() 함수를 만들어 완성된 URL을 반환하는 함수를 작성하세요.

```
my_url(sidoname='서울', key='asdf')
```

예시 출력)
https://api.go.kr?sidoname=서울&key=asdf&

In [ ]:
```
# 입력받은 가변 키워드 인자를 활용하여 'https://api.go.kr?'를 BASE_URL로한 URL을 생성하시오.
```

In [ ]:
```
def my_url(**kwargs):
    url = 'https://api.go.kr?'
    # kwargs : dictionary
    # kwargs.items() : dict_items([('sidoname', '서울'), ('key', 'asdf')])
    print(kwargs.items())
    for name, value in kwargs.items():
        url += f'{name}={value}&'
    return url
```

In [ ]:
```
my_url(sidoname='서울', key='asdf')
```

In [ ]:
```
# 함수 정의는 아마도.. **kwargs
dict(a='apple', b='banana')
```

In [ ]:
```
# 함수 정의는 아마도.. *objects
print('안녕','하세요')
```

In [ ]:
```
def my_sum(a,b):
    return a + b
```

In [ ]:
```
my_sum(1,3)
```

In [ ]:
```
def students(*args):
    print(args)
```

In [ ]:
```
students('대영','태현')
```

In [ ]:
```
def students(*student_list):
    print(student_list)
```

In [ ]:
```
students('대영', '태현')
```