| 파이썬 스터디 |

3주차

index

1. 리스트
- 리스트란?
- 연산
- 리스트 함수
- 리스트와 for문
- 다차원 리스트

2. 함수
- 함수란?
- 매개변수, 인수
- 리턴 값
- global 변수

3. 기타
- 문자열 함수
- 매개변수의 기본값
- lambda

1. 리스트

| 리스트란?

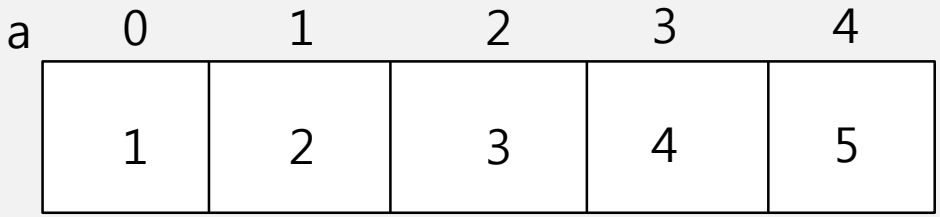
```
a = 1
b = 2
c = 3
....

a = [1,2,3...]
```

- 변수의 개수가 적을때는 한개 한개 따로 선언해도 됨
- 그러나 변수의 개수가 많아지면 하나하나 모두 선언할 수 없음
- 따라서 이를 해결하기 위해 리스트라는 개념 등장

리스트란?

```
a = [1,2,3,4,5]
```

- 옆과 같이 선언하면 총 5개의 변수를 만드는 것과 동일
- 각 요소는 첨자를 활용해 접근 가능
- 각 요소는 0번째 위치부터 시작해서 n번째 위치까지 선언
  ex)a[0] = 1, a[2] = 3, a[4] = 5
- a[-1] 은 리스트의 맨 마지막 원소를 가리킴



| 리스트의 초기화

```
a = [ ]
b = list()
c = [1, 2, 3]
d = ['A', 'B', 'CDEF']
e = [1, 2, [3, 4]]

arr = [0] * 10
```

- a와 b – 빈 리스트 선언
  - 처음에 값을 넣어주지 않고 후에 값을 넣어주려고 할 때 사용
- c,d,e – 처음에 값이 초기화되어 있는 리스트의 선언
- 리스트 여러개를 0으로 초기화하려면 *연산자를 활용하면 된다.

**리스트 내에는 숫자, 문자, 문자열, 리스트 등 어떠한 값이라도 모두 들어갈 수 있음

| 리스트 슬라이싱

```
a = [1, 2, 3, 4, 5]

print(a[1:3])
print(a[1:2])
print(a[1:])

b = a[2:]
print(b)
```

```
[2, 3]
[2]
[2, 3, 4, 5]
[3, 4, 5]
>>>
```

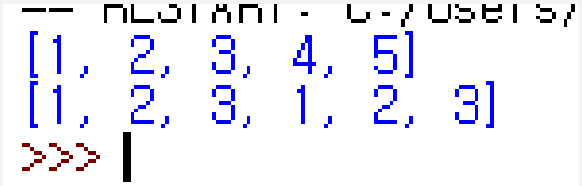
- 리스트의 첨자의 범위를 지정해줄 수 있음 ([시작인덱스:끝+1인덱스])
- 리스트의 첨자를 [1:]로 한다면 리스트의 첫번째 인덱스부터 마지막 인덱스까지 슬라이싱
- 리스트를 슬라이싱하여 새로운 리스트에 저장 가능

리스트 연산

- 더하기 연산(+), 반복 연산(*)
- 수정 연산(=), 길이 연산(len)
- 삭제연산 (del)
- 비교 연산

| 덧셈, 반복 연산

```
a = [1,2,3]
b = [4,5]

c = a + b
print(c)

c = a*2
print(c)
```



- 덧셈연산을 진행하면 앞의 리스트의 뒤에 리스트가 붙음
- 곱셈 연산을 진행하면 리스트를 반복호출

| 수정, 길이 연산

```
a = [1,2,3,4,5]

a[2] = 10

print(a)
print(len(a))
```

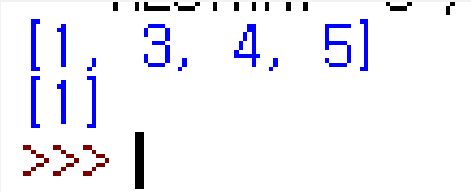
```
== RESTART: C:/Users
[1, 2, 10, 4, 5]
5
>>>
```

- a[인덱스]로 리스트의 요소에 접근가능
- 대입연산자를 활용하여 리스트의 값 수정 가능
- len(객체)
- len 함수는 객체의 길이를 구해주는 함수

삭제 연산

```
a = [1,2,3,4,5]

del(a[1])
print(a)

del(a[1:])
print(a)
```

- del(객체)
- del 함수는 객체를 삭제하는 함수
- del(a[1:])로 슬라이싱 해서 1번째 인덱스부터 리스트의 마지막 인덱스까지 한번에 삭제 가능



| 비교 연산

```
a = [1,2,3]
b = [1,2,3]
c = [4,5,6]

print(a==b)
print(a==c)

b.append(4)
print(a>b)
print(a<b)
```

```
== RESTART:
True
False
False
True
>>>
```

- 두 리스트의 길이가 같을때
  -리스트내의 요소가 모두 같으면 True
  -리스트내의 요소가 다르면 False
- 두 리스트의 길이가 다를때
  -길이가 긴쪽이 비교연산에서 큰 부분

리스트 함수

- 추가관련 함수: append, insert, extend
- 정렬관련 함수: sort, reverse
- 값추출 관련 함수: index, count
- 삭제관련 함수: remove, pop

| append

```
a = [1,2,3]
b = [4,5]

a.append(4)
a.append(5)
a.append(b)

print(a)
```

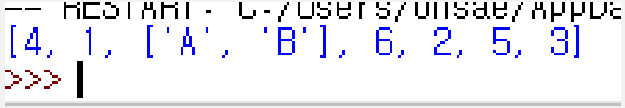
```
== RESTART: C:/Users/onsa
[1, 2, 3, 4, 5, [4, 5]]
>>>
```

- a.append(n)
- 리스트의 맨 마지막에 n을 삽입
- n이 리스트일경우 다차원배열의 형태로 추가됨

| insert

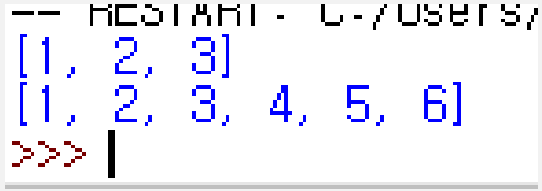
```
a = [1,2,3]
b = ['A','B']

a.insert(0,4)
a.insert(3,5)
a.insert(2,6)
a.insert(2,b)

print(a)
```

- a.insert(idx, n)
- 리스트의 idx 위치에 n의 값을 추가
- n에 리스트가 들어올 경우 다차원 배열의 형태로 추가됨



| extend

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

b = [4,5,6]
a.extend(b)
print(a)
```



- a.extend(list)
- 리스트의 뒤에 리스트를 붙임 (+연산과 동일)
- 리스트 내 다차원 배열의 형태로 붙이는 것이 아니라 b의 리스트내의 요소를 a의 리스트에 append 하는 것
- list 부분에는 리스트만 들어올 수 있음 (정수, 문자열 등 불가)

| sort

```
a = [3,4,1,2,5]
b = ['C', 'E', 'A', 'B', 'D']
c = [6,9,10,7,8]

a.sort()
b.sort()
c.sort(reverse = True)

print(a)
print(b)
print(c)
```

- a.sort()
- 리스트를 오름차순으로 정렬
- 문자도 아스키코드순으로 정렬
- sort(reverse = True)라고 쓰면 역순으로 정렬

```
== RESTART: C:/Users/onsae
[1, 2, 3, 4, 5]
['A', 'B', 'C', 'D', 'E']
[10, 9, 8, 7, 6]
>>>
```

reverse

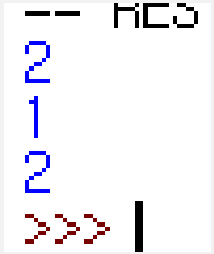
```
a = [3,7,1,4,6,2]
b = ['abcd', 'ghij', 'z', 'h']
c = [3,8,[1,3,5]]

a.reverse()
b.reverse()
c.reverse()

print(a)
print(b)
print(c)
```

- a.reverse()
- 리스트를 역순으로 바꿔줌
- 정렬 후 역순으로 바꾸는 게 아니라 그냥 리스트 자체를 뒤집는 것
- 리스트 내 리스트가 있으면 그 리스트는 뒤집지 않음

```
== RESTART: C:/Users/onsae/Ap
[2, 6, 4, 1, 7, 3]
['h', 'z', 'ghij', 'abcd']
[[1, 3, 5], 8, 3]
>>>
```

| index

```
a = [1,2,3,4,5]
b = ['A','B','C']
c = [1,2,[3,4]]
d = [3,4]

print(a.index(3))
print(b.index('B'))
print(c.index(d))
```



- a.index(n)
- 리스트에서 n이라는 요소가 있으면 그 인덱스를 리턴
- n에는 리스트가 들어갈 수 있음

count

```
a = [1,2,1,3,1,5]
b = ['A', 'A', 'B']
c = [[1,2],[1,2],3,4]
d = [1,2]

print(a.count(1))
print(b.count('A'))
print(c.count(d))
```

- a.count(n)
- 리스트 내에 n값이 몇번 나왔는지 리턴
- n에는 리스트가 들어갈 수 있음

remove

```
a = [1,2,3,1,2,3]

a.remove(3)
print(a)

a.remove(3)
print(a)

a.remove(2)
print(a)
```

```
[1, 2, 1, 2, 3]
[1, 2, 1, 2]
[1, 1, 2]
>>>
```

- a.remove(n)
- 리스트내에서 가장 처음 나오는 n의 값 삭제

| pop

```
a = [1,2,3,4,5]

print(a.pop())
print(a)

print(a.pop(1))
print(a)
```

```
5
[1, 2, 3, 4]
2
[1, 3, 4]
>>>
```

- a.pop(idx)
- 리스트의 idx번째 위치의 값을 반환하고 그 위치의 값을 삭제
- 만약 idx에 아무런값도 입력되지 않으면 맨 마지막값 반환 후 삭제

| 리스트와 for문

```
a = [1,2,3,4,5]

for i in a:
    print(i, end = ' ')
print('')

str_array = ['abc', 'def', 'ghi']
for i in str_array:
    print(i, end = ' ')
```

```
1 2 3 4 5
abc def ghi
>>>
```

- for문 in 뒤에 리스트를 사용 가능
- i 변수가 a의 값을 하나씩 참조해가면서 리스트의 끝이 되면 리스트 탈출
- i 변수가 str_array의 값을 하나씩 참조해가면서 리스트의 끝이 되면 탈출

| 리스트와 for문 - 가로입력

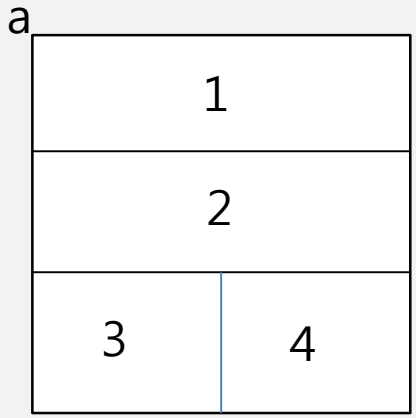
```
arr = []

n = int(input())
num = input()

sum = 0
for i in num.split():
    i = int(i)
    arr.append(i)

for i in range(0,n):
    sum = sum + arr[i]

print(sum)
```

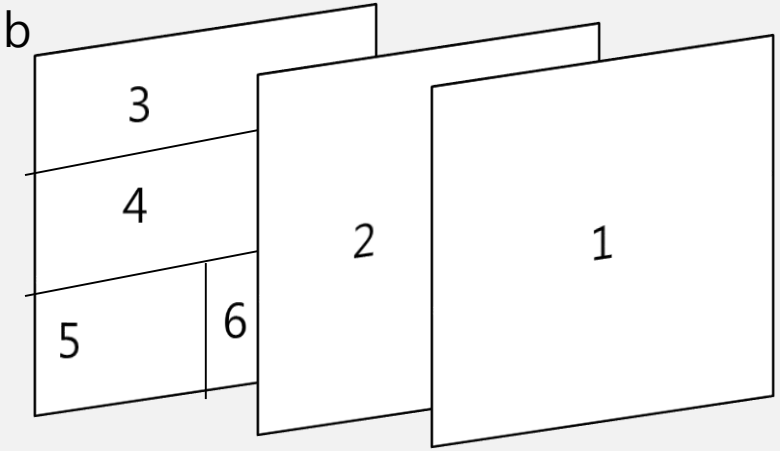
- num을 문자열 형태로 입력받는다.
- 입력받으면 공백을 기준으로 구분하는 문자열이 형성됨
- 공백을 기준으로 구분하므로 split()
- i는 문자열 변수이므로 정수 변환 후 배열에 값 추가

```
10
1 2 3 4 5 6 7 8 9 10
55
>>>
```

다차원 배열

```
a = [1,2,[3,4]]
print(a[2][0])
print(a[2][1])

b = [1,2,[3,4,[5,6]]]
print(b[2][2][0])
print(b[2][2][1])
```





2. 함수

함수란

```
arr = []
arr2 = []

n = int(input())
for i in range(0,n):
    k = int(input())
    arr.append(k)

for i in range(0,n):
    k = int(input())
    arr2.append(k)

for i in range(0,n):
    print(arr[i])

for i in range(0,n):
    print(arr2[i])
```

```
def input_list(arr, n):
    for i in range(0,n):
        k = int(input())
        arr.append(k)

def print_list(arr, n):
    for i in range(0,n):
        print(arr2[i])

arr = []
arr2 = []

n = int(input())

input_list(arr, n)
input_list(arr2, n)

print_list(arr, n)
print_list(arr2, n)
```

| 함수란?

- 같은 작업을 반복할 때 보기 편하게 바꿔줌, 특정한 기능을 수행하는 도구

- 함수는 사용하기 직전에 선언해도 됨.
  그러나 가독성을 위해서 맨 위에 모두 선언 하는게 보기 좋음

- 함수를 호출하려면 함수 이름뒤에 필수로 괄호가 붙어야 함
  (괄호가 함수인지 변수인지 구분하는 부분)

- 함수 내에서 선언한 변수는 함수가 종료되면 삭제됨
  -> 함수 내에서 선언한 변수를 함수 외부에서 사용 불가
  반대는 가능

| 함수란?

```
def a():
    n = 3

a()
print(n)
```

- 에러 발생

  why? n의 값 참조 불가

```
n = 3

def a():
    print(n)

a()
```

- n의 값이 출력됨

  why? n은 전역변수 역할

| 매개변수, 인수

```
def add(a, b):
    print(a+b)

add(3,4)
add('A', 'B')
add([1,2], [3,4])
```

```
7
AB
[1, 2, 3, 4]
```

- 함수에는 값을 넘겨줄 수 있음
- 값을 넘겨줄 때 자료형은 넘겨주는 쪽에서 맞춰줌
- 매개변수와 인수의 개수가 맞아야 제대로 함수 호출 가능

| 매개변수, 인수

```
def add_sum(*array):
    sum = 0

    for i in array:
        sum += i

    print(sum)

add_sum(1,2)
add_sum(1,2,3,4,5)
```

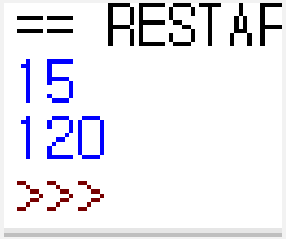
```
3
15
>>>
```

- 매개변수에 넘겨주는 자료의 개수를 정하지 않았을 때 튜플의 형태로 데이터를 넘겨준다.
- 튜플이란?
  -> 리스트와 비슷한 개념
  ->리스트는 값을 변경할 수 있는 반면 튜플은 값 변경 불가
- 튜플의 형태로 데이터가 넘어와 for 문에서 변수를 통해 값 참조가능

| 매개변수, 인수

- 옆과 같이 매개변수가 있고, 튜플형태의 매개변수가 있다면 왼쪽의 매개변수를 채운 후에 튜플에 값이 채워짐

- ex) result: '+'가 op에 들어간 후 튜플 num에 1,2,3,4,5가 들어감

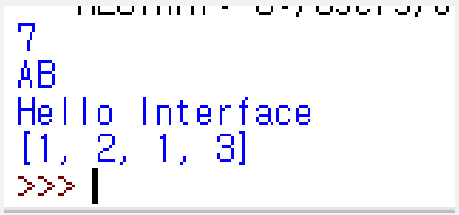



```
def operation(op, *num):
    if op == '+':
        res = 0
        for i in num:
            res += i

    elif op == '*':
        res = 1
        for i in num:
            res *= i

    print(res)

result = operation('+', 1,2,3,4,5)
result2 = operation('*', 1,2,3,4,5)
```

| 리턴 값

```
def add(a,b):
    return a+b

sum = add(3,4)
str = add('A', 'B')
str2 = add('Hello ', 'Interface')
list = add([1,2], [1,3])

print(sum)
print(str)
print(str2)
print(list)
```

- 함수는 값을 반환할 수 있음 (뱉을 수 있음)
- 반환하는 값의 자료형은 그 데이터의 형에 맞춰서 자동으로 변경됨
- ex) add(3,4)를 하면 return에 의해 이 부분이 7로 바뀌어서 sum = 7이 됨



| 리턴 값

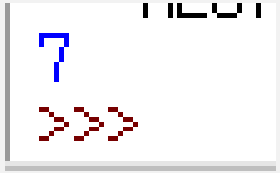
```
def operation(a,b):
    return a+b, a-b, a*b, a/b

plus, minus, mul, div = operation(10,5)

print(plus, minus)
print(mul, div)
```

```
15 5
50 2.0
>>>
```

- return을 할 때 값을 여러개 넘겨줄 수 있음
- 이때도 마찬가지로 튜플의 형태로 리턴되어서 각 변수에 차례대로 매핑됨

| 리턴 값

```
def operation(a,b):
    return a+b
    return a*b

res = operation(3,4)
print(res)
```

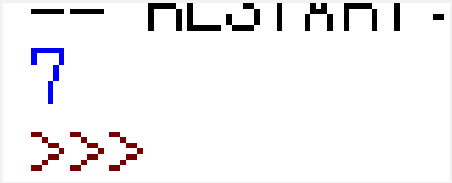


- return을 만나면 함수가 종료됨
- 따라서 옆의 함수는 return a+b를 만나서 함수가 끝났으므로 아래 return a*b는 실행될 수 없음

**return을 만나면 함수가 끝난다는 특성을 활용하여 조건을 만족할 때 return을 넣어서 함수를 끝나게 만들수도 있음.

global 변수

```
def add(a,b):
    global sum

    sum = a+b

add(3,4)
print(sum)
```



- 함수내에서 선언한 변수는 함수가 끝나면 자동으로 소멸됨 -> 함수에서 사용한 변수를 외부에서도 사용하고 싶음 -> 전역변수화
- 이 변수를 사용하기 위해서는 변수 이름 앞에 global 키워드를 붙여서 사용
- global 키워드를 붙이면 함수가 끝나고 나서도 해당 변수가 사라지지 않아 다른 곳에서도 사용가능
- 그러나 무분별한 global 변수의 활용은 잘못된 참조등이 발생할 수 있으므로 알맞게 사용하는게 중요

3. 기타

| 문자열 함수

- 위치반환: index, find
- 개수 반환: count
- 삽입: join
- 문자열 변환: upper, lower, replace
- 공백 제거: lstrip, rstrip, strip
- 나누기: split

| index, find

```
str = "Hello Interface"

print(str.find('H'))
print(str.find('G'))

print(str.index('I'))
print(str.index('Z'))
```

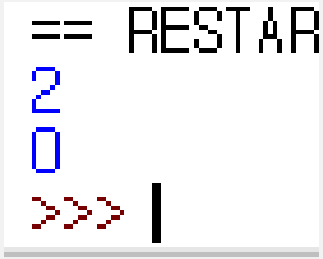
- str.find(char), str.index(char)
- 두 함수모두 문자열에 char가 있는지 확인 후 인덱스 반환
- find 함수는 char가 없을경우 -1 반환
- index 함수는 char가 없을경우 에러 발생시킴

```
0
-1
6
Traceback (most recent call last):
  File "C:/Users/onsae/AppData/Local/Programs
, in <module>
    print(str.index('Z'))
ValueError: substring not found
>>>
```

count

```
str = "Hobby"

print(str.count('b'))
print(str.count('G'))
```

- str.count(char)
- 문자열내에 char가 몇번 나오는지 리턴
- 한번도 나오지 않을경우 0 리턴



| join

```
str = 'ABCD'
str2 = ['A', 'B', 'C']
sep = ','
sep2 = 'Hi'

print(sep.join(str))
print(sep2.join(str))
print(sep.join(str2))
```

```
== RESTART: C:
A,B,C,D
AHiBHiCHiD
A,B,C
>>>
```

- separator.join(object)
- object의 각 사이마다 separator를 삽입
- 오브젝트에는 문자열, 리스트등이 올 수 있음

| upper, lower

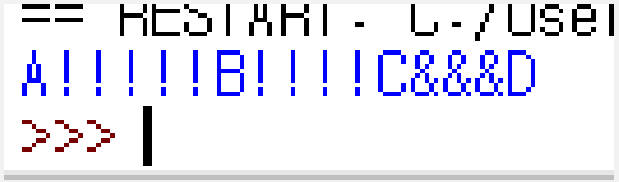
```
str = 'aBcdeFg'
str2 = 'AbCDEfG'

print(str.upper())
print(str2.lower())
```

- str.upper(), str.lower()
- upper는 str에 있는 모든 문자를 대문자로 바꿔줌
- lower는 str에 있는 모든 문자를 소문자로 바꿔줌

replace

```
str = 'A,,,,,B,,,,C...D'

str = str.replace(',', '!')
str = str.replace('.', '&')

print(str)
```

- str.replace(a, b)
- 문자열 내에 있는 모든 a라는 문자를 b라는 문자로 바꿔줌



| lstrip, rstrip, strip

```
s1 = '    ABC'
s2 = 'ABC    '
s3 = '   ABC   '

print(s1.lstrip())
print(s2.rstrip())
print(s3.strip())
```

```
ABC
ABC
ABC
>>>
```

- str.lstrip(), str.rstrip(), str.strip()
- lstrip 함수는 문자열의 가장 맨 왼쪽의 모든 공백들을 삭제
- rstrip 함수는 문자열의 가장 맨 오른쪽의 모든 공백들을 삭제
- strip 함수는 문자열의 양 옆의 모든 공백들을 삭제
- 공백 뒤나 앞에 문자가 있을시 삭제X

| split

```
str = "Life is to short"
str2 = 'a:b:c:d'

str = str.split()
str2 = str2.split(':')

print(str)
print(str2)
```

```
== RESTART: C:/Users/onsa
[1, 2, 3, 4, 5, [4, 5]]
>>>
```

- str.split(a)
- 문자열을 a라는 문자를 기준으로 구분해서 리스트를 만듬
- a에 아무것도 안들어오면 공백을 기준으로 구분해서 리스트 생성

| 매개기본

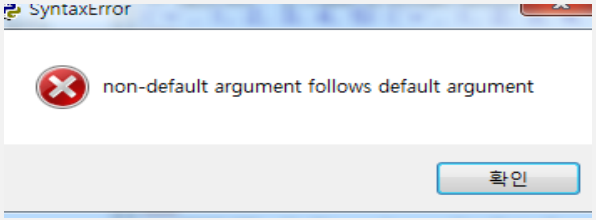
```
def operation(a,b,op='+'):
    if op=='+':
        return a+b
    elif op=='*':
        return a*b

print(operation(1,2))
print(operation(3,4,'+'))
print(operation(5,6,'*'))
```

```
>>> RESTART: C
3
7
30
>>>
```

- 매개변수도 초기화를 시킬 수 있음
- 옆과 같이 선언하면 매개변수의 개수가 부족해도 함수호출 가능
- 첫번째 함수 호출을 보면 op 부분에 값이 넘겨지지 않았으므로 기본값인 +가 들어감
- 기본값은 언제든지 변경가능

| 매개기본

```
def operation(a,op='+',b):
    if op=='+':
        return a+b
    elif op=='*':
        return a*b

print(operation(1,2))
print(operation(3,4,'+'))
print(operation(5,6,'*'))
```



- 매개변수의 기본값은 항상 맨 오른쪽부터 차례대로 채워줘야 함
- 매개변수를 중간부터 초기화 하면 함수를 호출할 때 인수를 어느부분에 넣어야 할지 애매해짐
- ex)옆의 첫번째 호출을 보면 a에는 1이 그리고 op에 2가 들어가서 b에는 값이 들어가지 않아 에러 발생

매개기본

```
add = lambda a,b: a+b
mul = lambda a,b: a*b

print(add(1,2))
print(mul(5,10))
```

```
def add(a,b):
    return a+b

def mul(a,b):
    return a*b

print(add(1,2))
print(mul(5,10))
```

- lambda는 함수를 한 줄로 간단히 정의할 때 사용
- 왼쪽과 오른쪽은 같은 내용