[오픈소스SW 활용] HW-2

오픈소스SW 활용 3분반, 32170578, 김산

[Item 11] Know How to Slice Sequences

- list
- dictionary
- tuple

Dictionary 자료구조는 hash function을 이용하여 동작한다

- 키값에 해싱함수를 적용해서 나온 결과를 bustket이라는 곳에 대응시켜 value를 얻는다.

In [35]:

```
a = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
assert a[:5] == a[0:5]
assert a[5:] == a[5:len(a)]
print('Middle two: ', a[3:5])
print('All but ends:', a[1:7])
```

```
Middle two:  ['d', 'e']
All but ends: ['b', 'c', 'd', 'e', 'f', 'g']
```

- slice index
  - [0:5:2] : 0~5까지(5빼고) 2씩 증가 [start:end:increment]
  - slice했다 해서 list의 값이 변하지 않는다.

In [36]:

```
print(a[:]) # ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
print(a[:5]) # ['a', 'b', 'c', 'd', 'e']
print(a[:-1]) # ['a', 'b', 'c', 'd', 'e', 'f', 'g']
print(a[4:]) # ['e', 'f', 'g', 'h']
print(a[-3:]) # ['f', 'g', 'h']
print(a[2:5]) # ['c', 'd', 'e']
print(a[2:-1]) # ['c', 'd', 'e', 'f', 'g']
print(a[-3:-1]) # ['f', 'g']
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
['a', 'b', 'c', 'd', 'e']
['a', 'b', 'c', 'd', 'e', 'f', 'g']
['e', 'f', 'g', 'h']
['f', 'g', 'h']
['c', 'd', 'e']
['c', 'd', 'e', 'f', 'g']
['f', 'g']
```

인덱스 -1과 같이 음수를 통해 역순으로 참조할 수 있다

In [39]:

```
first_20_items = a[:20]
last_20_items = a[-20:]

print(first_20_items)
print(last_20_items)
# 인덱스보다 아이템의 수가 부족한 경우 있는 내용만 출력된다
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

In [40]:

```
a[-0:]
# a₩[-0₩]은 a₩[:₩]와 같이 리스트의 모든 내용을 출력한다
```

Out[40]:

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

In [41]:

```
b = a[3:]
print('Before: ', b)
b[1] = 99
print('After: ', b)
print('No change:', a)
```

```
Before:  ['d', 'e', 'f', 'g', 'h']
After:  ['d', 99, 'f', 'g', 'h']
No change: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

In [43]:

```
print('Before ', a)
a[2:7] = [99, 22, 14]
print('After ', a)
# 인덱스 범위보다 삽입할 값이 더 적을경우 리스트의 크기가 줄어든다
```

```
Before  ['a', 'b', 99, 22, 14, 'h']
After  ['a', 'b', 99, 22, 14]
```

In [45]:

```
b = a[:]
assert b == a and b is not a
```

In [46]:

```
b = a
print('Before a', a)
print('Before b', b)
a[:] = [101, 102, 103]
assert a is b # Still the same list object
print('After a ', a) # Now has different contents
print('After b ', b) # Same list, so same contents as a
```

```
Before a ['a', 'b', 99, 22, 14]
Before b ['a', 'b', 99, 22, 14]
After a  [101, 102, 103]
After b  [101, 102, 103]
```

- a = b 대입 연산을 하게 되면같은 object가 된다.(같은 reference가 됨)
- 따라서 리스트에 대한 copy를 생각하면 b = a[:]와 같이 사용해야 함

이러한 python의 슬라이스 방식은 다음과 같이 문자열을 처리할때 특히 더 강력했던 것 같습니다.

In [1]:

```
py_string = 'Python'

# contains indices 0, 1 and 2
print(py_string[0:3])  # Pyt

# contains indices 1 and 3
print(py_string[1:5:2]) # yh
```

```
Pyt
yh
```

[Item 12] Avoid Striding and Slicing in a Single Expression

In [48]:

```
x = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
odds = x[::2]
evens = x[1::2]
print(odds)
print(evens)
# Usage : alist[start:end:stride]
```

```
['red', 'yellow', 'blue']
['orange', 'green', 'purple']
```

In [49]:

```
x = b'mongoose'
y = x[::-1]
print(y)
```

```
b'esoognom'
```

In [50]:

```
x = '寿司'
y = x[::-1] #이러한 stride는 유니코드나 바이트 문자열에 대해서도 적용된다
print(y)
```

```
司寿
```

c언어에서는 for루프를 내림차순으로 진행할때 for(int i = 10; i > 0; i--) 와 같이 사용하였지만 python에서는 stride를 통해 내림차순으로 루프를 표현할 수 있었습니다.

In [53]:

```
w = '寿司'
x = w.encode('utf-8')
y = x[::-1]
z = y.decode('utf-8')
# 자료형 불일치로 인한 에러
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-53-0d50076d0fe4> in <module>
      2 x = w.encode('utf-8')
      3 y = x[::-1]
----> 4 z = y.decode('utf-8')
      5 # 자료형 불일치로 인한 에러

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb8 in position 0: invalid start byte
```

In [52]:

```
x = ['a','b','c','d','e','f','g','h']
```

In [55]:

```
print(x[2::2]) # ['c', 'e', 'g']
print(x[-2::-2]) # ['g', 'e', 'c', 'a']
print(x[-2:2:-2]) # ['g', 'e']
print(x[2:2:-2]) # []
```

```
b'₩xbf₩x8f'
b'₩x8f₩xbf₩xe5'
b'₩x8f'
b''
```

- 마지막 x[2:2:-2]의 경우 start index와 end index가 같기 때문에 아무것도 출력되지 않는다.
- 가능한 양수 인덱스를 사용
- itertools 의 islice 를 사용하여 헷갈리는것을 방지할 수 있다.

[Item 13] Prefer Catch-All Unpacking Over Slicing

In [56]:

```
car_ages = [0, 9, 4, 8, 7, 20, 19, 1, 6, 15]
car_ages_descending = sorted(car_ages, reverse=True)
oldest, second_oldest = car_ages_descending
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call last)
<ipython-input-56-99634ca0e408> in <module>
      1 car_ages = [0, 9, 4, 8, 7, 20, 19, 1, 6, 15]
      2 car_ages_descending = sorted(car_ages, reverse=True)
----> 3 oldest, second_oldest = car_ages_descending

ValueError: too many values to unpack (expected 2)
```

In [57]:

```
oldest = car_ages_descending[0]
second_oldest = car_ages_descending[1]
others = car_ages_descending[2:]
print(oldest, second_oldest, others)
```

```
20 19 [15, 9, 8, 7, 6, 4, 1, 0]
```

In [58]:

```
oldest, second_oldest, *others = car_ages_descending
print(oldest, second_oldest, others)
```

```
20 19 [15, 9, 8, 7, 6, 4, 1, 0]
```

In [59]:

```
oldest, *others, youngest = car_ages_descending
print(oldest, youngest, others)
*others, second_youngest, youngest = car_ages_descending
print(youngest, second_youngest, others)
```

```
20 0 [19, 15, 9, 8, 7, 6, 4, 1]
0 1 [20, 19, 15, 9, 8, 7, 6, 4]
```

In [60]:

```
*others = car_ages_descending
fist, *middle, *second_middle, last = [1, 2, 3, 4]
```

```
  File "<ipython-input-60-de7bf1ede8b8>", line 1
    *others = car_ages_descending
    ^
SyntaxError: starred assignment target must be in a list or tuple
```

In [61]:

```
car_inventory = {
 'Downtown': ('Silver Shadow', 'Pinto', 'DMC'),
 'Airport': ('Skyline', 'Viper', 'Gremlin', 'Nova'),
}
((loc1, (best1, *rest1)),
 (loc2, (best2, *rest2))) = car_inventory.items()
print(f'Best at {loc1} is {best1}, {len(rest1)} others')
print(f'Best at {loc2} is {best2}, {len(rest2)} others')
```

```
Best at Downtown is Silver Shadow, 2 others
Best at Airport is Skyline, 3 others
```

In [62]:

```
short_list = [1, 2]
first, second, *rest = short_list
print(first, second, rest)
```

```
1 2 []
```

In [63]:

```
it = iter(range(1, 3))
first, second = it
print(f'{first} and {second}')
```

```
1 and 2
```

In [64]:

```
def generate_csv():
    yield ('Date', 'Make' , 'Model', 'Year', 'Price')
    for i in range(100):
        yield ('2019-03-25', 'Honda', 'Fit' , '2010', '$3400')
        yield ('2019-03-26', 'Ford', 'F150' , '2008', '$2400')

all_csv_rows = list(generate_csv()) # too noisy
header = all_csv_rows[0]
rows = all_csv_rows[1:]
print('CSV Header:', header)
print('Row count: ', len(rows))
```

```
CSV Header: ('Date', 'Make', 'Model', 'Year', 'Price')
Row count:  200
```

In [65]:

```
it = generate_csv()
header, *rows = it
print('CSV Header:', header)
print('Row count: ', len(rows))
```

```
CSV Header: ('Date', 'Make', 'Model', 'Year', 'Price')
Row count:  200
```

In [66]:

```
for str in rows:
    print(str)
```

```
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
```

```
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
```

```
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
```

```
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
('2019-03-25', 'Honda', 'Fit', '2010', '$3400')
('2019-03-26', 'Ford', 'F150', '2008', '$2400')
```

*rows 와 같이 starred expression을 사용하면 unpacking하여 값을 전달한다.

이러한 방식으로 unpacking을 사용하면 인덱스를 사용하여 header, rows를 구분해주는 과정 없이 간결하게 표현할 수 있는 것 같습니다

[Item 14] Sort by Complex Criteria using the key Parameter

In [68]:

```python
numbers = [93, 86, 11, 68, 70]
numbers.sort()
print(numbers)
```

```
[11, 68, 70, 86, 93]
```

In [69]:

```
class Tool:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight

    def __repr__(self):
        return f'Tool({self.name!r}, {self.weight})'

tools = [
    Tool('level', 3.5),
    Tool('hammer', 1.25),
    Tool('screwdriver', 0.5),
    Tool('chisel', 0.25),
]

tools.sort()
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-69-a39019e34d43> in <module>
    14 ]
    15
---> 16 tools.sort()

TypeError: '<' not supported between instances of 'Tool' and 'Tool'
```

Tool class에는 비교연산자가 존재하지 않기 때문에 위와 같은 에러가 발생한다

In [70]:

```
print('Unsorted:', repr(tools))
tools.sort(key=lambda x: x.name)
print('₩nSorted: ', tools)
```

```
Unsorted: [Tool('level', 3.5), Tool('hammer', 1.25), Tool('screwdriver', 0.5), Tool('chisel',
0.25)]

Sorted:  [Tool('chisel', 0.25), Tool('hammer', 1.25), Tool('level', 3.5), Tool('screwdriver',
0.5)]
```

In [71]:

```
print('Unsorted:', repr(tools))
tools.sort(key=lambda x: x.weight)
print('₩nSorted: ', tools)
```

```
Unsorted: [Tool('chisel', 0.25), Tool('hammer', 1.25), Tool('level', 3.5), Tool('screwdrive
r', 0.5)]

Sorted:  [Tool('chisel', 0.25), Tool('screwdriver', 0.5), Tool('hammer', 1.25), Tool('level',
3.5)]
```

In [72]:

```
places = ['home', 'work', 'New York', 'Paris']
places.sort()
print('Case sensitive: ', places)
places.sort(key=lambda x: x.lower())
print('Case insensitive:', places)
```

```
Case sensitive:  ['New York', 'Paris', 'home', 'work']
Case insensitive: ['home', 'New York', 'Paris', 'work']
```

- case sensitive의 경우 비교시 대, 소문자를 구분하기 때문에 대문자요소가 앞에 오고(아스키 코드 값에서 대문자가 먼저 나타난다))
- case insensitive의 경우 비교시 소문자로 lower하여 비교하기 때문에 알파벳 순으로 출력된다

In [74]:

```
power_tools = [
   Tool('sander', 4),
   Tool('drill', 4),
   Tool('circular saw', 5),
   Tool('jackhammer', 40),
]
power_tools.sort(key=lambda x: (x.weight, x.name))
print(power_tools)
# weight, name모두 고려하여 정렬
```

```
[Tool('drill', 4), Tool('sander', 4), Tool('circular saw', 5), Tool('jackhammer', 40)]
```

In [75]:

```
power_tools.sort(key=lambda x: (-x.weight, x.name))
print(power_tools)
```

```
[Tool('jackhammer', 40), Tool('circular saw', 5), Tool('drill', 4), Tool('sander', 4)]
```

In [76]:

```
power_tools.sort(key=lambda x: (x.weight, -x.name))
print(power_tools)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-76-7a62605bf47c> in <module>
----> 1 power_tools.sort(key=lambda x: (x.weight, -x.name))
      2 print(power_tools)

<ipython-input-76-7a62605bf47c> in <lambda>(x)
----> 1 power_tools.sort(key=lambda x: (x.weight, -x.name))
      2 print(power_tools)

TypeError: bad operand type for unary -: 'str'
```

- string에는 -가 정의되어 있지 않다 -> operator overriding을 통해 해결할 수 있다

In [77]:

```
power_tools.sort(key=lambda x: x.name) # Name ascending
print(power_tools)
power_tools.sort(key=lambda x: x.weight, reverse=True) # Weight descending
print(power_tools)
```

```
[Tool('circular saw', 5), Tool('drill', 4), Tool('jackhammer', 40), Tool('sander', 4)]
[Tool('jackhammer', 40), Tool('circular saw', 5), Tool('drill', 4), Tool('sander', 4)]
```

- 파이썬의 built in data type은 sorting method를 제공한다
- The unary minus operator can be used to reverse individual sort orders for types that allow it.

[Item 15] Be Cautious When Relying on dict Insertion Ordering

In [1]:

```
# Python 3.5
baby_names = {
 'cat': 'kitten',
 'dog': 'puppy',
}
print(baby_names)
```

```
{'cat': 'kitten', 'dog': 'puppy'}
```

In [2]:

```
# Python 3.6+
baby_names = {
 'cat': 'kitten',
 'dog': 'puppy',
}
print(baby_names)
```

```
{'cat': 'kitten', 'dog': 'puppy'}
```

In [3]:

```
# Python 3.5
print(list(baby_names.keys()))
print(list(baby_names.values()))
print(list(baby_names.items()))
print(baby_names.popitem()) # Randomly chooses an item
```

```
['cat', 'dog']
['kitten', 'puppy']
[('cat', 'kitten'), ('dog', 'puppy')]
('dog', 'puppy')
```

In [4]:

```
print(list(baby_names.keys()))
print(list(baby_names.values()))
print(list(baby_names.items()))
print(baby_names.popitem()) # Last item inserted
```

```
['cat']
['kitten']
[('cat', 'kitten')]
('cat', 'kitten')
```

In [5]:

```
# Python 3.5
def my_func(**kwargs):
    for key, value in kwargs.items():
        print('%s = %s' % (key, value))
my_func(goose='gosling', kangaroo='joey')
```

```
goose = gosling
kangaroo = joey
```

In [6]:

```
# Python 3.5
# Python 3.7+
class MyClass:
    def __init__(self):
        self.alligator = 'hatchling'
        self.elephant = 'calf'
a = MyClass()
for key, value in a.__dict__.items():
    print('%s = %s' % (key, value))
```

```
alligator = hatchling
elephant = calf
```

In [10]:

```
votes = { 'otter': 1281, 'polar bear': 587, 'fox': 863 }

def populate_ranks(votes, ranks):
    names = list(votes.keys())
    names.sort(key=votes.get, reverse=True)
    for i, name in enumerate(names, 1):
        ranks[name] = i

def get_winner(ranks):
    return next(iter(ranks))
```

In [11]:

```
ranks = {}
populate_ranks(votes, ranks)
print(ranks)
winner = get_winner(ranks)
print(winner)
```

```
{'otter': 1, 'fox': 2, 'polar bear': 3}
otter
```

In [12]:

```
from collections.abc import MutableMapping
class SortedDict(MutableMapping):
    def __init__(self):
        self.data = {}
    def __getitem__(self, key):
        return self.data[key]
    def __setitem__(self, key, value):
        self.data[key] = value
    def __delitem__(self, key):
        del self.data[key]
    def __iter__(self):
        keys = list(self.data.keys())
        keys.sort()
        for key in keys:
            yield key
    def __len__(self): #앞위에 underscore가 2개씩있는 함수는 operator overloading을 지원하는 special method
        return len(self.data)
```

개인적으로 이러한 special method는 다음과 같이 클래스 내부의 리스트 멤버를 클래스레벨에서 활용 하여 유용하게 쓸수 있을것 같았습니다.

In [ ]:

```
class Account:
    # ... (see above)

    def __len__(self):
        return len(self._transactions)

    def __getitem__(self, position):
        return self._transactions[position]
```

In [13]:

```
sorted_ranks = SortedDict()
populate_ranks(votes, sorted_ranks)
print(sorted_ranks.data)
winner = get_winner(sorted_ranks)
print(winner)
```

```
{'otter': 1, 'fox': 2, 'polar bear': 3}
fox
```

딕셔너리에서는 alpabet순서를 따르기 때문에 이러한 오류가 발생

In [14]:

```
#Solution 1 : Conservative and robust solution
def get_winner(ranks):
    for name, rank in ranks.items():
        if rank == 1:
            return name
winner = get_winner(sorted_ranks)
print(winner)
```

```
otter
```

In [15]:

```
# Solution 2 : Add an explicit check
def get_winner(ranks):
    if not isinstance(ranks, dict):
        raise TypeError('must provide a dict instance')
    return next(iter(ranks))
get_winner(sorted_ranks)
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-15-c3de71bcc1cc> in <module>
      4         raise TypeError('must provide a dict instance')
      5     return next(iter(ranks))
----> 6 get_winner(sorted_ranks)

<ipython-input-15-c3de71bcc1cc> in get_winner(ranks)
      2 def get_winner(ranks):
      3     if not isinstance(ranks, dict):
----> 4         raise TypeError('must provide a dict instance')
      5     return next(iter(ranks))
      6 get_winner(sorted_ranks)

TypeError: must provide a dict instance
```

In [16]:

```
# Solution 3 : Use type annotations
from typing import Dict,MutableMapping
def populate_ranks(votes: Dict[str, int],
    ranks: Dict[str, int]) -> None:
        names = list(votes.keys())
        names.sort(key=votes.get, reverse=True)
        for i, name in enumerate(names, 1):
            ranks[name]  = i
def get_winner(ranks: Dict[str, int]) -> str:
    return next(iter(ranks)) # lead to the error
sorted_ranks = SortedDict()
populate_ranks(votes, sorted_ranks)
print(sorted_ranks.data)
winner = get_winner(sorted_ranks)
print(winner)
```

```
{'otter': 1, 'fox': 2, 'polar bear': 3}
fox
```

In [17]:

```
# (another way) find the key for a certain value
def get_key(ranks, val):
    for key, value in ranks.items():
        if val == value:
            return key
def get_winner(ranks: Dict[str, int]) -> str:
    return get_key(ranks, 1)

class SortedDict(MutableMapping[str, int]):
    def __init__(self):
        self.data = {}
    def __getitem__(self, key):
        return self.data[key]
    def __setitem__(self, key, value):
        self.data[key] = value
    def __delitem__(self, key):
        del self.data[key]
    def __iter__(self):
        keys = list(self.data.keys())
        keys.sort()
        for key in keys:
            yield key
    def __len__(self):
        return len(self.data)

votes = {
 'otter': 1281,
 'polar bear': 587,
 'fox': 863,
}
sorted_ranks = SortedDict()
populate_ranks(votes, sorted_ranks)
print(sorted_ranks.data)
winner = get_winner(sorted_ranks)
print(winner)
```

```
{'otter': 1, 'fox': 2, 'polar bear': 3}
otter
```

Role of Underscore _ in Python

In [ ]:

```
for _ in range(50):
__init__(self)
_ = 2
```

- 특별히 이름을 지어주지 않아도 될때
- access modifier

1. Use in Python interpreter
2. Ignore values
3. Use in loop
4. Separate digits of numbers
5. Naming
    - Single Pre Underscore
    - Single Post Underscore
    - Double Pre Undescores
    - Double Pre And Post Underscores

1. Python automatically stores the value of the last expression

In [19]:

```
4 + 5
```

Out[19]:

```
9
```

In [20]:

```
_
```

Out[20]:

```
9
```

In [21]:

```
_ + 7
```

Out[21]:

```
16
```

2. Ignore values

In [22]:

```
a, _, b = (1, 2, 3) # a = 1, b = 3
print(a, b)
## ignoring multiple values
## *(variable) used to assign multiple value to a variable as list while unpacking
## it's called "Extended Unpacking", only available in Python 3.x
a, *_, b = (7, 6, 5, 4, 3, 2, 1)
print(a, b)
```

```
1 3
7 1
```

3. Use in loop

In [23]:

```
## lopping ten times using _
for _ in range(5):
    print(_)
## iterating over a list using _
## you can use _ same as a variable
languages = ["Python", "JS", "PHP", "Java"]
for _ in languages:
    print(_)
_ = 5
while _ < 10:
    print(_, end = ' ') # default value of 'end' id '₩n' in python. we're changing it to space
    _ += 1
```

```
0
1
2
3
4
Python
JS
PHP
Java
5 6 7 8 9
```

이렇게 _(언더스코어)를 통한 표현은 변수의 이름을 명시적으로 지정할 필요가 없고, 또 변수 이름이 마땅히 생각이 안날때 언더스코어를 통해 지정해 준다면, 코딩하는데 불필요한 고민을 좀 덜어줄 것 같습니다.

4. Separate digits of numbers

In [24]:

```
## different number systems
## you can also check whether they are correct or not by coverting them into integer using "int" method
million = 1_000_000
binary = 0b_0010
octa = 0o_64
hexa = 0x_23_ab
print(million)
print(binary)
print(octa)
print(hexa)
```

```
1000000
2
52
9131
```

숫자 표현을 끊어서 표현해 주어서 가독성이 좋은것 같습니다.

5. Naming

1. Single pre-underscore, _name

- Used for internal use
- Python doesn't import the names which starts with a single pre underscore.

In [26]:

```
class Test:
    def __init__(self):
        self.name = "Dankook Unversity"
        self._num = 7
obj = Test()
print(obj.name)
print(obj._num)
```

```
Dankook Unversity
7
```

In [27]:

```
## filename:- my_functions.py
def func():
    return "Dankook Unversity"
def _private_func():
    return 7
```

single underscore로 시작하는 `_private_func()` 의 경우 from import 를 통해 외부에서 사용할 수 없음

```
>>> def function(class):
 File "<stdin>", line 1
 def function(class):
 ^
SyntaxError: invalid syntax
>>> def function(class_):
... pass
...
>>>
>>>
```

1. Single post-underscore, name_

- Use Python Keywords as a variable, function or class names, you can use this convention for that
- Can avoid conflicts with the Python Keywords by adding an underscore at the end of the name which you want to use

underscore를 통해 nameing comfict를 방지할 수 있다.

1. Double pre-underscore, __name ,

- Tell the Python interpreter to rewrite the attribute name of subclasses to avoid naming conflicts
- Name Mangling:- interpreter of the Python alters the variable name in a way that it is challenging to clash when the class is inherited
- 더블 언더스코어의 경우 파이썬 인터프리터가 임의로 변수이름을 변경
- 따라서 클래스의method를 통해 출력해야 한다

In [28]:

```
def __init__(self):
  self.a = 1
  self._b = 2
  self.__c = 3

class Sample():
  def __init__(self):
    self.a = 1
    self._b = 2
    self.__c = 3
class SecondClass(Sample):
  def __init__(self):
    super().__init__()
    self.a = "overridden"
    self._b = "overridden"
    self.__c = "overridden"
obj1 = Sample()
obj2 = SecondClass()
print(obj2.a)
print(obj2._b)
print(obj2._SecondClass__c)

class SimpleClass:
  def __init__(self):
    self.__datacamp = "Excellent"
  def get_datacamp(self):
    return self.__datacamp
obj = SimpleClass()
print(obj.get_datacamp())
```

```
overridden
overridden
overridden
Excellent
```

In [29]:

```
class Sample():
  def __init__(self):
    self.__num__ = 7
obj = Sample()
obj.__num__
```

Out[29]:

```
7
```

[Item 16] Prefer get over in and KeyError to Handle Missing Dictionary Keys

Problem 1

Solution 1

In [7]:

```
counters = {
    'pumpernickel': 2,
    'sourdough': 1,
}
key = 'wheat'
if key in counters: # How about "if key not in counters"?
    counter = counters[key]
else:
    counter = 0
counters[key] = counter + 1
print(counters)
```

```
{'pumpernickel': 2, 'sourdough': 1, 'wheat': 1}
```

Solution 2

In [9]:

```
counters = {
    'pumpernickel': 2,
    'sourdough': 1,
}
key = 'wheat'

try:
    counter = counters[key]
except KeyError:
    counter = 0
counters[key] = counter + 1
print(counters)
```

```
{'pumpernickel': 2, 'sourdough': 1, 'wheat': 1}
```

Solution 3

In [11]:

```
counters = {
  'pumpernickel': 2,
  'sourdough': 1,
}
key = 'wheat'

count = counters.get(key, 0)
counters[key] = counter + 1
print(counters)
```

```
{'pumpernickel': 2, 'sourdough': 1, 'wheat': 1}
```

- get의 2번째 인자로 key에 해당하는 value가 없을시 2번째 인자를 출력하게 할 수 있다 get의 2번째 인자로 key에 해당하는 value가 없을시 2번째 인자를 출력하게 할 수 있다
- get은 key에 해당하는 값이 없으면 None을 출력한다

In [13]:

```
person = {'name': 'Phill', 'age': 22}

print('Name: ', person.get('name'))
print('Age: ', person.get('age'))

# value is not provided
print('Salary: ', person.get('salary'))

# value is provided
print('Salary: ', person.get('salary', 0.0))
```

```
Name:  Phill
Age:  22
Salary:  None
Salary:  0.0
```

Solution 4

In [12]:

```
counters = {
  'pumpernickel': 2,
  'sourdough': 1,
}
key = 'wheat'

count = counters.setdefault(key, 0)
counters[key] = count + 1
```

Problem 2

Solution 1

In [14]:

```
votes = {
  'baguette': ['Bob', 'Alice'],
  'ciabatta': ['Coco', 'Deb'],
}
key = 'brioche'
who = 'Elmer'

if key in votes:
  names = votes[key]
else:
  votes[key] = names = [] # Check this triple assignment statement!
names.append(who)
print(votes)
```

```
{'baguette': ['Bob', 'Alice'], 'ciabatta': ['Coco', 'Deb'], 'brioche': ['Elmer']}
```

Solution 2

In [16]:

```
votes = {
  'baguette': ['Bob', 'Alice'],
  'ciabatta': ['Coco', 'Deb'],
}
key = 'brioche'
who = 'Elmer'

try:
  names = votes[key]
except KeyError:
  votes[key] = names = []
names.append(who)
print(votes)
```

```
{'baguette': ['Bob', 'Alice'], 'ciabatta': ['Coco', 'Deb'], 'brioche': ['Elmer']}
```

Solution 3

In [18]:

```
votes = {
  'baguette': ['Bob', 'Alice'],
  'ciabatta': ['Coco', 'Deb'],
}
key = 'brioche'
who = 'Elmer'

names = votes.get(key)
if names is None:
  votes[key] = names = []
names.append(who)
print(votes)
```

```
{'baguette': ['Bob', 'Alice'], 'ciabatta': ['Coco', 'Deb'], 'brioche': ['Elmer']}
```

Solution 4

In [19]:

```
votes = {
    'baguette': ['Bob', 'Alice'],
    'ciabatta': ['Coco', 'Deb'],
}
key = 'brioche'
who = 'Elmer'

if (names := votes.get(key)) is None:
    votes[key] = names = []
names.append(who)
print(votes)
```

```
{'baguette': ['Bob', 'Alice'], 'ciabatta': ['Coco', 'Deb'], 'brioche': ['Elmer']}
```

Solution 5

In [20]:

```
votes = {
    'baguette': ['Bob', 'Alice'],
    'ciabatta': ['Coco', 'Deb'],
}
key = 'brioche'
who = 'Elmer'

names = votes.setdefault(key, [])
names.append(who)
print(votes)
```

```
{'baguette': ['Bob', 'Alice'], 'ciabatta': ['Coco', 'Deb'], 'brioche': ['Elmer']}
```

[Item 17] Prefer defaultdict over setdefault to Handle Missing Items in Internal State

Problem 1

Solution 1

In [21]:

```
visits = {
    'Mexico': {'Tulum', 'Puerto Vallarta'},
    'Japan': {'Hakone'},
}
visits.setdefault('France', set()).add('Arles') # short
print(visits)
```

```
{'Mexico': {'Tulum', 'Puerto Vallarta'}, 'Japan': {'Hakone'}, 'France': {'Arles'}}
```

Solution 2

In [23]:

```
if (japan := visits.get('Japan')) is None: # long
    visits['Japan'] = japan = set()
    japan.add('Kyoto')
print(visits)
```

```
{'Mexico': {'Tulum', 'Puerto Vallarta'}, 'Japan': {'Hakone'}, 'France': {'Arles'}}
```

Solution 1 & 2

In [24]:

```
from pprint import pprint
visits = {
    'Mexico': {'Tulum', 'Puerto Vallarta'},
    'Japan': {'Hakone'},
}
visits.setdefault('France', set()).add('Arles') # Short
if (japan := visits.get('Japan')) is None: # Long
    visits['Japan'] = japan = set()
japan.add('Kyoto')
original_print = print
print = pprint

print(visits)
print = original_print
```

```
{'France': {'Arles'},
 'Japan': {'Kyoto', 'Hakone'},
 'Mexico': {'Tulum', 'Puerto Vallarta'}}
```

Solution 3

In [25]:

```
class Visits:
    def __init__(self):
        self.data = {}
    def add(self, country, city):
        city_set = self.data.setdefault(country, set())
        city_set.add(city)
visits = Visits()
visits.add('Russia', 'Yekaterinburg')
visits.add('Tanzania', 'Zanzibar')
print(visits.data)
```

```
{'Russia': {'Yekaterinburg'}, 'Tanzania': {'Zanzibar'}}
```

매 호출마다 새로운 인스턴스를 만들기 때문에 비효율적입니다

Solution 4 : defaultdic

In [26]:

```
from collections import defaultdict
class Visits:
  def __init__(self):
    self.data = defaultdict(set)
  def add(self, country, city):
    self.data[country].add(city)
    visits = Visits()
visits.add('Russia', 'Yekaterinburg')
visits.add('Tanzania', 'Zanzibar')
print(visits.data)
```

```
{'Russia': {'Yekaterinburg'}, 'Tanzania': {'Zanzibar'}}
```

[Item 18] Know How to Construct KeyDependent Default Values with __missing__

Problem

Design a program to manage social network profile pictures on the file system

Solution 1

In [33]:

```
pictures = {}
path = 'profile_9991.png'
with open(path, 'wb') as f:
  f.write(b'image data here 9991')
if (handle := pictures.get(path)) is None:
  try:
    handle = open(path, 'a+b')
  except OSError:
    print(f'Failed to open path {path}')
    raise
  else:
    pictures[path] = handle
handle.seek(0)
image_data = handle.read()
print(pictures)
print(image_data)
```

```
{'profile_9991.png': <_io.BufferedRandom name='profile_9991.png'>}
b'image data here 9991'
```

Solution 2

In [32]:

```
pictures = {}
path = 'profile_9922.png'
with open(path, 'wb') as f:
    f.write(b'image data here 9922')

try:
    handle = pictures[path]
except KeyError:
    try:
        handle = open(path, 'a+b')
    except OSError:
        print(f'Failed to open path {path}')
        raise
    else:
        pictures[path] = handle

handle.seek(0)
image_data = handle.read()

print(pictures)
print(image_data)
```

```
{'profile_9922.png': <_io.BufferedRandom name='profile_9922.png'>}
b'image data here 9922'
```

Solution 3

In [36]:

```
pictures = {}
path = 'profile_9933.png'

with open(path, 'wb') as f:
    f.write(b'image data here 9933')

from collections import defaultdict

def open_picture(profile_path):
    try:
        return open(profile_path, 'a+b')
    except OSError:
        print(f'Failed to open path {profile_path}')
        raise
pictures = defaultdict(open_picture)
handle = pictures[path]
handle.seek(0)
image_data = handle.read()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-5b38a9dc61d4> in <module>
     14         raise
     15 pictures = defaultdict(open_picture)
---> 16 handle = pictures[path]
     17 handle.seek(0)
     18 image_data = handle.read()

TypeError: open_picture() missing 1 required positional argument: 'profile_path'
```

Solution 4

In [38]:

```
pictures = {}
path = 'profile_9944.png'
with open(path, 'wb') as f:
  f.write(b'image data here 9944')
def open_picture(profile_path):
  try:
    return open(profile_path, 'a+b')
  except OSError:
    print(f'Failed to open path {profile_path}')
    raise
class Pictures(dict):
  def __missing__(self, key):
    value = open_picture(key)
    self[key] = value
    return value

pictures = Pictures()
handle = pictures[path]
handle.seek(0)
image_data = handle.read()

print(image_data)
```

```
b'image data here 9944'
```

[Item 19] Never Unpack More Than Three Variables When Functions Return Multiple Values

Problem

- Determine various statistics for a population of alligators.
- Given a list , we need to calculate the minimium and maximum lengths in the propulation. ### Solution

In [40]:

```
def get_stats(numbers):
  minimum = min(numbers)
  maximum = max(numbers)
  return minimum, maximum

lengths = [63, 73, 72, 60, 67, 66, 71, 61, 72, 70]
minimum, maximum = get_stats(lengths)
print(f'Min: {minimum}, Max: {maximum}')
```

```
Min: 60, Max: 73
```

In [41]:

```
def my_function():
    return 1, 2

first, second = my_function()
assert first == 1
assert second == 2
```

Problem

Calculate how big each alligatore is relative to the population average

Solution

In [44]:

```
def get_avg_ratio(numbers):
    average = sum(numbers) / len(numbers)
    scaled = [x / average for x in numbers]
    scaled.sort(reverse=True)
    return scaled
longest, *middle, shortest = get_avg_ratio(lengths)
print(f'Longest: {longest:>4.0%}')
print(f'Shortest: {shortest:>4.0%}')
```

```
Longest: 108%
Shortest:  89%
```

Problem

Calculate the average length, median length, and total population size of the alligater

Solution

In [57]:

```
def get_avg_ratio(numbers):
    minimum = min(numbers)
    maximum = max(numbers)
    count = len(numbers)
    average = sum(numbers) / count

    sorted_numbers = sorted(numbers)
    middle = count // 2
    if count % 2 == 0:
        lower = sorted_numbers[middle-1]
        upper = sorted_numbers[middle]
        median = (lower + upper) / 2
    else:
        median = sorted_numbers[middle]

    return minimum, maximum, average, median, count
minimum, maximum, average, median, count = get_avg_ratio(lengths)
print(f'Min:{minimum}, Max:{maximum}')
print(f'Average:{average:>4.0%}, Median:{median:>4.0%}, Count:{count}')
```

```
Min:60, Max:73
Average:6750%, Median:6850%, Count:10
```

- 위와같이 return value가 너무 많으면, typo가 나타날 가능성이 있다.
- 따라서 namedtuple을 사용하는것을 추천 [item37]

[Item 20] Prefer Raising Exceptions to Returning None

In [31]:

```
def careful_divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return None
x, y = 1, 0
result = careful_divide(x, y)
if result is None:
    print('Invalid inputs')
```

```
Invalid inputs
```

In [32]:

```
x, y = 0, 5
result = careful_divide(x, y)
if not result:
    print('Invalid inputs') # This runs! But shouldn't
```

```
Invalid inputs
```

Solution 1

In [33]:

```
def careful_divide(a, b):
    try:
        return True, a / b
    except ZeroDivisionError:
        return False, None
x, y = 1, 0
success, result = careful_divide(x, y)
if not success:
    print('Invalid inputs')
x, y = 0, 5
_, result = careful_divide(x, y) # ignore the value
if not result:
    print('Invalid inputs') # This runs! But shouldn't
```

```
Invalid inputs
Invalid inputs
```

Solution 2

In [34]:

```
def careful_divide(a, b):
    try:
        return a / b
    except ZeroDivisionError as e:
        raise ValueError('Invalid inputs')

x, y = 5, 2
try:
    result = careful_divide(x, y)
except ValueError:
    print('Invalid inputs')
else:
    print('Result is %.1f' % result)
```

```
Result is 2.5
```

Solution 3

In [ ]:

```
def careful_divide(a: float, b: float)->float:
 """Divides a by b
 Raises:
 ValueError: When the inputs cannot be divided.
 """
 try:
   return a / b
 except ZeroDivisionError as e:
   raise ValueError('Invalid inputs')
x, y = 5, 2
try:
   result = careful_divide(x, y)
except ValueError:
   print('Invalid inputs')
else:
   print('Result is %.1f' % result)
```

[Item 21] Know How Closures Interact with Variable Scope

Python Closure

In [79]:

```
def make_printer(msg):
    msg = "hi there"

    def printer():
        print(msg)

    return printer

myprinter = make_printer("Hello World!")
myprinter()
myprinter()
myprinter()
```

```
hi there
hi there
hi there
```

python closure는 굳이 사용할 일이 많지 않을것 같지만 매개변수를 수정하여 전달하기 보다 이름으로 표현하는것이 더 가독성이 좋을때, 또는 다음과 같이 innerfunction을 활용할때 특히 유용한 것 같습니다

In [9]:

```
def has_permission(page):
    def permission(username):
        if username.lower() == "admin":
            return f"'{username}' has access to {page}."
        else:
            return f"'{username}' doesn't have access to {page}."
    return permission

check_admin_page_permision = has_permission("Admin Page")
```

In [11]:

```
check_admin_page_permision("admin")
```

Out[11]:

```
"'admin' has access to Admin Page."
```

In [12]:

```
check_admin_page_permision("john")
```

Out[12]:

```
"'john' doesn't have access to Admin Page."
```

Python Closure with nonlocal keyword

In [91]:

```
def make_counter():
  count = 0

  def inner():
    nonlocal count
    count += 1
    return count

  return inner

counter = make_counter()
c = counter()
print(c)
c = counter()
print(c)
c = counter()
print(c)
```

```
1
1
1
```

In [94]:

```
class Summer():
  def __init__(self):
    self.data = []
  def __call__(self, val):
    self.data.append(val)
    _sum = sum(self.data)
    return _sum
summer = Summer()
s = summer(1)
print(s)

s = summer(2)
print(s)

s = summer(3)
print(s)

s = summer(4)
print(s)
```

```
1
3
6
10
```

Nested function은 주로 함수 내에서 private한 계산을 수행해야 할때와 같이 모듈화 작업에 유용한 것 같다.

Closure vs. Class

In [96]:

```
class Summer():
  def __init__(self):
    self.data = []
  def __call__(self, val):
    self.data.append(val)
    _sum = sum(self.data)
    return _sum
summer = Summer()
s = summer(1)
print(s)
s = summer(2)
print(s)
s = summer(3)
print(s)
s = summer(4)
print(s)
```

```
1
3
6
10
```

In [100]:

```
def make_summer():
    data = []
    def summer(val):
        data.append(val)
        _sum = sum(data)
        return _sum
    return summer

summer = make_summer()
s = summer(1)
print(s)
s = summer(2)
print(s)
s = summer(3)
print(s)
s = summer(4)
print(s)
```

```
1
3
6
10
```

하지만 closure를 사용하니 매번 함수를 추가적으로 객체에 할당하는 과정이 필요한 것 같습니다.

따라서 기능별로 모듈화해야한다면 class로 표현하는것이 더 유리할것으로 보입니다.

Problem

- Sort a list of numbers but prioritize one group of numbers to come first

Solution 1

In [103]:

```
def sort_priority(values, group):
    def helper(x):
        if x in group:
            return (0, x)
        return (1, x)
    values.sort(key=helper)
numbers = [8, 3, 1, 2, 5, 4, 7, 6]
group = {2, 3, 5, 7}
sort_priority(numbers, group)
print(numbers)
```

```
[2, 3, 5, 7, 1, 4, 6, 8]
```

Solution 2

In [104]:

```
def sort_priority2(numbers, group):
    found = False
    def helper(x):
        if x in group:
            found = True # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
sort_priority2(numbers, group)
print(numbers)
```

```
[2, 3, 5, 7, 1, 4, 6, 8]
```

Solution 3

In [106]:

```
def sort_priority3(numbers, group):
    found = False
    def helper(x):
        nonlocal found # Added!!!
        # global found # how about!!!
        if x in group:
            found = True
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found

sort_priority3(numbers, group)
print(numbers)
```

```
[2, 3, 5, 7, 1, 4, 6, 8]
```

Solution 4

In [110]:

```
class Sorter:
    def __init__(self, group):
        self.group = group
        self.found = False
    def __call__(self, x):
        if x in self.group:
            self.found = True
            return (0, x)
        return (1, x)
sorter = Sorter(group)
numbers.sort(key=sorter)
assert sorter.found is True

print(numbers)
```

```
[2, 3, 5, 7, 1, 4, 6, 8]
```

[Item 22] Reduce Visual Noise with Variable Positional Arguments

Problem

Solution 1

In [1]:

```
def log(message, values):
    if not values:
        print(message)
    else:
        values_str = ', '.join(str(x) for x in values)
        print(f'{message}: {values_str}')
log('My numbers are', [1, 2])
log('Hi there', [])
```

```
My numbers are: 1, 2
Hi there
```

Solution 2

In [2]:

```
def log(message, *values):
    if not values:
        print(message)
    else:
        values_str = ', '.join(str(x) for x in values)
        print(f'{message}: {values_str}')
log('My numbers are', [1, 2])
log('Hi there')
```

```
My numbers are: [1, 2]
Hi there
```

In [4]:

```
def my_generator():
    for i in range(10):
        yield i
def my_func(*args):
    print(args)
it = my_generator()
my_func(*it)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

solution 3

In [6]:

```
def log(sequence, message, *values):
    if not values:
        print(f'{sequence} - {message}')
    else:
        values_str = ', '.join(str(x) for x in values)
        print(f'{sequence} - {message}: {values_str}')
log(1, 'Favorites', 7, 33) # New with *args OK
log(1, 'Hi there') # New message only OK
log('Favorite numbers', 7, 33) # Old usage breaks
```

```
1 - Favorites: 7, 33
1 - Hi there
Favorite numbers - 7: 33
```

[Item 23] Provide Optional Behavior with Keyword Arguments

In [8]:

```
def remainder(number, divisor):
    return number % divisor
assert remainder(20, 7) == 6 # positional arguments
assert remainder(number=20, divisor=7) == 6 # keyword arguments
assert remainder(20, divisor=7) == 6 # mixed!
assert remainder(divisor=7, number=20) == 6 # change the order
```

텐서플로우에서는 다양한 인자를 keyword arguments로 전달받는데 이는 다양한 인자를 헷갈리지 않게 키워드로 전달하여 편하게 사용할 수 있도록 해주는것같습니다.

```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

In [9]:

```
my_kwargs = {
 'number': 20,
 'divisor': 7,
}
assert remainder(**my_kwargs) == 6
```

In [10]:

```
my_kwargs = {
 'divisor': 7,
}
assert remainder(number=20, **my_kwargs) == 6
```

In [11]:

```
my_kwargs = {
 'number': 20,
}
other_kwargs = {
 'divisor': 7,
}
assert remainder(**my_kwargs, **other_kwargs) == 6
```

In [12]:

```
def print_parameters(**kwargs):
   for key, value in kwargs.items():
      print(f'{key} = {value}')
print_parameters(alpha=1.5, beta=9, gamma=4)
```

```
alpha = 1.5
beta = 9
gamma = 4
```

In [13]:

```
def flow_rate(weight_diff, time_diff):
   return weight_diff / time_diff
weight_diff = 0.5
time_diff = 3
flow = flow_rate(weight_diff, time_diff)
print(f'{flow:.3} kg per second')
```

```
0.167 kg per second
```

In [14]:

```
def flow_rate(weight_diff, time_diff, period=1):
   return (weight_diff / time_diff) * period

flow_per_second = flow_rate(weight_diff, time_diff)
flow_per_hour = flow_rate(weight_diff, time_diff, period=3600)
```

In [15]:

```
def flow_rate(weight_diff, time_diff, period=1, units_per_kg=1):
   return ((weight_diff * units_per_kg) / time_diff) * period
```

[Item 24] Use None and Docstrings to Specify Dynamic Default Arguments

In [16]:

```
from time import sleep
from datetime import datetime
def log(message, when=datetime.now()): # when is not a static value
    print(f'{when}: {message}')
log('Hi there!')
sleep(0.1)
log('Hello again!')
```

```
2021-04-27 16:54:48.117401: Hi there!
2021-04-27 16:54:48.117401: Hello again!
```

In [20]:

```
def log(message, when=None):
    """Log a message with a timestamp.
    Args:
    message: Message to print.
    when: datetime of when the message occurred.
    Defaults to the present time.
    """
    if when is None:
        when = datetime.now()
    print(f'{when}: {message}')
log('Hi there!')
sleep(0.1)
log('Hello again!')
```

```
2021-04-27 16:56:04.682770: Hi there!
2021-04-27 16:56:04.783635: Hello again!
```

In [21]:

```
import json
def decode(data, default={}):
    try:
        return json.loads(data)
    except ValueError:
        return default
foo = decode('bad data')
foo['stuff'] = 5
bar = decode('also bad')
bar['meep'] = 1
print('Foo:', foo)
print('Bar:', bar)
assert foo is bar #
```

```
Foo: {'stuff': 5, 'meep': 1}
Bar: {'stuff': 5, 'meep': 1}
```

In [ ]:

```
def decode(data, default=None):
   """Load JSON data from a string.
   Args:
   data: JSON data to decode.
   default: Value to return if decoding fails.
   Defaults to an empty dictionary.
   """
   try:
      return json.loads(data)

foo = decode('bad data')
foo['stuff'] = 5
bar = decode('also bad')
bar['meep'] = 1
print('Foo:', foo)
print('Bar:', bar)
assert foo is not bar
```

In [25]:

```
from typing import Optional
def log_typed(message: str, when: Optional[datetime]=None) -> None:
   """Log a message with a timestamp.
   Args:
   message: Message to print.
   when: datetime of when the message occurred.
   Defaults to the present time.
   """
   if when is None:
      when = datetime.now()
   print(f'{when}: {message}')
```

[Item 25] Enforce Clarity with KeywordOnly and Positional-Only Arguments

In [27]:

```
def safe_division(number, divisor, ignore_overflow, ignore_zero_division):
    try:
        return number / divisor
    except OverflowError:
        if ignore_overflow:
            return 0
        else:
            raise
    except ZeroDivisionError:
        if ignore_zero_division:
            return float('inf')
        else:
            raise
result = safe_division(1.0, 10**500, True, False)
print(result)
result = safe_division(1.0, 0.0, False, True)
print(result)
```

```
0
inf
```

In [28]:

```
def safe_division(number, divisor, ignore_overflow=False, ignore_zero_division=False):
    try:
        return number / divisor
    except OverflowError:
        if ignore_overflow:
            return 0
        else:
            raise
    except ZeroDivisionError:
        if ignore_zero_division:
            return float('inf')
        else:
            raise
result = safe_division(1.0, 10**500, ignore_overflow=True)
print(result)
result = safe_division(1.0, 0.0, ignore_zero_division=True)
print(result)
```

```
0
inf
```