

# 데이터분석 프로그래밍

## 리스트와 튜플

임현기

# 리스트

- 여러 개의 데이터를 하나로 묶어서 저장/처리하는 것이 필요함
  - 파이썬에서는 리스트와 딕셔너리를 제공

```
>>> height = 178.9    # float 타입의 데이터를 저장한다.
```

사람이 100명이면  
변수도 100개,  
사람이 1,000명이면  
???????

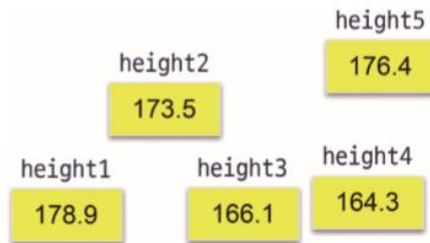
```
>>> height1 = 178.9    # float 타입의 데이터를 저장한다.  
>>> height2 = 173.5    # float 타입의 데이터를 저장한다.  
>>> height3 = 166.1    # float 타입의 데이터를 저장한다.  
>>> height4 = 164.3    # float 타입의 데이터를 저장한다.  
>>> height5 = 176.4    # float 타입의 데이터를 저장한다.
```



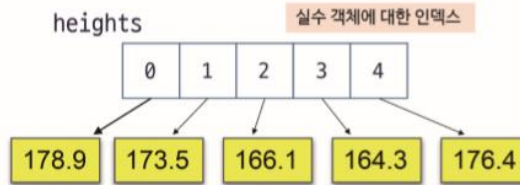
# 리스트

```
>>> heights = [178.9, 173.5, 166.1, 164.3, 176.4]
>>> heights
[178.9, 173.5, 166.1, 164.3, 176.4]
```

리스트를 사용하면  
하나의 변수에 여러  
개의 값을 담을 수 있다.



개별 변수로 표현된 데이터



리스트로 표현된 데이터

리스트 내의 개별  
데이터를 항목 또는  
요소라고 한다.

# 리스트

---

- 리스트 생성 방법
  - 다른 변수처럼 생성하고, [] 안에 항목들을 적어 줌

```
>>> bts = ['V', 'Jungkook', 'Jimin']
```

- 초기에 빈 리스트 생성 가능

```
>>> bts = []
```

# 리스트

- 공백 리스트에 항목 추가
  - append 메소드 사용

```
>>> bts = []  
>>> bts.append("V")  
>>> bts  
['V']
```

```
>>> bts.append("Jin")  
>>> bts.append("Suga")  
>>> bts  
['V', 'Jin', 'Suga']
```

리스트 내에 새 항목을  
추가한다.

– + 사용

```
>>> bts = ['V', 'Jin', 'Suga', 'Jungkook']  
>>> bts = bts + ['RM'] # 덧셈 연산자로 멤버 'RM'을 추가함  
>>> bts  
['V', 'Jin', 'Suga', 'Jungkook', 'RM']
```

리스트 내에 새 항목  
'RM'을 추가한다.

# 리스트

```
>>> list(range(5))      # 0에서 5사이의 정수열을 생성(5는 포함안됨)
[0, 1, 2, 3, 4]
>>> list(range(0, 5))  # list(range(5))와 동일한 결과
[0, 1, 2, 3, 4]
>>> list(range(0, 5, 1)) # list(range(0, 5))와 동일한 결과
[0, 1, 2, 3, 4]
>>> list(range(0, 5, 2)) # 생성하는 값을 2씩 증가시킴
[0, 2, 4]
>>> list(range(2, 5))   # 2에서 5-1까지의 연속된 수 2, 3, 4를 생성
[2, 3, 4]
```

range() 함수를  
사용해서 여러 항목을  
한꺼번에 생성할 수  
있다.

# 리스트 연산

---

- 리스트 덧셈 연산

```
>>> bts = ['V', 'J-Hope'] + ['RM', 'Jungkook', 'Jin']  
>>> bts  
['V', 'J-Hope', 'RM', 'Jungkook', 'Jin']
```

```
>>> mystery = [0, 1, 2] * 3    # [0, 1, 2]가 3회 반복되어 저장됨  
>>> mystery  
[0, 1, 2, 0, 1, 2, 0, 1, 2]
```

– 정수끼리의 합이 아님

# 리스트 연산

두 리스트를 합치는 연산

```
>>> numbers = [10, 20, 30] + [40, 50, 60]
>>> numbers
[10, 20, 30, 40, 50, 60]
```

```
>>> bts = ['V', 'J-Hope', 'RM', 'Jungkook', 'Jin', 'Jimin', 'Suga']
>>> 'V' in bts
True
>>> 'V' not in bts
False
```

'V'라는 멤버가 bts에  
있는가?



빈 리스트를 생성한 다음 사용자로부터 제일 좋아하는 3개의 과일을 입력받아서 리스트에 저장한다. 사용자로부터 과일을 입력받아서 리스트에 그 과일이 포함되어 있으면 '이 과일은 당신이 좋아하는 과일입니다.'를 출력하고, 그렇지 않으면 '이 과일은 당신이 좋아하는 과일이 아닙니다.'를 출력하는 프로그램을 작성한다.



#### 원하는 결과

좋아하는 과일 이름을 입력하시오: 사과  
좋아하는 과일 이름을 입력하시오: 키위  
좋아하는 과일 이름을 입력하시오: 바나나  
과일의 이름을 입력하세요: 바나나  
이 과일은 당신이 좋아하는 과일입니다.

---

```
fruits = []

name = input('좋아하는 과일의 이름을 입력하시오: ')
fruits.append(name)
name = input('좋아하는 과일의 이름을 입력하시오: ')
fruits.append(name)
name = input('좋아하는 과일의 이름을 입력하시오: ')
fruits.append(name)

name = input('과일의 이름을 입력하세요: ')
if name in fruits:
    print('이 과일은 당신이 좋아하는 과일입니다.')
else:
    print('이 과일은 당신이 좋아하는 과일이 아닙니다.')
```

# 리스트 함수

- 하나의 리스트 안에 여러 타입 존재 가능

```
>>> slist1 = ['Kim', 178.9, 'Park', 173.5, 'Lee', 176.1]
>>> slist1
['Kim', 178.9, 'Park', 173.5, 'Lee', 176.1]
>>> slist1[0], slist1[2], slist1[4]
('Kim', 'Park', 'Lee')
```

– 일반적으로 권장하지 않음

- 리스트를 묶는 리스트 가능

```
>>> slist2 = [ ['Kim', 178.9], ['Park', 173.5], ['Lee', 176.1] ]
>>> slist2
[ ['Kim', 178.9], ['Park', 173.5], ['Lee', 176.1] ]
```

학생들의 이름과 키를  
묶어서 리스트로 만들고  
이것들을 모아서 하나의  
리스트로 만들 수도 있다

```
>>> type(slist1)
<class 'list'>
>>> type(slist2)
<class 'list'>
```

# 리스트 함수

---

- 내장 함수
  - len(): 리스트의 길이
  - max(), min(): 최대값, 최소값
  - sum(): 합계
  - any(): 0이 아닌 원소가 하나라도 있으면 True

```
>>> n_list = [200, 700, 500, 300, 400]
>>> len(n_list)      # 리스트의 길이를 반환한다
5
>>> max(n_list)      # 리스트 항목중 최대값을 반환한다
700
>>> min(n_list)      # 리스트 항목중 최소값을 반환한다
200
>>> sum(n_list)      # 리스트 항목의 합을 반환한다
2100
```

# 리스트 함수

---

```
>>> sum(n_list) / len(n_list) # 전체의 합을 원소의 개수로 나누면 평균값을 얻을 수 있다
420.0
>>> list(range(1, 11)) # 1에서 10까지의 수를 생성하여 리스트에 넣는다
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> a_list = [0, ''] # 임의의 리스트를 생성
>>> any(n_list)      # n_list에 0이 아닌 원소가 하나라도 있는가
True
>>> any(a_list)      # a_list에 0이 아닌 원소가 하나라도 있는가
False
```

서울, 부산, 인천의 인구를 가지고 있는 리스트 `city_pop`을 생성해보자. 각 도시들의 인구는 `Seoul`, `Busan`, `Incheon` 등의 변수에 저장되어 있다고 가정한다. 우리는 변수를 사용해서 다음과 같이 리스트를 생성할 수 있다.

```
# 인구 통계(단위: 천명)
Seoul = 9765
Busan = 3441
Incheon = 2954
city_pop = [ Seoul, Busan, Incheon ] # 변수들로 리스트 생성
print(city_pop)                     # 리스트 데이터를 출력
```

세 도시에 대전(Daejeon)을 추가해 보도록 하자. 대전의 인구는 1,531천명이라고 가정하자. 그리고 이 네 도시 중에서 가장 인구가 많은 도시의 인구와 가장 인구가 적은 도시의 인구, 그리고 네 도시의 인구의 평균을 출력해 보도록 하자. 이를 위하여 `min()`, `max()` 함수를 사용하는 대신 `for` 반복문을 사용하는 방법으로 구현해 보아라.

#### 원하는 결과

```
[9765, 3441, 2954]
최대 인구: 9765
최소 인구: 1531
평균 인구: 4422.75
```

---

<문제에서 제시된 코드를 여기에 삽입한다>

```
Daejeon = 1531
city_pop.append(Daejeon)

max_pop = 0
min_pop = 1000000
pop_sum = 0
n = 0

for pop in city_pop:    # 순환문을 돌면서 최댓값, 최솟값을 구한다
    if pop > max_pop :
        max_pop = pop
    if pop < min_pop :
        min_pop = pop
    pop_sum += pop
    n += 1

print('최대 인구:', max_pop)
print('최소 인구:', min_pop)
print('평균 인구:', pop_sum / n)
```


# 리스트 인덱스

- 인덱스: 리스트 안에 특성 데이터를 꺼낼 때 사용되는 정수
  - 첫번째 데이터는 인덱스 0

```
>>> letters = ['A', 'B', 'C', 'D', 'E', 'F']
```

```
>>> letters[0] # 리스트의 첫 항목에 접근  
'A'
```

letters	'A'	'B'	'C'	'D'	'E'	'F'
인덱스	0	1	2	3	4	5





# 리스트 인덱스

```
>>> letters[1]  
'B'  
>>> letters[2]  
'C'
```

파이썬에서는 인덱스를  
음수로 줄 수도 있다!  
리스트의 맨 끝에 있는  
데이터를 얻고자 할 때 유용

```
>>> letters[-1] # 리스트의 마지막 항목에 접근  
'F'
```

letters	'A'	'B'	'C'	'D'	'E'	'F'
인덱스	0	1	2	3	4	5
음수 인덱스	-6	-5	-4	-3	-2	-1



# 리스트 인덱스

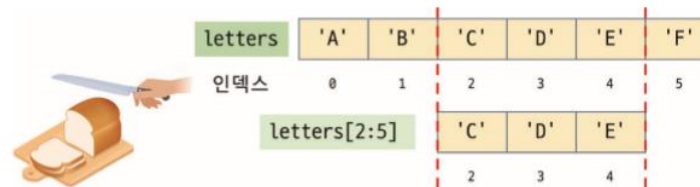
---

```
>>> bts = ['V', 'RM', 'Jungkook', 'J-Hop', 'Suga', 'Jin', 'Jimin']
>>> len(bts)           # 리스트의 원소 개수를 얻는다.
7
>>> bts[len(bts)-1]    # 마지막 원소
'Jimin'
>>> bts[-1]           # 음수 인덱스로 마지막 원소를 쉽게 접근할 수 있다.
'Jimin'
>>> min(bts)           # 리스트 원소 중 가장 작은 값을 찾는다. (문자열은 사전식 순서)
'J-Hop'
>>> max(bts)           # 리스트 원소 중 가장 큰 값을 찾는다. (문자열은 사전식 순서)
'V'
```

# 리스트 슬라이싱

- 슬라이싱: 여러 원소들을 선택할 때

```
>>> letters = ['A', 'B', 'C', 'D', 'E', 'F']  
>>> letters[2:5]  
['C', 'D', 'E']
```



```
>>> letters[:3]  
['A', 'B', 'C']
```

```
>>> letters[3:]  
['D', 'E', 'F']
```

# 리스트 슬라이싱

콜론만 있으면 리스트의 처음부터 끝까지

```
>>> letters[:]  
['A', 'B', 'C', 'D', 'E', 'F']
```

리스트를 처음부터 끝까지 읽어오되,  
스텝(step) 만큼 건너뛰기

```
>>> letters[::2]  
['A', 'C', 'E']  
>>> letters[::-1]      # 뒤에서부터 앞으로 읽어오며 원소를 생성한다  
['F', 'E', 'D', 'C', 'B', 'A']
```



### 도전문제 7.2

- 1) `numbers = [4, 5, 6, 7, 8, 9]`와 같은 리스트가 있다고 하자. 6을 추출하는 문장을 적어보자.
- 2) 위의 리스트에서 슬라이싱을 통해 `[5, 6]` 리스트를 추출하도록 하자.
- 3) 시작인덱스를 생략하여 `[4, 5, 6]` 리스트를 추출하도록 하자.
- 4) 시작인덱스를 1로하고 스텝값을 2로 하여 `[5, 7, 9]`를 추출하도록 하자.

---

서울, 부산, 인천의 인구를 가지고 있는 리스트 `population`이 있다고 하자.

# 다음과 같은 리스트가 생성되어 있다.

```
population = ["Seoul", 9765, "Busan", 3441, "Incheon", 2954]
```

- 1) 이 리스트에서 서울의 인구인 두 번째 요소를 출력해보자.
- 2) 이 리스트에서 마지막 요소인 인천의 인구를 출력해보자. 이때, 음수로 된 인덱스를 사용해본다.
- 3) 각 도시의 이름을 `step` 값을 이용하여 출력해 보자.
- 4) 각 도시의 인구를 `step` 값을 이용하여 출력한 후 이 값들의 합을 출력하도록 하자.

#### 원하는 결과

서울 인구: 9765

인천 인구: 2954

도시 리스트: ['Seoul', 'Busan', 'Incheon']

인구의 합: 16160

---

```
# 다음과 같은 리스트가 생성되어 있다.  
population = ["Seoul", 9765, "Busan", 3441, "Incheon", 2954]  
  
print('서울 인구:', population[1])      # 문제 1)  
print('인천 인구:', population[-1])     # 문제 2)  
  
cities = population[0::2]                # 문제 3)  
print('도시 리스트:', cities)  
pops = population[1::2]                  # 문제 4)  
print('인구의 합:', sum(pops))
```



### 도전문제 7.3

- 1) `s = "Python is strong"`과 같은 문자열이 있다고 하자. 이 문자열에서 가장 첫 알파벳 문자 'P'를 추출하여라.
- 2) 위의 문자열 `s`에서 슬라이싱을 통해 'strong' 문자열을 추출하도록 하자.
- 3) 시작 인덱스를 생략한 슬라이싱을 통해 'Python' 문자열을 추출하도록 하자.
- 4) 슬라이싱을 통해 'is' 문자열을 추출하여라.



# 리스트 수정

---

```
>>> slist = ['Kim', 178.9, 'Park', 173.5, 'Lee', 176.1]
```

```
>>> slist[2:4] = ['Paik', 180.0]  
>>> slist  
['Kim', 178.9, 'Paik', 180.0, 'Lee', 176.1]
```

# 리스트 수정

```
>>> slist[3] = 175.0
```

'Park'의 키를 175.0로 변경하고 싶으면  
인덱스를 사용하여 다음과 같이 수정

```
>>> slist  
['Kim', 178.9, 'Park', 175.0, 'Lee', 176.1]
```

```
>>> slist.insert(4, "Hong")  
>>> slist.insert(5, 168.1)  
>>> slist  
['Kim', 178.9, 'Paik', 180.0, 'Hong', 168.1, 'Lee', 176.1,]
```

함수를 사용하여 동적으로 항목을  
추가하려면 append()나 insert() 함수를  
사용할 수 있다

# 리스트 수정

---

```
x = ['Tick', 'Tock', 'Song']  
y = ['Ping', 'Pong']  
x.append(y)  
print('x:', x)
```

x: ['Tick', 'Tock', 'Song', ['Ping', 'Pong']]

```
x = ['Tick', 'Tock', 'Song']  
y = ['Ping', 'Pong']  
x.extend(y)  
print('x:', x)
```

x: ['Tick', 'Tock', 'Song', 'Ping', 'Pong']

# 리스트 수정

메소드	하는 일
<code>index( x )</code>	원소 <code>x</code> 를 이용하여 위치를 찾는 기능을 한다.
<code>append( x )</code>	원소 <code>x</code> 를 리스트의 끝에 추가한다.
<code>count( x )</code>	리스트 내에서 <code>x</code> 원소의 개수를 반환한다.
<code>extend([x1, x2])</code>	<code>[x1, x2]</code> 리스트를 기존 리스트에 삽입한다.
<code>insert(index, x)</code>	원하는 <code>index</code> 위치에 <code>x</code> 를 추가한다.
<code>remove( x )</code>	<code>x</code> 원소를 리스트에서 삭제한다.
<code>pop(index)</code>	<code>index</code> 위치의 원소를 삭제한 후 반환한다. 이때 <code>index</code> 는 생략될 수 있으며 이 경우 리스트의 마지막 원소를 삭제하고 이를 반환한다.
<code>sort()</code>	값을 오름차순 순서대로 정렬한다. 키워드 인자 <code>reverse=True</code> 이면 내림차순으로 정렬한다.
<code>reverse()</code>	리스트를 원래 원소들의 역순으로 만들어 준다.

# 리스트 삭제

```
>>> bts = [ "V", "J-Hope", "Suga", "Jungkook" ]  
>>> bts.remove("Jungkook")  
>>> bts  
['V', 'J-Hope', 'Suga']
```

remove() 메소드는 다음과 같이 리스트에서  
지정된 항목을 삭제

```
if "Suga" in bts:  
    bts.remove("Suga")
```

조건이 참인 경우에만 bts 리스트에서 이 항목을  
삭제하면 안전한 프로그램이 된다

# 리스트 삭제

---

```
>>> bts = ["V", "J-Hope", "Suga", "Jungkook"]
>>> last_member = bts.pop()      # 마지막 항목 'Jungkook'을 삭제하고 반환한다
>>> last_member                  # 삭제된 항목을 출력하자
'Jungkook'
>>> bts                          # bts 리스트에 마지막 항목이 삭제되었는가 확인하자
['V', 'J-Hope', 'Suga']
```

# 리스트 삭제

---

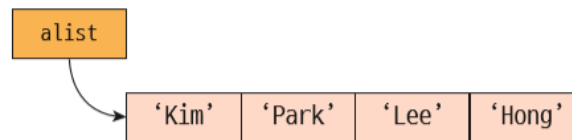
- del 명령어는 파이썬의 키워드로 해당항목을 메모리에서 삭제

```
>>> bts = [ "V", "J-Hope", "Suga", "Jungkook"]  
>>> del bts[0]                # 리스트의 첫 항목을 삭제하는 명령어  
>>> bts  
['J-Hope', 'Suga', 'Jungkook']
```

# 리스트의 객체 생성

- 리스트 변수의 이름은 리스트의 주소

```
>>> alist = ['Kim', 'Park', 'Lee', 'Hong']
```

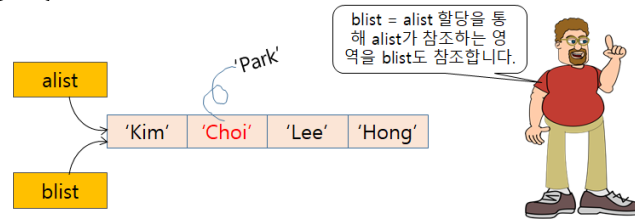


```
>>> blist = alist
>>> blist[1] = 'Choi' # blist의 두 번째 항목값을 변경함
>>> alist
['Kim', 'Choi', 'Lee', 'Hong']
```



# 리스트의 객체 생성

- 참조: 변수의 이름이 존재하는 메모리 상의 객체를 가리키는 것



- id() 함수를 통해 객체의 식별번호를 확인할 수 있음



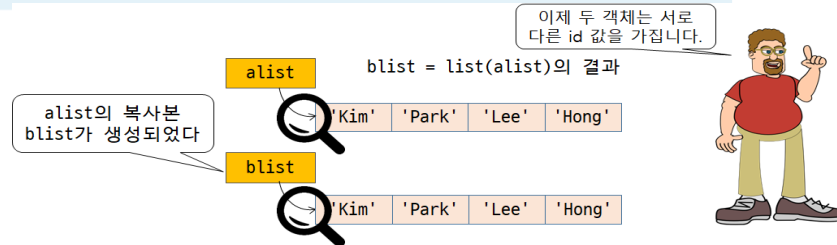
```
>>> id(alist) # 아이디 값은 이 책과 다른 값이 나올 수 있다
140050268419592
>>> id(blist) # alist의 아이디 값과 동일한 값이 나온다
140050268419592
```

두 객체의 아이디가 같다!

# 리스트의 객체 생성

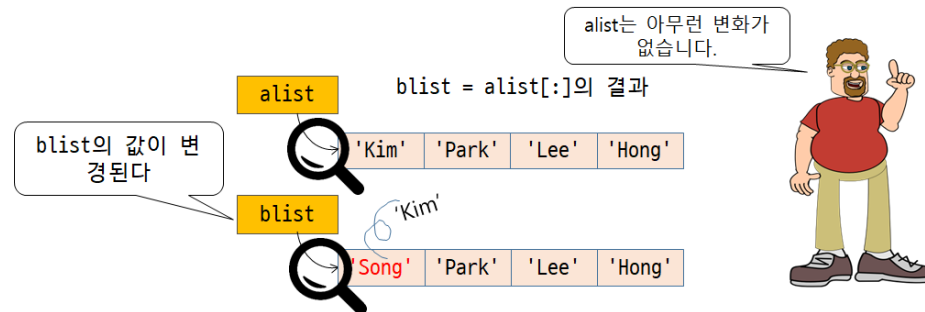
- 동일한 내용의 리스트를 생성하려면 list() 함수

```
>>> alist = ['Kim', 'Park', 'Lee', 'Hong']
>>> blist = list(alist)
>>> id(alist)      # 책의 값과 여러분이 실행한 값은 다를 수 있다
140050268419592
>>> id(blist)      # blist의 아이디는 alist의 아이디와 다를 것이다.
140050268535624
```



# 리스트의 객체 생성

```
>>> blist = alist[:]          # 이렇게 하여도 된다.
>>> id(alist)
140050268419592
>>> id(blist)                 # blist의 아이디는 alist의 아이디와 다를 것이다.
140050129233608
>>> blist[0] = 'Song'
>>> alist                     # alist의 복사본 blist를 고쳤으므로 아무런 변화가 없다
['Kim', 'Park', 'Lee', 'Hong']
>>> blist                     # alist의 복사본의 첫 번째 항목이 변경됨
['Song', 'Park', 'Lee', 'Hong']
```



# 리스트 탐색

- 특정 항목을 찾을 때 index() 함수 사용

```
>>> bts = [ "V", "J-Hope", "Suga", "Jungkook" ]  
>>> bts.index("Suga")  
2
```

- 탐색 전 in 연산자로 확인하는 것이 필요

```
if "Suga" in bts:  
    print(bts.index("Suga"))
```

# 리스트 정렬

---

```
>>> numbers = [ 9, 6, 7, 1, 8, 4, 5, 3, 2 ]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> bts = [ "V", "J-Hope", "Suga", "Jungkook" ]
>>> bts.sort() # bts 리스트를 알파벳 순으로 정렬
>>> print(bts)
['J-Hope', 'Jungkook', 'Suga', 'V']
```

– 역순 정렬을 하려면 reverse=True를 붙임

# 리스트 정렬

---

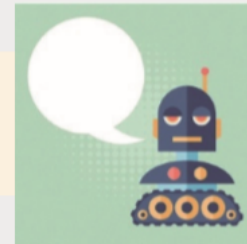
```
>>> numbers.sort(reverse=True)
>>> print(numbers)
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
>>> numbers = [ 9, 6, 7, 1, 8, 4, 5, 3, 2 ]
>>> new_list = sorted(numbers)
>>> print(new_list)      # numbers 리스트를 항목의 크기 순으로 정렬
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(numbers)      # sorted() 함수는 정렬한 결과를 반환하므로 원래 리스트에는 변화가 없음
[9, 6, 7, 1, 8, 4, 5, 3, 2]
```

```
>>> numbers = [ 9, 6, 7, 1, 8, 4, 5, 3, 2 ]
>>> new_list = sorted(numbers, reverse=True)
>>> print(new_list)
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

리스트에 여러 개의 명언을 저장한 후에 그 중에서 하나를 랜덤하게 골라서 오늘의 명언으로 제공한다. 리스트에 저장한 항목 중에서 하나를 랜덤하게 고르려면 다음과 같은 `random` 모듈의 함수 `choice()`를 사용하면 간편하다.

```
import random
quotes = [...]
random.choice(quotes)
```



#### 원하는 결과

```
#####
#           오늘의 명언           #
#####
```

고생 없이 얻을 수 있는 진실로 귀중한 것은 하나도 없다.

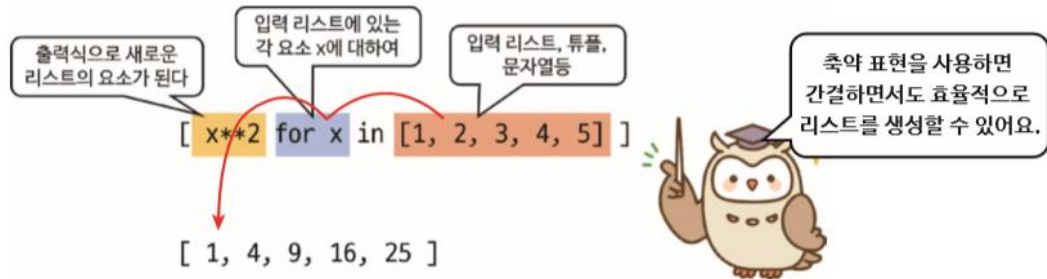
```
import random

quotes = []
quotes.append("꿈을 지녀라. 그러면 어려운 현실을 이길 수 있다.")
quotes.append("분노는 바보들의 가슴속에서만 살아간다..")
quotes.append("고생 없이 얻을 수 있는 진실로 귀중한 것은 하나도 없다.")
quotes.append("사람은 사랑할 때 누구나 시인이 된다.")
quotes.append("시작이 반이다.")

print("#####")
print("# 오늘의 명언 #")
print("#####")
print("")
dailyQuote = random.choice(quotes)
print(dailyQuote)
```



# 리스트 함축



```
s = []  
for x in [1, 2, 3, 4, 5]:  
    s.append(x**2)
```

리스트 함축을 사용하지 않는다면  
다음과 같이 번거로운 반복문을  
사용해야 할 것

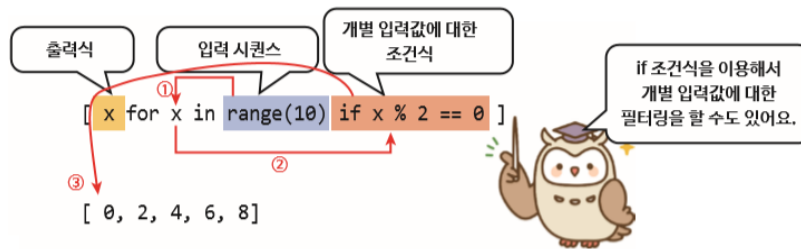
# 리스트 함축

---

```
>>> [x for x in range(10)]      # 0에서 9까지 숫자를 포함하는 리스트
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x * x for x in range(10)] # 0에서 9까지 숫자의 제곱 값
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> st = 'Hello World'
>>> [x.upper() for x in st]     # 문자열 각각에 대해 upper() 메소드 적용
['H', 'E', 'L', 'L', 'O', ' ', 'W', 'O', 'R', 'L', 'D']
```

```
>>> a = ['welcome', 'to', 'the', 'python', 'world']
>>> first_a = [ s[0].upper() for s in a] # 첫 알파벳에 대한 대문자 변환
>>> print(first_a)
['W', 'T', 'T', 'P', 'W']
```

# 리스트 함축



```
>>> [x for x in range(10) if x % 2 == 0]
[0, 2, 4, 6, 8]
>>> [x**2 for x in range(10) if x % 2 == 0] # 출력식에 제곱을 할 수 있다
[0, 4, 16, 64, 64]
```

# 리스트 함축

```
s = ["Hello", "12345", "World", "67890"]  
numbers = [x for x in s if x.isdigit()]  
print(numbers)
```

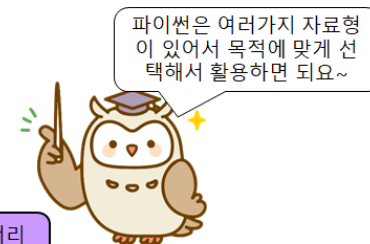
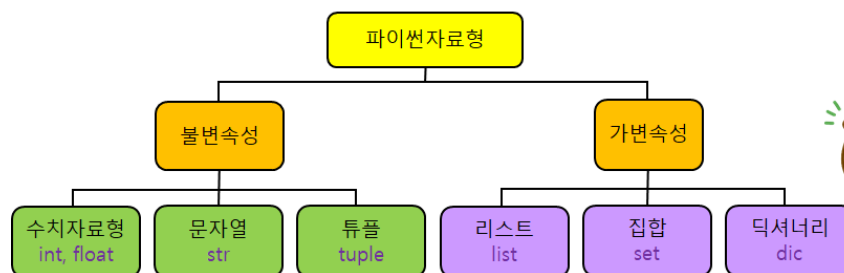
```
['12345', '67890']
```

```
>>> [int(x) for x in input('정수를 여러개 입력하세요 : ').split()]  
정수를 여러개 입력하세요 : 100 200 300  
[100, 200, 300]  
>>> [int(x) for x in input('정수를 여러개 입력하세요 : ').split() if x.isdigit()]  
정수를 여러개 입력하세요 : 100 이백 300 400  
[100, 300, 400]
```

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

# 튜플

- 파이썬의 데이터형
  - 불변속성
  - 가변속성
- 튜플은 불변속성
  - 리스트와 유사
  - 내용이 정해지면 변경 불가
  - 접근 속도가 빠름



# 튜플

---

색상을 저장하는 튜플

```
>>> colors = ("red", "green", "blue")  
>>> colors  
( 'red', 'green', 'blue' )
```

다수의 정수를 저장하는 튜플

```
>>> numbers = (1, 2, 3, 4, 5 )  
>>> numbers  
(1, 2, 3, 4, 5)
```

# 튜플

- 인덱싱, 슬라이싱 가능
- +, \* 연산 가능

```
>>> t1 = (1, 2, 3, 4, 5)
>>> t1[0]          # 튜플의 인덱싱-리스트 인덱싱과 동일한 방식
1
>>> t1[1:4]        # 튜플의 슬라이싱 결과로 튜플을 반환함
(2, 3, 4)
>>> t2 = t1 + t1    # 튜플의 결합 연산
>>> t2
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

서울, 부산, 인천, 광주, 대전의 인구를 가지고 있는 리스트 `city_info`가 있다고 하자. LAB 7-3은 서로 다른 자료형인 도시명과 인구수를 하나의 리스트에 넣었다. 그러나 리스트에는 동일한 자료형을 항목으로 사용하는 것이 바람직하므로 다음과 같이 튜플의 리스트를 만들 수 있다.

# 다음과 같은 리스트가 생성되어 있다.

```
city_info = [('서울', 9765), ('부산', 3441), ('인천', 2954),  
             ('광주', 1501), ('대전', 1531)]
```

최대 인구를 가진 도시를 찾아 보고, 최소 인구를 가진 도시도 찾아 보라. 그리고 아래와 같이 전체 도시들의 평균 인구를 계산하는 코드도 작성해 보라.

#### 원하는 결과

최대인구: 서울, 인구: 9765 천명  
최소인구: 광주, 인구: 1501 천명  
리스트 도시 평균 인구: 3838.4 천명



---

<문제에서 제시된 코드를 여기에 삽입한다>

```
max_pop = 0
min_pop = 9999999999999999
total_pop = 0

max_city = None
min_city = None

for city in city_info:
    total_pop += city[1]
    if city[1] > max_pop :
        max_pop = city[1]
        max_city = city
    if city[1] < min_pop :
        min_pop = city[1]
        min_city = city

print('최대인구: {0}, 인구: {1} 천명'.format(max_city[0], max_city[1]))
print('최소인구: {0}, 인구: {1} 천명'.format(min_city[0], min_city[1]))
print('리스트 도시 평균 인구: {0} 천명'.format(total_pop / len(city_info)) )
```

# 클래스와 객체

- 파이썬은 객체지향 프로그래밍 언어
  - 자료형, 함수, 모듈 모두 객체
  - 객체를 클래스로 정의
  - 인스턴스 객체: 클래스의 틀을 가지는 객체

```
>>> animals = ['lion', 'tiger', 'cat', 'dog']
>>> animals.sort()      # animals 리스트의 내부 문자열을 알파벳 순으로 정렬한다
>>> animals
['cat', 'dog', 'lion', 'tiger']
>>> animals.append('rabbit') # animals 리스트에 새 원소를 추가한다
>>> animals
['cat', 'dog', 'lion', 'tiger', 'rabbit']
>>> animals.reverse()    # animals 리스트를 원래 원소의 역순으로 재배열한다
>>> animals
['rabbit', 'tiger', 'lion', 'dog', 'cat']
```

문자열 항목들을 속성으로  
집어 넣어 animals라는 리스트  
클래스의 인스턴스 객체를  
만들었다

# 클래스와 객체

- 메소드: 특정 클래스에 속한 객체들이 사용할 수 있는 함수
  - 클래스 정의
  - 클래스의 인스턴스 객체 생성
  - 각 객체들이 메소드를 이용

## 클래스 정의

이름: 리스트	속성	복수의 데이터 나열, 데이터 개수, ...
	기능 - 메소드	순서대로 정렬하기, 데이터 추가, 순서 뒤집기 등의 기능 <code>sort()</code> , <code>append()</code> , <code>reverse()</code>

## 프로그램에서 변수에 할당되어 동작하는 객체

리스트 클래스에 속하는 인스턴스 객체

메소드를 사용했을 때 속성의 변화

animals

'lion' 'tiger' 'cat' 'dog'

fruits

'pear' 'apple' 'banana'

odd\_numbers

1 3 5 7 9

animals.reverse()

animals 'dog' 'cat' 'tiger' 'lion'

fruits.sort()

fruits 'apple' 'banana' 'pear'

odd\_numbers.append(11)

odd\_numbers 1 3 5 7 9 11