

Week1-3

1. 학습정리

1. Python data structure

- 컬렉션
 - deque
 - 리스트와 거의 동일
 - 리스트에 비해 빠르고 효율적임
 - 기능
 - leftappend()
 - 앞에 추가
 - rotate(int)
 - int 만큼 자리 이동
 - extend([1,2,3])
 - 여러개 한번에 추가
 - extend([1,2,3])
 - 앞쪽에 역순으로 붙음 (3,2,1,~~)
 - 장점
 - 효율적 메모리 구조 -> 성능 향상
 - defaultdict
 - 아직 키 값이 없는 걸 출력하려 할 때 디폴트 값으로 보여줌
 - 디폴트 값은 함수 형태로 해야함
 - lambda : 0 형태로 가능
 - Counter
 - 시퀀스에서 값의 개수를 세줌

```
ball_or_strike_list = ["B", "S", "S", "S", "S", "B", "B"]
```

```
c = Counter(ball_or_strike_list) # a new counter from keyword args
c
```

```
Counter({'B': 3, 'S': 4})
```

- 반대도 가능

```
c = Counter(cats=4, dogs=8) # a new counter from keyword args
print(list(c.elements()))
```

```
['cats', 'cats', 'cats', 'cats', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs']
```

- 집합연산도 지원
- namedtuple
 - 저장되는 데이터의 변수를 사전에 지정
 - 한번 정의해서 사용 가능
 - 저장될 때는 key=value 형태로 저장됨
 - 이렇게 있다는 것 정도만
- %timeit 함수()

- 주피터 노트북에서 함수의 속도를 측정
- """ 문자열 문자열 문자열 """
 - 이런 형태면 이미 공백기준으로 split 된 문자열임

2. Pythonic code

- split&join
- list comprehension
 - for + append 보다 빠름
 - ex

```
>>> word_1 = "Hello"
>>> word_2 = "World"
>>> result = [i+j for i in word_1 for j in word_2]
# Nested For loop
>>> result
['HW', 'Ho', 'Hr', 'Hl', 'Hd', 'eW', 'eo', 'er',
 'el', 'ed', 'lW', 'lo', 'lr', 'll', 'ld', 'lW',
 'lo', 'lr', 'll', 'ld', 'oW', 'oo', 'or', 'ol', 'od']
```

- if not else

```
>>> result = [i+j for i in case_1 for j in case_2 if not(i==j)]
# Filter: i랑 j과 같다면 List에 추가하지 않음
# [i+j if not(i==j) else i for i in case_1 for j in case_2]
```

- 2차원에선 배열 만들 땐 뒤의 for 루프가 먼저 작동

Two dimensional vs One dimensional


```
>>> case_1 = ["A","B","C"]
>>> case_2 = ["D","E","A"]
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
>>> result = [i+j for i in case_1 for j in case_2]
>>> result
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
>>> result = [ [i+j for i in case_1] for j in case_2]
>>> result
[['AD', 'BD', 'CD'], ['AE', 'BE', 'CE'], ['AA', 'BA', 'CA']]
```

- zip(리스트, 리스트)
 - 두 개의 리스트의 값을 병렬적으로 추출 -> 튜플로 묶음

```
[ c for c in zip(alist, blist)]
```

```
[('a1', 'b1'), ('a2', 'b2'), ('a3', 'b3')]
```

- lambda
 - 이름 없는 함수

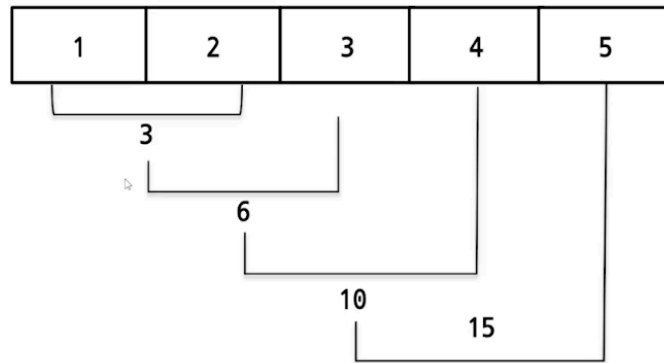
| General function | Lambda function |
|--|--|
| <pre>def f(x, y): return x + y</pre>  | <pre>f = lambda x, y: x + y print(f(1, 4))</pre> |

- map(함수, 리스트)
 - 리스트의 각 원소에 함수 적용
 - 리스트 컴프리헨션으로 대체가능
- reduce(함수, 리스트)

- map function과 달리 list에 똑같은 함수를 적용해서 통합

```
from functools import reduce
```

```
print(reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]))
```



×

- 대용량 데이터 다룰 때 사용
- iterable object
 - 리스트, 문자열, 튜플 등과 같이 차례대로 출력하는 시퀀스 형 자료형
 - 내부적으로 `__iter__`, `__next__` 라는 함수가 구현되어 있음
 - `iter(list)` 후 `next(이터레이터)` 로 하면 출력되고 다음으로
 - c++ 이터레이터와 비슷
- generator
 - for 루프에서 메모리에 올리지 않고 필요할 때만 생성해 줌
 - 항상 112바이트만 사용
 - 메모리 절약 가능
 - generator comprehension

generator comprehension

gener

- list comprehension과 유사한 형태로 generator 형태의 list 생성
- generator expression 이라는 이름으로도 부름
- `[]` 대신 `()` 를 사용하여 표현

```
gen_ex = (n*n for n in range(500))  
print(type(g))
```

- function passing argument
 - keyword argument
 - 파라미터 패싱할 때
 - 함수(변수이름 = 값) 으로 하면 지정해놓은 키워드로 패싱함
 - default argument

- 파라미터 기본값 지정
- 가변인자 : variable-length asterisk
 - Asterisk(*) 기호를 사용하여 파라미터를 표시
 - 파라미터 앞에 * 를 쓰면 가변인자가 됨
 - 튜플타입으로 전달됨
- 키워드 가변인자
 - 파라미터 앞에 ** 를 쓰면 키워드 가변인자
 - dict 타입으로 전달받음
- 순서
 - 앞에서 키워드 형태로 쓰면 뒷부분은 다 키워드로 넣어야 함
- asterisk (*)
 - * 기호를 의미
 - 시퀀스형 자료형 앞에 *을 붙이면 언패킹함
 - $*(1,2,3,4,5) = 1,2,3,4,5$
 - dict 타입을 언패킹 할 땐 ** 를 씀
 - $b=1, c=2$ 이렇게 언패킹됨

2. 피어세션 정리

- 과제 코드 리뷰
 - 돌아가며 자신의 코드 설명
 - 파이썬닉한 코드 작성법 조언
- 수업 중 어려웠던 것 질답