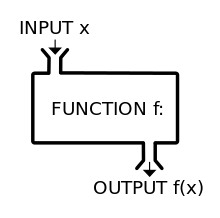
Week 2

Function(함수)



위의 그림처럼 input(입력)을 넣으면 output(출력)이 출력되는 것을 말한다.

파이썬에는 파이썬을 깔면 있는 함수들이 많다. 파이썬 자체에 있는 함수를 내장 함수라 하고

파이썬을 깔면 생기는 파이썬 라이브러리에 있는 모듈을 불러와 쓰는 외장 함수가 있다.

typecast

말 그대로 자료의 type을 바꿔주는 함수이다.

기본적으로 자료형을 확인하는 함수는 type이다.

type(object) : 입력값 객체의 자료형이 무엇인지 알려주는 함수

자료형에는 많은 type이있다. 정리를 해보자

int() : 정수형으로 변환

float() : 실수형으로 변환

complex() : 복소수형으로 변환

str() : 문자형으로 변환

위와 같이 있는 것이 기본이다. 다른 유형도 있는데 한번 나열해보면

chr(i) : 아스키(ASCII)  코드값(i)을 입력으로 받고 그 코드에 해당하는 **문자**를 리턴하는 함수 -> 출력값의 type은 str이다

ord(c) : 문자(c)의 아스키 코드값을 리턴 -> 리턴값의 type은 int이다

oct(x) : 정수 형태의 숫자(x)  를 8진수 문자열로 리턴 -> 리턴값의 type은 str

hex(x) : 정수 형태의 숫자(x)  를 16진수 문자열로 리턴 -> 리턴값의 type은 str

반복 가능한 자료형(리스트, 튜플, 집합, 문자열 등)의 타입을 바꿔주는 함수도 있는데

list(s) : 반복 가능한 자료형(s)을 입력받아 리스트로 리턴 -> 리턴값의 type은 list

tuple(s) : 반복 가능한 자료형(s)을 입력받아 튜플로 리턴 -> 리턴값의 type은 tuple

set(s) : 반복 가능한 자료형을 입력받아 집합 자료형으로 리턴 -> 리턴값의 type은 set

더 많은 내장함수가 있는데 수업 진도를 나가면서 관련 함수를 쓰는게 더 나을 것 같다.

Week 2에 나온 함수도 살펴보자

abs(x) : 어떤 숫자를 입력으로 받으면 해당 숫자의 절대값을 리턴해준다

min(반복 가능한 자료형) : 반복 가능한 자료형을 입력 받으면 그 중 최소값을 리턴

max(반복 가능한 자료형): 반복 가능한 자료형을 입력 받으면 그 중 최대값을 리턴

round(x, y) :  x의 가장 가까운 숫자를 y번째 자리까지 표시해서 리턴 (y를 입력하지 않으면 정수형이 나온다, 반올림이 적용된다.)

help(함수명) :  함수명을 입력하면 해당 함수에 대한 사용법(?) 이 나온다.

id() : 객체의 주소값을 리턴해준다

def 에 대해 알아보자 (이 글에서 할당은 assignment를 말한다.)

def는 함수를 만들 때 사용하는 예약어이다.

밑에 말한 기본적으로 설정되어있는 함수 말고 코딩을 할 때 필요한 함수를 내가 스스로 생성할 수 있다는 것이다.

기본적인 구조는

 >>>def 함수명(입력인수) :

            수행할 문장 1

            수행할 문장 2

            ...

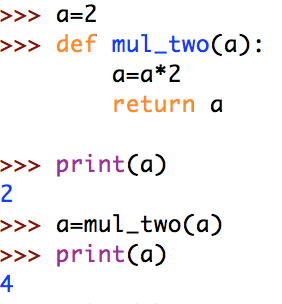
            return 출력값

이다.

그리고 local variable(지역변수) 에 대해서 얘기하는데  함수 안에서 변수는 함수 안에서만 적용되고 함수 밖에서는 적용이 안된다 이런 원리인데

즉 함수 안에서 a라는 변수랑  함수 밖에서 a랑은 다르다는 것이다

물론 함수 안에서 함수 밖의 변수를 변경하는 방법이 있다.



 예시에서 보는 것 처럼 함수 밖에서 변수 a를 선언하고 def 함수에도 a를 사용했다. def 함수를 만들고 a를 확인해보면 함수 밖의 변수가 프린트되어지는 것을 확인할 수 있고

def 함수에서 리턴값을 a라고 했으니 a(함수 밖의 a)=mul\_two(a)=a(함수 안의 a)가 되어 a=mul\_two(a)를 입력한 후 다시 a를 보면 값이 바뀌어져있는걸 확인할 수 있다.

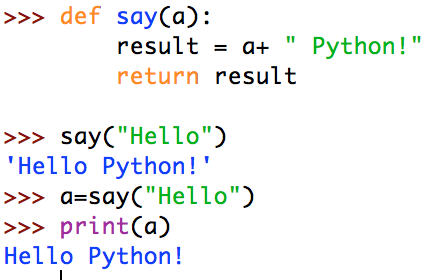
이렇게 해도 아직 mul\_two함수 안의 a는 함수 밖의 a와는 다르다

global이라는 명령어를 이용하는 방법도 있는데

함수 안에서 global a라고 입력하면 a는 전역변수(?)가 되어 함수 밖에서 a와 함수 안에서 a가 같아지게 된다. 함수가 외부 변수의 영향을 받으면 너무 변수 이름을 작명할 때 복잡해지므로 안 좋은 코딩 방법이 될 수 있다고 하신 것 같다.

함수의 일반적 형태는 입력 인수와 결과값이 모두 있는 형태인데 입력값이 없거나 결과값이 없거나 둘 다 없는 경우의 함수도 있다

* 입력값, 결과값이 모두 있는 함수(Normal Function)



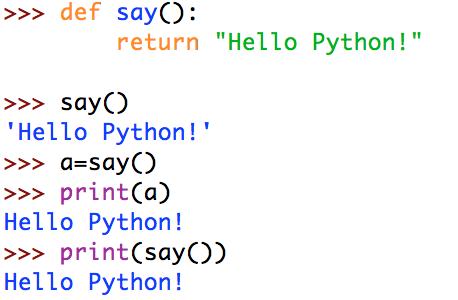
첫 문장부터 살펴보면 def로 say라는 함수를 선언하고 매개변수로 a를 지정했다.  그래서 a에 할당한 인수와  " Python!" 을 합친 결과값을 리턴 하는 함수를 만든거다.

그리고 say("Hello")를 입력하면 "Hello Python!" 이라는 결과값을 보여준다

그 다음 문장은 이 결과값을 함수 밖 a라는 변수에 할당한 것이고 다음 문장은 그 결과값을 프린트 한 것이다

결국 print(a)는 print("Hello Python!")와 똑같은 결과를 보여준다.

* 입력값이 없고 결과값은 있는 함수(Function with no parameter)



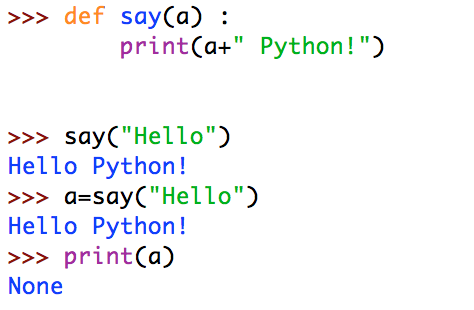
첫 문장부터 살펴보면 def로 say라는 함수를 매개변수 없이 선언했다.  그리고 say함수는  "Hello  Python!" 을  결과값으로 리턴 하는 함수다.

그리고 say()를 입력하면 "Hello Python!" 이라는 결과값(리턴값)을 보여준다

그 다음 문장은 이 결과값을  a라는 변수에 할당한 것이고 다음 문장은 그 결과값을 프린트 한것이다

결국 print(a)는 print("Hello Python!"), print(say()(결과값))와 똑같은 결과를 보여준다.

* 입력값은 있고 결과값은 없는 함수(Function with no return value)



첫 문장부터 살펴보면 def로 say라는 함수를 선언하고 매개변수로 a를 지정했다.  say(a)함수는 a에 입력한 인수와  " Python!" 을 합친 문자열을 프린트해주는 함수이다.

그리고 say("Hello")를 입력하면 Hello Python! 이라는 프린트 되어진 문자열을 보여준다.

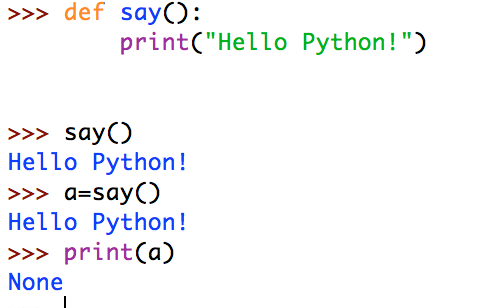
그 다음 문장은 이 결과값이 없는 것(?)을 함수 밖 a라는 변수에 할당한 것이고 그리고 결과값이 없으므로 say(a)라는 함수가 실행되어 Hello Python!이 프린트 되어져 보여진다.

 다음 문장은 그 결과값이 없는 것을 프린트 한 것이다

say(a)함수에는 결과값이 없으므로 print(a)를 하면 None 이라는 False(거짓)을 나타내는 자료형이 나온다.

참고로 None은 결과값이 아니다.

* 입력값, 결과값 모두 없는 함수(Function with no parameter and no return value)



첫 문장부터 살펴보면 def로 say라는 함수를 입력인수 없이 선언했다.  say()함수는 say함수는  "Hello  Python!" 라는 문자열을 프린트 해준다.

그리고 say()를 입력하면 Hello Python! 이라는 프린트 되어진 문자열을 보여준다.

그 다음 문장 부터는 위에 입력값은 있고 결과값이 없는 함수와 상황이 비슷하다. 결과값이 없는 것(?)을 a라는 변수에 할당한 것이고 그리고 결과값이 없으므로

say()라는 함수가 실행되어 Hello Python!이 프린트 되어져 보여진다.

 다음 문장은 그 결과값이 없는 것을 프린트 한 것이다

say(a)함수에는 결과값이 없으므로 print(a)를 하면 None 이라는 False(거짓)을 나타내는 자료형이 나온다.

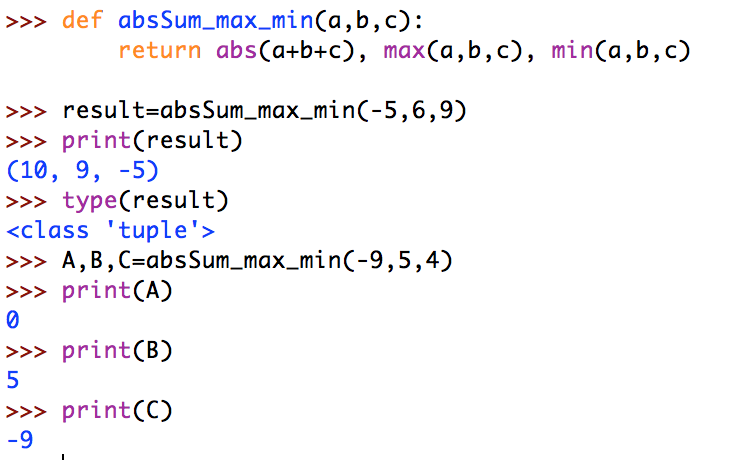
참고로 None은 결과값이 아니다.

헷갈릴 부분은 프린트 되어진 것을 결과값이라고 착각하는 것이다. **프린트 되어진 것과 리턴값은 다른 것**이다.

리턴값의 특징에 대해 생각해보면

1. 함수의 리턴값은 언제나 하나이다.
2. 함수가 return을 만나면 그 함수의 실행이 끝난다.(함수를 빠져나간다.)

1. 특징



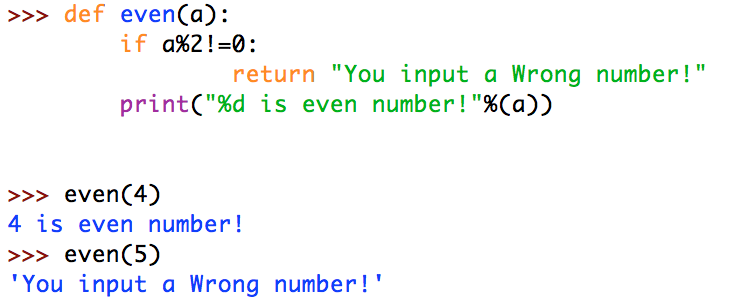
absSum\_max\_min이라는 함수를 선언하고 입력인수는 세 개 a,b,c를 받는다. 리턴값은 a,b,c의 합의 절대값, a,b,c의 최대값, a,b,c의 최소값으로 설정했다

그리고 result라는 변수에 리턴값을 할당하고 프린트 해서 보니 (10, 9, -5)가 나온다 리턴값의 자료형은 확인해보니 튜플형이다

이렇듯 리턴값이 (10, 9, -5)라는 **하나**의 튜플이 나온 것이다

그 다음 한 것은 A, B, C라는 변수에 리턴값으로 나올 세 개의 튜플 요소를 할당한 것이다 그래서 프린트 해서 보면 튜플 요소가 순서대로 할당 된 것을 볼 수 있다.

2. 특징



여기서 나온 함수의 조건문이나 문자열 포매팅은 후에 배우는 것 같으므로 나중에 해야겠다.

보면 even(a)라는 함수는 a가 2로 나누었을때 0이아니면 리턴값을 주는 형태이다. 그리고 프린트가 또 써있는데 확인해보면

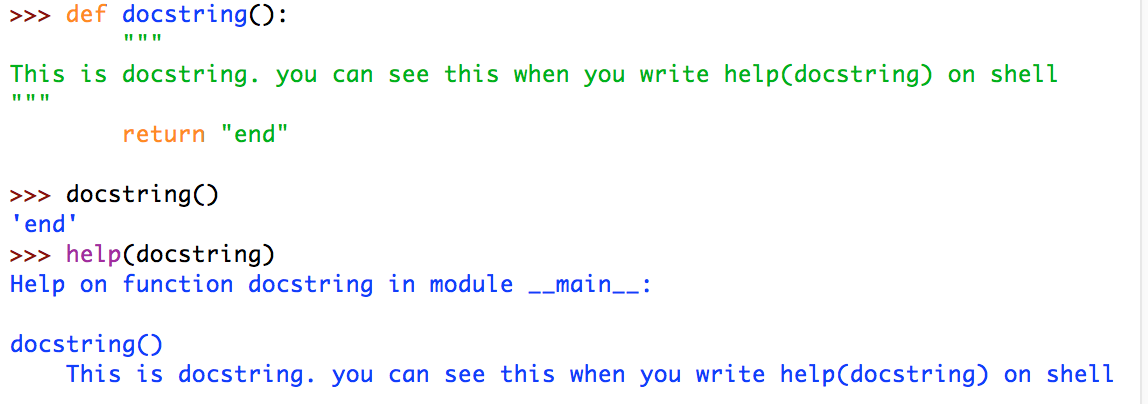
even(4)는 프린트한 4 is even number! 문장이나오고 even(5)는 "You input a Wrong number!" 리턴값이 나온다 .

입력인수를 5를 받을 때는 함수가 if조건문을 만족하고 리턴값을 낸다. 그리고 함수가 끝나서 밑에 쓰여진 프린트된 문장이 실행이 안되는 것이다.

차이점이 even(4)는 위에서 말한 입력값은 있고 결과값이 없는 함수가 된 것이고 even(5)는 입력값, 결과값 모두 있는 함수가 된 것이다.

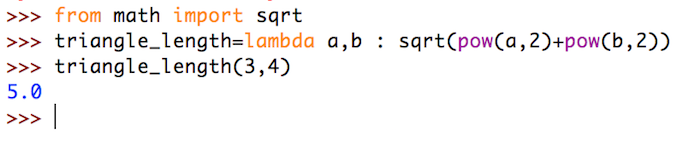
날짜, 주일 , 생일 계산 하는 것을 배우기전에 docstring이라는 것을 배우는데 간단히 말해서 설명문 같은 것이다.

남이나 내가 코딩을 하고 함수를 만들었는데 이 함수가 뭘 하는지 정확하게 기억하고, 또 남이 봤을 때 이 함수가 이렇게 사용하는 것이구나 라고 알 수있게 설명하는 것이다

이렇게 """를 쓰면 그 중간에 쓰는 것이 docstring이 되고 끝날때는 다시 """으로 닫아준다 쓴 docstring을 볼려면 함수를 선언하고 해당 함수의 이름을 help에다 넣으면 볼 수 있다

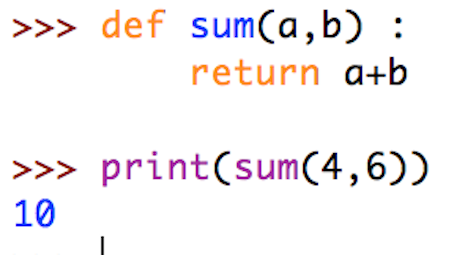
함수 관련해서 함수 생성 할 때 def 와 똑같은 기능을 하는 것이 있다 lambda인데 한 두줄로 완성되는 함수라면 굳이 def말고 람다를 쓰는것이 더 편한다. 또한 def를 쓸 수 없는 곳에도 주로 쓰인다.

예를들어 직각삼각형의 빗변의 길이를 구하는 함수를 만든다고 했을 때 def말고 람다를 써보면



lambda는 def보다 간결하게 사용할 수 있고 리스트 안에 람다를 사용한다든지 def 가 들어갈 수 없는 곳에서 함수를 만들어 사용할 수 있어서 굉장히 유용하다.

* Argument(인자) 매개변수(parameter)의 차이



여기서 매개변수(parameter)는 a와 b이고 인자(Argument)는 4와 6이다.

즉 인자(Argument) 는 어떤 함수를 호출할 때 매개변수에 전달되어지는 값이고

매개변수(parameter)는 그 인자가 assign되어지는 변수이다

* 파이썬에서 그럼 인자에 대해 자세히 알아보자

파이썬에서는 인자를 세 종류로 나눈다.

1.기본인자(Positional argument) 2. 디폴트(기본값) 인자(default argument) (1과 2는 같은데 인자에 디폴트값을 매긴 것도 일단 다르게 분류한다.)

3. 가변형 인자 (Arbitrary argument) 4. 키워드 인자(Keyword argument)

* def function(arg, darg=X, \*args, \*kwargs) :

인자 나열순서 : 기본인자 > 디폴트 인자 > 가변형 인자 > 키워드 인자

인자 우선배정 순서 : 기본인자 > 디폴트 인자 > 가변형 인자 > 키워드 인자

* Positional argument(arg)(기본인자)

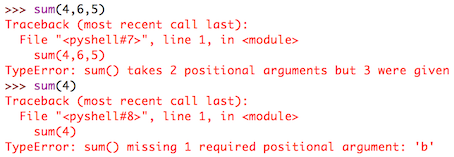
 기본인자 우리가 지금 수업을 들을 때, 그리고 가장 많이 쓴다. 위의 예제에서 봤듯이  sum(4,6)  처럼 a=4, b=6이되어 사용되어진다.

함수를 호출하고 인자를 전달했을 때 디폴트값이 없이 단독으로 쓰인 매개변수에 가장 먼저 인수를 배정한다.

또한 함수에 인자를 넣을 때 기본인자(positional argument)(디폴트값 없이 단독으로 쓰인) 의 개수만큼 넣어줘야 한다.

그 이하를 넣거나 이상을 넣으면 TypeError 가 난다.

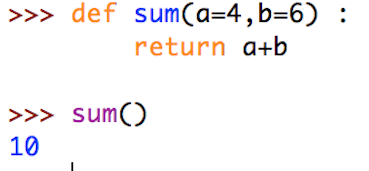
위에 만들었던 sum으로 실험을 해보자



* Default argument(darg=X)(디폴트 인자)

말만 다르게 한거지 기본인자에 디폴트(기본값)를 넣어준 것이다. 수업시간에 안 다루는 내용이라 생소할 수있지만 알아두면 나쁠 건 없다고 생각해 공부해보았다.

디폴트 값을 이용해서 sum코드를 써보면

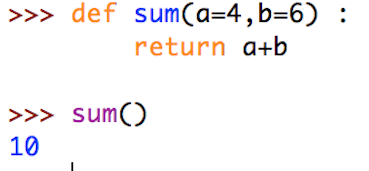


보시다 싶이 함수를 정의 할때 매개변수 a,b 에다 초기값을 설정한 것이다

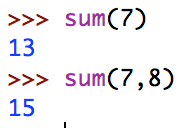
그리고 입력인자 없이 그냥 sum()  을 쓰면 디폴트값으로 지정해둔 4와 6이 더해져 10이라는 결과값을 리턴한다.

이렇게 디폴트 인자를 쓸 때 몇가지 특징, 주의할 점이 있다

1. 인자를 쓰지않으면 디폴트값을 사용한다.



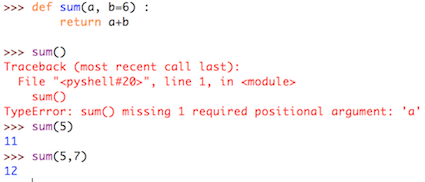
2.  인자 값을 쓰고 함수를 부르면 맨 처음 배정된 매개변수부터(예시에선 a부터 b  순으로) 맨 나중에 배정된 매개변수 순으로 인자값을 전달한다.



3.인자를 쓸 때 인자 나열순서를 지켜야 하므로 디폴트 인자 뒤에는 기본인자가 올 수 없다.

https://khlug.org/image/1025ac37a4e495ba.png

그럼 하나는 기본인자 하나는 디폴트 인자로 설정해보자

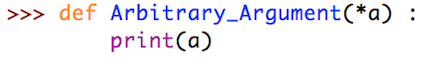


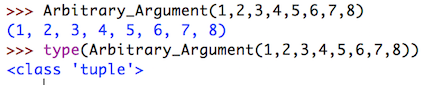
* Arbitrary argument(\*args)(가변형 인자)

매개변수앞에 \*을 붙이면 가변형 인자를 받을 수 있다.

즉 입력인자가 몇 개이든 상관없이 받을 수 있다.

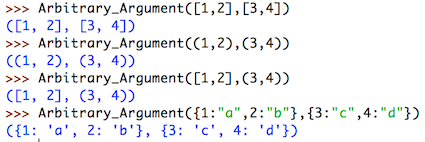
그리고 그 입력인자들을 모아서 튜플 형태로 만들어준다.





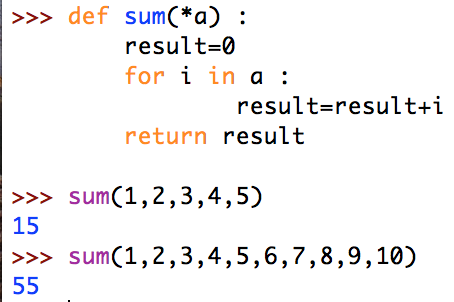
 a는 변수명이므로 \*args나 \*Python처럼 막 써도 된다.

입력인자를 튜플로 정리하므로 입력인자에 리스트나 튜플,  딕셔너리를 넣어도 되는지  실험해보자



결과적으로 리스트, 튜플, 딕셔너리 모두 포함 가능하고 섞어서 해도 모두 튜플로 다 받아준다.

가장 많이 알려져있는 여러개의 값 모두 더하는 함수를  만들어 보자



* Keyword argument(\*\*kwargs)(키워드 인자)

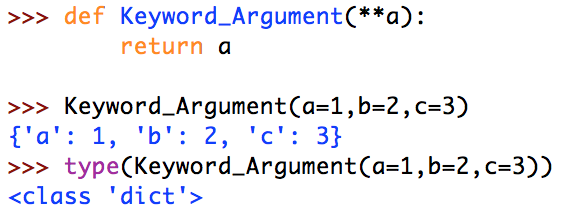
매개변수 앞에 \*\*을 붙이면 키워드 인자를 받을 수 있다.

입력인자들을 모아서 딕셔너리 형태로 만들어준다.

이것도 마찬가지로 입력인자가 몇개이든 상관이 없는데 중요한 건 딕셔너리 형태이므로 key=value 형태로 인자를 넣어야 한다.

인자값을 입력하는 순서는 상관이 없다 . 어차피 해당 key에 맞는 value값을 넣어주는 형태이므로 인자 순서는 상관 없다.

일단 keyword argument로 코드를 만들어보자



입력한 방법은 key=value 값으로 입력을 했고

나온 결과는 딕셔너리 자료형이라는 것을 알 수 있다.

 딕셔너리를 공부하면 알겠지만 딕셔너리는 순서를 따지지 않는다. 추가되는 순서나 이런 것은 규칙이 없다

그래서 위에 언급한 듯이 "인자값을 입력하는 순서는 상관이 없다 . 어차피 해당 key에 맞는 value값을 넣어주는 형태이므로 인자 순서는 상관 없다." 라고 정리를 했다.

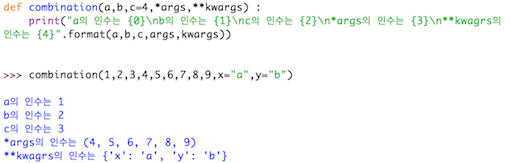
이렇듯 이 네개의 모든 인자를 섞어서 쓰려면 위에서 언급한

* def function(arg, darg=X, \*args, \*kwargs) :

인자 나열순서 : 기본인자 > 디폴트 인자 > 가변형 인자 > 키워드 인자

인자 우선배정 순서 : 기본인자 > 디폴트 인자 > 가변형 인자 > 키워드 인자

이걸 기억해야한다.



코드에서 볼 수있듯이 인수가 디폴트값이 안 정해진 기본인자 a,b에 들어가고 그 다음에 디폴트값이 있는 c에 들어가고 그다음에 가변형 인자에 4,5,6,7,8,9가 들어간 후

key=value값을 집어넣은 키워드 인자에 들어가는 것을 볼 수 있다.