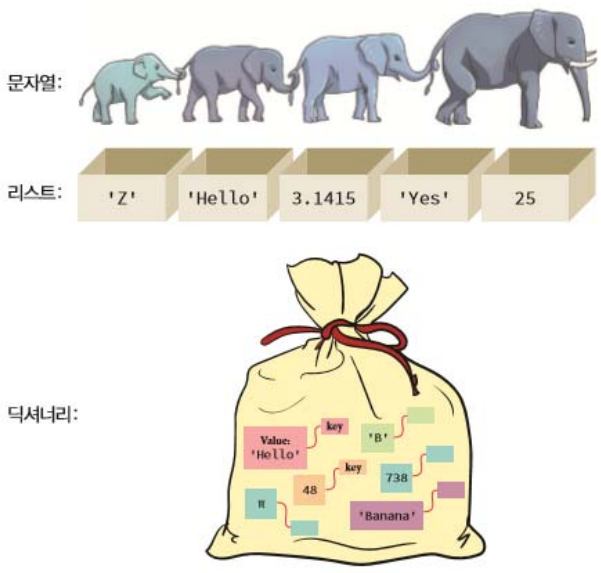
7장. 튜플, 세트, 딕셔너리, 문자열

자료구조란?

- 프로그램에서 자료들을 저장하는 여러 가지 구조들이 있다. 이를 **자료 구조(data structure)**라 부른다.



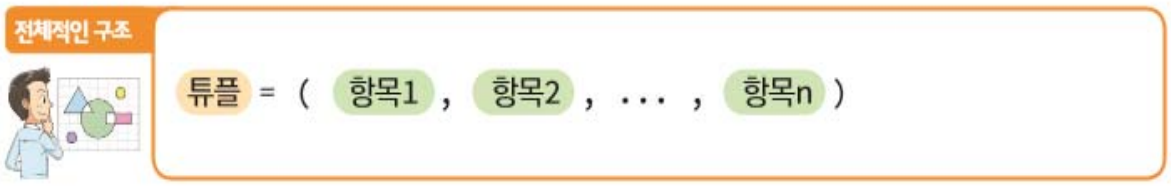
시퀀스

- 시퀀스에 속하는 자료 구조들은 동일한 연산을 지원한다.
  - 인덱싱(indexing), 슬라이싱(slicing), 덧셈 연산(adding), 곱셈 연산(multiplying)

- 리스트는 앞에서 자세하게 살펴본바 있다. 여기서는 나머지 시퀀스들을 탐구해보자.

튜플

- **튜플(tuple)**은 변경될 수 없는 리스트




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

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

튜플은 변경할 수 없다

```
>>> t1 = (1, 2, 3, 4, 5);
>>> t1[0] = 100
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    t1[0]=100
TypeError: 'tuple' object does not support item assignment
```

```
>>> numbers = ( 1, 2, 3, 4, 5 )
>>> colors = ("red", "green", "blue")
>>> t = numbers + colors
>>> t
(1, 2, 3, 4, 5, 'red', 'green', 'blue')
```

기본적인 튜플 연산들

파이썬 수식	결과	설명
`len((1, 2, 3))`	3	튜플의 길이
`(1, 2, 3) + (4, 5, 6)`	(1, 2, 3, 4, 5, 6)	접합
`('Hi!',) * 4`	('Hi!', 'Hi!', 'Hi!', 'Hi!')	반복
`3 in (1, 2, 3)`	True	멤버쉽
`for x in (1, 2, 3): print x,`	1 2 3	반복

함수	설명
`cmp(t1, t2)`	2개의 튜플을 비교한다.
`len(t)`	튜플의 길이를 반환한다.
`max(t)`	튜플에 저장된 최대값을 반환한다.
`min(t)`	튜플에 저장된 최소값을 반환한다.
`tuple(seq)`	리스트를 튜플로 변환한다.

튜플 대입 연산

```
>>> student1 = ("철수", 19, "CS")
>>> (name, age, major) = student1
>>> name
'철수'
>>> age
19
>>> major
'CS'
```

Lab: 함수의 튜플 반환 예제

- 원의 넓이와 둘레를 동시에 반환하는 함수를 작성, 테스트해보자.

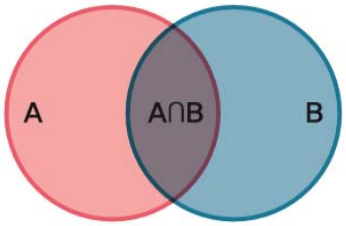
```
원의 반지름을 입력하시오: 10
원의 넓이는 314.1592653589793이고 원의 둘레는 62.831853071795860이다.
```

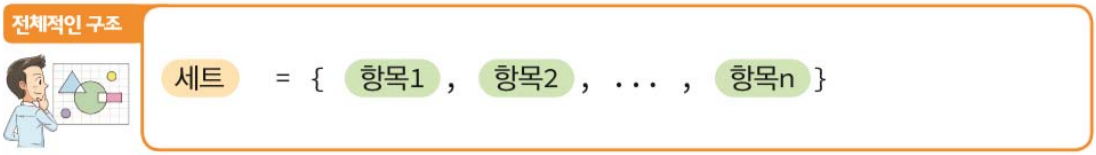
Solution

```
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)

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

세트(Set)

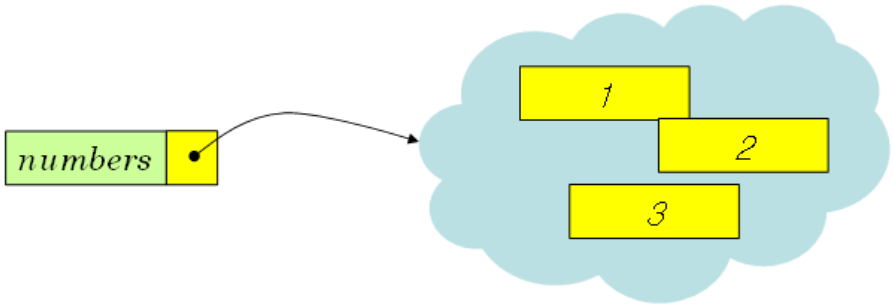
- 세트(set)는 우리가 수학에서 배웠던 집합이다.
- 세트는 중복되지 않은 항목들이 모인 것
- 세트의 항목 간에는 순서가 없다.





예제

```
>>> numbers = {2, 1, 3}
>>> numbers
{1, 2, 3}

>>> len(numbers)
3

>>> fruits = { "Apple", "Banana", "Pineapple" }
>>> mySet = { 1.0, 2.0, "Hello World", (1, 2, 3) }
```



in 연산자

```
>>> numbers = {2, 1, 3}
>>> if 1 in numbers:
      print("집합 안에 1이 있습니다.")

집합 안에 1이 있습니다.
```

```
>>> numbers = {2, 1, 3}
>>> for x in numbers:
      print(x, end=" ")

1 2 3
```

세트에 요소 추가하기

```
>>> numbers = { 2, 1, 3 }
>>> numbers[0]
...
TypeError: 'set' object does not support indexing


>>> numbers.add(4)
>>> numbers
{1, 2, 3, 4}
```

부분 집합 연산

```
>>> A = {1, 2, 3}
>>> B = {1, 2, 3}
>>> A == B                  # 같은가?
True

>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B < A                   # 큰가?
True

>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B.issubset(A)           # 부분집합인가?
True
```

집합 연산

```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5}

>>> A | B          # 합집합
{1, 2, 3, 4, 5}


>>> A & B          # 교집합
{3}

>>> A - B          # 차집합
{1, 2}
```

Lab: 파티 동시 참석자 알아내기

- 파티에 참석한 사람들의 명단이 세트 A와 B에 각각 저장되어 있다. 2개 파티에 모두 참석한 사람들의 명단을 출력하려면 어떻게 해야 할까?

```
2개의 파티에 모두 참석한 사람은 다음과 같습니다.
{'Park'}
```

Solution

```
partyA = set(["Park", "Kim", "Lee"])
partyB = set(["Park", "Choi"])

print("2개의 파티에 모두 참석한 사람은 다음과 같습니다.")
print ( partyA.intersection(partyB))
```

사전 (Dictionary)

- 딕셔너리는 키(key)와 값(value)의 쌍을 저장할 수 있는 객체

키(key)	값(value)
"Kim"	"01012345678"
"Park"	"01012345679"
"Lee"	"01012345680"

딕셔너리 생성

전체적인 구조

딕셔너리 = { 키1 : 값1 , 키2 : 값2 , ... }

```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679',
'Lee':'01012345680' }

>>> contacts
{'Kim': '01012345678', 'Lee': '01012345680', 'Park':
'01012345679'}
```

항목 접근하기

```
>>> contacts = {'Kim':'01012345678',
'Park':'01012345679', 'Lee':'01012345680' }

>>> contacts['Kim']
'01012345678'

>>> contacts.get('Kim')
'01012345678'
```

```
>>> if "Kim" in contacts:
        print("키가 딕셔너리에 있음")
```

항목 추가 & 삭제하기

```
>>> contacts['Choi'] = '01056781234'
>>> contacts
{'Kim': '01012345678', 'Choi': '01056781234', 'Lee':
'01012345680', 'Park': '01012345679'}
```

```
>>> contacts = {'Kim':'01012345678',
'Park':'01012345679', 'Lee':'01012345680' }

>>> contacts.pop("Kim")
'01012345678'

>>> contacts
{'Lee': '01012345680', 'Park': '01012345679'}
```

항목 순회하기

```
>>> scores = { 'Korean': 80, 'Math': 90, 'English': 80}
>>> for item in scores.items():
        print(item)


('Math', 90)
('English', 80)
('Korean', 80)
>>>
```

Lab: 영한 사전 만들기

- 우리는 영한 사전을 구현하여 보자. 어떻게 하면 좋은가? 공백 딕셔너리를 생성하고 여기에 영어 단어를 키로 하고 설명을 값으로 하여 저장하면 될 것이다.

```
단어를 입력하시오: one
하나
```

```
단어를 입력하시오: python
없음
```

Solution

```
english_dict = dict()

english_dict['one'] = '하나'
english_dict['two'] = '둘'
english_dict['three'] = '셋'

word = input("단어를 입력하시오: ");
print (english_dict.get(word, "없음"))
```

Lab: 단어 카운터

- 사용자가 지정하는 파일을 읽어서 파일에 저장된 각각의 단어가 몇 번이나 나오는지를 계산하는 프로그램을 작성하여 보자.

```
파일 이름: proverbs.txt
{'a': 1, 'done.': 1, 'that': 1, 'well.': 1, 'ends': 1,
'Well': 1, 'flock': 1, 'feather': 1, "All's": 1, 'Birds':
1, 'together.': 1, 'of': 1, 'fast.': 1, 'begun': 1,
'half': 1, 'well': 1, 'travels': 1, 'news': 1, 'is': 1,
'Bad': 1}
```

Solution

```
fname = input("파일 이름: ")
file = open(fname, "r")

table = dict()
for line in file:
    words = line.split()
    for word in words:
        if word not in table:
            table[word] = 1
        else:
            table[word] += 1

print(table)
```

Lab: 축약어 풀어쓰기

- 현대인들은 축약어를 많이 사용한다. 예를 들어서 "B4(Before)" "TX(Thanks)" "BBL(Be Back Later)" "BCNU(Be Seeing You)" "HAND(Have A Nice Day)"와 같은 축약어들이 있다. 축약어를 풀어서 일반적인 문장으로 변환하는 프로그램을 작성하여 보자.

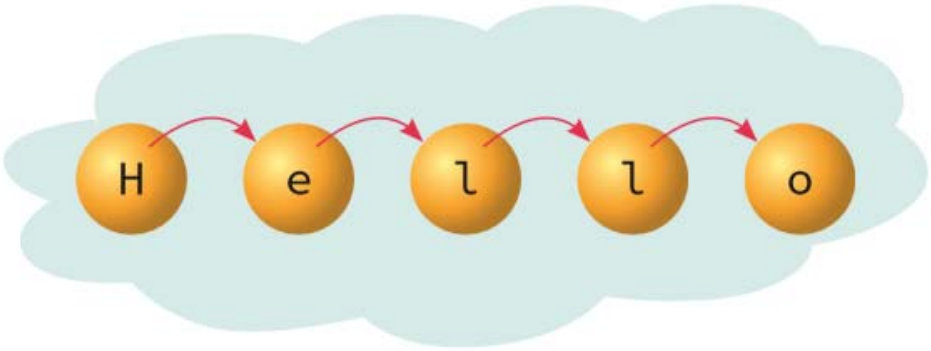
```
번역할 문장을 입력하시오: TX Mr. Park!
Thanks Mr.Park!
```

Solution

```
table = { "B4": "Before",
          "TX": "Thanks",
          "BBL": "Be Back Later",
          "BCNU":"Be Seeing You",
          "HAND":"Have A Nice Day" }

message = input('번역할 문장을 입력하시오: ')
words = message.split()
result = ""
for word in words:
    if word in table:
        result += table[word] + " "
    else:
        result += word

print(result)
```

문자열

- **문자열**은 문자들의 시퀀스로 정의된다. 글자들이 실(string)로 묶여 있는 것이 문자열이라고 생각하면 된다.



문자열의 예

```
s1 = str("Hello")
s2 = "Hello"

s1 = "Hello"
s2 = "World"
s3 = "Hello" + "World"
```

개별 문자 접근하기

```
>>> word = 'abcdef'
>>> word[0]
'a'
>>> word[5]
'f'
```

a	b	c	d	e	f
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

슬라이싱

```
>>> word = 'Python'
>>> word[0:2]
'Py'
>>> word[2:5]
'tho'
```

in 연산자와 not in 연산자

```
>>> s="Love will find a way."
>>> "Love" in s
True
>>> "love" in s
False
```

```
s = input("문자열을 입력하시오")
if 'c' in s:
    print('c가 포함되어 있음')
else:
    print('c가 포함되어 있지 않음')
```

문자열 비교하기

```
a = input("문자열을 입력하시오: ")
b = input("문자열을 입력하시오: ")
if( a < b ):
     print(a, "가 앞에 있음")
else:
     print(b, "가 앞에 있음")
```

```
문자열을 입력하시오: apple
문자열을 입력하시오: orange
apple 가 앞에 있음
```

문자열에서 단어 분리

```
>>> s = 'Never put off till tomorrow what you can do today.'
>>> s.split()
['Never', 'put', 'off', 'till', 'tomorrow', 'what', 'you', 'can',
'do', 'today.']
```

Lab: 머리 글자어 만들기

- 머리 글자어(acronym)은 NATO(North Atlantic Treaty Organization)처럼 각 단어의 첫 글자를 모아서 만든 문자열이다. 사용자가 문장을 입력하면 해당되는 머리 글자어를 출력하는 프로그램을 작성하여 보자.

```
문자열을 입력하시오: North Atlantic Treaty Organization
NATO
```

Solution

```
phrase  = input("문자열을 입력하시오: ")

acronym = ""
for word in phrase.upper().split():
      acronym += word[0]

print( acronym )
```

Lab: 문자열 분석

- 문자열 안에 있는 문자의 개수, 숫자의 개수, 공백의 개수를 계산하는 프로그램을 작성하여 보자.

문자열을 입력하시오: A picture is worth a thousand words.

{'digits': 0, 'spaces': 6, 'alphas': 29}

Solution

```
sentence = input("문자열을 입력하시오: ")

table = { "alphas": 0, "digits":0, "spaces": 0 }

for i in sentence:
    if i.isalpha():
        table["alphas"] += 1
    if i.isdigit():
        table["digits"] += 1
    if i.isspace():
        table["spaces"] += 1

print(table)
```

핵심 정리

- **튜플**은 리스트와 유사하지만 변경할 수 없는 객체이다.
- **세트**는 집합으로 요소들은 중복되지 않아야 한다.
- **딕셔너리**는 사전으로 키와 값의 쌍으로 저장된다. 키를 이용하여 값을 찾을 수 있다.