# 기초 PYTHON 프로그래밍

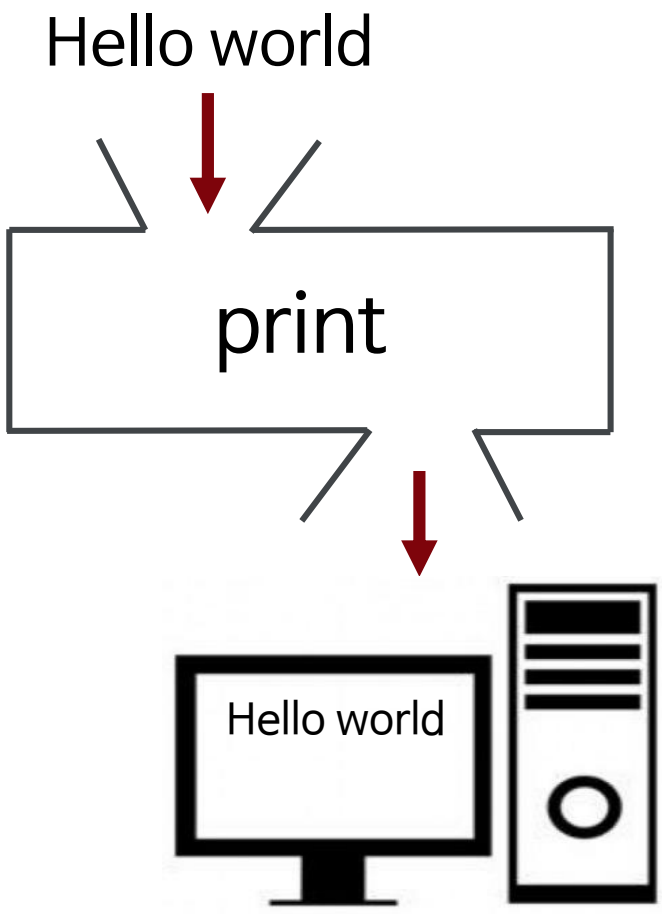
## 11. 함수

1. 함수
2. 파이썬 내장 함수
3. 사용자 정의 함수
4. 함수의 인수와 반환값
5. 함수의 위치와 main 작성하기
6. 전역 변수와 지역 변수
7. 함수의 인수
8. 람다 함수 (lambda)

# 1. 함수

- 블랙 박스 (black box)
  - 함수는 입력과 출력을 갖는 black box이다.
  - 주어진 입력에 대해서 어떤 과정을 거쳐 출력이 나오는지가 숨겨져 있다.
- 함수의 재사용성 (reuse)
- 함수 이름, 입력, 출력 중요함.

```
>>> print('Hello world')
```



# 2. 파이썬 내장 함수

- 내장 함수 (built-in functions)
  - 파이썬 언어에서 미리 만들어서 제공하는 함수들
  - IDLE에서 dir(__builtins__)라고 입력하면 파이썬에서 제공하는 내장 함수 목록을 볼 수 있다.
  - 내장 함수에 어떤 것들이 있는지 학습하고 적절히 사용할 줄 아는 것이 중요하다.
- 사용자 정의 함수 (user-defined functions)
  - 사용자가 직접 만드는 함수
  - 함수 작성 문법을 익히고 직접 작성해 보는 것이 중요하다.

# 2. 파이썬 내장 함수

```
>>> dir(__builtins__)
['ArithmeticError', …… 'ZeroDivisionError', '_', '__build_class__',
'__debug__', '__doc__', '__import__', '__loader__', '__name__',
'__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray',
'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright',
'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit',
'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash',
'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object',
'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str',
'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

# 2. 파이썬 내장 함수

- 군집자료형에 유용한 함수들
  - len(), max(), min(), sum(), sorted(), reversed() 함수

```
>>> L = [6,4,9,8,10,50,7]
>>> len(L)
7
>>> max(L)
50
>>> min(L)
4
>>> sum(L)
94
>>> sorted(L)
[4, 6, 7, 8, 9, 10, 50]
>>> L
[6, 4, 9, 8, 10, 50, 7]
```

```
>>> T = (4,6,10,2,1,5)
>>> a = len(T)
>>> b = max(T)
>>> c = min(T)
>>> d = sum(T)
>>> print(a,b,c,d)
6 10 1 28

>>> S = sorted(T)
>>> print(S)
[1, 2, 4, 5, 6, 10]
>>> type(S)
<class 'list'>
```

```
>>> S = {4,6,2,7,5}
>>> a = len(S)
>>> b = max(S)
>>> c = min(S)
>>> d = sum(S)
>>> print(a,b,c,d)
5 7 2 24

>>> T = sorted(S)
>>> print(T)
[2, 4, 5, 6, 7]
>>> type(T)
<class 'list'>
```

# 2. 파이썬 내장 함수

- reversed 함수 - 군집자료형 데이터를 역순으로 바꾼다.
  - 순서가 있는 list, tuple, str 에만 적용할 수 있다.

```
>>> L = [1,3,5,7,9]
>>> M = reversed(L)
>>> print(M)
<list_reverseiterator object at 0x03DC92B0>
>>> M = list(reversed(L))
>>> print(M)
[9, 7, 5, 3, 1]
```

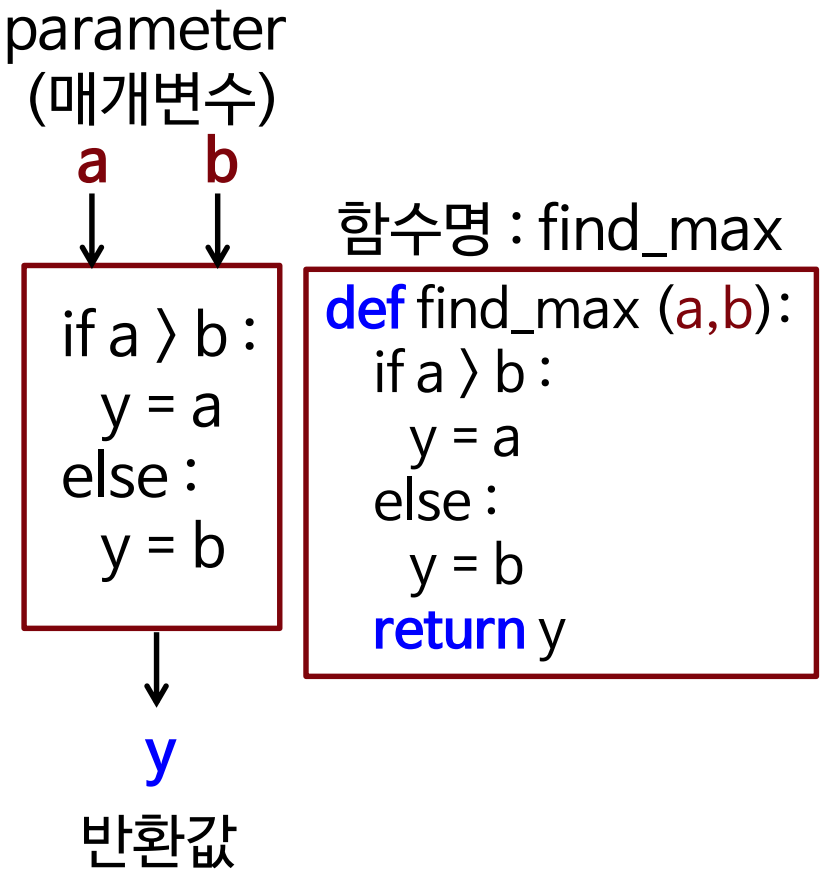
```
>>> T = (2,4,6,8)
>>> K = list(reversed(T))
>>> print(K)
[8, 6, 4, 2]
```

```
>>> S = 'HELLO'
>>> P = list(reversed(S))
>>> print(P)
['O', 'L', 'L', 'E', 'H']
```

# 3. 사용자 정의 함수

- 함수 정의하기

```
def 함수명 ( parameter ) :
    """ 이 함수는 두 정수에서 큰 값
    을 찾아서 반환하는 함수이다. """
    
    return 반환값
```




- docstring : """ …… """
- 함수명 - 변수명 규칙과 같다.
  - 영문 대소문자, 숫자, _ 로 구성됨.
  - 숫자로 시작할 수 없음.

# 3. 사용자 정의 함수

- 함수 호출하기 - 함수명에 인수를 넘기면서 호출함.

```
def find_max (a,b):
    if a > b :
        y = a
    else :
        y = b
    return y
```

a,b : 매개변수 (parameter)

```
# main

m = find_max(10,20)    # 10, 20 : 인수 (arguments)
n = find_max(7,3)      # 7, 3 : 인수 (arguments)
print(m,n)
```

# 4. 함수의 인수와 반환값

- 인수, 반환값이 없는 함수
  - 함수에 입력이 없으면 빈 괄호로 둔다.
  - 함수에 출력이 없을 수도 있다 (return 구문이 없다).

```
def hello():
    print('hello world')
    print('hello python~')

# main - 여기에서 프로그램 수행 시작

print('start of the program')
hello()
print('middle of the program')
hello()
print('end of the program')
```

```
start of the program
hello world
hello python~
middle of the program
hello world
hello python~
end of the program
```

# 4. 함수의 인수와 반환값

- 인수가 여러 개인 함수 (인수 개수만큼 parameter 필요함)
  - 예) main 에서 국어, 영어, 수학 성적을 입력받아 calculate함수의 입력으로 넘긴다. calculate 함수는 평균을 반환한다.

```
def calculate(kor, eng, math):
    total = kor + eng + math
    average = total / 3
    return average

# main
kor_score = int(input('국어 성적을 입력하시오 : '))
eng_score = int(input('영어 성적을 입력하시오 : '))
math_score = int(input('수학 성적을 입력하시오 : '))

avg = calculate(kor_score, eng_score, math_score)
print('평균 :', avg)
```

```
국어 성적을 입력하시오 : 90
영어 성적을 입력하시오 : 85
수학 성적을 입력하시오 : 92
평균 : 89.0
```

# 4. 함수의 인수와 반환값

- 함수에서 return 문이 여러 번 나오는 경우
  - return 문이 여러 번 나오더라도 가장 먼저 return을 만나는 순간 함수는 값을 반환하고 종료한다.

```
def find_max (a,b):
    if a > b :
        y = a
    else :
        y = b
    return y
```

함수명 : find_max

```
def find_max (a,b):
    if a > b :
        return a
    else:
        return b
```

# 4. 함수의 인수와 반환값

- 반환값이 여러 개인 경우
  - 함수에 반환값이 2개 이상인 경우 튜플로 묶어서 반환한다.
  - 예) 두 수를 입력받아서 두 수의 합과 두 수의 곱을 반환하는 함수

```
def add_multiply(x,y):
    sum = x + y
    mult = x * y
    return sum, mult   # 튜플로 반환한다.

# main
a = int(input('Enter a : '))
b = int(input('Enter b : '))
m,n = add_multiply(a,b)
print(m,n)
```

# 5. 함수의 위치와 main 작성하기

- 함수의 위치
  - 함수는 호출 전에 정의되어 있어야 한다.

```
print('start of the program')
hello()
print('middle of the program')
hello()
print('end of the program')

def hello():
    print('hello world')
```

```
start of the program
Traceback (most recent call last):
  File "C:/Users/sogang-pc/AppData/Loc
ne 2, in <module>
    hello()
NameError: name 'hello' is not defined
```

```
a = 10
b = 20
print('before test definition')

def test():
    print('I am test')
    print(a, b)

print('start')
test()
print('end')
```

```
before test definition
start
I am test
10 20
end
```

# 5. 함수의 main 작성하기

- main 함수 작성하기
  - main은 프로그램 시작점을 의미한다.

```
def find_max (a,b):
    if a > b : return a
    else: return b

x = find_max(10, 20)
y = find_max(7,3)
print(x,y)
```

```
def find_max (a,b):
    if a > b : return a
    else: return b

def main():
    x = find_max(10, 20)
    y = find_max(7,3)
    print(x,y)

if __name__ == '__main__':
    main()
```

# 6. 전역 변수와 지역 변수

- 전역 변수(global variable) : 프로그램 전체에서 사용 가능
- 지역 변수(local variable) : 함수 내에서만 사용 가능
- scoping rule : 변수를 찾을 때 지역 변수 → 전역 변수 순서로 찾는다.

```
def test():
    print(a)

# main

a = 100   # 전역변수
print(a)
test()
print(a)
```

```
100
100
100
```

```
def test():
    b = 20   # 지역변수
    print(a,b)

# main

a = 100   # 전역변수
print(a)
test()
print(a)

print(b)   # 에러
```

```
100
100 20
100
```

```
def test():
    a = 200  # 지역변수
    print(a)

# main

a = 100   # 전역변수
print(a)
test()
print(a)
```

```
100
200
100
```

# 6. 전역 변수와 지역 변수

- global 선언
  - 전역 변수를 함수 내에서 바꾸고자 하면, global 선언이 필요함

```
def test():
    a = 200

# main

a = 100
test()
print(a)
```

```
100
```

```
def test():
    global a
    a = 200

# main

a = 100
test()
print(a)
```

```
200
```

# 6. 전역 변수와 지역 변수

- global 선언
  - 함수에서 만든 지역 변수를 전역 변수로 사용하고자 한다면 global 선언이 필요함

```
def test():
    a = 100
    print(a)

# main

test()
print(a)
```

에러 발생

```
def test():
    global a
    a = 100
    print(a)

# main

test()
print(a)
```

```
100
100
```

# 6. 전역 변수와 지역 변수

- 함수 매개변수는 지역 변수이다.

```
def test(a):
    a += 200
    print(a)

# main

a = 100
test(a)
print(a)
```

지역 변수 a

전역 변수 a

```
300
100
```

함수 test를 호출할 때 100을 인자로 넘기면 **지역 변수 a**가 생성되고 값으로 100을 갖는다.

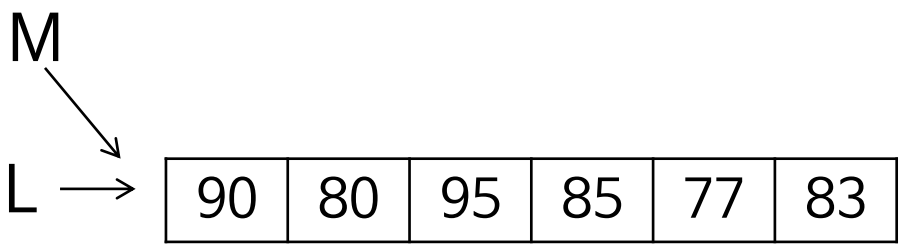
# 6. 전역 변수와 지역 변수

- mutable 객체가 인수인 경우 (리스트, 집합, 사전)
  - 리스트 인수

```
def test(M):
    M[2] += 2
    M[4] += 10

# main

L = [90,80,95,85,77,83]
test(L)
print(L)
```




만약에 L 복사본이 생겨서 원본 L을 바꾸고 싶지 않다면 다음과 같이 test 함수에 인수를 넘겨야 한다.

test(L[:]) 또는 test(L[::])

```
[90, 80, 97, 85, 87, 83]
```

# 7. 함수의 인수

- 기본값(default value)이 있는 인수
  - 함수를 호출할 때 인수를 넘겨주지 않아도 인수가 자신의 기본값을 취하도록 하는 기능.

```
def inc(a, step=1):
    return a + step

b = inc(10)
print(b)    # 11

c = inc(10, 50)
print(c)    # 60
```

- 첫 번째 매개 변수 a 에는 반드시 인수를 넘겨야 한다.
- 두 번째 매개 변수인 step에는 인수를 넘기면 그 값이 step에 저장되고, 인수를 넘기지 않으면 1이 step 값으로 이용된다.

# 7. 함수의 인수

- 함수를 정의할 때 인수의 기본값 사용시 주의점
  - 기본값이 있는 인수가 먼저 올 수 없다.

```
def inc(step=1, a):
    return a + step
```

에러

  - 인수가 여러 개인 경우의 예

```
def test(a, b=1, c=2):          # OK
        return a + b + c

def test2(a, b, c=10):          # OK
        return a + b + c

def test3(a, b=1, c):           # 에러
        return a + b + c
```

```
>>> L = [4,3,5,1,2]
>>> L.sort()
>>> print(L)
[1, 2, 3, 4, 5]
>>> M = [3,5,1,2,4]
>>> M.sort(reverse=True)
>>> print(M)
[5, 4, 3, 2, 1]
>>> a,b = 10, 20
>>> print(a,b)
10 20
>>> print(a,b,sep=':')
10:20
>>> print(a,end=' ^^ '); print(b)
10 ^^ 20
```

# 7. 함수의 인수

- 키워드 인수
  - 함수 호출 시에 인수 이름(매개변수)과 값을 같이 전달하기

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

```
a = area(10,5)
b = area(x=4, y=9)      # 매개변수와 값을 같이 적어 준다
c = area(y=5, x=10)     # 매개변수와 값을 같이 적으면 순서 상관없다
d = area(10, y=5)       # OK (일반 인수 뒤에 키워드 인수 올 수 있다)
e = area(x=10, 5)       # 에러
```

# 7. 함수의 인수

- 키워드 인수
  - 키워드 인수의 위치는 일반 인수 이후이다.

```
def volume(x,y,z):
    return x * y * z
```

일반 인수 : positional argument

```
volume(1, 3, 5)          # 15
volume(y=7, z=5, x=2)    # 70
volume(z=2, x=4, y=5)    # 40
volume(5, z=10, y=2)     # 100
volume(5, x=2, z=20)     # 에러 (x에 두 개의 값이 할당된다는 에러)
volume(z=10, 20, x=2)    # 에러 (일반 인수가 키워드 인수 뒤에 오면 안됨)
```

# 7. 함수의 인수

## ◆ 가변 인수 사용하기

- 정해지지 않은 수의 인수를 함수에 전달하기
- 함수를 정의할 때 인수 목록에 반드시 넘겨야 하는 고정 인수를 우선 나열하고, 나머지를 마지막에 튜플 형식으로 한꺼번에 받는다.

```
def friends(*name):
    for n in name:
        print(n.title())

friends('alice', 'paul')
print('------')
friends('cindy', 'sally', 'david', 'tom')
```

```
Alice
Paul
------
Cindy
Sally
David
Tom
```

```
def arg(a, b, *c):
    print(a,b,c)
    print(sum(c))

# main
arg(10,20)
print('----------')
arg(7,8,9)
print('----------')
arg(1,2,3,4,5)
```

```
10 20 ()
0
----------
7 8 (9,)
9
----------
1 2 (3, 4, 5)
12
```

# 7. 함수의 인수

- 정의되지 않은 키워드 인수 처리하기
  - ** 형식으로 기술
  - 전달받는 형식은 사전이다. 즉, 키는 키워드(변수명)가 되고, 값은 키워드 인수로 전달되는 값이 된다.

```
def name_age(**lists):
    print(lists)

name_age(Alice=10, Paul=12)
print('----------------------------')
name_age(Cindy=5, David=7, Tom=10)
```

```
{'Alice': 10, 'Paul': 12}
----------------------------
{'David': 7, 'Tom': 10, 'Cindy': 5}
```

# 8. 람다 함수 (lambda)

- 람다 함수의 정의
  - 이름없는 한 줄짜리 함수이다.
  - 람다 함수는 return 문을 사용하지 않는다.
  - 람다 함수의 몸체는 문이 아닌 하나의 식이다.
  - 람다 함수는 함수를 함수 인자로 넘길 때 유용하다.

lambda 〈인수들〉 : 〈반환할 식〉

```
def add(x,y):
    return x+y
```



```
lambda x, y: x + y
```

# 8. 람다 함수 (lambda)

- map 내장 함수

map( 함수명 , 함수에 대한 인수 집합 )

시퀀스 자료형 (리스트, 튜플, 문자열)

```
〉〉〉 names = ['Alice', 'Paul', 'Bob', 'Robert']
〉〉〉 A = map(len, names)
〉〉〉 print(A)
〈map object at 0x036091D0〉
〉〉〉 list(A)
[5, 4, 3, 6]
```

# 8. 람다 함수 (lambda)

- map 내장 함수 예

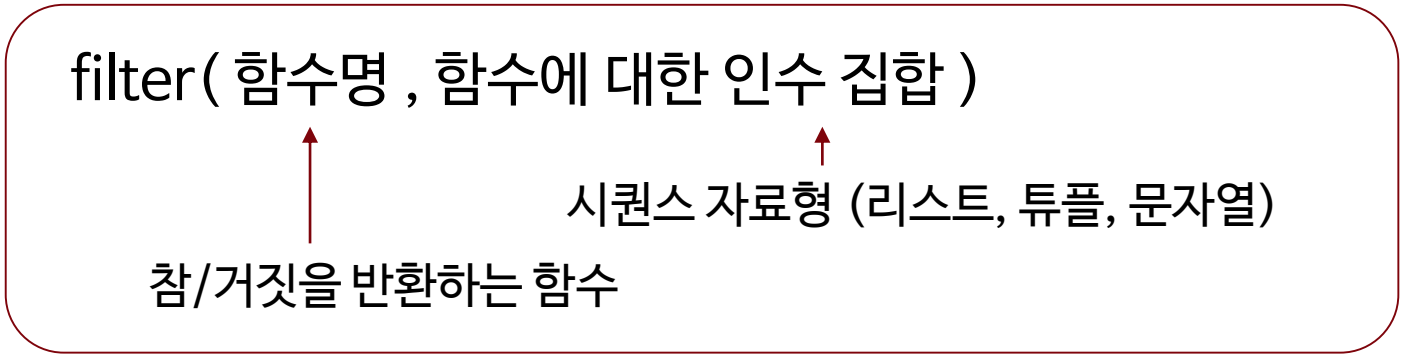
```
>>> def f(x):
        return x * x
>>> X = [1, 2, 3, 4, 5]
>>> list(map( f, X ))        # 함수 f 에 X의 값들을 하나씩 적용한다
[1, 4, 9, 16, 25]

>>> X = [1, 2, 3, 4, 5]
>>> Y = map( lambda a:a * a, X )
>>> list(Y)
[1, 4, 9, 16, 25]

# range(10)의 원소를 식 x²+4x+5 에 대입한 결과를 내 준다
>>> Y = map( lambda x: x * x + 4 * x + 5, range(10) )
>>> list(Y)
[5, 10, 17, 26, 37, 50, 65, 82, 101, 122]
```

# 8. 람다 함수 (lambda)

- filter 내장 함수




```
>>> X = [1,3,5,7,9]
>>> result = filter( lambda x:x>5, X )
>>> print(result)
<filter object at 0x03FAD0D0>
>>> list(result)
[7, 9]

>>> list(filter( lambda x:x%2==1, range(11)))
[1, 3, 5, 7, 9]
```