List

- [ ]와 ,로 표현하는 데이터 형으로 여러 가지 형태의 정보를 하나의 리스트로 묶어서 나타냄

In [1]:

```
squares = [1, 4, 9, 16, 25]
zoo_animal = ['tiger', 'lion', 'monkey']
smith_information = ['Smith', 1990, 'London']
lee_information = ['Lee', 1975, 'Seoul']
customer = [smith_information, lee_information]
```

- len() 함수를 이용하여 길이를 구함

In [2]:

```
len(squares)
```

Out[2]:

5

In [3]:

```
len(zoo_animal)
```

Out[3]:

3

In [4]:

```
len(customer)
```

Out[4]:

2

List의 인덱스(index)

In [5]:

```
squares = [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- index를 이용한 list 접근
  - 첫 아이템의 index는 0부터 시작

In [6]:

```
squares[0]
```

Out[6]:

1

In [7]:

```
squares[1]
```

Out[7]:

4

In [8]:

```
squares[4]
```

Out[8]:

25

list의 slice 연산

In [9]:

```
squares = [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [10]:

```
squares[0:3]
```

Out[10]:

[1, 4, 9]

혹은

In [11]:

```
squares[:3]
```

Out[11]:

[1, 4, 9]

- 인덱스 0의 item부터 인덱스 3의 item 이전까지

In [12]:

```
squares[3:6]
```

Out[12]:

[16, 25, 36]

- 인덱스 3의 item부터 인덱스 6의 item 이전까지

In [13]:

```
squares[6:9]
```

Out[13]:

[49, 64, 81]

혹은

In [14]:

```
squares[6:]
```

Out[14]:

```
[49, 64, 81]
```

- 인덱스 0의 item부터 인덱스 6의 item 이전까지 2의 간격으로

In [15]:

```
squares[0:6:2]
```

Out[15]:

```
[1, 9, 25]
```

list 내용 변경

In [16]:

```
squares = [1, 4, 10, 17, 25]
```

In [19]:

```
squares[2] = 9
print(squares)
```

```
[1, 4, 9, 17, 25]
```

In [20]:

```
squares[3] = 16
print(squares)
```

```
[1, 4, 9, 16, 25]
```

In [21]:

```
squares = [1, 4, 10, 17, 25]
print(squares)
```

```
[1, 4, 10, 17, 25]
```

In [22]:

```
squares[2:4] = [9, 16]
print(squares)
```

```
[1, 4, 9, 16, 25]
```

In [23]:

```
squares = squares + [36, 49, 64]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64]
```

In [24]:

```
squares.append(81)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [25]:

```
squares.append(10**2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [26]:

```
squares[5:10] = []
print(squares)
```

```
[1, 4, 9, 16, 25]
```

In [27]:

```
squares[1] = []
print(squares)
```

```
[1, [], 9, 16, 25]
```

In [28]:

```
squares[1:2] = []
print(squares)
```

```
[1, 9, 16, 25]
```

In [29]:

```
squares.insert(1, 4)
print(squares)
```

```
[1, 4, 9, 16, 25]
```

list내의 list

In [30]:

```
A = [[0,1,2,3,4], ['a', 'b']]
```

In [31]:

```
A[0]
```

Out[31]:

```
[0, 1, 2, 3, 4]
```

In [32]:

```
A[1]
```

Out[32]:

```
['a', 'b']
```

In [33]:

```
A[0][0]
```

Out[33]:

```
0
```

In [34]:

```
A[0][1]
```

Out[34]:

```
1
```

In [35]:

```
A[1][0]
```

Out[35]:

```
'a'
```

In [36]:

```
A[1][1]
```

Out[36]:

```
'b'
```

list의 다양한 method

In [37]:

```
animals = ['dog', 'cat', 'duck', 'lion', 'snake']
```

In [38]:

```
animals.index('cat')
```

Out[38]:

```
1
```

In [39]:

```
animals.remove('duck')
animals
```

Out[39]:

```
['dog', 'cat', 'lion', 'snake']
```

- list data type은 다양한 method들을 포함하며 아래의 링크에서 확인
- https://docs.python.org/2/tutorial/datastructures.html#more-on-lists (https://docs.python.org/2/tutorial/datastructures.html#more-on-lists)

참조와 복사

- 하나의 객체를 L1, L2가 동시에 지칭하여, L1의 원소가 변하면 L2의 원소도 같이 변함

In [40]:

```
L1 = [1, 2, 3]
L2 = L1
L1[0] = 4
```

In [43]:

```
print(L1)
```

```
[4, 2, 3]
```

In [44]:

```
print(L2)
```

```
[4, 2, 3]
```

참조와 복사(2)

- L1에 새로운 리스트를 대응하면 원래의 참조를 끊고 새로운 대입값을 적용

In [45]:

```
L1 = [1, 2, 3]
L2 = L1
L1 = [4, 5, 6]
```

In [46]:

```
print(L1)
```

```
[4, 5, 6]
```

In [47]:

```
print(L2)
```

```
[1, 2, 3]
```

참조와 복사(3)

- 같은 대상을 참조하는 것이 아니라 대상을 복사하여 사용하고 싶을 때

In [48]:

```
L1 = [1, 2, 3]
L2 = L1[:]
```

혹은

In [49]:

```
L2 = list(L1)
```

In [50]:

```
L1[0] = 4
print(L1)
```

```
[4, 2, 3]
```

In [51]:

```
print(L2)
```

```
[1, 2, 3]
```

복사의 예 (Cloning)

- Python 언어의 모든 매개 변수는 참조로 전달된다.
- 즉, 함수 내에서 매개 변수가 참조하는 것을 변경하면 변경 내용은 호출 함수 밖에서도 반영된다.
- 이를 pass by reference(참조)라고 한다.
- 다음 예제를 보면 함수 내부에서 변화가 함수 바깥으로 영향을 미치는 것을 볼 수 있다.

In [3]:

```
def changeme( mylist ):
    mylist.append([1,2,3,4]);
    print("Values inside the function: ", mylist)
    return

mylist = [10,20,30];
changeme( mylist );
print("Values outside the function: ", mylist)
```

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

- 만약 이를 원치 않으면, 다음과 같이 복사하여 사용한다.

In [4]:

```
def changeme( mylist ):
    mylist = mylist[:]
    mylist.append([1,2,3,4]);
    print("Values inside the function: ", mylist)
    return

mylist = [10,20,30];
changeme( mylist );
print("Values outside the function: ", mylist)
```

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30]
```

복사의 예(2)

다음의 두 removeDup를 비교해 보자.

In [53]:

```
def removeDup(L1, L2):
    for e1 in L1:
        if e1 in L2:
            L1.remove(e1)

L1=[1,2,3,4]
L2=[1,2,5,6]
removeDup(L1, L2)
print('L1 = ', L1)
```

```
L1 =  [2, 3, 4]
```

In [54]:

```
def removeDup(L1, L2):
    for e1 in L1[:]:
        if e1 in L2:
            L1.remove(e1)

L1=[1,2,3,4]
L2=[1,2,5,6]
removeDup(L1, L2)
print('L1 = ', L1)
```

```
L1 =  [3, 4]
```

string과 list의 차이점

- string과 list는 인덱스를 통해 접근할 수 있는 등 비슷한 점이 많음

In [55]:

```
my_string = 'test'
print(my_string[0])
```

```
t
```

- list는 인덱스를 통해 원소를 변화시키는 것이 가능(mutable)
  - my_list[i] = new_element

- string은 불가능 (immutable)

In [56]:

```
my_string[0] = 'k'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-56-76a7872484ea> in <module>()
----> 1 my_string[0] = 'k'

TypeError: 'str' object does not support item assignment
```

list concatenation

In [57]:

```
x = [1, 2, 3]
x.extend([4, 5, 6])   # x is now [1, 2, 3, 4, 5, 6]
print(x)
```

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

만약 x를 변화시키고 싶지 않으면

In [58]:

```
x = [1, 2, 3]
y = x + [4, 5, 6]   # y is [1, 2, 3, 4, 5, 6] x는 그대로
print(x)
print(y)
```

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

- 하나씩 추가할 경우는 append를 많이 씀

In [59]:

```
x = [1, 2, 3]
x.append(0)     # x is now [1, 2, 3, 0]
print(x)
```

```
[1, 2, 3, 0]
```

List comprehension

In [60]:

```
squares = [x**2 for x in range(4)]
print(squares)
```

[0, 1, 4, 9]

In [61]:

```
[(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x!=y]
```

Out[61]:

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

In [62]:

```
[abs(x) for x in [-1, 2, -3]]
```

Out[62]:

[1, 2, 3]

In [63]:

```
from math import pi
[str(round(pi, i)) for i in range(1, 6)]
```

Out[63]:

['3.1', '3.14', '3.142', '3.1416', '3.14159']

In [65]:

```
mixed = [1, 2, 'a', 3, 4.0]
print([x**2 for x in mixed if type(x) == int])
```

[1, 4, 9]

Tuple - 투플

- Tuple은 list와 매우 비슷함
  - 단 immutable
  - 내용을 바꾸는 연산을 제외한 list의 많은 연산을 똑같이 수행 가능
  - []가 아닌 ( ) 혹은 아무런 괄호 없이 표현

In [66]:

```
my_tuple = (1, 2)
my_tuple[1] = 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-45ea6b3730b7> in <module>()
      1 my_tuple = (1, 2)
----> 2 my_tuple[1] = 3

TypeError: 'tuple' object does not support item assignment
```

- 여러 값을 같이 반환할 때 유용

In [67]:

```
def sum_product(x, y): return x+y, x*y
```

In [68]:

```
sum_product(2, 3)
```

Out[68]:

```
(5, 6)
```

- 여러 값을 동시에 부여 (list로도 가능)

In [69]:

```
x, y = (1, 2)
x, y = y, x     #Python에서 두 변수 값을 swap하는 법
print(x, y)
```

```
2 1
```

Example : Tuple

In [70]:

```
t1 = ()
t2 = (1, 'two', 3)
t3 = (10,)   # 원소가 하나인 tuple
```

In [71]:

```
print(t1)
```

```
()
```

In [72]:

```
print(t2)
```

```
(1, 'two', 3)
```

In [73]:

```
print(t3)
```

```
(10,)
```

In [74]:

```
t1 = (1, 'two', 3)
t2 = (t1, 3.25)
```

In [75]:

```
print(t2)
```

```
((1, 'two', 3), 3.25)
```

In [76]:

```
print(t1 + t2)
```

```
(1, 'two', 3, (1, 'two', 3), 3.25)
```

In [77]:

```
print((t1 + t2)[3])
```

```
(1, 'two', 3)
```

In [78]:

```
print((t1 + t2)[2:5])
```

```
(3, (1, 'two', 3), 3.25)
```

Example : 공약수

In [79]:

```
def findDivisors (n1, n2):
    divisors = ()  #empty tuple
    for i in range(1, min (n1, n2) + 1):
        if n1%i == 0 and n2%i == 0:
            divisors = divisors + (i, )
    return divisors
```

In [80]:

```
divisors = findDivisors(20, 100)
print(divisors)
```

```
(1, 2, 4, 5, 10, 20)
```

for문과 list의 연계

In [82]:

```
my_list = [10, 'hi', -1.2, 'first']
```

In [83]:

```
for x in my_list:
    print(x)
```

```
10
hi
-1.2
first
```

In [84]:

```
student_list = ['Tom', 'Jack', 'Jane', 'Susan']
student_list.sort()
for student in student_list:
    print(student)
```

```
Jack
Jane
Susan
Tom
```

enumerate

- enumerate을 이용하여 인덱스와 리스트의 값에 접근

In [85]:

```
student_list = ['Tom', 'Jane', 'Susie', 'Chris']
for index, value in enumerate(student_list):
    print(index, value)
```

```
0 Tom
1 Jane
2 Susie
3 Chris
```

- for i in range(len(some_list)):
  do something(i) # not Pythonic
- for i, _ in enumerate(some_list) :
  do something(i) # Pythonic

zip

- 두 개 이상의 리스트를 원소 별로 합쳐서 투플들의 리스트로 생성

In [86]:

```
x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)    # [(1, 4), (2, 5), (3, 6)]
print(zipped)
```

```
<zip object at 0x00000205DB9A0848>
```

String, tuple, list 공통적인 기능

- seq[i] : i번째 원소
- len(seq) : 길이
- seq1 + seq2 : 연결
- n * seq : n번 반복
- seq[start:end] : 일부
- e in seq : 만약 e가 seq에 속하면 True, 아니면 False
- e not in seq : 만약 e가 seq에 속하면 False, 아니면 True
- for e in seq : 반복문

Dictionary

- Dictionary는 list처럼 여러 원소들을 하나의 collection으로 사용
- List에서는 index를 통하여 원소에 접근하였지만, dictionary에서는 key를 통하여 접근
- Dictionary의 기본 구조
  - {key1: element1, key2: element2, … }
- key는 일반적으로 string이나 숫자를 사용
  - 예제 – 학생 이름과 점수
    - {'Tom': 92, 'Jane': 95, 'Amy': 73, 'Nick': 81}
    - 학생 이름이 key로 사용되었으며, 점수가 값으로 사용됨

key를 통한 접근

In [87]:

```
scores = {'Tom': 92, 'Jane': 95, 'Amy': 73, 'Nick': 81}
```

In [88]:

```
scores['Tom']
```

Out[88]:

92

In [89]:

```
scores[0]  # error 발생 : index를 통해 접근할 수 없음
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-89-ee3bcbc05378> in <module>()
----> 1 scores[0]  # error 발생 : index를 통해 접근할 수 없음

KeyError: 0
```

- list에서와 같이 key를 통하여 원소의 내용을 바꿀 수 있음 (mutable)

In [91]:

```
scores['Tom'] = 100
print(scores)
```

```
{'Tom': 100, 'Jane': 95, 'Amy': 73, 'Nick': 81}
```

- 기존에 'Robin'이 없으므로 새로운 원소 추가

In [92]:

```
scores['Robin'] = 50
print(scores)
```

```
{'Tom': 100, 'Jane': 95, 'Amy': 73, 'Nick': 81, 'Robin': 50}
```

- 기존의 원소 제거

In [93]:

```
del scores['Nick']
print(scores)
```

```
{'Tom': 100, 'Jane': 95, 'Amy': 73, 'Robin': 50}
```

Dictionary와 key의 순서

In [94]:

```
scores = {'Tom': 92, 'Jane': 95, 'Amy': 73, 'Nick': 81}
```

In [95]:

```
scores.keys()
```

Out[95]:

```
dict_keys(['Tom', 'Jane', 'Amy', 'Nick'])
```

- dictionary는 순서가 정해지지 않음(unordered)

In [96]:

```
for key in scores:
    print(key, scores[key])
```

```
Tom 92
Jane 95
Amy 73
Nick 81
```

구조화된 데이터를 dictionary로 표현

In [97]:

```
tweet = {
    "user" : "joelgrus",
    "text" : "Data Science is Awesome",
    "retweet_cout" : 100,
    "hashtags" : ["#data", "#science"]
}

tweet_keys = tweet.keys()    #list of keys
tweet_values = tweet.values()   #list of values
tweet_items = tweet.items()   #list of (key, values)
```

In [98]:

```
"user" in tweet_keys  # True but slow
```

Out[98]:

```
True
```

In [99]:

```
"user" in tweet         # True faster
```

Out[99]:

```
True
```

In [100]:

```
"joelgrus" in tweet_values  #True
```

Out[100]:

```
True
```

Counter

- sequence를 dictionary와 비슷한 형태로 바꾸어 줌

In [101]:

```
from collections import Counter
c = Counter([0, 1, 2, 0])
print(c)
```

```
Counter({0: 2, 1: 1, 2: 1})
```

- c의 결과는 다음과 같다 : Counter({0: 2, 1: 1, 2: 1})
- 0이 2개, 1이 1개, 2가 1개 있음을 나타냄

Generator and Iterator

- List의 문제 중 하나는 그 길이가 매우 커질 수 있다는 점.
- 예를 들어 Python2에서 range(1000000)은 백만 개의 원소를 가짐.
- 때로는 이 데이터를 모두 생성하는 것은 낭비일 수 있음.
- generator는 for문 등을 이용하여 반복 가능하지만, 그 값들을 생성하는 것은 필요한 경우만 실행.
- yield를 이용하여 generator를 만드는 방법

In [103]:

```
def lazy_range(n):
    i = 0
    while i<n:
        yield i
        i += 1
```

generator

for i in lazy_range(10):
  do_something_with(i)

- Python은 실제로 위와 비슷한 xrange라 불리우는 함수가 있으며,
  - Python 3에서는 range 자체가 generator를 반환.
- 다음 세가지는 같은 역할을 함,
  - for i in range(n):
  - for i in lazy_range(n):
  - for i in xrange(n):
- .next()를 이용하여 반복되는 값을 확인
  - x = lazy_range(10)♂x.next()♂x.next()♂x.next()

generator(2)

- 무한 시퀀스를 만들 수도 있다.

In [104]:

```
def natural_numbers():
    n = 1
    while True:
        yield n
        n += 1
```

- for를 이용한 generator 생성

In [105]:

```
lazy_evens_below_20 = (i for i in lazy_range(20) if i%2==0)
```

dictionary의 item()

- dict의 items() method는 yield를 이용하여 key-value 형태의 generator를 반환

In [106]:

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
```

In [108]:

```
for k, v in knights.items():
    print(k, v)
```

```
gallahad the pure
robin the brave
```

Sorting - 정렬

- Python 리스트는 sort method를 이용하여 정렬 가능

In [109]:

```
x = [4, 1, 2, 3]
y = sorted(x)    # y는 [1, 2, 3, 4], x는 변하지 않음
print(x)
print(y)
```

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

In [110]:

```
x.sort()            # x가 이제 [1, 2, 3, 4]
print(x)
```

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

- 응용 : 절대값 기준으로 내림차순으로 정렬

In [111]:

```
x = sorted([-4, 1, -2, 3], key=abs, reverse=True)
print(x)
```

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

랜덤 (random) 리스트

In [112]:

```
import random
four_uniform_randoms = [random.random() for _ in range(4)]
print(four_uniform_randoms)
```

```
[0.8502839930740965, 0.9575258814239622, 0.8421408083556148, 0.47254988392479125]
```

- random.random()은 0과 1사이에서 uniform하게 분포된 랜덤 숫자를 반환
- _는 다시 사용하지 않을 dummy variable을 표현
  - x로 바꾸어도 상관없음
- 랜덤 샘플 추출 (중복 제외)

In [113]:

```
lottery_numbers = range(60)
random.sample(lottery_numbers, 6)   # 6개의 랜덤 샘플
```

Out[113]:

[7, 27, 58, 49, 22, 50]

- 랜덤 선택 (중복 허용)

In [114]:

```
four_with_replacement = [random.choice(range(10)) for _ in range(4)]
print(four_with_replacement)
```

[5, 7, 1, 8]

In [ ]:

```
len()
```