

Python Functions

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

```
>>> L = [3, 5, 5]           >>> S = {3, 5, 6}
>>> L.append(4)           >>> S.remove(5)
```
- Functions belonging to User-Defined Classes

```
>>> class Box(object):           >>> bbb = Box()
    def calc_space(self):         >>> space_value = bbb.calc_space()
    .....
```
- Functions from Other Modules (consisting of Functions and Classes)

```
>>> import sam_module
```
- Functions from Python Standard Library (consisting of Functions and Classes)

```
>>> import math
```

Python Built-in Functions [1/16]

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

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

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

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

all(x)은 반복 가능한(iterable) 자료형 x를 입력 인수로 받으며, 이 x가 모두 참이면 True, 거짓이 하나라도 있으면 False를 리턴한다.

```
>>> 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 리턴한다.

Python Built-in Functions: any()

any

any(x)는 x 중 하나라도 참이 있을 경우 True를 리턴하고, x가 모두 거짓일 경우에만 False를 리턴한다. all(x)의 반대 경우라고 할 수 있다.

다음의 예를 보자.

```
>>> 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 code값을 입력으로 받아 그 코드에 해당하는 문자를 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
```

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, 0.099999999999999978)
```

Python Built-in Functions: enumerate()

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
```

Python Built-in Functions: eval()

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)
```

보통 `eval`은 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

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'
```

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 리턴하는 함수이다.

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

hex(x)는 정수값을 입력받아 16진수(hexadecimal)로 변환하여 리턴하는 함수이다.

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

Python Built-in Functions: open()

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

b는 w, r, a와 함께 사용된다.

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

위 예의 rb는 "바이너리 읽기 모드"를 의미한다.

아래 예의 fread와 fread2는 동일한 방법이다.

```
>>> 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')
```

Python Built-in Functions: range()

range

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

인수가 2개일 경우

입력으로 주어지는 2개의 인수는 시작 숫자와 끝 숫자를 나타낸다. 단, 끝 숫자는 해당 범위에 포함되지 않는다는 것에 주의하자.

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

인수가 3개일 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]
```

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

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

pow

pow(x, y)는 x의 y 제곱한 결과값을 리턴하는 함수

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

str

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

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]
```

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

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 `False`를 리턴한다.

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

위의 예는 `a`가 `Person` 클래스에 의해서 생성된 인스턴스임을 확인시켜 준다.

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

`b`는 `Person` 클래스에 의해 생성된 인스턴스가 아니므로 `False`를 리턴한다.

Python Built-in Functions: list()

`list(s)`는 반복 가능한 자료형 `s`를 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]
```

Towards Sophisticated Coding

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<u><code>enumerate()</code></u>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<u><code>filter()</code></u>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<u><code>zip()</code></u>
<code>compile()</code>	<code>globals()</code>	<u><code>map()</code></u>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

- Lambda 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 $A \rightarrow B \rightarrow C$ 같은경우에 중간에 B에서 function k를 호출하면,
 $A \rightarrow B \rightarrow \text{function } k \rightarrow B \rightarrow C$ 이런식으로 중간에 code reading흐름이 끊어짐

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

Sorted builtin-function 같은 경우에 보통 key function을 정해주는데, 제 곱수가 가장 작은 순서대로 소팅을 하고 싶다.

```
>> a = [-1, -8, 3, -4, 2, 5, -7]
>> a.sorted(key= lambda x : x*x, reverse=True)
>> print a
[-1,2,3,-4,5,-7,8]
```

Python 람다(lambda) 함수 [2/2]

lambda (not built-in function)

lambda는 함수를 생성할 때 사용하는 예약어로, def와 동일한 역할을 한다. 보통 함수를 한줄로 간결하게 만들 때 사용한다. 우리말로 "람다"라고 읽고 def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰인다. 사용법은 다음과 같다.

lambda 인수1, 인수2, ... : 인수를 이용한 표현식

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

lambda를 이용한 sum 함수는 인수로 a, b를 받아 서로 더한 값을 돌려준다. 위의 예제는 def를 사용한 아래 함수와 하는 일이 완전히 동일하다.

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

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 applies a function to all the items in the list
`map (function_to_apply, list_of_inputs)`
- Most of the times we want to pass all the list elements to a function one-by-one and then collect the output

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

Map allows us to implement this in a much simpler and nicer way.

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

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

- Most of the times we use lambdas with “map”, so we did the same below. 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]

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

map(f, iterable)은 함수(f)와 반복 가능한(iterable) 자료형을 입력으로 받는다. map은 입력받은 자료형의 각 요소가 함수 f에 의해 수행된 결과를 묶어서 리턴하는 함수이다.

```
# two_times.py
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]
```

Map을 이용해서 간소화

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

lamda을 이용해서 간소화

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

- 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]
>>> map(lambda x,y:x+y, a,b)
[18, 14, 14, 14]
>>> map(lambda x,y,z:x+y+z, a,b,c)
[17, 10, 19, 23]
>>> map(lambda x,y,z:x+y-z, a,b,c)
[19, 18, 9, 5]
```

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

As the name suggests, `filter` creates a list of elements for which a function returns true.
short and concise example:

```
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]

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

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

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])))
```

Python Reduce Function [1/3]

(from functools module)

- The function `reduce(func, seq)` continually applies the function `func()` to the sequence `seq`. It returns a single value.
- If `seq = [S1, S2, S3, ... , Sn]`, calling `reduce(func, seq)` works like this:
 - At first the first two elements of `seq` will be applied to `func`, i.e. `func(S1, S2)`
 - The list on which `reduce()` works looks now like this : `[func(S1, S2), S3, ..., Sn]`
 - In the next step `func` will be applied on the previous result and the third element of the list, i.e. `func(func(S1, S2), S3)`
 - The list looks like this now : `[func(func(S1, S2), S3), ..., Sn]`
 - Continue like this until just one element is left and return this element as the result of `reduce()`

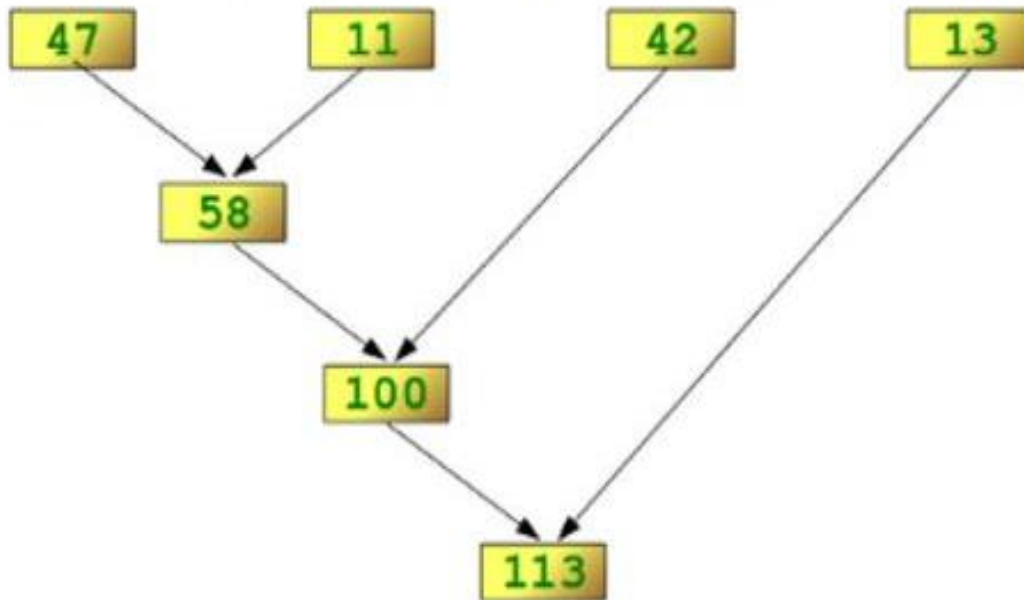
Python Reduce Function [2/3]

(from functools module)

```
>>> 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:



Python Reduce Function [3/3]

(from functools module)

```
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
>>>
```

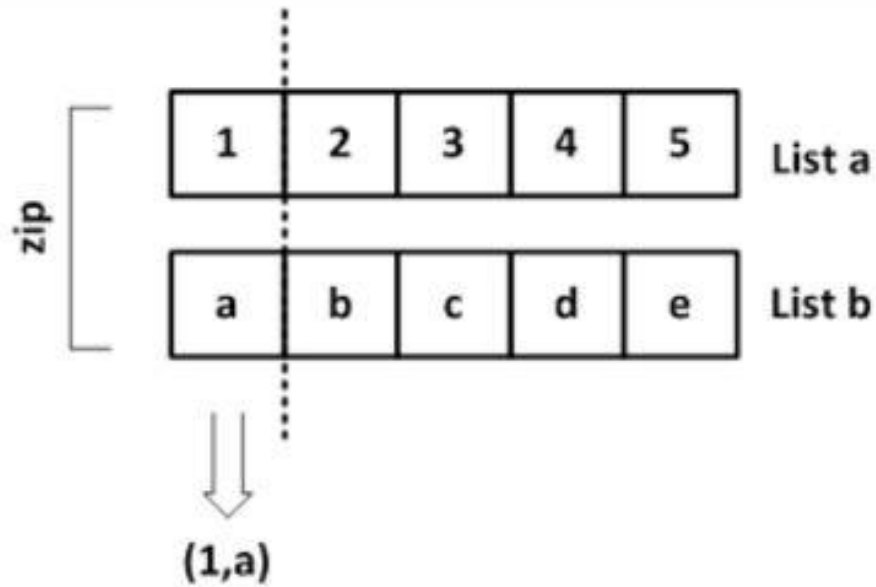
Calculating the sum of the numbers from 1 to 100:

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

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

- zip 함수는 보통 List 여러개를 slice 할때 사용
- 여러가지 List를 김밥말듯이 말아서
- 그리고 iterator 등의 함수로 slice

```
>> 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

`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