**Python**

**A**

**[ abs ]**

어떤 숫자를 입력 받았을 때, 해당하는 숫자의 절댓값을 반환.

Input : 숫자

Output : int형

>>> abs(3.5)

>>> 3

**B**

**C**

**[ Counter ]**

from collections import Counter

dict로 카운트해서 반환

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

>>> b = dict( Counter(a) )

>>> {1: 3, 2: 3, 3: 2, 4: 1, 5: 1}

**[ combinations ]**

배열의 조합 찾기 ( 순열도 확인하기 )

from itertools import combinations

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

>>> b = list( combinations( a, 2 ) )

>>> [(1, 2), (1, 3), (2, 3)]

**D**

**[ deque ]**

Double-ended queue의 준말, 앞과 뒤에서 즉, 양방향에서 데이터를 처리할 수 있는 queue형 자료구조.

스크린샷이(가) 표시된 사진

자동 생성된 설명

*선언*

from collections import deque

deq = dequeue()

*함수*

1. append
2. appendleft
3. extend
4. extendleft
5. pop
6. popleft
7. rotate

*참고*

<https://excelsior-cjh.tistory.com/96>

**[ datetime ]**

시간 / 날짜 계산할 때 좋음

from datetime import datetime, timedelta

FMT = ‘%Y-%H:%M’

start = '2014-12:00'

finish = '2015-13:12'

time = datetime.strptime(finish,FMT) - datetime.strptime(start,FMT) # 초 단위로 환산됨.

time = datetime.strptime(finish,FMT) + timedelta(minutes = 숫자) 도 가능

결과는 fortmat(time)으로 확인할 수 있음.

**[ divmod ]**

몫, 나머지를 한번에 구할 수 있음.

divmod( a, b ) 🡪 [a//b, a%b]

divmod(5,3)[0] : 1

divmod(5,3)[1] : 2

**E**

**F**

**G**

**H**

**I**

**J  
[ join ]**

만약

String 타입의 변수 a와 arr이라는 list가 있을 때,

a.join(list)

>>> a = ‘’

>>> b = [‘ㄱ’,’ㄴ’,’ㄷ’]

>>> c = a.join(b)

>>> c = ‘ㄱㄴㄷ’

**K**

**L**

**M**

**[ map ]**

map ( 함수, <iterable object> )

: 입력 받은 <iterable object>의 각 요소를 함수로 수행된 결과를 묶어서 반환.

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

>>> [2, 4, 6, 8]

Input : 함수, <iterable object>

Output : list

**N**

**O**

**[ ord ]**

Input 문자열의 ASCII 코드 값을 return.

>>> ord(‘a’)

>>> 97

**P**

**[ permutations ]**

배열의 순열 찾기 ( 조합도 확인하기 )

from itertools import permutations

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

>>> b = list( permutations ( a, 2 ) )

>>> [(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]

**[ pass ]**

아무것도 안하기

>>> try :

>>> ~~~

>>> except :

>>> pass

**Q**

**[ Queue ]**

First-In-First-Out 또는 Last-In-Last-Out

import queue

#FIFO Queue

data = queue.Queue()

data.put(1)

data.put(2)

data.get() # 1

data.get() # 2

#LIFO Queue

data = queue.LifoQueue()

data.put(1)

data.put(2)

data.get() # 2

data.get() # 1

#Priority Queue

data = queue.PriorityQueue()

data.put((1,2)) # (priority, value)

data.put((2,1))

data.get()[1] # 2

PriorityQueue의 경우, 우선순위를 둘 수도 있지만, 일반 Queue처럼 사용하는 것도 가능.

**R**

**[ replace ]**

특정 문자열을 변경하여 반환

>>> a = ‘apple’

>>> b = a.replace( ‘pp’, ‘ss’ )

>>> assle

**[ round ]**

반올림

>>> round( 1.5 )

>>> 2

**[ re ] 정규 표현식**

출처 : https://wikidocs.net/4308

문자 클래스로 만들어진 정규식은 *“[ ] 사이의 문자들과 매치”* 라는 의미를 갖는다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

1. Dot(.)

정규 표현식의 Dot(.) 메타 문자는 줄바꿈 문자인 \n을 제외한 모든 문자와 매치됨을 의미

예) a.b 🡺 a + 모든 문자 + b

즉, a와 b라는 문자 사이에 적어도 한 개의 문자가 있다면, match.

1. 반복
   1. \*  
      \*은 \* 바로 앞에 있는 문자 a가 0부터 무한대로 반복될 경우 매치됨을 의미

예) ca\*t 🡺 ct : O , cat : O , caaaat : O

* 1. +  
     +는 최소 1번 이상 반복될 때 사용. \*가 반복 횟수 0부터라면, +는 반복 횟수 1부터.

예) ca+t 🡺 ct : X , cat : O , caaaat : O

* 1. {n}  
     반드시 n번 반복
  2. {min, max}

Min~max회 반복

* 1. ?  
     {0,1}과 같은 의미, 즉 있어도 되고 없어도 된다. 단, 있다면 반드시 한 번만 있을 것.   
     예) ab?c 🡺 abc : O , ac : O ,

1. Re module  
   import re

p = re.compile( 정규표현식 )

스크린샷이(가) 표시된 사진

자동 생성된 설명

m = p.match(‘확인하고 싶은 문자열’)

if m:

~~~

else:

~~~

스크린샷이(가) 표시된 사진

자동 생성된 설명

**S**

**[ split ]**

String 타입을 특정 기준으로 쪼개서 반환.

>>> a = ‘hi hello’

>>> b = a.split(‘ ‘)

>>> [ ‘hi’, ‘hello’ ]

Input : string, split 기준

Output : list

**[ sort ]**

list 정렬

>>> list\_name.sort( reverse = True ) 🡪 역순 : 큰 순서대로 정렬, default는 False.

>>> list\_name.sort( key = lambda x : ( x[0], x[1] ) 🡪 백준 11650

Output : list

**[ sorted ]**

dict 정렬

key를 기준으로 정렬

>>> temp = sorted ( dict\_name.items(), key = ( lambda x : x[0] ), reverse = True)

value를 기준으로 정렬

>>> temp = sorted ( dict\_name.items(), key = ( lambda x : x[1] ), reverse = True)

Output : list

**T**

**[ transpose ]**

Numpy의 함수 중 하나로 행렬을 회전시킬 때 사용.

import numpy as np

a = [[0,0],[1,1]]

b = np.transpose(a)

b >>> [[0,1],[0,1]]

**U**

**V**

**W**

**X**

**Y**

**Z**

**[ zip ]**

동일한 갯수의 원소를 갖는 시쿼스 자료형(예를 들면, 배열)을 묶어주는 역할

>>> a = [ 1, 2 ]

>>> b = [ 3, 4 ]

>>> c = list ( zip( a, b ) )

>>> [ ( 1, 2), (3, 4) ]

응용

[ 1, 2 ]와 [ 3, 4 ]를 이용해서 [ 4, 6 ]을 만들고 싶을 때.

>>> a = [ 1, 2 ]

>>> b = [ 3, 4 ]

>>> c = [ c+d for c, d in zip( a, b ) ]

>>> [ 4, 6 ]

**ETC**

**[ : ]**

string 또는 list 조작.

>>> a = ‘abcdefg’

>>> b = a[2:]

>>> ‘cdefg’

>>> c = a[:3]

>>> ‘abcd’

>>> d = a[::2]

>>> ‘aceg’

>>> e = a[::-2]

>>> ‘geca’