UNIT11

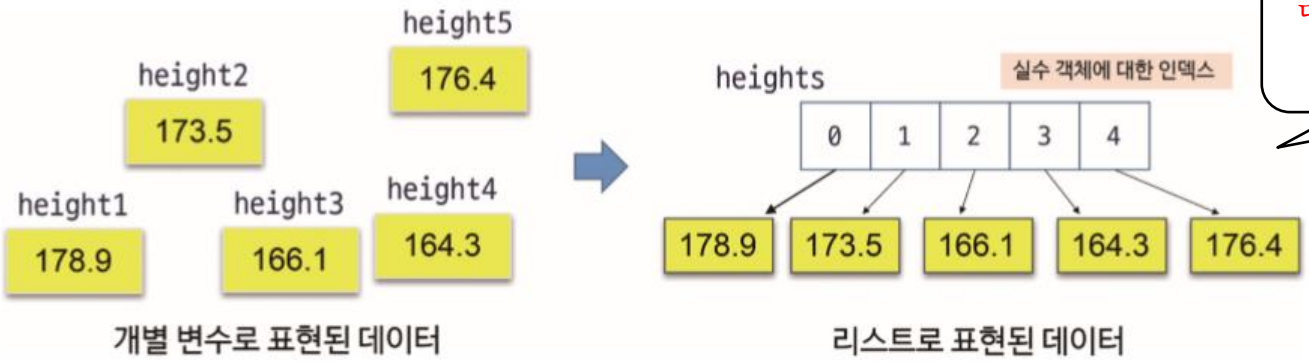
인공지능 파이썬 코딩의 주춧돌, 배열

• 리스트는 무엇이고 왜 필요한가

- 이럴 때는 "리스트(list)"를 만들고 여기에 데이터를 저장하면 된다. 파이썬에서 리스트는 대괄호를 이용해서 만들 수 있다. 예를 들어, 학생 5명의 키를 리스트에 저장하려면 다음과 같이 한다.

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

리스트를 사용하면하나의 변수에 여러 개의 값을 담을 수 있다.



리스트 내의 개별 데이터를 항목 또는 요소라고 한다.

여러 개의 항목이 들어가는 리스트를 만들자

- 파이썬에서 리스트는 다른 변수처럼 생성할 수 있다. 즉 항목들을 [ ] 안에 적어주고 변수에 저장하면 리스트 변수가 생성된다. 다음과 같은 아이돌 그룹 BTS의 멤버 3명으로 리스트를 간단하게 만들어 보자.

- 만일 초기에 멤버가 없을 경우 다음과 같은 방식도 가능하다. 이때는 빈 리스트가 만들어 진다.

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

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

• 여러 개의 항목이 들어가는 리스트를 만들자

- 항목이 없는 공백 리스트는 도대체 어디에 쓸모가 있을까? 공백 리스트에는 코드로 항목을 추가할 수 있다. 많은 경우에 우리는 리스트에 몇 개의 항목이 들어갈 것인지를 예측할 수 없다. 이런 경우에 공백 리스트가 유용하다. 리스트에 항목을 새롭게 추가하려면 append(항목) 메소드를 사용한다.

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

- 리스트에 요소를 추가하는 제일 간단한 방법은 + 연산자를 이용하는 것이다.

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

리스트 내에 새 항목을 추가한다.

```
>>> 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.

```
>>> 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, 4을 생성
[2, 3, 4]
```

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

리스트 연산을 해보자

- 파이썬에서는 리스트에 대해서도 다양한 연산이 가능하다. 예를 들어서 우리는 덧셈 연산으로 다음과 같이 두 개의 리스트를 합칠 수 있다.

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

- 합쳐진다
  사용하여야 한다.

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

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?

입력을 받아 맛있는 과일의 리스트를 만들어 보자

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.
```

입력을 받아 맛있는 과일의 리스트를 만들어 보자

```
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.')
```

리스트에 사용 가능한 함수를 알아보자

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

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

리스트에 사용 가능한 함수를 알아보자

- 리스트에 다음과 같은 수치값이 있을 경우 **len(), max(), min(), sum(), any()** 등의 내장함수를 사용하여 편리하게 활용할 수 있다. **len()**은 리스트의 길이를 반환하며, **max()**는 리스트 항목 중 최댓값, **min()**은 리스트 항목중 최솟값을 반환한다.
- **list(range())**를 통해서 **range()** 함수가 생성한 값을 리스트에 넣을 수 있다. **any()** 함수는 리스트에 0이 아닌 원소가 하나라도 있을 경우 **True**를 반환한다. 따라서 0과 "(공백문자열만으로 이루어진 리스트)는 **any()** 함수가 **False**를 반환한다.

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

```
>>> 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
```

리스트에 도시의 인구를 저장해보자

서울, 부산, 인천의 인구를 가지고 있는 리스트 city_pop을 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 도시중에서 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
```

리스트에 도시의 인구를 저장해보자

<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)
```

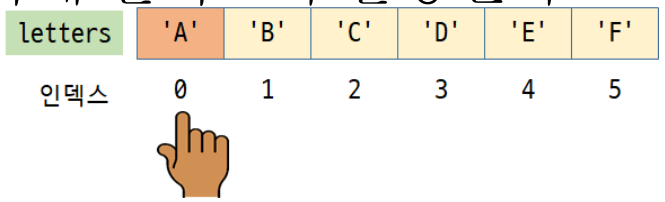
인덱스를 사용하여 리스트의 항목에 접근하자

- 파이썬 리스트에 저장된 데이터를 꺼내려면 어떻게 하면 될까? 예를 들어서 다음과 같이 알파벳 문자를 저장하고 있는 리스트가 있다고 하자.

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

- 리스트에서 데이터를 추출하려면 인덱스index라는 정수를 사용한다. 리스트의 첫 번째 데이터는 인덱스 0을, 두 번째 요소는 인덱스 1을 가지게 된다. 나머지 요소들도 유사하게 인덱스가 할당된다.

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

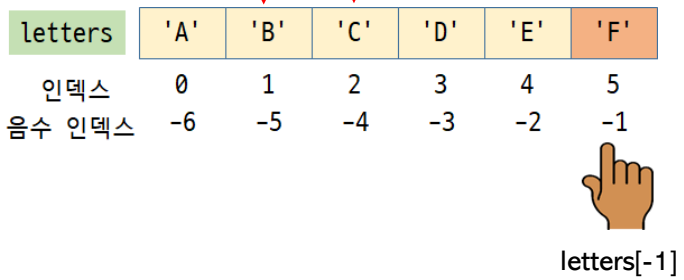


인덱스를 사용하여 리스트의 항목에 접근하자

```
>>> 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'
```



인덱스를 사용하여 리스트의 항목에 접근하자

- 다음은 **BTS** 멤버의 영문 이름으로 이루어진 리스트이다. 이 예제는 리스트에 대하여 인덱싱을 이용하여 항목을 접근하거나 **min()**, **max()** 등의 함수를 통해서 항목을 가져오는 기능이 실려있다.

```
>>> 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'
```

리스트의 원소 값을 자유롭게 조작해 보자

- 우리는 리스트의 요소들을 조작할 수 있다. 리스트 요소 변경은 간단하다. 원하는 요소를 인덱스를 이용하여 지정한 후에 할당연산자로 새 값을 할당하면 된다.

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

- 리스트의 슬라이싱을 사용하여 한 번에 여러 원소의 값을 변경할 수도 있다. 예를 들어서 **Park**의 이름을 **'Paik'**으로, 키를 **180.0**으로 바꿔주고 싶은 경우, **slist[2:4]**를 써준 뒤 이름과 키를 리스트로 적으면 된다.

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

리스트의 원소 값을 자유롭게 조작해 보자

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

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

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

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

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

리스트의 원소 값을 자유롭게 조작해 보자

- 리스트의 슬라이싱을 사용하여 한 번에 여러 원소의 값을 변경할 수도 있다. 예를 들어서 Park의 이름을 'Paik'으로, 키를 180.0으로 바꿔주고 싶은 경우, slist[2:4]를 써준 뒤 이름과 키를 리스트로 적으면 된다.

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

- 함수를 사용하여 동적으로 항목을 추가하려면 append()나 insert() 함수를 사용할 수 있다. append()는 리스트의 뒤쪽에만 추가할 수 있으나 insert(index, item)는 지정된 index 위치에 항목을 추가한다.

리스트의 원소 값을 자유롭게 조작해 보자

- 리스트의 메소드와 하는 일을 정리하면 다음 표와 같다.

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

항목을 삭제하는 방법은 여러 가지가 있다

- 우리는 앞에서 리스트의 항목을 추가하거나 변경시키는 연산자와 메소드를 알아보았다. 리스트의 항목을 삭제하는 것도 가능할까? 물론이다. 삭제를 위해서는 **remove()**, **pop()** 메소드 그리고 **del** 명령을 사용할 수 있다.

- **remove()** 메소드는 다음과 같이 리스트에서 지정된 항목을 삭제한다.

항목을 삭제하는 방법은 여러 가지가 있다

remove() 메소드는 다음과 같이 리스트에서 지정된 항목을 삭제

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

조건이 참인 경우에만 **bts** 리스트에서 이 항목을 삭제하면 안전한 프로그램이 된다

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

항목을 삭제하는 방법은 여러 가지가 있다.

- **pop()** 역시 리스트에서 항목을 삭제하는 역할을 하는데 **remove()** 와는 달리 리스트의 마지막 항목을 삭제하여 그 항목값을 반환한다.

```
>>> 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']
```

항목을 삭제하는 방법은 여러 가지가 있다.

- **del** 명령어는 **리스트의 메소드가 아니므로 주의를 기울여야 한다.** 이 명령어는 파이썬의 키워드인데 다음과 같이 인덱스를 사용하여 해당하는 항목을 메모리에서 삭제할 수도 있다.

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

리스트를 탐색해보자

- 우리는 리스트에서 특정한 항목을 찾을 수 있다. 얼마나 편리한 기능인가? C와 같은 언어에서는 이 기능을 사용하기 위해서는 상당한 코드가 있어야 한다. 반면 파이썬에서는 특정 항목이 리스트의 어떤 인덱스에 있는지를 알려면 index()를 사용한다.

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

- remove()와 마찬가지로 만약 탐색하고자 하는 항목이 리스트에 없다면 오류가 발생한다. 따라서 탐색하기 전에 in 연산자를 이용하여 리스트 안에 항목이 있는지 부터 확인하는 편이 안전하다.

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

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

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

리스트를 탐색해보자

- 만약 리스트의 모든 항목을 한 번씩 방문하려면 어떻게 해야 할까? 아주 간단하다. 다음과 같은 구문을 사용하면 된다. 이 방식은 반복문을 공부하며 이미 다른 바가 있다. 유용한 방법이라 리스트에서 다시 한 번 복습해 본다.

```
bts = ["V", "J-Hope", "Suga", "Jungkook" ]
for member in bts:
    print(member)
```

```
V
J-Hope
Suga
Jungkook
```

리스트를 크기에 따라 정렬해보자

- 리스트는 정렬할 수도 있다. **정렬**sorting이란 리스트 안의 항목들을 크기 순으로 나열하는 것이다. 정렬을 위해서는 리스트의 `sort()`메소드를 사용하면 된다.

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

- 리스트 안에 들어 있는 항목들이 문자열일 경우 사전 순서대로 정렬되는 것도 확인해 볼 수 있다.

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

- 만약 리스트를 역으로 정렬하고 싶으면 `sorted()`를 호출할 때, `reverse=True`을 붙인다.

```
>>> 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]
```

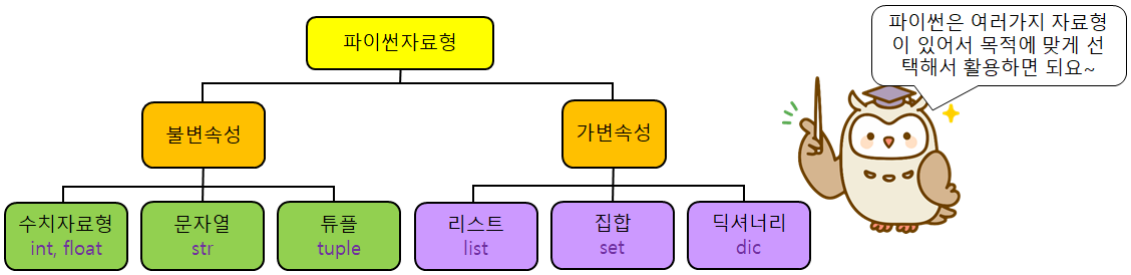
오늘의 명언을 골라주는 기능을 만들자

```
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)
```

한번 생성하면 그 값을 고칠 수 없는 자료형 : 튜플

- 파이썬의 데이터 형을 크게 분류하면 다음 그림과 같이 **불변속성**immutable 데이터 형, **가변속성**mutable 데이터 형으로 나눌 수 있다. **튜플**tuple은 리스트와 아주 유사하다. 하지만 튜플의 내용은 한 번 지정되면 변경될 수 없으며 대표적인 불변속성 자료형이다.

- 튜플은 한 번 그 내용이 정해지면 변경이 불가능하므로 구조가 단순하고 **리스트에 비하여 접근 속도가 빠르다**는 장점도 있다. 따라서 두 자료형은 목적에 따라서 적합한 경우에 선택하여 사용한다.



한번 생성하면 그 값을 고칠 수 없는 자료형 : 튜플

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

색상을 저장하는 튜플

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

다수의 정수를 저장하는 튜플

한번 생성하면 그 값을 고칠 수 없는 자료형 : 튜플

- 다시 한번 강조하면 튜플은 **변경될 수 없는 객체**immutable이므로 튜플의 요소는 변경될 수 없다. 따라서 다음과 같이 변경을 시도하면 에러를 출력한다.

- 튜플도 시퀀스의 일종이기 때문에 인덱싱과 슬라이싱은 문자열이나 리스트와 동일하

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

```
>>> 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)
```

함수는 튜플을 돌려줄 수 있다

C나 C++, Java와 같은 프로그래밍 언어에서는 함수가 하나의 값만을 반환할 수 있다. 파이썬에서는 함수가 튜플을 반환하게 하면 함수가 여러 개의 값을 동시에 반환할 수 있다. 원의 넓이와 둘레를 동시에 반환하는 함수를 작성하고 테스트 해보자.

원하는 결과

```
원의 반지름을 입력하시오: 10
원의 넓이는 314.159265358979300이고 원의 둘레는 62.831853071795860이다.
```

함수는 튜플을 돌려줄 수 있다

```
import math

def calCircle(r):
    # 반지름이 r인 원의 넓이와 둘레를 동시에 반환하는 함수 (area, circum)
    area = math.pi * r * r
    circum = 2 * math.pi * r
    return area, circum    # 튜플을 반환함

radius = float(input("원의 반지름을 입력하시오: "))
(a, c) = calCircle(radius)
print("원의 넓이는 "+str(a)+"이고 원의 둘레는"+str(c)+"이다.")
```

잠깐 – 임시 변수 없는 데이터 교환

두 변수의 값을 서로 교환하여 바꾸는 일이 종종 필요하다. 초보 프로그래머가 종종 실수하는 예는 다음처럼 서로 할당을 하는 것이다.

```
a = b
b = a
```

그러나 첫 문장 수행후 a의 값은 b로 바뀌기 때문에 두 번째 문장은 의미가 없어지고, 두 변수는 모두 원래의 b가 가지고 있던 값으로 변해 같은 값이 되어 버린다. 이를 피하기 위해 임시변수를 사용해야 한다.

```
temp = a
a = b
b = temp
```

그런데 튜플을 이용하면 이런 임시 변수없이 간단히 데이터를 교환할 수 있다.

```
a, b = b, a
```