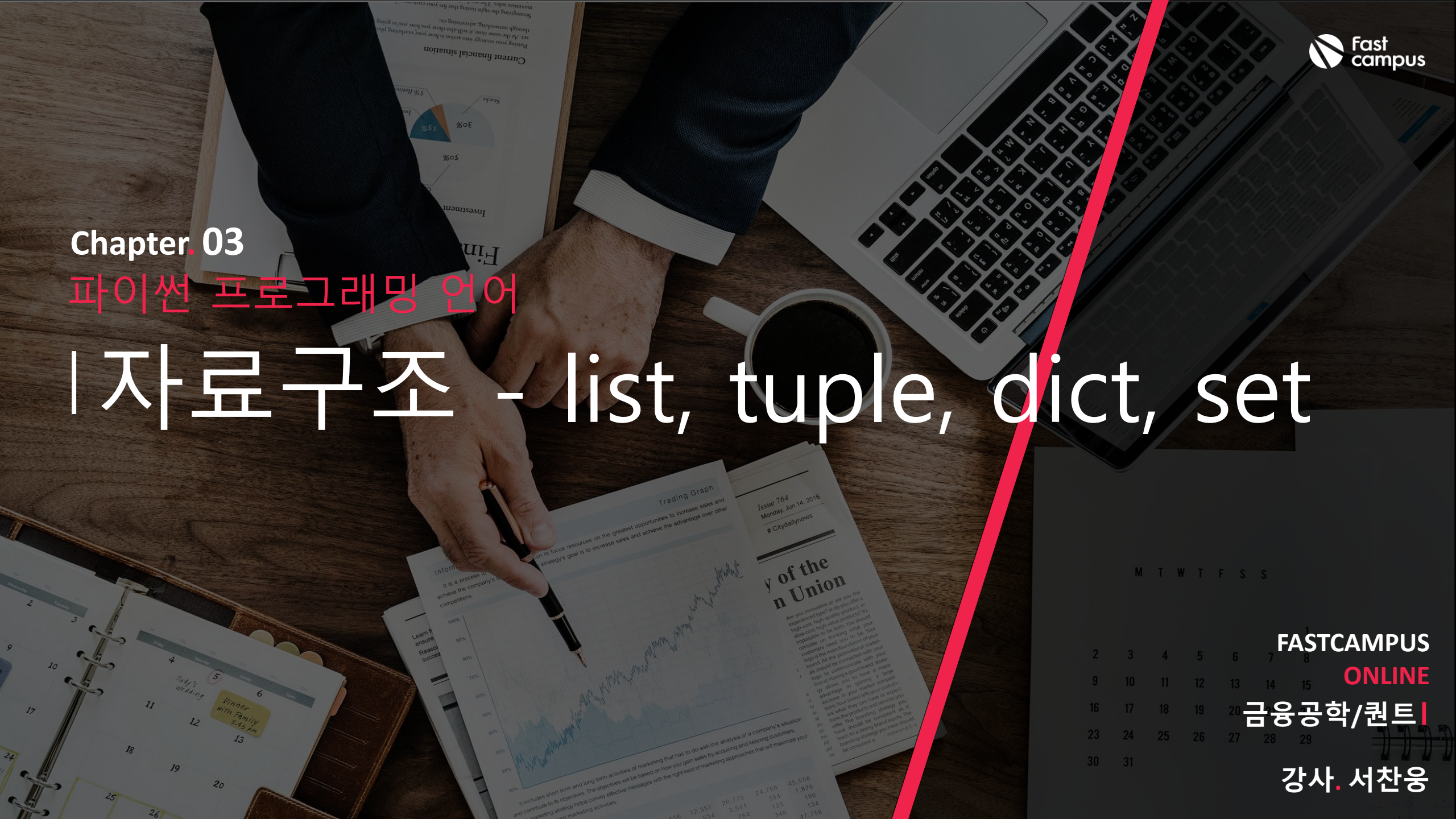


Chapter. 03

파이썬 프로그래밍 언어

# I 자료구조 - list, tuple, dict, set

FASTCAMPUS
ONLINE
금융공학/퀀트I

강사. 서찬웅

# I 이번시간에 공부할 내용

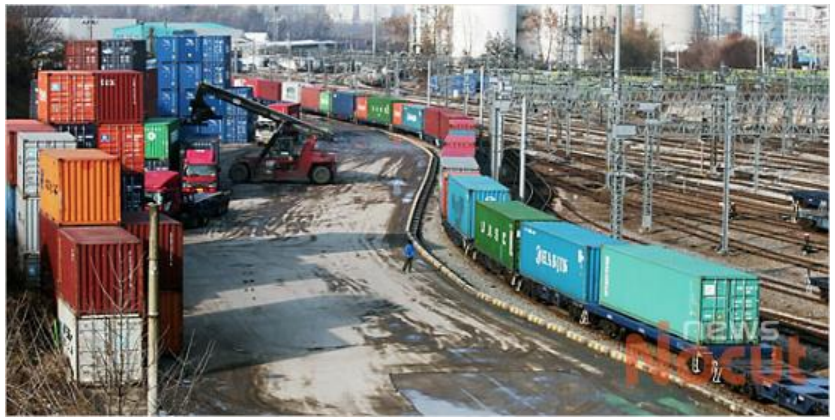


1. 리스트(list)
2. 튜플(tuple)
3. 사전(dictionary)
4. 리스트, 사전 정렬
5. 집합(set)

FAST CAMPUS
ONLINE
서찬웅 강사.

# I 리스트

- list는 원소를 주어진 순서대로 보관하는 데이터 유형입니다.
- list에 보관하는 원소나 값은 매우 다양한 데이터 타입이 될 수 있습니다.
- list 안에 list를 보관할 수도 있습니다.
- list는 열차처럼 열차 한량 한량들이 모여서 전체 열차를 구성하듯 데이터가 모여 list 전체를 구성 합니다.
- 열차에 사람이나, 화물, 자동차, 탱크를 수송할 수 있듯이 list 또한 파이썬의 데이터 유형을 저장할 수 있습니다.
- 리스트는 시퀀스형 자료형입니다. 시퀀스형 자료형의 특성을 가지고 있습니다.



# I 리스트



- list를 생성해 보겠습니다.
  - list 이름 = [원소1, 원소2, 원소3..... 원소N]
  - 빈 list 만들 때는 아래와 같습니다.
    - list 이름 = list()
    - list 이름 = []

```
stock = ['삼성전자', '엘지전자', 'SK하이닉스']
```

- 인덱스를 사용하여 값을 출력할수 있습니다.

```
print (stock[1])
```

```
엘지전자
```

- 원하는 인덱스의 값을 변경하고 싶을 때는 시퀀스 접근 방식으로 접근하여 할당 연산자(=)를 사용하여 대입합니다.

```
stock[1] = '현대자동차'
```

```
print (stock[1])
```

```
현대자동차
```

# I 리스트

- list는 스퀀스형 자료형입니다.
- 앞의 예제의 데이터를 그림으로 표현하면 아래와 같습니다.

| stock | 삼성전자 | 현대자동차 | SK하이닉스 |
|---|---|---|---|
| 양의 index | 0 | 1 | 2 |
| 음의 index | -3 | -2 | -1 |

- for문을 사용해서 출력을 해보겠습니다.

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

```
삼성전자
현대자동차
SK하이닉스
```

```
for x in stock[::-1]:
    print (x)
```

```
SK하이닉스
현대자동차
삼성전자
```

# I 리스트



- list에 새로운 값을 추가해보겠습니다.
  - list에 하나의 데이터를 추가하고 싶다면 append() 메소드를 사용하여 데이터를 입력합니다.

```
stock.append('gs')
```

```
stock
```

```
['삼성전자', '현대자동차', 'SK하이닉스', 'gs']
```

```
del stock[3]
```

```
stock
```

```
['삼성전자', '현대자동차', 'SK하이닉스']
```

del() 함수를 사용하면
1. 원하는 인덱스의 데이터를 삭제할 수 있습니다.
2. 전체 list를 삭제할 수도 있습니다.

  - list의 마지막 뒤에 여러 개의 데이터를 입력하고 싶다면 extend() 메소드를 사용합니다.
  - extend() 메소드의 인자값은 리스트 유형으로 전달합니다.

```
stock.extend(['롯데제과', '신세계'])
```

```
stock
```

```
['삼성전자', '현대자동차', 'SK하이닉스', '롯데제과', '신세계']
```

  - insert() 메소드를 사용하면 원하는 인덱스에 데이터를 저장할 수 있습니다.

```
stock.insert(2, '엘지전자')
```

```
stock
```

```
['삼성전자', '현대자동차', '엘지전자', 'SK하이닉스', '롯데제과', '신세계']
```

# I 리스트



- slicing으로 데이터를 접근해보겠습니다.

```
stock[1:3]
```

```
['현대자동차', '엘지전자']
```

```
stock[:2]
```

```
['삼성전자', '현대자동차']
```

```
stock[::2]
```

```
['삼성전자', '엘지전자', '롯데제과']
```

- 문자열에서 학습한 시퀀스 특성 그대로 list에서도 적용됩니다.
- index() 메소드를 사용하면 list의 데이터 중 같은 값이 있다면 처음 발견한 위치의 index 값을 반환합니다.
- 하지만 값이 없다면 ValueError를 일으킵니다.

```
stock.index('엘지전자')
```

```
2
```

```
stock.index("한화")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-35-8013a3b60418> in <module>
----> 1 stock.index("한화")

ValueError: '한화' is not in list
```

# I 리스트

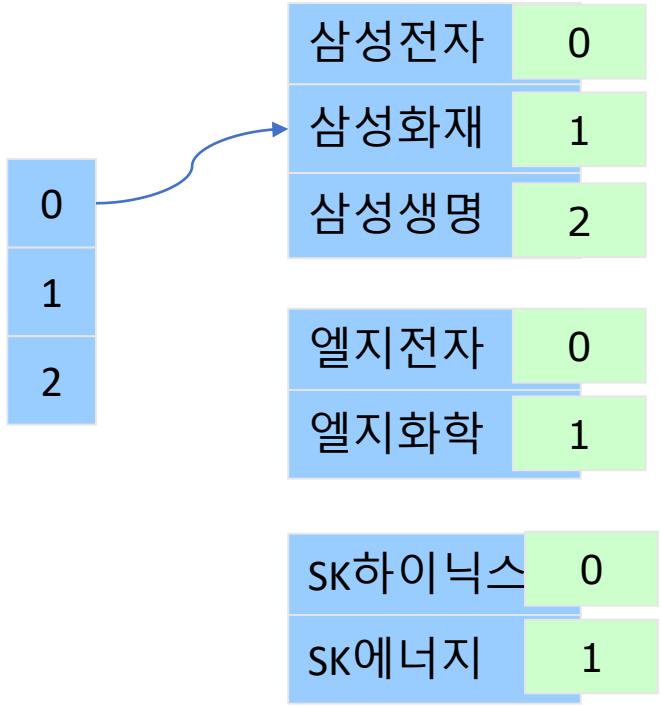
- 중첩 list에 대해서 알아보겠습니다.
- list안에 데이터는 사용자가 원하는 값을 저장할 수 있다고 이야기 했습니다.
- 당연히 list안에 list를 저장할 수 있습니다.

```
stock2 = [['삼성전자', '삼성화재', '삼성생명'], ['엘지전자', '엘지화학'], ['SK하이닉스', 'SK에너지']]
```

- 중첩 list에서 데이터의 접근도 [][]을 이용하면 됩니다.

```
stock2[0][1]
```

```
'삼성화재'
```



# I 리스트

- 자주 사용하는 메소드

| 메소드 | 설명 |
|---|---|
| append | 자료를 리스트 끝에 추가 (혹은 스택의 push) |
| insert | 자료를 지정된 위치에 삽입 |
| index | 요소 검색(Search) |
| count | 요소 개수 알아내기 |
| sort | 리스트 정렬 |
| reverse | 자료 순서 바꾸기 |
| remove | 지정 자료 값 한 개 삭제 |
| pop | 리스트의 마지막 값을 읽어내고 삭제 (스택의 pop) |
| extend | 리스트를 추가 |

# I 리스트

- list의 데이터 정렬하기
  - sort() 메소드를 사용하면 데이터를 정렬할 수 있습니다.
  - 역순으로 정렬할땐 reverse 옵션을  사용합니다.

- sort() 메소드를 사용할 때 참고할 점
  - sort 메소드는 key 옵션을 사용하여 문자열을 int형으로 인식하게 만들고 정렬합니다.
  - reverse option를 사용하면 역순으로 정렬합니다.

```
L = ['123', '34', '56', '23456']
```

```
L.sort()
```

```
L
```

```
['123', '23456', '34', '56']
```

```
L.sort(key=int)
```

```
L
```

```
['34', '56', '123', '23456']
```

```
L.sort(reverse=True)
```

```
L
```

```
['56', '34', '23456', '123']
```

```
L.sort(reverse=True, key=int)
```

```
L
```

```
['23456', '123', '56', '34']
```



FAST CAMPUS
ONLINE
서찬웅 강사.

# I 리스트

- list comprehension(리스트 내장) 이란?
  - list를 쉽게 생성하기 위한 기능입니다.
  - 0부터 9까지 숫자를 제곱승을 구해서 list에 저장하는 코드를 살펴보겠습니다.
  - list comprehension 문법은 아래와 같습니다.

    [ expression for expr in sequence1
    for expr2 in sequence2 …
    for expN in sequenceN
    if condition ]

```
compre_1 = list()
for x in range(0,10):
    compre_1.append(x**2)
```

```
compre_1
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

  - 위의 문법을 사용해서 compre_1과 동일한 데이터를 가진 list를 만들어보겠습니다.

표현식 for expr in seqence

```
compre_2 = [x**2 for x in range(0,10)]
```

```
compre_2
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# I 리스트



- list comprehension를 사용하여 데이터를 생성할 때 조건문을 넣을 수 있습니다.
- 아래 예제는 0 ~ 29까지 숫자 중에서 짝수만 tmp라는 list에 저장합니다.

```
tmp = [k for k in range(0,30) if k % 2 == 0]
```

```
tmp
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

- 아래 예제는 구구단에서 짝수단만 저장하는 예제입니다.

```
[x * y for x in range(2,10) for y in range(1,10) if x % 2 == 0]
```

# I 리스트 문제



- 문제 1 : 앞에서 만든 윤년을 list comprehension 방식으로 저장해 봅시다.

- 문제 2 : 문자열 메소드중에 split() 메소드는 결과값을 list으로 반환합니다.
  문자열의 join() 메소드는 list의 데이터를 하나의 문자열로 만들어 줍니다.
  위 2가지 특성을 활용하여 다음 문제를 풀어보겠습니다.

```
str_text = '''first line
second line
third line'''
```

1. 앞서 여러 줄 문자열을 다음과 같이 출력합니다.
   first line:second line:third line

2. 앞서 여러 줄 문자열에서 두 번째 줄 첫 번째 단어를 second 출력

3. 단어 사이를 : 문자로 연결하여 다음과 같이 출력
   first:line:second:line:third:line

# I 튜플

- tuple은 불변 리스트라고 불립니다. 곧 한번 만들면 변경이 안됩니다.
- list는 값을 넣거나 빼거나 변경이 자유로웠지만, tuple은 안됩니다.
- list처럼 값이 순서대로 저장되어 있고, 인덱스 값으로 데이터에 접근이 가능합니다.
- 형식
  - tuple 이름 = (원소_1, 원소_2, ….. , 원소 N)
  - tuple은 소괄호를 사용하여 생성됩니다.

- 왜 사용할까요?
  - 데이터 바뀔 일이 없거나 바뀌면 안되는 경우 사용합니다.
  - 데이터가 의도치 않게 바뀔 가능성이 있다면 tuple로 사전에 예방합니다.

# I 튜플

- 아래는 tuple 생성 예제입니다.
- tuple은 데이터 변경을 허용하지 않습니다.

```
days_of_the_week = ('일요일', '월요일', '화요일', '수요일', '목요일', '금요일', '토요일')
```

```
print (type(days_of_the_week))
```

```
<class 'tuple'>
```

```
days_of_the_week[0] = '이번주 일요일'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-147-83ef49a797ec> in <module>
----> 1 days_of_the_week[0] = '이번주 일요일'

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

# I 튜플

- list와 tuple의 차이점에 대해서 알아보겠습니다.
- 공통점
  - 임의의 객체를 저장한다
  - 시퀀스 자료형이다
- 차이점
  - 변경 불가능
  - 메소드를 갖지 않는다
  - 포맷 문자열 지원, 함수 호출시 가변 인수 지원 기능 등이 추가로 있습니다.
- 일반적인 용도 차이
  - 데이터가 변경될 경우 – 리스트
  - 변경되지 않을 데이터를 표현할 경우 – 튜플
- 상호 변환 가능
  - list(), tuple()

# I 튜플

- 패킹과 언패킹
  - 한 데이터에서 여러 개의 데이터를 넣는 것을 패킹(packing)이라고 합니다.

```
t = 1, 2, "Hello"
```

```
t
```

```
(1, 2, 'Hello')
```

  - 패킹과 반대로 한 데이터에서 데이터를 각각 꺼내 오는 것을 언패킹(unpacking)이라고 합니다.
  - 언패킹시에 전달 받아야 할 변수의 개수가 맞지 않는다면 오류가 발생합니다.

```
x, y = t
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-173-757d34c4b6b9> in <module>
----> 1 x, y = t

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

  - 리스트도 언패킹은 지원합니다.

```
test = [1,2,3]
```

```
a, b, c = test
```

```
x , y, z  = t
```

```
x
```

```
1
```

```
y
```

```
2
```

```
z
```

```
'Hello'
```

# I 튜플

- 추가적으로 tuple에 대한 설명을 하겠습니다.
  - 언패킹시에 좌변의 변수중 하나에 *를 붙이면 언패킹되고 남은 데이터들이 리스트 형식으로 저장됩니다.
  - 예제를 통해서 살펴보겠습니다.

가장 왼쪽 변수 a1에 *가 붙여 있기 때문에 a2, a3, a4에 데이터를 할당하고 남은 데이터를 a1에 할당합니다.
a2 : 6, a3 : ‘test1’, a4: ‘test2’

```
t2 = 1,2,3,4,5,6, 'test1', 'test2'
```

```
*a1, a2, a3, a4 = t2
```

```
a1
```

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

a1에 데이터 하나를 할당하고 두번째 변수 a2에 *가 붙여 있기 때문에 a3, a4를 할당하고 남은 데이터 전체를 할당합니다.

```
a1, *a2, a3, a4 = t2
```

```
a2
```

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

a1에 데이터 하나 , a2에도 데이터 하나, a3에도 데이터 하나 마지막 a4에 *가 붙여 있기 때문에 남은 데이터를 할당합니다.

```
a1, a2, a3, *a4 = t2
```

```
a4
```

```
[4, 5, 6, 'test1', 'test2']
```

# I 튜플



- for 문 표현식에서 for 변수에 list, tuple의 데이터를 언패킹되어 전달할 수 있습니다.
- 아래 예제를 통해서 확인해보겠습니다.

```
stock3 = [['삼성전자', '45300'], ['엘지전자', '73900']]
```

```
for name, money in stock3:
    print (name, money)
```

```
삼성전자 45300
엘지전자 73900
```

- 중첩된 list 안의 list의 값이 name, money에 언패킹되어 전달됩니다.
- tuple로 변경해도 동일한 결과가 나옵니다.

# I 튜플

- 튜플의 존재의 이유
  - 어떤 연산을 더 효율적으로 만들어 주고, 또 어떤 연산을 안전하게 만들어 주기 때문입니다.
  - 한번 생성되고 나면 변경 할 수 없기 때문에 튜플은 항목을 추가/삭제 할 수 있는 자료형이 아니라 여러 부분으로 이루어진 단일 객체라고 이해하는 것이 편할 것입니다.
  - 개발자들은 List가 더 유연성이 있기 때문에 튜플 을 완전히 무시하고 리스트만 사용하려는 경향이 있고, 리스트가 튜플에 비해 컴퓨터 메모리 사용이 많다는 사실만 참고.

```
tuple_name = ('삼성전자', '엘지전자', '삼성SDS')
list_name = ['삼성전자', '엘지전자', '삼성SDS']
print(tuple_name.__sizeof__())
print(list_name.__sizeof__())
```

```
48
64
```

# I 사전(dictionary)



- Dict은 key-value 쌍으로 데이터를 보관하는 유형입니다.
  - Key – Value을 원소라고 부릅니다.
  - Dict은 중괄호 안에 쉼표로 구분하여 원소를 입력하여 만듭니다.
- Dict_이름 = { Key_1 : value_1 , Key_2 : value_2 }
- 빈 dict는 'dict_이름 = {}' 중괄호를 사용하면 만들 수 있습니다.
- 내부적으로 해시 기법을 사용하기 때문에 검색 속도가 빠릅니다.
- Key 값은 중복을 허용하지 않습니다.

```
first = {'Python' : '서찬웅', '패스트':'캠퍼스'}
```

key value key value

```
first
```

```
{'Python': '서찬웅', '패스트': '캠퍼스'}
```

# I 사전(dictionary)



- Dict에 새로운 원소를 추가하는 방법을 알아 보겠습니다.

```
stock_dict = {'삼성전자' : '45300', '엘지전자' : '73900'}
```

```
stock_dict
```

```
{'삼성전자': '45300', '엘지전자': '73900'}
```

```
stock_dict['삼성sds'] = '22400'
```

```
stock_dict
```

```
{'삼성전자': '45300', '엘지전자': '73900', '삼성sds': '22400'}
```

  - 기존에 생성된 dict에 [key] = value 형식으로 입력하면 새로운 원소가 추가됩니다.
- 원소를 삭제하고 싶다면 del dict[key] 실행하면 됩니다.

```
del stock_dict['삼성전자']
```

```
stock_dict
```

```
{'엘지전자': '73900', '삼성sds': '22400'}
```

# I 사전(dictionary)



- Dict에 원소가 존재하는지 확인하는 방법에 대해서 알아보겠습니다.

```
'엘지전자' in stock_dict
```

```
True
```

```
'삼성전자' in stock_dict
```

바로 전에 del를 이용해서 삼성전자를 삭제했기 때문에 False

```
False
```

- 시퀀스형 자료형에서 사용했던 in를 사용하면 dict안에 해당 key 값이 존재하는지 확인할 수 있습니다.

# I 사전(dictionary)



- 이중이상으로 중첩된 dict에 대해서 알아보겠습니다.

```
stock_dict2 = {'삼성전자' : {'고가' : 46000, '저가': 45250, '시가' : 45750}, '엘지전자' : {'고가': 74900, '저가': 73600, '시가':74600}}
```

```
stock_dict2
```

```
{'삼성전자': {'고가': 46000, '저가': 45250, '시가': 45750},
 '엘지전자': {'고가': 74900, '저가': 73600, '시가': 74600}}
```

  - Dict의 값은 데이터 유형에 상관없이 사용할 수 있습니다.
  - 예제처럼 dict안에 dict을 넣을수도 있습니다.
- 이중, 혹은 그 이상으로 된 dict에서 value 값을 출력해 보겠습니다.
  - Stock_dict2의 첫번째 Key '엘지전자'로 접근하고 두 번째 key인 '시가'로 접근하여 해당 value를 출력

```
stock_dict2['엘지전자']['시가']
```

```
74600
```

# I 사전(dictionary)



- Dict의 메소드 keys(), values(), items()를 사용하여 데이터를 출력하는 방법에 대해서 알아보겠습니다.
  - keys() : 키에 대한 View를 반환
  - values() : 값에 대한 View를 반환
  - items() : (key, values)의 View를 반환
    - view는 dict 항목들을 동적으로 볼 수 있는 객체를 의미합니다.

```
stock_dict2.keys()
```

```
dict_keys(['삼성전자', '엘지전자'])
```

```
stock_dict2.values()
```

```
dict_values([{'고가': 46000, '저가': 45250, '시가': 45750}, {'고가': 74900, '저가': 73600, '시가': 74600}])
```

```
stock_dict2.items()
```

```
dict_items([('삼성전자', {'고가': 46000, '저가': 45250, '시가': 45750}), ('엘지전자', {'고가': 74900, '저가': 73600, '시가': 74600})])
```

- items() 메소드를 사용하여 데이터를 출력해보겠습니다.

```
for key, value in stock_dict2.items():
    print ("{} 주식현황 ".format(key))
    for key2, value2 in value.items():
        print ("{} = {}".format(key2, value2))
```

# I 사전(dictionary)



- dict의 정렬을 해보겠습니다.
  - sorted() 함수를 사용하면 쉽게 정렬을 할 수가 있습니다.

```
dict_sorted = {'f' : 10, 'b' : 5, 'a' : 15, 'd' : 1}
```

```
sorted(dict_sorted, key=lambda x : dict_sorted[x])
```

value 기준 정렬

```
['d', 'b', 'f', 'a']
```

```
sorted(dict_sorted, key = lambda x : x)
```

key 기준 정렬

```
['a', 'b', 'd', 'f']
```

  - 중첩 dict일 때 value 값으로 정렬하는 예제

```
stock_dict3 = {'삼성전자' : {'고가' : 46000, '저가':  45250, '시가' : 45750},
               '엘지화학' : {'고가' : 371000,'저가' : 366500,'시가' : 370000},
              'SK텔레콤' : {'고가' : 245000, '저가' : 243000, '시가': 244000},
               '엘지전자' : {'고가': 74900, '저가': 73600, '시가': 74600}}
```

```
sorted(stock_dict3, key = lambda y : (stock_dict3[y]['시가']))
```

```
['삼성전자', '엘지전자', 'SK텔레콤', '엘지화학']
```

lambda는 함수편에서 공부할 내용입니다.
함수편에서 자세한 설명이 나옵니다.

# I 사전(dictionary)



- dict의 메소드 리스트
  - clear() – 항목 모두 삭제
  - copy() – 사전 복사
  - get(key[, x]) – 키가 없으면 x를 취한다
  - setdefault(key[, x]) – 키가 없으면 x를 키에 설정한다
  - update(D) – D사전의 내용으로 갱신
  - popitem() – (키, 값) 을 리턴하고 항목 제거
  - pop(key) - key 항목의 값을 반환하고 사전에서 제거
- 사전 내장(dictionary comprehension)
  - 사전 내장은 중괄호{}를 사용하여 키:값 형식으로 항목을 표현
  - 사전 내장은 리스트 내장과 유사한 방식으로 동작하지만 사전을 만들어 낸다.

```
{w:k for k,w in [(1, 'one'),(2, 'two'),(3, 'three')]}
```

```
{'one': 1, 'three': 3, 'two': 2}
```

# I 참고



- 시퀀스(Sequence) 자료형 : 문자열, List, Tuple

- 자료형에 포함된 각 객체는 순서를 가지고 있으며, Index를 사용하여 접근 가능.

- Q : Index는 왜 1이 아닌 0부터 시작하는가?

- A : 컴퓨터는 이진수, 즉 비트(이진수)를 이용해 모든 것을 저장한다.
- 옛날에는 저장 매체의 가격이 매우 비싸서 하나의 비트라도 낭비하지 않기
- 위해서 메모리 위치와 List Index와 같은 것들도 0부터 시작하는 것.

- 가변과 불변 Sequence

| 가변 나열형 | 불변 나열형 |
|---|---|
| list | str, tuple, range |

# I set(집합)

- set은 여러 값을 순서 없이 그리고 중복 없이 모아 놓은 자료형입니다.
- 순서가 없기 때문에 인덱스를 사용할수 없습니다.
- 반복 가능한(Iterable) 객체로부터 집합을 만들 수 있지만 모든 데이터가 집합의 원소로 사용 할수 있는 것은 아니다. Hashable 이면서 변경 불가능한 자료형만이 집합의 원소로 사용할 수 있습니다.
  - 정수, 실수, 복소수, 부울, 문자열, 튜플만이 집합에 저장할 수 있습니다.
  - list, set를 저장할려고 하면 TypeError가 발생합니다.

```
a = set()
```

```
b = {1, 2, 3}
```

```
a
```

```
set()
```

```
b
```

```
{1, 2, 3}
```

```
c = {}
```

```
type(c)
```

```
dict
```

빈 집합을 생성할 때 {}를 사용할 수 없는 이유와 빈 집합일 경우 출력이 중괄호 {} 대신 set() 함수인 이유는 중괄호{} 가 빈 사전으로 인식되기 때문이다.

```
a = {1,1,1,1,5,6,2,4,4,2,4,4,3,2,2,2,2,2,3,3,3,3,3}
a
```

```
{1, 2, 3, 4, 5, 6}
```

```
a[1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-89-8bc71255a22e> in <module>
----> 1 a[1]

TypeError: 'set' object is not subscriptable
```

# I set의 메소드

| 연산 | 동등한 표현 | 내용 |
|---|---|---|
| **s.update(t)** | **s \|= t** | s와 t의 합집합을 s에 저장 |
| **s.intersection_update(t)** | **s &= t** | s와 t의 교집합을 s에 저장 |
| **s.difference_update(t)** | **s -= t** | s와 t의 차집합을 s에 저장 |
| **s.symmetric_difference_update(t)** | **s ^= t** | s와 t의 배타집합을 s에 저장 |
| **s.add(x)** | | 원소 x를 s에 추가 |
| **s.remove(x)** | | 원소 x를 s에서 제거; 없으면 KeyError 예외 발생 |
| **s.discard(x)** | | 원소 x가 있다면 s에서 제거 |
| **s.pop()** | | s에서 임의의 원소를 하나 리턴하고 집합에서는 제거; 빈 집합이면 KeyError 예외 발생 |
| **s.clear()** | | 집합 s의 모든 원소 삭제 |

# I set의 연산



- set은 수학의 집합하고 동일한 연산을 진행할 수 있습니다.
- union(합집합), intersection(교집합), difference(차집합), symmetric_difference(대칭 차집합)이 있습니다.

```
A = {1, 2, 3, 4, 5, 6}
B = {4, 5, 6, 7, 8, 9}
C = {4, 10}
```

```
A.union(B)
```

합집합 A | B 와 동일

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
A.intersection(B)
```

교집합 A & B

```
{4, 5, 6}
```

```
A.intersection(B, C)
```

인수가 2개 이상도 가능

```
{4}
```

```
A.difference(B)
```

차집합 A - B

```
{1, 2, 3}
```

```
A.symmetric_difference(B)
```

대칭 차집합 A ^ B

```
{1, 2, 3, 7, 8, 9}
```

✓ 결과가 반영되기를 원하면 update(), intersection_update(), difference_update(), symmetric_difference_update() 메서드를 사용하면 된다.

# I set의 연산



- set은 중복을 허용하지 않기 때문에 이 성질을 이용하면 list 자료형에서 중복된 값을 제거할 수 있습니다.

```
a = [1,1,1,1,1,2,2,2,2,2,3,3,3,3,4,4,4,4,4,4,5,5,5,6,3,4,5,5,3,3,4,5,7,7,7]
```

```
set(a)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

```
list(set(a))
```

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

# I 정리

- 리스트(list) 생성
- 튜플(tuple) 생성
- 사전(dictionary) 생성
- 리스트 내장(list comprehension)
- 사전 내장(dict comprehension)
- 패킹, 언패킹
- 정렬
- 집합(set)



FAST CAMPUS
ONLINE
서찬웅 강사.

# 감사합니다

FAST CAMPUS
ONLINE
서찬웅 강사.