Week 03. Python 기초 - 3

Python 내장 함수

Type

`type(object)` 은 입력값의 자료형이 무엇인지 알려주는 함수이다.

In [1]:

```
print(type("abc"))
print(type([ ]))
print(type(open("test", 'w')))
```

```
<class 'str'>
<class 'list'>
<class '_io.TextIOWrapper'>
```

abs(절대값)

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

In [2]:

```
print(abs(3))
print(abs(-3))
print(abs(-1.2))
```

```
3
3
1.2
```

dir (객체의 변수와 함수)

`dir` 은 객체가 자체적으로 가지고 있는 변수나 함수를 보여 준다.

In [3]:

```
dir([1, 2, 3])
```

Out[3]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

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.

In [2]:

```
class Person:
    pass

a = Person()

isinstance(a, Person)
```

Out[2]:

True

input

`input([prompt])` 은 사용자 입력을 받는 함수이다.

> ※ [ ] 기호는 괄호 안의 내용을 생략할 수 있다는 관례적인 표기법임을 기억하자.

In [2]:

```
b = input("이름을 입력 해 보세요: ")
print('b:',b)
```

이름을 입력 해 보세요: 오록규
b: 오록규

len

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

In [6]:

```
print(len("python"))
print(len([1,2,3]))
print('-----------------')

a = [10,20,30]

for i in range(len(a)):
    print(a[i])
```

6
3

10
20
30

list

`list(s)` 는 반복 가능한 자료형을 입력받아 리스트로 만들어 리턴하는 함수이다.

In [7]:

```
print(list("python"))
print(list((1,2,3)))
```

```
['p', 'y', 't', 'h', 'o', 'n']
[1, 2, 3]
```

enumerate

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

> ※ 보통 enumerate 함수는 아래 예제처럼 for문과 함께 자주 사용된다.

In [14]:

```
for i, name in enumerate(['body', 'foo', 'bar']):
    print(i, name)
```

```
0 body
1 foo
2 bar
```

sorted

`sorted` 함수는 정렬된 새로운 순차 자료형을 반환한다.

리스트 자료형에도 `sort`라는 함수가 있다. 하지만 리스트 자료형의 `sort` 함수는 리스트 객체 그 자체를 정렬만 할 뿐 정렬된 결과를 리턴하지는 않는다.

In [9]:

```
print(sorted([3, 1, 2]))
print(sorted(['a', 'c', 'b']))
print(sorted("zero"))
print(sorted((3, 2, 1)))
```

```
[1, 2, 3]
['a', 'b', 'c']
['e', 'o', 'r', 'z']
[1, 2, 3]
```

zip

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

In [7]:

```
# zip
print(list(zip([1, 2, 3], [4, 5, 6])))

#unzip
print(list(zip((1, 4), (2, 5), (3, 6))))

# zip to dict
print(dict(zip([1, 2, 3], [4, 5, 6])))

print('-'*30)
print(list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9])))
print(list(zip("abc", "def")))
```

```
[(1, 4), (2, 5), (3, 6)]
[(1, 2, 3), (4, 5, 6)]
{1: 4, 2: 5, 3: 6}
------------------------------
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

어렵지만, 중요한 함수들 (자주 사용하진 않습니다)

lambda

`lambda`는 함수를 생성할 때 사용하는 예약어로, `def`와 동일한 역할을 한다. 보통 함수를 한줄로 간결하게 만들 때 사용한다.

`def`를 사용해야 할 정도로 복잡하지 않거나 단 한번만 함수로서 사용을 해야 할 경우 주로 쓰인다.

In [15]:

```
prod = lambda a, b: a*b

# def sum(a,b):
#     return a+b

print(prod(3,4))

print('--------------------')

myList = [lambda a,b:a+b, lambda a,b:a*b]
print(myList[0](3,4)) # 앞의 함수 - 두 수의 합
print(myList[1](3,4)) # 뒤의 함수 - 두 수의 곱
```

```
12
--------------------
7
12
```

In [14]:

```
points = [{ 'x' : 2, 'y' : 3 },{ 'x' : 4, 'y' : 1 } ]

points.sort(key=lambda i : i['y'])

print(points)
```

```
[{'y': 1, 'x': 4}, {'y': 3, 'x': 2}]
```

filter

`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.

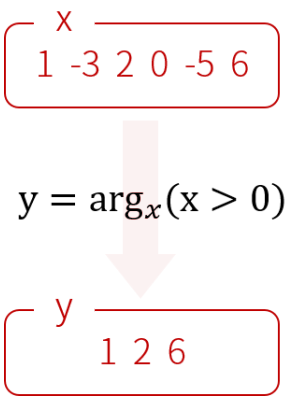
In [16]:

```
def positive(x):
    return x > 0

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

```
[1, 2, 6]
```

두 번째 인수인 리스트의 요소들이 첫 번째 인수인 positive 함수에 입력되었을 때 리턴값이 참인 것만 묶어서 돌려준다. 앞의 예에서는 1, 2, 6만 양수여서 x > 0 이라는 문장이 참이 되므로 [1, 2, 6]이라는 결과값을 리턴하게 된 것이다.



위 예제를 더 간단히 하면

In [17]:

```
print(list(filter(lambda x: x > 0, [1,-3,2,0,-5,6])))
```

```
[1, 2, 6]
```

map

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

In [18]:

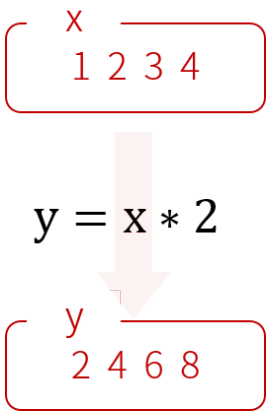
```
def two_times(x):
    return x*2

list(map(two_times, [1, 2, 3, 4, 5]))
```

Out[18]:

```
[2, 4, 6, 8, 10]
```

- 먼저 리스트의 첫 번째 요소인 1이 two_times 함수의 입력값으로 들어가고, 1 * 2의 과정을 거쳐서 2가 된다.
- 다음으로 리스트의 두 번째 요소인 2가 2 * 2의 과정을 거쳐 4가 된다. 따라서 결과값 리스트는 이제 [2, 4]가 된다.
- 총 4개의 요소값이 모두 수행되면 최종적으로 [2, 4, 6, 8]이 리턴된다. 이것이 map 함수가 하는 일이다.



앞의 예는 lambda를 사용하면 다음처럼 간략하게 만들 수 있다.

In [19]:

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

Out[19]:

```
[2, 4, 6, 8]
```

Python 표준 라이브러리 - 외장함수

- 상당히 많은 표준 라이브러리들을 제공
- 표준 라이브러리를 불러다 쓰기 위해서는 `import` 문을 사용

파이썬에서 모듈은 다른 .py 파일에서 추가해서 사용할 수 있는 함수와 변수 선언을 담고 있는 .py 파일이다.

math

`math` 에 있는 `sqrt()` 라는 함수를 불러다 쓰기 위해서는, 아래 예제와 같이 `import math` 를 실행하고, `math.sqrt()` 함수를 호출하면 된다.

In [33]:

```
import math

n = math.sqrt(9.0)
print(n)   # 3.0 출력
```

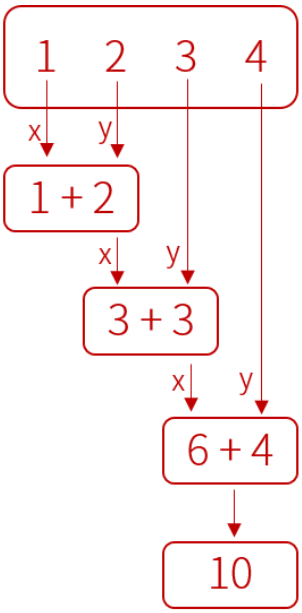
```
3.0
```

Reduce

`reduce(f(x,y), iterable)` 의 첫 인수는 함수(`f`), 두번째 인수는 시퀀스(순서가 있는)자료형(`iterable`) 이다.

두번째 인수인 시퀀스의 자료들은 순차적으로 reduce가 받은 함수의 첫 번째 인자(`x`), 두 번째 인자(`y`)로 전달된다. 첫 인수 `x` 는 함수의 계산 결과가 누적적으로 적용됨. 일반적으로 `lambda` 함수와 같이 사용됨

단, 처음 계산에서는 시퀀스의 두 개의 요소가 함수에 전달된다.



In [34]:

```
from functools import reduce

# 1부터 5까지 더하기
# 다음 코드는 이 수식과 동일하다. ((1+2)+3)+4

reduce(lambda x, y: x + y, [1, 2, 3, 4])
```

Out[34]:

```
10
```

random

`random`은 난수(규칙이 없는 임의의 수)를 발생시키는 모듈임

In [18]:

```
import random

# 0.0 ~ 1.0 사이의 실수 중 난수값을 리턴함
print(random.random())

# 1 ~ 10 사이의 정수 중 난수값을 리턴함
print(random.randint(1, 10))
```

```
0.9323750883839059
5
```

shuffle 섞기

In [19]:

```
a = [1, 2, 3, 4, 5]
random.shuffle(a)
print(a)
```

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

choice 뽑기

In [20]:

```
# 무작위로 뽑기
choose = random.choice(a)
print(choose)
```

```
1
```

날짜와 시간(datetime)

데이터분석 및 시간에 대한 내용을 다룰 때 꼭 필요한 외부 함수

datetime 클래스

날짜와 시간 정보를 가지는 객체

- `now([tz])` : 현재 시각
- `strptime(date_string, format)` : 문자열 -> datetime
- `fromtimestamp()` : timestamp -> datetime
- `fromordinal(ordinal)` : proleptic Gregorian ordinal (엑셀 날짜) -> datetime
- `combine(date, time)` : date + time -> datetime

In [21]:

```
from datetime import datetime, date, time
import time

now = datetime.now()
print(now)
print(now.weekday())   # 0:월, 1:화, 2:수, 3:목, 4:금, 5:토, 6:일
print('--------------------')

dt = datetime(2017, 3, 28, 10, 20, 30)
print ('year:', dt.year)
print ('month:', dt.month)
print ('day:', dt.day)
print('date:', dt.date(), dt.time())
```

```
2017-05-21 00:12:30.388504
6
--------------------
year: 2017
month: 3
day: 28
date: 2017-03-28 10:20:30
```

replace

In [22]:

```
print('replaced_date:',dt.replace(day=1, second=2))
print('--------------------')
```

```
replaced_date: 2017-03-01 10:20:02
--------------------
```

strptime

- `%Y` : 년, zero-padded decimal
- `%m` : 월, zero-padded decimal (`%B` : 월을 문자로)
- `%d` : 일, zero-padded decimal
- `%A` : 요일
- `%H` : 시, zero-padded decimal
- `%M` : 분, zero-padded decimal
- `%S` : 초, zero-padded decimal

In [23]:

```
# 문자열을 datetime 객체로 바꿔준다.
dt_st = datetime.strptime("2017-03-11 11:32", "%Y-%m-%d %H:%M")
print(dt_st)

print(dt_st.strftime("%A %d. %B %Y"))
```

```
2017-03-11 11:32:00
Saturday 11. March 2017
```

timedelta

In [24]:

```
dt1 = datetime(2017, 3, 20, 20, 0)
dt2 = datetime(2017, 3, 22, 22, 30)
delta = dt2 - dt1

print(type(delta),delta)
print('dt1 + delta:', dt1 + delta)
```

```
<class 'datetime.timedelta'> 2 days, 2:30:00
dt1 + delta: 2017-03-22 22:30:00
```

sleep

In [25]:

```
import time
for i in range(5):
    print(i)
    time.sleep(1)
```

```
0
1
2
3
4
```

함수 고도화

키워드 인수

여러 개의 매개변수를 가지고 있는 함수를 호출할 때, 그 중 몇 개만 인수를 넘겨주고 싶을 때

이때 매개변수의 이름을 지정하여 직접 값을 넘겨줄 수 있는데 이것을 `키워드인수` 라 부릅니다.

매개 변수이름(키워드)를 사용하여 각각의 매개 변수에 인수를 넘겨 주도록 지정해 줍니다.

In [2]:

```
def keyword_func(a, b=5, c=10):
    result = a + b * c
    print(result, a, b, c)

keyword_func(3, 7) # a = 3, b = 7, c = 10
keyword_func(25, c=24) # a = 25, b = 5, c = 24
keyword_func(c=50, a=100) # a = 100, b = 5, c = 50
```

```
73 3 7 10
145 25 5 24
350 100 5 50
```

VarArgs 매개변수

함수에 임의의 개수의 매개 변수를 지정해주고 싶을 때 `*` , `**` 로 구분함

In [13]:

```
def total(initial=5, *args, **kwargs):
    count = initial
    print(args, kwargs)

    for number in args: # number = (1,2,3)
        count += number

    for key in kwargs: # keywords = {'vegetables': 50, 'fruits': 100}
        count += kwargs[key]

    return count


print(total(10, 1, 2, 3, vegetables=50, fruits=100))

# initial = 10
# *numbers = (1,2,3)
# **keywords = {'vegetables': 50, 'fruits': 100}
```

```
(1, 2, 3) {'vegetables': 50, 'fruits': 100}
166
```

- `*args` 과 같이 매개 변수를 지정해 주면 함 수에 넘겨진 모든 위치 기반 인수들이 'args' 이라는 이름의 튜플로 묶여서 넘어옵니다.
- `**kwargs` 와 같이 앞에 별 두 개가 달린 매개 변수를 지정 해 주면 함수에 넘겨진 모든 키워드 인수들이 `**kwargs` 이라는 이름의 딕셔너리로 묶여서 넘어옵니다.