Python 3

Python data structure

- stack & queue
  - stack: LIFO
  - queue: FIFO
- tuple & set
  - tuple: 값 변경 불가능한 리스트
  - set: 순서 없이 저장, 중복 불허
  - remove(), discard(): 해당 원소 삭제
  - clear(): 모든 원소 삭제
- dict
  - {Identifier(Key): Value}
- Collections module
  - deque, Counter, OrderedDict, namedtuple
  - defaultdict

```
d = defaultdict(lambda: 0)
for word in text.split():
  d[word] += 1
  print("Word count\n", d)
```

Pythonic code

- split & join
- list comprehension
  - 기존 list 사용해 간단히 다른 list 만드는 기법
  - 포괄적인 (포함되는) list
  - [i + j for i in list_1 for j in list_2 if not (i==j)]
- enumerate & zip
  - zip

```
list_a = ['a1', 'a2', 'a3']
list_b = ['b1', 'b2', 'b3']
[[a, b] for a, b in zip(list_a, list_b)]

# [['a1', 'b1'], ['a2', 'b2'], ['a3', 'b3']]
```

- lambda & map & reduce
  - lambda: PEP8부터 권장하지 않고 있음 (여전히 많이 쓰이긴 함) * 가독성이 별로임 * 테스트의 어려움
  - map: list comprehension으로 대부분 대체 가능

    - reduce: 코드 직관성이 떨어져 역시 권장하지 않고 있음 * map과 달리, list에 똑같은 함수를 적용해서 통합 * 대용량 데이터를 다룰 때 유요할 수 있음
- generator
    - iterable object를 특수한 형태로 사용
    - element가 사용되는 시점에 값을 메모리에 반환 (iterator에 비해 메모리 절약)

```
def general_list(value):
  result = []
  for i in range(value):
    result.append(i)
  return result

sys.getsizeof(general_list(50))
# 520

def generator_list(value):
  result = []
  for i in range(value):
    yield i

sys.getsizeof(generator_list(50))
# 112
```

    - generator comprehension * gen_ex = (n*n for n in range(100))
    - when to use * list 타입 데이터 반환 함수 (가독성, 중간에 loop 중단될 수 있을 때) * 큰 데이터를 처리할 때 * 파일 데이터를 처리할 때
- asterisk
    - keyword arguments

```
def function(my_name, your_name):
  ...
function(your_name="B", my_name="A")
```

    - default arguments * def function(my_name=”A”, your_name)
    - variable-length arguments (가변인자) * “*” 기호를 사용해 함수의 parameters 표현

```
def function(a, b, *args):
  ...
function(1,2,3,4,5)
```

    - keyword variable-length arguments * parameter 이름 지정 없이 사용 * “**” 사용해 parameters 표현 * 입력된 값은 dict type으로 사용 가능

```
def function(**kwargs):
  ...
function(first=3, second=4, third=12)
```

- unpacking a container

```
*(1, 2, 3, 4) == 1, 2, 3, 4
*{"b":1, "c":2, "d":3} == b=1, c=2, d=3
```