

# 빠르게 활용하는 파이썬3 프로그래밍

## 변수 & 함수



# 변수

- 객체를 가리키는 것

**a=3**

- 3이라는 값을 가지는 정수 자료형(객체)이 자동으로 메모리에 생성된다.
- a는 변수의 이름이며, 3이라는 정수형 객체가 저장된 메모리 위치를 가리키게 된다.
- 즉, 변수 a는 객체가 저장된 메모리의 위치를 가리키는 레퍼런스(Reference)라고도 할 수 있다.



# 변수

- 변수를 만드는 여러 가지 방법

```
>>> a, b = ('python', 'life')  
>>> (a, b) = 'python', 'life'
```

```
>>> [a,b] = ['python', 'life']
```

```
>>> a = b = 'python'
```

```
>>> a = 3  
>>> b = 5  
>>> a, b = b, a  
>>> a  
5  
>>> b  
3
```

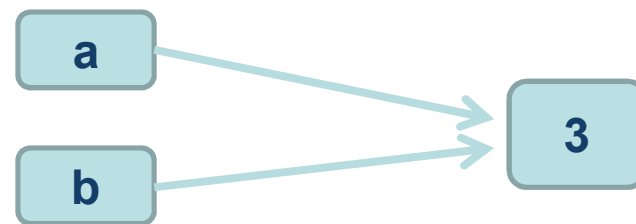
# 변수

- 메모리에 생성된 변수 없애기

- ✓ **가비지 콜렉션(Garbage Collection)** : 객체를 가리키고 있는 변수가 하나도 없을 때 객체는 메모리에서 사라지게 됨.

```
>>> a = 3
>>> b = 3
>>> del(a)
>>> del(b)
```

**#3**이라는 정수형 객체가 메모리에 생성





# 변수

- 리스트를 변수에 넣고 복사하고자 할 때

```
>>> a = [1, 2, 3]
>>> b = a
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 4, 3]
```

## 1. [:] 이용

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 2, 3]
```

## 2. copy 모듈 이용

```
>>> from copy import copy
>>> b = copy(a)
```

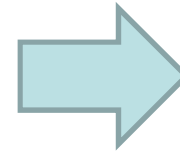
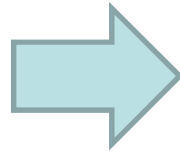
```
>>> b is a
False
```





# 함수

**Input**



**Output**

- 반복적으로 사용되는 가치 있는 부분"을 한 뭉치로 묶어서 "어떤 입력값을 주었을 때 어떤 결과값을 돌려준다"라는 식의 함수로 작성하는 것이 현명하다.



# 함수

- 함수는 여러 개의 문장(statement)을 하나로 묶어 줌
- 이미 정의 되어 있는 함수를 사용하거나 필요한 함수를 정의함
- 한 번 혹은 여러 번 호출 될 수 있으며 함수 종료 시 결과값을 전달.
- 프로그램을 구조적, 논리적으로 만들어 준다.

- A. 함수의정의
- B. return
- C. 인수전달
- D. 스코핑 룰
- E. 함수 인수
- F. Lambda 함수
- G. 재귀적 함수 호출
- H. Pass
- I. `__doc__`속성과 help 함수
- J. 이터레이터
- K. 제네레이터



# 함수의 정의

- 함수의 선언은 def로 시작하고 콜론(:)으로 끝낸다.

```
def 함수명(입력 인수):  
    <수행할 문장1>  
    <수행할 문장2>  
    ...
```

- 함수의 시작과 끝은 코드의 들여쓰기로 구분하기때문에 시작과 끝을 명시해 줄 필요가 없다.
- 헤더(header)파일,  
인터페이스(interface)/구현(implementation)같은  
부분으로 나누지 않음





# 함수 선언

- 함수 선언 문법

```
def <함수명>(인수1, 인수2, ...인수N):  
    <구문>  
    return <반환값>
```

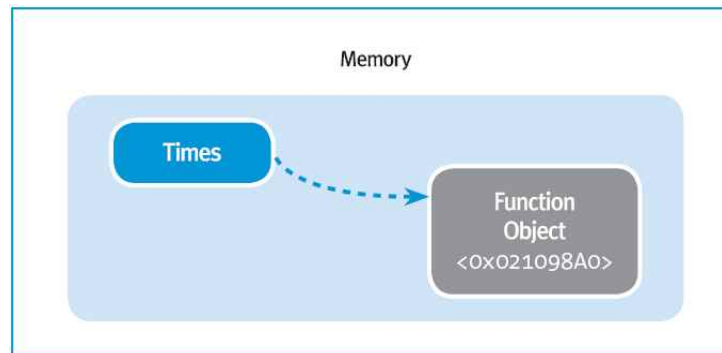
- 간단한 함수 선언해 보기
  - 입력 받은 2개의 인수를 서로 곱한 값을 리턴한다.

```
>>> def Times(a,b):  
        return a*b
```

```
>>> Times(10,10)  
100
```

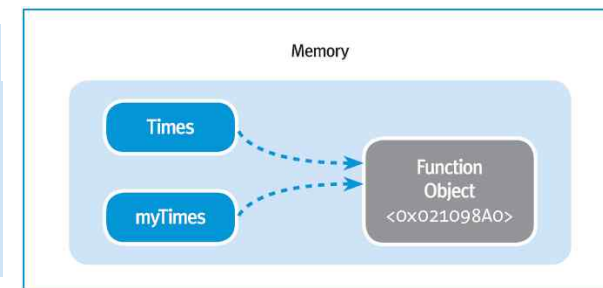
# 함수를 선언하면...

- 메모리에 함수 객체가 생성된다.
- 함수 객체를 가리키는 레퍼런스가 생성된다.



- 함수 레퍼런스를 통해서 함수를 사용하게 된다.
  - 함수 레퍼런스는 다른 변수에 할당 할 수 있다.

```
>>> myTimes = Times
>>> r = myTimes(10, 10)
>>> r
100
```





# return

- 함수를 종료시키고 호출한 곳으로 돌아가게 한다.
- return은 어떠한 객체로 돌려줄 수 있다.
  - 여러 개의 값을 튜플로 묶어서 값을 전달 할 수 있음

```
>>> def swap(x, y):  
    return y, x  
  
>>> swap(1, 2)  
(2,1)
```

- return을 사용하지 않거나 return만 적을 때도 함수가 종료
  - 리턴값으로 None을 리턴

```
>>> def setValue(newValue):  
    x = newValue    ← 반환 값이 없는 경우  
  
>>> retval = setValue(10)  
  
>>> print(retval)  
None
```



# 함수

- 입력값과 결과값에 따른 함수의 형태

- 일반적인 함수

```
def sum(a, b):  
    result = a + b  
    return result
```

```
>>> a = sum(3, 4)  
>>> print(a)  
7
```

- 입력값이 없는 함수

```
>>> def say():  
...     return 'Hi'  
...  
>>>
```

```
>>> a = say()  
>>> print(a)  
Hi
```

- 입력값도 결과값도 없는 함수

```
>>> def say():  
...     print('Hi')  
...  
>>>
```

```
>>> say()  
Hi
```



# 함수

## - 결과값이 없는 함수

```
>>> def sum(a, b):  
...     print("%d, %d의 합은 %d입니다." % (a, b, a+b))  
...  
>>>
```

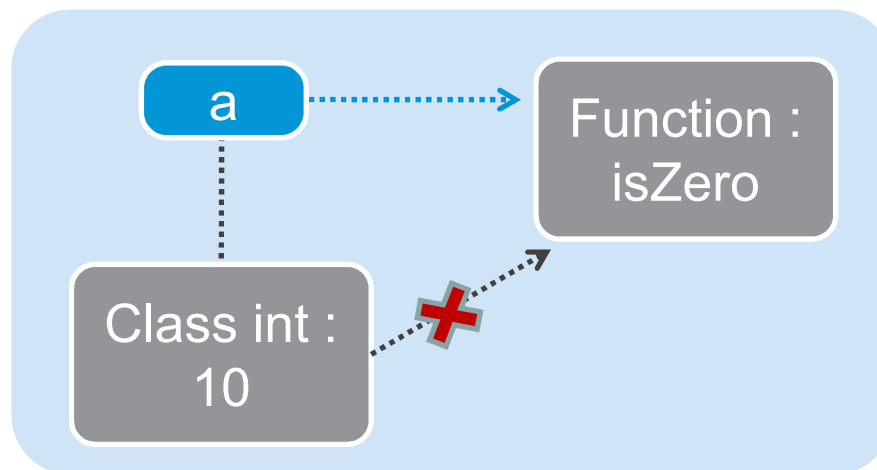
```
>>> sum(3, 4)  
3, 4의 합은 7입니다.
```

```
>>> a = sum(3, 4)  
>>> print(a)  
None
```

# 파이썬 함수에서 인수 전달-1

- 파이썬에서 함수 인수는 레퍼런스를 이용해 전달
  - 함수의 인수는 호출자 내부 객체의 레퍼런스

```
>>> a = 10
>>> def isZero(arg1):
>>>     return arg1 == 0
>>> isZero(a)
False
```

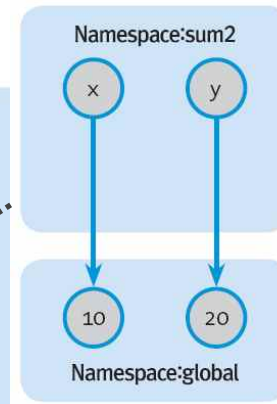




# 파이썬 함수에서 인수 전달-2

- 호출자가 전달하는 변수의 타입에 따라 다르게 처리
  - 변경가능 변수 (mutable)
  - 불가능 변수 (immutable)
- 변경 불가능 변수 예제

```
>>> a = 10
>>> b = 20
>>> def sum1(x, y):
>>>     return x + y
>>> sum1(a, b)
30
```



```
>>> x = 10
>>> def sum2(x, y):
>>>     x = 1
>>>     return x + y
>>> sum2(x, b)
```

21

```
>>> x
10
```

← 이 부분에서 값이 1인 객체가 생성되고 x에 레퍼런스를 할당합니다

← x를 인수로 넣어 줍니다

← 함수 내부에서 변경한 사항이 외부에 영향을 미치지 않습니다

# 파이썬 함수에서 인수 전달-3

- 변경가능한 변수를 인수로 전달.

```
>>> def change(x):  
    x[0] = 'H'      ← list x의 첫 번째 아이템을 H로 바꿉니다  
>>> wordlist = ['J', 'A', 'M']  
>>> change(wordlist)  
>>> wordlist  
['H', 'A', 'M']    ← change가 호출자의 객체에게 영향을 미칩니다
```

```
>>> def change(x):  
    x = x[:]        ← 입력받은 인수를 모두 x에 복사합니다  
    x[0] = 'H'      ← list x의 첫 번째 아이템을 H로 바꿉니다  
    return None  
>>> wordlist = ['J', 'A', 'M']  
>>> change(wordlist)  
>>> wordlist  
['J', 'A', 'M']    ← change가 호출자의 객체에 영향을 미치지 않습니다
```



# 스코핑 룰(Scoping rule)

- 이름공간 (Name Space)
  - 변수의 이름이 저장되어 있는 장소
  - 함수 내부의 이름공간, 지역 영역(Local scope)
  - 함수 밖의 영역, 전역 영역(Global scope)
  - 파이썬 자체에서 정의한 내용에 대한 영역, 내장 영역 (Built-in Scope)
- LGB 규칙
  - 변수 이름을 찾을 때 Local Scope -> Global Scope -> Built-in Scope 순서로 찾는다.

```
>>> x = 1
>>> def func(a):
    return a + x
>>> func(1)
2
>>> def func2(a):
    x = 2
    return a + x
>>> func2(1)
3
```

- 지역 영역에서 전역 영역의 이름을 접근할 때 global을 이용



# 파이썬에서 인수 모드

- 기본 인수 값
  - 함수를 호출 할 때 인수를 지정해 주지 않아도 기본 값이 할당되도록 하는 방법.

```
>>> def Times(a = 10, b = 20):  
    return a * b  
  
>>> Times()  
200  
  
>>> Times(5)  
100
```

- 키워드 인수
  - 인수 이름으로 값을 전달하는 방식
  - 변수의 이름으로 특정 인수를 전달할 수 있다.

```
>>> def connectURI(server, port):  
    str = "http://" + server + ":" + port  
    return str  
  
>>> connectURI("test.com", "8080")  
'http://test.com:8080'  
  
>>> connectURI(port="8080", server="test.com")  
'http://test.com:8080'
```

# 재귀적(recursive) 함수 호출

- 함수 내부에서 자기 자신을 계속 호출 하는 방법
  - 변수를 조금씩 변경하면서 연속적으로 반복된 연산을 할 때 유용함.

```
>>> def gop(a,b):  
    print (a*b)
```

```
>>> def hap(a,b):  
    print (a+b)
```

```
>>> def hap_gop(a,b):  
    hap(a,b)  
    gop(a,b)
```

```
>>> hap_gop(1,2)  
3  
2  
.
```

```
>>> def countdown(n):  
    if n == 0:  
        print ("Blastoff!")  
    else:  
        print (n)  
        countdown(n-1)
```

```
>>> countdown(3)  
3  
2  
1  
Blastoff!  
.
```



# 재귀적(recursive) 함수 호출

- 함수 내부에서 자기 자신을 계속 호출 하는 방법
  - 변수를 조금씩 변경하면서 연속적으로 반복된 연산을 할 때 유용함.

```
>>> def factorial(x):  
    if x == 1:  
        return 1  
    return x * factorial(x - 1)  
  
>> factorial(10)  
3628800
```





# pass 구문(statement)

- 아무 일도 하지 않습니다.

```
>>> while True:  
    pass
```

- 아무것도 하지 않는 함수, 모듈, 클래스를 만들어야 할 경우가 있는데. 이 때 pass가 사용될 수 있다.

```
>>> def sample():  
    pass
```

```
>>> sample()
```

```
>>>
```

```
>>>
```



## \_\_doc\_\_ 속성과 help 함수

- help 함수를 이용해 함수의 설명을 볼 수 있다.

```
>>> help(print)
```

- 사용자가 만든 함수도 help를 사용해 설명을 볼 수 있다.

```
>>> def plus(a, b):  
    return a + b  
  
>>> help(plus)  
Help on function plus in module __main__:  
plus(a, b)
```

- 조금 더 자세한 설명을 추가 하려면 \_\_doc\_\_ 속성을 이용한다.


```
>>> plus.__doc__ = "return the sum of parameter a, b "  
  
>>> help(plus)  
Help on function plus in module __main__:  
plus(a, b)  
    return the sum of parameter a, b
```



# 이터레이터 (Iterater)

- 순회가능한 객체의 요소를 순서대로 접근 할 수 있는 객체
  - 내부 반복문을 관리해 주는 객체
  - 이터레이터 안의 `__next__()`를 이용해 순회 가능한 객체의 요소를 하나씩 접근 할 수 있다.

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> next(it)
'a'
>>> next(it)
'b'
>>> it.__next__()
'c'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
    next(it)
StopIteration
```



# 제네레이터 (Generator)

- return 대신 yield라는 구문을 이용해 함수 객체를 유지한 채 값을 호출자에 넘겨줌
  - 값을 넘겨준 후 함수 객체는 그대로 유지
  - 함수의 상태를 그대로 유지하고 다시 호출 할 수 있기 때문에 순회가능한 객체를 만들 때 매우 편리함.

```
>>> def reverse(data):  
    for index in range(len(data) - 1, -1, -1):  
        yield data[index]  
  
>>> for char in reverse('golf'):  
    print(char)  
  
f  
l  
o  
g
```



# 제네레이터 예제

```
>>> def abc():  
    data = "abc"  
    for char in data:  
        yield char  
  
>>> abc  
<function abc at 0x0205EB70>  
>>> abc()  
<generator object abc at 0x02061A30>  
  
>>> it = iter(abc())  
>>> next(it)  
'a'  
>>> next(it)  
'b'  
>>> next(it)  
'c'
```

- `abc` 자체는 함수이지만 리턴하는 값은 제네레이터 객체
- 함수를 유지하고 값을 전달할 수 있기 때문에 순회 가능한 객체를 만드는데 제네레이터가 유리

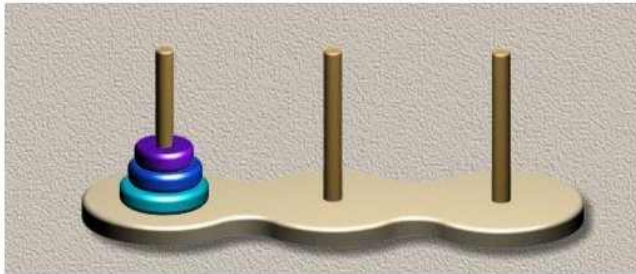


# 과제

- 과제 1. 학번을 입력하면 이름이 리턴되는 함수 "name"을 만들고 `_doc_`를 이용하여 설명을 첨부하고 출력결과를 보이게 캡처합니다.
- 과제 2. 재귀함수를 이용하여 하노이 탑을 수행하는 함수를 만든다.
- 과제 3. 재귀함수를 이용하여  $2^n$ 을 구하는 함수를 만든다.
- 과제 4. 제너레이터를 이용하여 피보나치 수열을 실행하는 함수를 만든다.
- 과제 5. 재귀함수를 이용하여 피보나치 수열을 실행하는 함수를 만든다.



# 하노이의 탑 규칙 및 힌트



## 1. 하노이 탑 정의

- 원판을 세 번째로 모두 옮겨놓아야 한다. 이때도 작은 원판이 큰 원판 위에 있어야 한다.
- 원판을 옮길 때는 반드시 한 번에 한 개씩 옮길 수 있고 두 번째 막대기를 이용할 수 있습니다.
- 옮기는 과정에서 절대로 큰 원판이 작은 원판 위에 놓이지 않아야 합니다.
- 최대공약수를 구하기 위해서는 다음과 같은 과정을 거쳐야 한다.

## 2. 하노이 탑의 규칙 찾기

- 원판이 1개일 때
  1. A에서 원판 1을 C로 이동
- 원판이 2개일 때
  1. A에서 원판1을 B로 이동
  2. A에서 원판2을 C로 이동
  3. B에서 원판1을 C로 이동
- 원판이 3개일 때
  1. A에서 원판1을 C로 이동
  2. A에서 원판2을 B로 이동
  3. C에서 원판1을 B로 이동
  4. A에서 원판3을 C로 이동
  5. B에서 원판1을 A로 이동
  6. B에서 원판2을 C로 이동
  7. A에서 원판1을 C로 이동

## 3. 하노이 탑 알고리즘

- 순서대로 1부터  $n$ 까지 원판이 있고 A, B, C 3개의 막대기가 있는 경우 하노이 탑 문제를 해결하는 방법은 다음과 같다.
  1. A 막대기에서 2번부터  $n$ 번째까지  $n-1$ 개의 원판을 B막대기로 이동한다.
  2. A 막대기에서 1번 원판을 C막대기로 이동시킨다.
  3. B 막대기에서 2번부터  $n$ 번째 까지  $n-1$ 개의 원판을 C로 이동시킨다.

# 과제 제출 방법

- ▶ 과제캡처 후 워드or한글파일에 첨부/정리하여 제출
- ▶ 파일형식 : [과제번호4]\_이름(조이름)\_학번
  - ▶ 제출 형식 어길 시 감점처리
- ▶ 제출 : **dbcyy1@gmail.com**로 제출
- ▶ 제출기간 : 4월5일 화요일 23시59분까지
- ▶ 첨부파일 넣었는지, 메일 반송되지 않았는지 꼭 확인하세요!



감사합니다.