Chapter 9

* Python Built-In Functions

* Parameter Passing

Python Functions

- Python Built-in Functions:
  - input(), print(), len(), abs(), set(), …..
- User Defined Functions
- Functions belonging to Python Built-In Data Types

```
>>> L = [3, 5, 5]
>>> L.append(4)
```

```
>>> S = {3, 5, 6}
>>> S.remove(5)
```

- Functions belonging to User-Defined Classes

```
>>> class Box(object):
        def  calc_space(self):
            ……
```

```
>>> bbb = Box()
>>> space_value = bbb.calc_space()
```

- Functions from Other Modules (consisting of Functions and Classes)

```
>>>  import sam_module
>>>   x = sam_module.func1( )
```

- Functions from Python Standard Library (consisting of Functions and Classes)

```
>>>  import math
>>>  x = math.log(10)
```

Python Built-in Functions

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	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()	
delattr()	hash()	memoryview()	set()	

Python Built-in Functions: abs( ), all( )

abs(x)는 어떤 숫자를 입력으로 받았을 때, 그 숫자의 절대값을 돌려주는 함수

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```

all(x)는 반복 가능한 (iterable) 자료형 x를 입력변수로 받으며, x가 모두 True이면 True, False가 한개라도 있으면 False를 return

```
>>> all([1, 2, 3])
True
```

리스트 자료형 [1, 2, 3]은 모든 요소가 참이므로 True를 리턴한다.

```
>>> all([1, 2, 3, 0])
False
```

리스트 자료형 [1, 2, 3, 0] 중에서 요소 0은 거짓이므로 False를 리턴

값	T or F
"python"	True
""	False
[1, 2, 3]	True
[]	False
()	False
{}	False
1	True
0	False
None	False

Python Built-in Functions: any( )

any(x)는 반복 가능한 (iterable) 자료형 x를 입력변수로 받으며, x가 한개라도 True이면 True, 모든 x가 False면 False를 return 한다

```
>>> any([1, 2, 3, 0])
True
```

리스트 자료형 [1, 2, 3, 0] 중에서 1, 2, 3이 참이므로 True를 리턴한다.

```
>>> any([0, ""])
False
```

리스트 자료형 [0, ""]의 요소 0과 ""은 모두 거짓이므로 False를 리턴한다.

Python Built-in Functions: chr( ), ord( )

chr(i)는 Ascii code 숫자값을 (0..256) 입력으로 받아 해당 Character를 출력하는 함수

```
>>> chr(97)
'a'
>>> chr(48)
'0'
```

ord(c)는 문자의 아스키 코드값을 리턴하는 함수이다.

(※ ord 함수는 chr 함수와 반대이다.)

```
>>> ord('a')
97
>>> ord('0')
48
```

참고

Representations of Characters [1/2]

256 Characters Encoding

ASCII	0000 0001	0001 0000	0001 0001	0001 0010	0001 0011	0001 0100	0001 0101	0001 0110	0001 0111	0001 1000	0001 1001	0001 1010	0001 1011	0001 1100	0001 1101	0001 1110	0001 1111
0000	$^{N}_{U}$	$^{S}_{H}$	$^{S}_{X}$	$^{E}_{X}$	$^{E}_{T}$	$^{E}_{Q}$	$^{A}_{K}$	$^{B}_{L}$	$^{B}_{S}$	$^{H}_{T}$	$^{L}_{F}$	$^{Y}_{T}$	$^{F}_{F}$	$^{C}_{R}$	$^{S}_{0}$	$^{S}_{I}$	
0001	$^{D}_{L}$	$^{D}_{1}$	$^{D}_{2}$	$^{D}_{3}$	$^{D}_{4}$	$^{N}_{K}$	$^{S}_{Y}$	$^{E}_{\Sigma}$	$^{C}_{N}$	$^{E}_{M}$	$^{S}_{B}$	$^{E}_{C}$	$^{F}_{S}$	$^{G}_{S}$	$^{R}_{S}$	$^{U}_{S}$	
0010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	$^{D}_{T}$	
1000	$^{8}_{0}$	$^{8}_{1}$	$^{8}_{2}$	$^{8}_{3}$	$^{I}_{N}$	$^{N}_{L}$	$^{S}_{S}$	$^{E}_{S}$	$^{H}_{S}$	$^{H}_{J}$	$^{V}_{S}$	$^{P}_{D}$	$^{P}_{V}$	$^{R}_{I}$	$^{S}_{2}$	$^{S}_{3}$	
1001	$^{D}_{C}$	$^{P}_{1}$	$^{P}_{Z}$	$^{S}_{E}$	$^{C}_{C}$	$^{M}_{M}$	$^{S}_{P}$	$^{E}_{P}$	$^{O}_{S}$	$^{O}_{O}$	$^{O}_{A}$	$^{C}_{S}$	$^{S}_{T}$	$^{O}_{S}$	$^{P}_{M}$	$^{A}_{P}$	
1010	$^{A}_{0}$	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	
1011	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	
1100	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	
1101	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
1110	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	
1111	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	

Figure 7.3 ASCII, the American Standard Code for Information Interchange.

Note: The original 7-bit ASCII is the top half of the table; the whole table is known as Extended ASCII (ISO-8859-1). The 8-bit symbol for a letter is the four row bits followed by the four column bits (e.g., A = 0100 0001, while z = 0111 1010). Characters shown as two small letters are control symbols used to encode nonprintable information (e.g., $^{B}_{S}$ = 0000 1000 is backspace). The bottom half of the table represents characters needed by Western European languages, such as Icelandic's eth (ð) and thorn (þ).

لماذا لا يتكلمون اللّغة العربية فحسب؟
Защо те просто не могат да говорят **български**?
Per què no poden simplement parlar en **català**?
他們爲什麼不說中文（台灣）？
Proč prostě nemluví **česky**?
Hvorfor kan de ikke bare tale **dansk**?
Warum sprechen sie nicht einfach **Deutsch**?
Μα γιατί δεν μπορούν να μιλήσουν **Ελληνικά**;
Why can't they just speak **English**?
¿Por qué no pueden simplemente hablar en **castella**
Miksi he eivät yksinkertaisesti puhu **suomea**?
Pourquoi, tout simplement, ne parlent-ils pas **franç**
למה הם פשוט לא מדברים **עברית**?
Miért nem beszélnek egyszerűen **magyarul**?
Af hverju geta þeir ekki bara talað **íslensku**?
Perché non possono semplicemente parlare **italian**
なぜ、みんな日本語を話してくれないのか
의 모든 사람들이 한국어를 이해한다면 얼마나
Waarom spreken ze niet gewoon **Nederlands**?
Hvorfor kan de ikke bare snakke **norsk**?
Dlaczego oni po prostu nie mówią po **polsku**?
Porque é que eles não falam em **Português (do Br**
Oare ăştia de ce nu vorbesc **româneşte**?
Почему же они не говорят **по-русски**?
Zašto jednostavno ne govore **hrvatski**?
Pse nuk duan të flasin vetëm **shqip**?
Varför pratar dom inte bara **svenska**?
ทำไมเขาถึงไม่พูดภาษาไทย
Neden **Türkçe** konuşamıyorlar?

e 7.4 "Why can't they just speak _____?" A
Veb page, www.trigeminal.com/samples/pro
displaying that question expressed in more th
ages. Can you name all of them in this partial

97번째 char? ➔ 0110 0001 ➔ "a"

참고

Representations of Characters [2/2]

- **ASCII: 7 bits for 128 characters (1960년)**
- **E-ASCII: 8 bits for 256 characters (1963년)** ➔ 보통 **Ascii**하면 이것을 의미
- **유니코드**(Unicode)
  - 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현되도록 설계된 산업 표준
  - 유니코드 협회(Unicode Consortium)
- **UTF** (Unicode Transformation Format) **-8** : 32bits (1992년)
  - A character encoding for all possible characters in Unicode
  - variable-length of 4 bytes code whose 1st byte is E-ASCII
  - Dominant character encoding for the World Wide Web, accounting for 84.6% of all Web pages

E-ASCII	.....	....	.....

Python Built-in Functions: dir( ), divmod( )

dir은 객체가 자체적으로 가지고 있는 변수나 함수를 보여 준다. 아래 예는 리스트와 딕셔너리 객체의 관련 함수들(메서드)을 보여 주는 예이다. 우리가 02장에서 살펴보았던 자료형 관련 함수들을 만나볼 수 있을 것이다.

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```

divmod(a, b)는 2개의 숫자를 입력으로 받는다. 그리고 a를 b로 나눈 몫과 나머지를 튜플 형태로 리턴하는 함수이다.

```
>>> divmod(7, 3)
(2, 1)
>>> divmod(1.3, 0.2)
(6.0, 0.09999999999999978)
```

7 / 3 ➔ 2.3333
7 // 3 ➔ 2
7 % 3 ➔ 1

Python Built-in Functions: enumerate( )

enumerate는 "열거하다"라는 뜻이다. 이 함수는 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력으로 받아 인덱스 값을 포함하는 enumerate 객체를 리턴한다.

```
>>> for i, name in enumerate(['body', 'foo', 'bar']):
...     print(i, name)
...
0 body
1 foo
2 bar
```

```
>>> x, y, z = enumerate( ['body',  'foo',  'bar'] )
>>> print(x, y, z)
(0, 'body') (1, 'foo') (2, 'bar')
```

```
>>> x, y, z = enumerate( {'body',  'foo',  'bar'} )
>>> print(x, y, z)
(0, 'body') (1, 'foo') (2, 'bar')
```

Python Built-in Functions: eval( )

eval(expression)은 실행 가능한 문자열(1+2, 'hi' + 'a' 같은 것)을 입력으로 받아 문자열을 실행한 결과값을 리턴하는 함수이다.

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```

( ) 안에 " xxx " 가 있으면 quotation mark를 제거하고 xxx를 실행한다.

보통 eval은 입력받은 문자열로 파이썬 함수나 클래스를 동적으로 실행하고 싶은 경우에 사용된다.

```
def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = (9/5) * celsius + 32
    print("The temperature is ",fahrenheit," degrees Fahrenheit.")
```

Python Built-in Functions: id( )

id(object)는 객체를 입력받아 객체의 고유 주소값(레퍼런스)을 리턴하는 함수이다.

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

위 예의 3, a, b는 고유 주소값이 모두 135072304이다. 즉, 3, a, b가 모두 같은 객체를 가리키고 있음을 알 수 있다.

Python Built-in Functions: max( ), min( )

max(iterable)는 인수로 반복 가능한 자료형을 입력받아 그 최대값을 리턴하는 함수이다.

```
>>> max([1, 2, 3])
3
>>> max("python")
'y'
```

Try!

>>> max(3)
... TypeError.....

min(iterable)은 max 함수와 반대로, 인수로 반복 가능한 자료형을 입력받아 그 최소값을 리턴하는 함수이다.

```
>>> min([1, 2, 3])
1
>>> min("python")
'h'
```

Python Built-in Functions: oct( ), hex( )

oct(x)는 정수 형태의 숫자를 8진수 문자열로 바꾸어 리턴

```
>>> oct(34)
'0o42'
>>> oct(12345)
'0o30071'
```

```
>>> int(0o30071)
Out[1]: 12345

>>> hex(0o30071)
Out[2]: '0x3039'

>>> int(0xea)
Out[3]: 234
```

hex(x)는 정수값을 입력받아 16진수(hexadecimal)로 변환하여 리턴

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

Python Built-in Functions: open( )

open(filename, [mode])은 "파일 이름"과 "읽기 방법"을 입력받아 파일 객체를 리턴하는 함수이다.
읽기 방법(mode)이 생략되면 기본값인 읽기 전용 모드(r)로 파일 객체를 만들어 리턴한다.

mode	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

```
>>> fread = open("read_mode.txt", 'r')
>>> fread2 = open("read_mode.txt")
```

즉, 모드 부분이 생략되면 기본값으로 읽기 모드인 r을 갖게 된다.

다음은 추가 모드(a)로 파일을 여는 예이다.

```
>>> fappend = open("append_mode.txt", 'a')
```

b는 w, r, a와 함께 사용된다.

```
>>> f = open("binary_file", "rb")
```

위 예의 rb는 "바이너리 읽기 모드"를 의미한다.

** JPG화일이나 Exec화일같은
것을 open할때는 rb mode로

Python Built-in Functions: range( )

range([start,] stop [,step])는 for문과 함께 자주 사용되는 함수이다. 이 함수는 입력받은 숫자에 해당되는 범위의 값을 반복 가능한 객체로 만들어 리턴한다.

인수가 1개인 경우

```
>>> print(range(5))
range(0,5)
```

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

인수가 2개인 경우:

2개의 입력인수는 시작과 끝

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
```

인수가 3개인 경우

세 번째 인수는 숫자 사이의 거리를 말한다.

```
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Python Built-in Functions: pow( ), str()

pow(x, y)는 x의 y 제곱한 결과값을 리턴하는 함수

```
>>> pow(2, 4)
16
>>> pow(3, 3)
27
```

str(object)은 문자열 형태로 객체를 변환하여 리턴하는 함수

```
>>> str(3)
'3'
>>> str('hi')
'hi'
>>> str('hi'.upper())
'HI'
```

Python Built-in Functions: sorted( )

sorted(iterable) 함수는 입력값을 정렬한 후 그 결과를 리스트로 리턴

```
>>> sorted([3, 1, 2])
[1, 2, 3]
>>> sorted(['a', 'c', 'b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3, 2, 1))
[1, 2, 3]
```

Sorting in List

```
>>> a = [3, 1, 2]
>>> result = a.sort()
>>> print(result)
None
>>> a
[1, 2, 3]
```

More on sorted()

You can also use the `list.sort()` method of a list. It modifies the list in-place (and returns None to avoid confusion). I
you don't need the original list, it's slightly more efficient.

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
```

Another difference is that the `list.sort()` method is only defined for lists. In contrast, the `sorted()` function accepts any iterable.

```
>>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
[1, 2, 3, 4, 5]
```

dictionary에 sorted( ) 적용하면 dictionary의 key들에 적용

```
>>> ListB = [24, 13, -15, -36, 8, 22, 48, 25, 46, -9]
>>> print(sorted(ListB, key=int, reverse=True))
[48, 46, 25, 24, 22, 13, 8, -9, -15, -36]
```

>>> print(ListB) # try this

Python Built-in Functions: reversed( )

```
1    # for string
2    seqString = 'Python'
3    print(list(reversed(seqString)))
4
5    # for tuple
6    seqTuple = ('P', 'y', 't', 'h', 'o', 'n')
7    print(list(reversed(seqTuple)))
8
9    # for range
10   seqRange = range(5, 9)
11   print(list(reversed(seqRange)))
12
13   # for list
14   seqList = [1, 2, 4, 3, 5]
15   print(list(reversed(seqList)))
```

```
['n', 'o', 'h', 't', 'y', 'P']
['n', 'o', 'h', 't', 'y', 'P']
[8, 7, 6, 5]
[5, 3, 4, 2, 1]
```

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];

aList.reverse();
print("List : ", aList )
```

```
List :  ['xyz', 'abc', 'zara', 'xyz', 123]
```

Python Built-in Functions: tuple( ), type( )

tuple(iterable)은 반복 가능한 자료형을 입력받아 튜플 형태로 바꾸어 리턴하는 함수이다. 만약 튜플이 입력으로 들어오면 그대로 리턴한다.

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
(1, 2, 3)
```

type(object)은 입력값의 자료형이 무엇인지 알려주는 함수이다.

```
>>> type("abc")
<class 'str'>
>>> type([ ])
<class 'list'>
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'>
```

Python Built-in Functions: int( ), len()

int(x)는 문자열 형태의 숫자나 소수점이 있는 숫자 등을 정수 형태로 리턴하는 함수로, 정수를 입력으로 받으면 그대로 리턴한다.

```
>>> int('3')
3
>>> int(3.4)
3
```

int(x, radix)는 radix 진수로 표현된 문자열 x를 10진수로 변환하여 리턴한다.

len(s)은 입력값 s의 길이(요소의 전체 개수)를 리턴하는 함수이다.

```
>>> len("python")
6
>>> len([1,2,3])
3
>>> len((1, 'a'))
2
```

Python Built-in Functions: isinstance( )

isinstance(object, class)는 첫 번째 인수로 인스턴스, 두 번째 인수로 클래스 이름을 받는다. 입력으로 받은 인스턴스가 그 클래스의 인스턴스인지를 판단하여 참이면 True, 거짓이면 False를 리턴한다.

```
>>> class Person: pass
...
>>> a = Person()
>>> isinstance(a, Person)
True
```

위의 예는 a가 Person 클래스에 의해서 생성된 인스턴스임을 확인시켜 준다.

```
>>> b = 3
>>> isinstance(b, Person)
False
```

b는 Person 클래스에 의해 생성된 인스턴스가 아니므로 False를 리턴한다.

Python Built-in Functions: list( )

list(s)는 반복 가능한 자료형 s를 입력받아 리스트로 만들어 리턴하는 함수이다.

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

list 함수에 리스트를 입력으로 주면 똑같은 리스트를 복사하여 돌려준다.

```
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```

Python Built-in Function: round(), slice()

```
round(80.23456, 2) :  80.23
round(100.000056, 3) :  100.0
round(-100.000056, 3) :  -100.0
```

The syntax of slice() are:

```
slice(stop)
slice(start, stop, step)
```

** slice() returns indices

- **start** - starting integer where the slicing of the object starts
- **stop** - integer until which the slicing takes place. The slicing stops at index **stop - 1**.
- **step** - integer value which determines the increment between each index for slicing

```
1   pyString = 'Python'
2
3   # contains indices (0, 1, 2)
4   # i.e. P, y and t
5   sObject = slice(3)
6
7   print(pyString[sObject])
8
9   # contains indices (1, 3)
10  # i.e. y and h
11  sObject = slice(1, 5, 2)
12
13  print(pyString[sObject])
```

```
Pyt
yh
```

slice objects

slice(3)
slice(None, 3, None)
slice(1, 5, 2)

range( ) 와 유사해보이지만 쓰임새가 다른다.

Python Built-in Function: iter(), next()

```
# define a list
my_list = [4, 7, 0, 3]

# get an iterator using iter()
my_iter = iter(my_list)

## iterate through it using next()

#prints 4
print(next(my_iter))

#prints 7
print(next(my_iter))

## next(obj) is same as obj.__next__()

#prints 0
print(my_iter.__next__())

#prints 3
print(my_iter.__next__())

## This will raise error, no items left
next(my_iter)
```

iter( ) : iterable object를 iterator object로 만들어주는 function

iterator object는 next()를 쓸수있는 object!

For loop으로 iterable object를 접근하는것과 유사하다

나중에 User-defined Class의 object들을 순차적으로 꺼내서 작업을 하려면 (즉, next() 를 쓰려면), iter()을 이용해서 iterator object를 만들어줘야 하는데, User-defined Class의 내부에 __iter__( )과 __next__( )를 만드는것을 User의 몫이다

Towards Sophisticated Coding

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	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()	
delattr()	hash()	memoryview()	set()	

- Lamda function
- Functools module: reduce( )

Python 람다(lambda) 함수 [1/2] (not built-in function)

- 람다 함수는 람다 대수학(lambda calculus)에서 유래된 이름
- 익명함수 (Function Name이 없는 Function)
  - 일회용 쓰고 버리는 형태의 Function
  - 함수를 이름없이 그냥 inline으로 쓰면 기본적으로 coding량이 줄어드는 효과
  - 보통 A → B →C 같는경우에 중간에 B에서 function k를 호출하면,

    A → B → function k → B → C 이런식으로 중간에 code reading흐름이 끊어짐

Syntax

```
lamda parameter1, parameter2,..  : function definition
```

```
>> a = lambda x , y : x * y
>> print a(3,4)
>> 12
```

```
>>> sum = lambda a, b: a+b
>>> sum(3,4)
7
```

=

```
>>> def sum(a, b):
...     return a+b
>>> sum(3,4)
7
```

Python 람다(lambda) 함수 [2/2]
(not built-in function)

그렇다면 def가 있는데 왜 lambda라는 것이 나오게 되었을까? 이유는 간단하다. lambda는 def 보다 간결하게 사용할 수 있기 때문이다. 또한 lambda는 def를 사용할 수 없는 곳에도 사용할 수 있다. 다음 예제에서 리스트 내에 lambda가 들어간 경우를 살펴보자.

```
>>> myList = [lambda a,b:a+b, lambda a,b:a*b]
>>> myList
[at 0x811eb2c>, at 0x811eb64>]
```

```
>>> myList[0]
at 0x811eb2c>
```

```
>>> myList[0](3,4)
7
```

```
>>> myList[1](3,4)
12
```

Python Built-in Functions: map( ) [1/4]

- map(function, iterable) applies a function to all the items in the list
- map(lambda expression, iterable) applies lambda expression to all the items in the list

```
def two_times(numberList):
    result = [ ]
    for number in numberList:
        result.append(number*2)
    return result

result = two_times([1, 2, 3, 4])
print(result)
```

two_times 함수는 리스트 요소를 입력받아 각 요소에 2를 곱한 결과값을 돌려준다

결과값: [2, 4, 6, 8]

```
>>> def two_times(x): return x*2
...
>>> list(map(two_times, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

map( ) & function 을 이용해서 간소화

```
>>> list(map(lambda a: a*2, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

map( ) & lamda을 이용해서 더욱 간소화

Python Built-in Functions: map( ) [2/4]

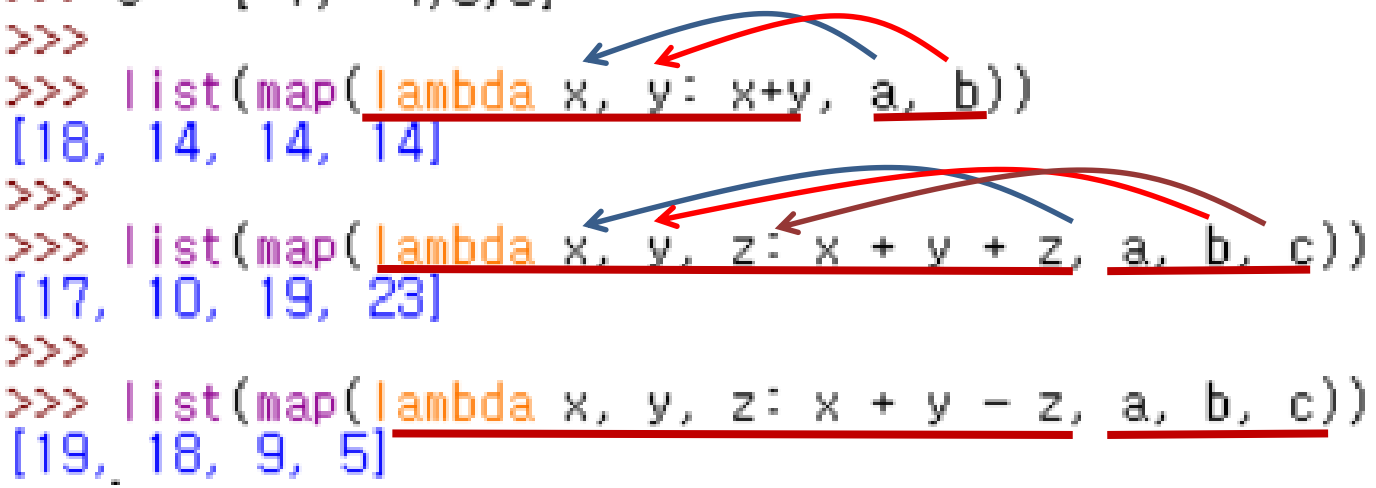
```
items = [1, 2, 3, 4, 5]
squared = []
for i in items:
    squared.append(i**2)
```

```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, items))
```

Python Built-in Functions: map( ) [3/4]

- map( ) can be applied to more than one list
- The lists have to have the same length
- map( ) will apply its lambda function to elements of the argument lists
  - Its first applies to the elements with the 0th index
  - Then, to the elements with the 1st index until the n-th index is reached

```
>>> a = [1,2,3,4]
>>> b = [17,12,11,10]
>>> c = [-1, -4,5,9]
>>>
>>> list(map(lambda x, y: x+y, a, b))
[18, 14, 14, 14]
>>>
>>> list(map(lambda x, y, z: x + y + z, a, b, c))
[17, 10, 19, 23]
>>>
>>> list(map(lambda x, y, z: x + y - z, a, b, c))
[19, 18, 9, 5]
```



Python Built-in Functions: map( ) [4/4]

- Instead of a list of inputs we can even have a list of functions!

```
def multiply(x):
    return (x*x)
def add(x):
    return (x+x)

funcs = [multiply, add]
for i in range(5):
    value = list(map(lambda x: x(i), funcs))
    print(value)

# Output:
# [0, 0]
# [1, 2]
# [4, 4]
# [9, 6]
# [16, 8]
```

funcs list가 function name들을 가지고 있으며, x에 function 이름을 공급

Python Built-in Functions: filter( ) [1/2]

filter란 무엇인가를 걸러낸다는 뜻으로, filter 함수도 동일한 의미를 가진다. filter 함수는 첫 번째 인수로 함수 이름을, 두 번째 인수로 그 함수에 차례로 들어갈 반복 가능한 자료형을 받는다. 그리고 두 번째 인수인 반복 가능한 자료형 요소들이 첫 번째 인수인 함수에 입력되었을 때 리턴값이 참인 것만 묶어서(걸러내서) 돌려준다.

```
#positive.py
def positive(l):
    result = []
    for i in l:
        if i > 0:
            result.append(i)
    return result

print(positive([1,-3,2,0,-5,6]))
```

결과값: [1, 2, 6]

filter 함수를 이용하면 위의 내용을 아래와 같이 간단하게 작성

```
#filter1.py
def positive(x):
    return x > 0

print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
```

True가 되는 element만 filtering

결과값: [1, 2, 6]

Python Built-in Functions: filter( ) [2/2]

filter( ) with lamda function

```
number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
print(less_than_zero)

# Output: [-5, -4, -3, -2, -1]
```

True가 되는 element만 filtering

The filter resembles a for loop but it is a builtin function and faster.

```
>>> fib = [0,1,1,2,3,5,8,13,21,34,55]
>>> result = list( filter(lambda x: x % 2 == 0, fib) )
>>> print(result)
[1, 1, 3, 5, 13, 21, 55]
>>> result = list( filter(lambda x: x % 2, fib) )
>>> print(result)
[0, 2, 8, 34]
>>>
```

reduce( ) from functools module [1/3]

- The function reduce(func, seq) continually applies the function func() to the sequence seq
  - It returns a single value

- If seq = $[S_1, S_2, S_3, \ldots, S_n]$, calling reduce(func, seq) works like this:

  - At first the first two elements of seq will be applied to func, i.e func(S_1, S_2)
    - The list on which reduce() works looks now like this : [func(S_1, S_2), $S_3, \ldots, S_n$]

  - In the next step func will be applied on the previous result and the third element of the list, i.e. func(func(S_1, S_2), S_3)
    - The list looks like this now : [func(func(S_1, S_2), S_3), …, S_n]

  - Continue like this until just one element is left and return this element as the result of reduce()

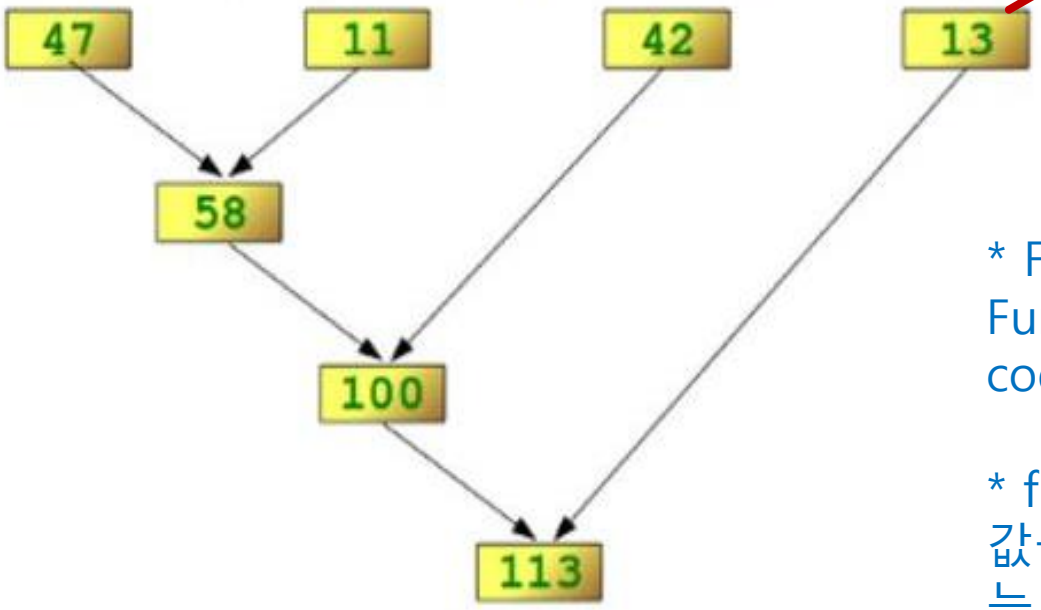
func(……func(func(func(S_1, S_2), S_3), S_4) …, S_n)

reduce( ) from functools module [2/3]

```
>>> from functools import reduce
```

```
>>> reduce(lambda x,y: x+y, [47,11,42,13])
113
```

The following diagram shows the intermediate steps of the calculation:



* Function Composition을 이용한 Functional Programming 방식으로 code를 표현 ➔ 간결한 code!

* filter( ) 와 map( ) 은 return 하는 값들이 여러 개이지만, reduce( ) 는 single value를 return한다

$$func(\ldots\ldots func(\ func(func(S_1, S_2), S_3), S_4)\ \ \ldots, S_n)$$

reduce( ) from functools module [3/3]

```
>>>from functools import reduce
>>>product = reduce((lambda x, y: x * y), [1, 2, 3, 4])

 # Output: 24
```

Determining the maximum of a list of numerical values by using reduce:

```
>>> f = lambda a,b: a if (a > b) else b
>>> reduce(f, [47,11,42,102,13])
102
```

Calculating the sum of the numbers from 1 to 100:

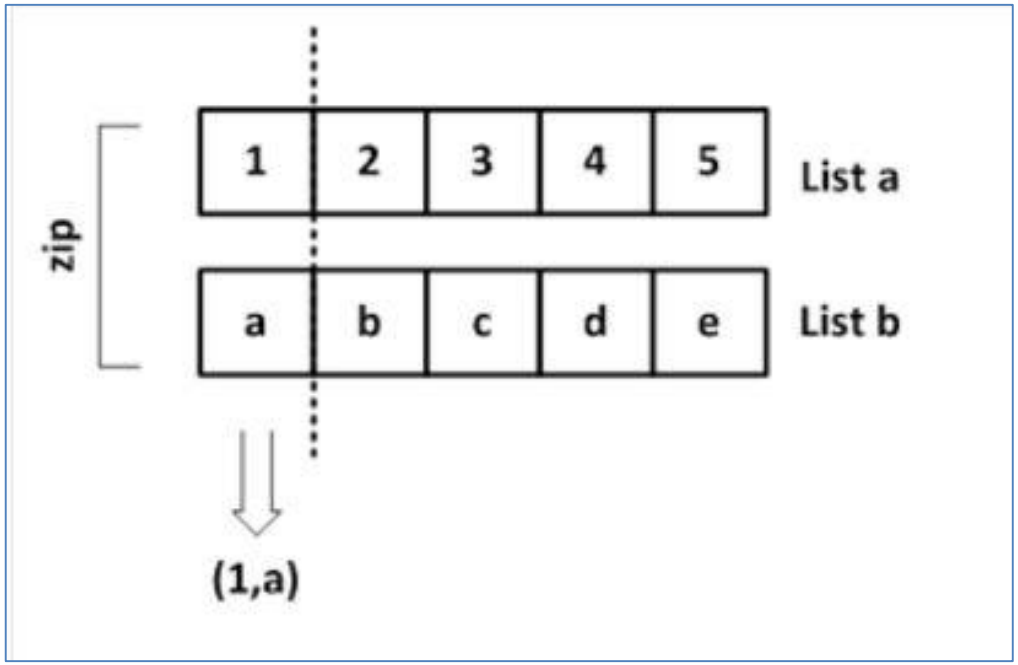
```
>>> reduce(lambda x, y: x+y, range(1,101))
5050
```

Python Built-in Functions: zip( ) [1/2]

- zip( ) 함수는 여러가지 List를 김밥말듯이 말아서 iterator 등의 함수로 slice해서 tuple들을 return한다

- zip( ) 의 parameter들에 들어있는 data의 갯수는 같아야 한다

```
>> a = [1,2,3,4,5]
>> b = ['a','b','c','d','e']
>> for x,y in zip (a,b):
        print( x, y )
```



Python Built-in Functions: zip( ) [2/2]

`zip(iterable*)` 은 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 하는 함수이다.

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

Owing to the built-in functions and some other help

- Now your code should be
  - Sophisticated
  - Descent
  - Powerful

Parameter Passing in Python

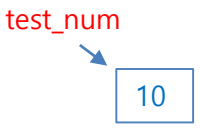
Parameter Passing in Function Call [1/5]

```
#Parameter Passing  Case 1 Numeric Data

def test_function(t_num):
        print("        Now, inside test_function: ")
        print("      t_num    =", t_num,          "        id = ",id(t_num))

        t_num = 4

        print("\n      Still, Inside test_function after changing values: ")
        print("      t_num    =", t_num,          "        id = ",id(t_num))


def my_function():

        test_num = 10

        print("\n **** Before calling test_function: ")
        print("test_num    =", test_num,          "        id = ",id(test_num))

        print("\n **** Calling test_function *****")
        test_function(test_num)

        print("\n **** After calling test_function:  ")
        print("test_num    =", test_num,          "        id = ",id(test_num))


my_function()
```

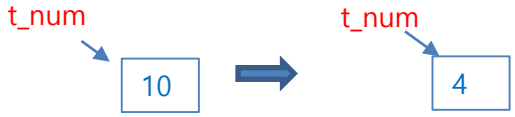
```
 **** Before calling test_function:
test_num    = 10        id =  1635596192

 **** Calling test_function *****
        Now, inside test_function:
      t_num    = 10        id =  1635596192

      Still, Inside test_function after changing values:
      t_num    = 4         id =  1635596096

 **** After calling test_function:
test_num    = 10        id =  1635596192
>>>
```

Int, Float, Char, String, Boolean, Tuple 같은 Immutable Data Type의 값들은 parameter 로 passing 되면서 물리적인 복제 (copy)를 한다

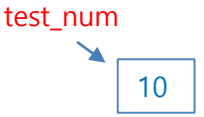
Before Function Call



During Function Call



After Function Call



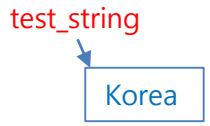
Parameter Passing in Function Call [2/5]

```
#Parameter Passing  Case 2:  Passing String

def test_function(t_string):
        print("        Now, inside test_function: ")
        print("t_string =", t_string,          "              id = ",id(t_string))

        t_string = "Japan"

        print("\n      Still, Inside test_function after changing values: ")
        print("t_string =", t_string,          "              id = ",id(t_string))


def my_function():

        test_string = "Korea"

        print("\n **** Before calling test_function: ")
        print("test_string =", test_string,          "              id = ",id(test_string))

        print("\n **** Calling test_function *****")
        test_function(test_string)

        print("\n **** After calling test_function:  ")
        print("test_string =", test_string,          "              id = ",id(test_string))


my_function()
```

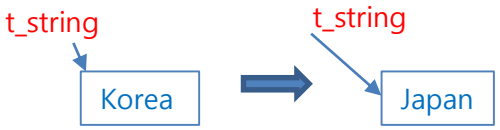
Int, Float, Char, String, Boolean, Tuple 같은 Immutable Data Type의 값들은 parameter 로 passing 되면서 물리적인 복제 (copy)를 한다

```
 **** Before calling test_function:
test_string = Korea              id =  54036576

 **** Calling test_function *****
       Now, inside test_function:
t_string = Korea                 id =  54036576

     Still, Inside test_function after changing values:
t_string = Japan                 id =  54036544

 **** After calling test_function:
test_string = Korea              id =  54036576
>>>
```

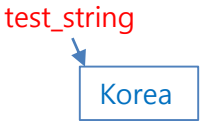
Before Function Call



During Function Call



After Function Call



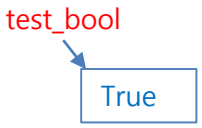
Parameter Passing in Function Call [3/5]

```
#Parameter Passing  Case 3:  Passing Boolean

def test_function(t_bool):
        print("        Now, inside test_function: ")
        print("t_bool   =", t_bool,          "                id = ",id(t_bool))

        t_bool = False

        print("\n      Still, Inside test_function after changing values: ")
        print("t_bool   =", t_bool,          "                id = ",id(t_bool))


def my_function():

        test_bool = True

        print("\n **** Before calling test_function: ")
        print("test_bool   =", test_bool,          "                id = ",id(test_bool))

        print("\n **** Calling test_function *****")
        test_function(test_bool)

        print("\n **** After calling test_function:  ")
        print("test_bool   =", test_bool,          "                id = ",id(test_bool))


my_function()
```

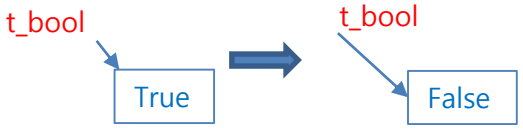
Int, Float, Char, String, Boolean, Tuple 같은 Immutable Data Type의 값들은 parameter로 passing 되면서 물리적인 복제 (copy)를 한다

```
 **** Before calling test_function:
test_bool   = True                id =  1635414864

 **** Calling test_function *****
        Now, inside test_function:
t_bool   = True                id =  1635414864

      Still, Inside test_function after changing values:
t_bool   = False                id =  1635414880

 **** After calling test_function:
test_bool   = True                id =  1635414864
>>>
```

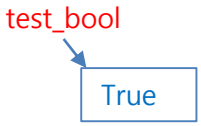
Before Function Call



During Function Call



After Function Call



Parameter Passing in Function Call [4/5]

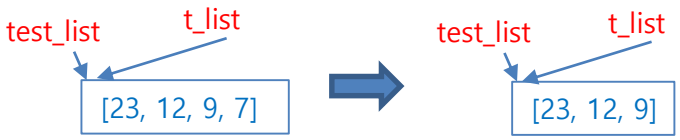
```
#Parameter Passing  Case 4:  Passing List

def test_function(t_list):
        print("        Now, inside test_function: ")
        print("    t_list   =", t_list,          " id = ", id(t_list))

        t_list.pop(3)

        print("\n      Still, Inside test_function after changing values: ")
        print("    t_list   =", t_list,          " id = ", id(t_list))


def my_function():

        test_list = [23, 12, 9, 7]

        print("\n **** Before calling test_function: ")
        print("test_list   =", test_list,          "   id = ",id(test_list))

        print("\n **** Calling test_function *****")
        test_function(test_list)

        print("\n **** After calling test_function:  ")
        print("test_list   =", test_list,          "    id = ",id(test_list))


my_function()
```

```
 **** Before calling test_function:
test_list   = [23, 12, 9, 7]    id =  62083840

 **** Calling test_function *****
       Now, inside test_function:
   t_list   = [23, 12, 9, 7]  id =  62083840

     Still, Inside test_function after changing values:
   t_list   = [23, 12, 9]  id =  62083840

 **** After calling test_function:
test_list   = [23, 12, 9]     id =  62083840
>>>
```

List, Set, Dictionary 들은 parameter 로 passing 되면서 address를 넘겨준다

Before Function Call

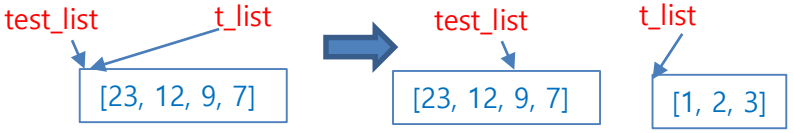
During Function Call



After Function Call

만약 t_list = [1, 2, 3]

During Function Call



Parameter Passing in Function Call [5/5]

```
#Parameter Passing  Case 5:  Passing Object

class FooClass():
        def __init__(self):
                self.test_var = 0

def test_function(t_object):
        print("     Now, inside test_function: ")
        print("     t_object.test_var =", t_object.test_var, "   id = ",id(t_object.test_var))

        t_object.test_var = 10000

        print("\n     Still, Inside test_function after changing values: ")
        print("     t_object.test_var =", t_object.test_var, "  id = ",id(t_object.test_var))


def my_function():

        test_object = FooClass()
        test_object.test_var = 100

        print("\n **** Before calling test_function: ")
        print("test_object.test_var =", test_object.test_var, "   id = ",id(test_object.test_var))

        print("\n **** Calling test_function *****")
        test_function(test_object)

        print("\n **** After calling test_function:  ")
        print("test_object.test_var =", test_object.test_var, "    id = ",id(test_object.test_var))


my_function()
```

User-defined object 들도 parameter 로 passing 되면서 address를 넘겨준다

```
 **** Before calling test_function:
test_object.test_var = 100   id =  1635597632

 **** Calling test_function *****
     Now, inside test_function:
     t_object.test_var = 100    id =  1635597632

     Still, Inside test_function after changing values:
     t_object.test_var = 10000  id =  51365328

 **** After calling test_function:
test_object.test_var = 10000    id =  51365328
```

Parameter Passing in Python Function Call

- Pointer를 허락하는 언어 (예, C) 에서의 Call-By-Value, Call-By-Reference를 Python, Java등에서 비교설명하는 것은 부적절
- Int, Float, Char, String, Boolean, Tuple 같은 Immutable Data Type의 값들은 parameter 로 passing 되면서 물리적인 복제 (copy)를 한다 : Call-by-Value 로 간주
- List, Set, Dictionary 들은 parameter 로 passing 되면서 address를 넘겨준다 : Call-by-Reference 로 간주