Chapter 02. 파이썬 문법

2. 자료형과 연산자

2.1 파이썬 자료형

1) 다양한 정보를 저장할 수 있는 자료형이 내장

2) 강력한 고수준 자료형 지원

3) 종류

1. **bool** : True, False를 나타내는 자료형. ex) True, False
2. **int, float, cmplex**: 정수, 실수, 복소수 등 숫자를 표현. ex) 123, 1.23, 1+2j
3. **str**: 유니코드 문자의 모임을 표현. ex) ‘abcd’, ‘Hello World’
4. **bytes**: 0~255 사이의 코드모임을 표현. ex) b‘Python’
5. **list**: 순서가 있는 객체의 집합을 표현. ex) [‘abcd’, 2, 1.23f]
6. **tuple**: 순서가 있는 객체의 집합을 표현. 내용 변경이 안됨. ex) (‘abcd’, 1234)
7. **set**: 집합을 표현. ex) { 123, ‘abcd’, 1.23f }
8. **dict**: 순서가 없는 객체를 저장하며 참조는 key로 한다 ex) { ‘a’: 1234, ‘b’:1.23f }

2.2 자료형의 분류

1) 접근 방법

- 직접(Direct) : int, float, complex, bool
- 시퀀스(Sequence) : bytes, str, list, tuple
- 매핑(Mapping) : dic

2) 변경 가능성

- 변경 가능(Mutable) : list, set, dict
- 변경 불가능(Immutable) : int, float, complex, bool, bytes, str, tuple

3) 저장 모델

- 리터럴(Literal) : int, float, complex, bool, bytes, str
- 저장(Container) : list, tuple, dict, set

2.3 bool 자료형

참이나 거짓을 나타내는 True, False 두 상수를 갖는다

[예제 bool.py]

```
a = 1
print(a > 10)
print(a < 10)

b = a == 1
print(type(b))

print(b + 10)
print(True + True)
```

2.4 수치 자료형

1) 정수형(int)

[예제 int.py]

10진, 2진, 8진, 16진 정수를 리터럴로 사용한다.

```
a = 23
print(type(a))
print(isinstance(a, int))

b = 0b1101
c = 0o23
d = 0x23

print(a, b, c, d)
```

파이썬 3.x 에서는 long형이 없어지고 int 타입으로 처리된다.
따라서 표현범위는 무제한이다.

```
e = 2**1024
print(type(e))
print(e)
```

2) 실수형(float)

[예제 float.py]

다른 언어에서처럼 소수점을 포함하거나 e나 E로 지수로 표현할 수 있다.

```
a = 1.2
print(type(a))
print(isinstance(a, float))
print(a.is_integer())

b = 3e3
c = -0.2e-4

print(a, b, c)
```

객체 함수 is_integer()는 실수 타입 객체의 값이 정수인지 판별한다.

타입 판별함수가 아니다.

```
a = 2.0
print(type(a))
print(a.is_integer())
```

3) 복소수형(complex)

[예제 complex.py]

복소수 타입의 객체는 실수부 + 허수부로 나뉘며 허수부에는 j 또는 J를 숫자 뒤에 붙힌다.

```
a = 4 + 5j
print(type(a))
print(isinstance(a, complex))
```

복소수 연산이 가능하며 실수부와 허수부 값만 따로 참조할 수도 있다.

```
b = 7 - 2j
print(a + b)
print(b.real, b.imag)
```

complex 함수를 이용하여 복소수 타입의 객체를 만들 수 있다.

```
b = 7 - 2j
print(a + b)
print(b.real, b.imag)
```

4) 수치 자료형의 연산자

4-1) 산술 연산자

[예제 arithmetic_operator.py]

사칙연산

```
print(2 * 3)
print(2 + 3)
print(2 - 3)
print(2 / 3)
print(2 / 3.0)
print(2.0 / 3)
print(2.0 / 3.0)
```

특히, 나눗셈(/)은 파이썬 3.x 부터는 혼란을 피하기 위해 실제 나눗셈을 한다.

//(몫 연산자), **(지수승), %(나머지 연산)

```
print(2 // 3)
print(2 ** 3)
print(2 % 3)
print(divmod(2, 3))
```

2. 자료형과 연산자

4-2) 산술 연산자 우선순위

1. +, - (단항 연산자) <-
2. ** <-
3. *, /, %, // ->
4. +, - (더하기, 빼기) ->

```
print(2 + 3 * 4)
print(-2 + 3 * 4)
print(-(2+3) * 4)

print(4 / 2 * 2)
print(4 / (2 * 2))

print(2 ** 3 ** 4)
print((2 ** 3) ** 4)
print(2 ** (3 ** 4))
```

4-3) 관계 연산자

[예제 relational_operator.py]

객체의 대소 비교하는 연산

```
print(1 > 3)
print(2 < 4)
print(4 <= 5)
print(4 >= 5)
print(6 == 9)
print(6 != 9)
```

다음의 복합 관계식도 지원한다.

```
a = 6
print(0 < a < 10)
print(0 < a and a < 10)
```

수치형 객체 뿐만 아니라, 다른 객체의 비교도 가능하다.

```
print('abcd' > 'abd')
print((1, 2, 4) < (1, 3, 1))
print([1, 3, 2] > [1, 2, 0])
```

== 는 객체의 값을 비교한다.(동질성) 같은 객체(동일성)을 비교할 때는 is 연산자를 사용한다.

```
a = 10
b = 20
c = a
print(a == b)
print(a is b)
print(a is c)
```

4-4) 논리 연산자

[예제 logical_operator.py]

진리 값(True 또는 False)을 피연산자(operand)로 취해서 논리 값 계산을 한다.

```
a = 20
print(not a < 30)
print(a < 30 and a != 30)
print(a == 30 or a > 30)
```

2. 자료형과 연산자

bool 타입 객체의 값은 True -> 1, False -> 0 값을 가진다.

```
print(True + 1)
print((a > b) + 1)
```

다른 타입의 객체도 bool 타입으로 변환이 가능하다.

```
print(bool(10), bool(0))
print(bool(3.14), bool(0.))
print(bool('abc'), bool(''))
print(bool([1, 2, 3]), bool([]))
print(bool((1, 2, 3)), bool(()))
print(bool({1:2}), bool({}))
print(bool(None))
```

논리식의 계산 순서

```
print([] or 'logical')
print('logical' and 'operator')
print(None and 1)
```

4-5) 비트 연산자

[예제 bitwise_operator.py]

비트 연산자는 정수 자료형에만 적용된다.

```
# ~ 연산자
print(~5)
print(~-1)

# << 연산자
a = 3
print(a << 2)

# >> 연산자
a = 4
print(a >> 1)
a = -4
print(a >> 1)

# bit and
a = 3
print(a & 2)
print(a | 8)
print(0x0f ^ 0x06)
```

4-6) 연산자 우선 순위

1.

f() 함수호출
x[idx:idx] 슬라이싱
x[index] 인덱싱
x.attr 어트리뷰트 참조

2.비트 연산자 not

~

3.산술 연산자

+,-
**
*,/,//,%
+,-

4. 비트연산자

<<, >>
&
^
|

5.관계 연산자

<, <=, >, >=, ==, !=
is, not is
in, not in

6.논리 연산자

not
and
or

5) 내장 수치 함수

연산자 이외에도 다음과 같은 내장 수치 함수들을 제공한다.

[예제 arthmetic_func.py]

```
print(abs(-3))
print(int(3.141592))
print(float(3))
print(complex(1))
print(divmod(5, 3))
print(pow(2, 10))
```

2.5 문자열(str)

‘ 또는 “ 로 묶인 문자들의 모임이다.

1) 문자열의 정의

[예제 str.py]

```
s = ''
str1 = 'hello world'
str2 = "hello world"
print(type(s), type(str1), type(str2))
print(isinstance(str2, str))
```

여러 줄 문자열도 정의할 수 있다.

```
str3 = """ABCDEFG
abcdef
가나다라마
123456789"""
print(str3)
```

2) 문자열 연산

기본적으로 시퀀스형이므로 시퀀스의 연산(인덱싱, 슬라이싱, 연결(+), 반복(*), len(), in, not in) 등의 연산이 가능하다.

[예제 str.py]

```
str1 = "First String"
str2 = "Second String"

print(str1 + str2)
print(str1*3)
print(str1[2])
print(str2[2:5])
print(len(str2))
```

문자열 객체의 내용은 변경 할수 없는 변경 불가(Immutable) 자료형이다.

```
str1[0] = 'f'
```

문자열 객체와 정수형 객체는 + 연산을 할 수 없다.

```
name = "길동"
age = 40
print("name: " + name + ", age:" + age)
```

3) 문자열 서식

문자열 서식(Formatting)을 사용하여 좀더 다양하게 표현할 수 있다.

[예제 str.py]

위의 문제는 format() 함수를 사용해서 해결할 수 있다.

```
print("name: " + name + ",age: " + format(age, "d"))
print("name: " + format(name, "s") + ",age: " + format(age, "d"))
```

2. 자료형과 연산자

튜플을 이용한 문자열 포매팅

```
f = 'name: %s, age: %d'

print(f % ('들리', 20))
print(f % ('또치', 10))
print(f % ('도우넛', 30))
```

딕션너리를 이용한 포매팅

```
f = 'name: %(name)s, age: %(age)d'
print(f)

print(f % {'name': '들리', 'age': 20})
print(f % {'name': '또치', 'age': 10})
print(f % {'name': '도우넛', 'age': 30})
```

4) 문자열 메서드

문자열 객체는 다양한 메서드를 제공한다.

[예제 str.py]

대소문자 관련 메서드

```
s = 'i like Python'

print(s.upper())
print(s.lower())
print(s.swapcase())
print(s.capitalize())
print(s.title())
```

2. 자료형과 연산자

검색관련 메서드

```
s = 'I Like Python. I Like Java Also'
print(s.count('Like'))

print(s.find('Like'))
print(s.find('Like', 5))
print(s.find('JS'))
print(s.rfind('Like'))

# print(s.index('JS'))
print(s.rindex('Like'))

print(s.startswith('I Like'))
print(s.startswith('Like', 2))
print(s.endswith('Also'))
print(s.endswith('Java', 0, 26))
```

2. 자료형과 연산자

편집과 치환 메서드

```
s = '   spam and ham    '
print(s.strip())
print(s.rstrip())
print(s.lstrip())

s = '<><abc><><defg><><>'
print(s.strip('<>'))

s = 'Hello Java'
print(s.replace( 'Java', 'Python'))
```

정렬 메서드

```
s = 'king and queen'

print(s.center(60))
print(s.center(60, '-'))
print(s.ljust(60, '-'))
print(s.rjust(60, '-'))
```

2. 자료형과 연산자

분리와 결합 메서드

```
s = 'spam and ham'
t = s.split();
print(t)
t = s.split(' and ');
print(t)

s2 = ":".join(t)
print(s2)

s3 = "one:two:three:four:five"
print(s3.split(':', 2))
print(s3.rsplit(':', 2))

lines = '''1st line
2nd line
3rd line
4th line
'''
print(lines.splitlines());
```

2. 자료형과 연산자

판별 메서드

```
print('1234'.isdigit())
print('abcd'.isalpha())

print('1234'.isalpha())
print('abcd'.isdigit())

print('abcd'.islower())
print('ABCD'.isupper())

print('\n\n'.isspace())
print('    '.isspace())
print(''.isspace())
```

‘0’ 로 채우기

```
print('20'.zfill(5))
print('1234'.zfill(5))
```

2. 자료형과 연산자

서식(포매팅) 메서드

```
f = 'name: {}, age: {}'
print(f)
print(f.format("둘리", 10))

print('name: {0}, age: {1}'.format('둘리', 10))
print('name: {1}, age: {0}'.format(30, '마이콜'))

print('{:3}의 제곱근은 {:.5}입니다.'.format(2, 2**0.5))
print('{1:03}의 제곱근은 {0:.5}입니다.'.format(2**0.5, 2))

f = "name: {name}, age: {age}"
print(f.format_map({'name':'도우넛', 'age': 10 }))
```

2.6 리스트(list)

순서를 가지는 객체들의 집합, 파이썬 자료형들 중에서 가장 많이 사용한다.

1) 리스트 생성과 연산

[예제 list.py]

시퀀스 자료형이기 때문에 시퀀스의 연산(인덱싱, 슬라이싱, 연결(+), 반복(*), len(), in, not in) 등의 연산이 가능하다. 특히, list는 del() 함수를 통한 삭제도 가능하다.

```
l = [1, 2, 'python']

print(l[-2], l[-1], l[0], l[1], l[2])
print(l[1:3])
print(l * 2)
print(l + [3, 4, 5])
print(len(l))
print(2 in l)

del l[0]
print(l)
```

2. 자료형과 연산자

리스트는 변경 가능(Mutable)한 자료형이다.

```
a = ['apple', 'banana', 10, 20]
a[2] = a[2] + 90
print(a)
```

슬라이스를 통한 치환

```
a = [1, 12, 123, 1234]

a[0:2] = [10, 20]
print(a)

a[0:2] = [10]
print(a)

a[1:2] = [20]
print(a)

a[2:3] = [30]
print(a)
```

2. 자료형과 연산자

슬라이스를 통한 삭제

```
a = [1, 12, 123, 1234]
a[1:2] = [ ]
print(a)
a[0:] = [ ]
print(a)
```

슬라이스를 통한 삽입

```
a = [1, 12, 123, 1234]

a[1:1] = ['a']
print(a)

a[5:] = [12345]
print(a)

a[:0] = [-12, -1, 0]
print(a)
```

2) 리스트의 메서드

[예제 list.py]

다음과 같은 메서드를 지원한다.

```
a = [1, 2, 3]
print(a)
a.append(5)
print(a)
a.insert(3, 4)
print(a)
print(a.count(2))
a.reverse();
print(a)
a.sort()
print(a)
a.remove(3)
print(a)
a.extend([6, 7, 8])
print(a)
```

3) 리스트를 스택으로 사용하기

[예제 list.py]

```
stack = [ ]

stack.append(10)
stack.append(20)
stack.append(30)

print(stack)

print(stack.pop())
print(stack.pop())
print(stack)
```

4) 리스트를 큐로 사용하기

[예제 list.py]

```
q = [ ]

q.append(100)
q.append(200)
q.append(300)

print(q)

print(q.pop(0))
print(q.pop(0))

print(q)
```

5) 리스트의 정렬

[예제 list.py]

리스트의 sort 메서드는 내장된 순서에 따라 정렬한다.

```
l = [1, 5, 3, 9, 8, 4, 2]
l.sort()
print(l)

l.sort(reverse=True)
print(l)
```

키값 기반의 사용자 정의 정렬하기

```
l = [10, 2, 22, 9, 8, 33, 4, 11]
l.sort(key=str);
print(l)

l.sort(key=int);
print(l)
```

6) 명령행(Command Line)에서 인수(Arguments) 처리하기

[예제 args.py]

```
import sys

print(sys.argv)

args = sys.argv[1:]
print(args)
```

2.7 세트(set)

순서가 없는 객체들의 집합이다. 수정이 가능한 자료형이다.

1) 세트의 생성과 연산

[예제 set.py]

시퀀스 자료형이 아니기 때문에 시퀀스의 연산 중 len(), in, not in 정도만 가능하다.

```
a = {1, 2, 3}
print(a, type(a))

print(len(a))
print(2 in a)
print(2 not in a)
```

2) 세트의 메서드

수정이 가능한 자료형이나 순서가 없기 때문에 메서드가 list에 비해 제한적이다.

```
s.add(4)
s.add(1)
s.discard(2)
s.remove(3)
s.update({2, 3})
s.clear()
```

2. 자료형과 연산자

수학에서의 집합 연산은 다음 메서드를 통해 가능하다.

```
s1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
s2 = {10, 20, 30}

s3 = s1.union(s2)
print(s3)

s4 = s1.intersection(s2)
print(s4)

s4 = s1.difference(s2)
print(s4)

s5 = s1.symmetric_difference(s2)
print(s5)

print(s1.issuperset(s4))
print(s5.issuperset(s1))
print(s2.issubset(s3))
```

2.8 튜플(tuple)

순서를 가지는 객체들의 집합이라는 것은 list와 같지만 수정 불가(immutable) 자료형이다.

1) 튜플의 생성과 연산

[예제 tuple.py]

시퀀스 자료형이기 때문에 시퀀스의 연산(인덱싱, 슬라이싱, 연결(+), 반복(*), len(), in, not in) 등의 연산이 가능하다.

```
t = (1, 2, 3)
print(t, type(t))

t = 1, 2, 'python'
print(t, type(t))

print(t[-2], t[-1], t[0], t[1], t[2])
print(t[1:3])
print(t[:])

print(t * 2)
print(t + (3, 4, 5))
print(len(t3))
print(5 in t3)
```

2. 자료형과 연산자

튜플은 변경 불가능(Immutable)한 시퀀스 형이다.

```
t = ('apple', 'banana', 10, 20)
t[2] = t[2] + 90
```

따라서 슬라이스를 통한 치환, 삭제, 추가등은 지원하지 않는다.

```
t = (1, 12, 123, 1234)
t[0:2] = (10, 20)
```

2) packing 과 unpacking

packing은 나열된 객체를 tuple로 저장하는 것을 말한다. unpacking은 반대의 작업으로 tuple, list 안의 객체를 개개의 변수로 할당할 수 있다.

[예제 unpacking.py]

```
t = 10, 20, 30, 'python'
print(t)
print(type(t))

# unpacking tuple
a, b, c, d = t
print(a, b, c, d)

# unpacking list
a, b, c, d = [10, 20, 30, 'python']
print(a, b, c, d)
```

unpacking에서는 왼쪽 변수가 부족한 경우 에러가 발생한다.

```
# unpacking error
a, b = (10, 20, 30, 40)
```

하지만, 확장 언팩킹에서는 왼쪽 변수가 적은 경우에도 다음과 같이 적용될 수 있다.

```
t = (1, 2, 3, 4, 5, 6)
a, *b = t
print(a,  b)

*a, b = t
print(a, b)

a, b, *c = t
print(a, b, c)

a, *b, c = t
print(a, b, c)
```

3) 내장 함수 list(), tuple(), set()를 사용해서 서로서로 변환이 가능하다.

[예제 tuple.py]

```
t = (1, 2, 3)

s = set(t)
print(s, type(s))

l = list(s)
print(l, type(l))

t = tuple(l)
print(t, type(t))
```

2.9 사전(dict)

- 순서를 가지지 않는 객체의 집합
- key 기반으로 값을 저장하고 참조하는 매핑형 자료형

1) 사전의 생성과 연산

[예제 dict.py]

시퀀스 자료형이 아니기 때문에 시퀀스의 연산 중 len(), in, not in 정도만 가능하다.

```
d = {'basketball': 5, 'soccer': 11, 'baseball': 9}
print(d, type(d))

print(d['basketball'])

d['volleyball'] = 6
print(d)

print(len(d))
print('soccer' in d)
print('volleyball' not in d)
```

2. 자료형과 연산자

다양한 사전의 생성 방법

```
d = dict()   # empty dict
print(d)

d = dict(one=1, two=2)  # keyword arguments
print(d)

d = dict([('one', 1), ('two', 2)])   # tuple list
print(d)

keys = ('one', 'two', 'three')
values = (1, 2, 3)
d = dict(zip(keys, values))
print(d)
```

2) 사전의 키

사전의 키는 객체의 값에 따라 해싱해야 하기 때문에 수정이 불가능한 객체여야 한다.
ex) bool, 수치형(int, float, complex), str, tuple

```
d = {}
print(d)

d[True] = 'true'
d[10] = '10'
d["twenty"] = '20'
d[(1, 2, 3)] = '6'

print(d)
```

```
d[[1, 2, 3]] = '6‘
```

3) 사전의 메서드

사전의 키들을 리스트로 반환

```
keys = d.keys()
print(keys)
print(type(keys))

for key in keys:
    print('{0}:{1}'.format(key, d[key]))
```

사전의 값들을 리스트로 반환

```
values = d.values()
print(values)
print(type(values))
```

(key, values) 튜플의 리스트를 반환

```
items = d.items()
print(items)
```

2. 자료형과 연산자

값을 가져오는 메서드

```
n = phone.get('둘리')
print(n)

n = phone.get('길동')
print(n)
# 에러
# n = phone['길동']

n = phone.get('길동', '000-0000-0000')
print(n)
print(phone)

n = phone.setdefault('길동', '000-0000-0000')
print(n)
print(phone)

n = phone.pop('둘리')
print(n)
print(phone)

n = phone.popitem()
print(n, type(n))
print(phone)
```

사전 업데이트와 전체 비우기

```
phone.update({'둘리': '010-1111-1111', '길동': '010-5555-5555'})
print(phone)

phone.clear()
print(phone)
```

2) 사전 순회하기

```
d = {'c': 3, 'a': 1, 'b': 2}
for key in d:
    print(key, end=' ')

for key in d:
    print(str(key) + ":" + str(d[key]), end=' ')

for key in d:
    print("{0}:{1}".format(key, d[key]), end=' ')

for key in d.keys():
    print("{0}:{1}".format(key, d[key]), end=' ')

for key, value in d.items():
    print("{0}:{1}".format(key, value), end=' ')
```

2.10 순차 자료형(Sequence 타입)을 위한 중요 내장 함수

1) range : 순차적인 정수 리스트를 만들 때 사용할 수 있다.

[예제 range.py]

range() 함수를 사용하여 순차적인 정수 리스트를 편리하게 만들어 사용할 수 있다.

```
seq = range(10)
print(seq, type(seq))
print(seq[0:])
print(len(seq))

for i in seq:
    print(i)

seq2 = range(5, 15)
for i in seq2:
    print(i)

seq3 = range(0, -10, -1)
for i in seq3:
    print(i)
```

2) enumerate : 순차 자료형에서 현재 아이템의 색인과 함께 처리하고자 할 때 흔히 사용한다.

[예제 enumerate.py]

```
i = 0
for value in ['red', 'yellow', 'blue', 'white', 'gray']:
    print('{0}: {1}'.format(i, value))
    i += 1

# 비교 : enumerate 함수를 사용했을 때

for i, value in enumerate(['red', 'yellow', 'blue', 'white', 'gray']):
    print('{0}: {1}'.format(i, value))
```

3) zip : 여러 개의 리스트나 튜플 또는 다른 순차 자료형을 서로 짝지어서 튜플의 리스트를 생성한다.

[예제 zip.py]

```
seq1 = {'foo', 'bar', 'baz'}
seq2 = {'one', 'two', 'three'}
z = zip(seq1, seq2)
print(z)
print(type(z))
```

보통, 순회는 다른 순차 자료형과 같다.

```
for t in z:
    print(t, type(t))
```

튜플 형태로 순회가 가능하다.

[예제 zip.py]

여러 개의 순차 자료형을 동시에 순회하는 경우에 사용하면 매우 유용하다.

```
z = zip(seq1, seq2)

for idx, (a, b) in enumerate(z):
    print('%d: %s, %s' % (idx, a, b))
```

다음 예는 짝지어진 순차 자료형을 다시 풀어내는 예제이다.

```
d1 = [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]

seq1, seq2 = zip(*d1)
print(seq1)
print(seq2)
```

2.11 리스트, 세트, 사전 내포(Comprehension)

1) 간결한 표현으로 새로운 리스트를 만들 수 있는 방법이다.

2) 기본 형태

expr for val in collection if condition

[예제 lits_comprehension.py]

```
results = []
for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    result = i * i
    results.append(result)
print(results)

results = [result*result for result in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]
print(results)
```

2. 자료형과 연산자

[예제 comprehension.py]

문자열 리스트에서 길이가 2 이하인 문자열 리스트 만들기

```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
strings = [s for s in strings if len(s) <= 2]
print(strings)
```

1~100 사이의 수중에 짝수 리스트 만들기

```
evens = [i for i in range(1, 101) if i % 2 == 0]
print(evens)
```

[실습] 1~100 사이에 3, 6, 9 가 있는 수 리스트 만들기

```
<<결과>>
[3, 6, 9, 13, 16, 19, 23, 26, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 43, 46, 49, 53, 56, 59, 60, 61, 62, 63,
64, 65, 66, 67, 68, 69, 73, 76, 79, 83, 86, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]

Process finished with exit code 0
```

3) 세트 Comprehension

대괄호 대신 중괄호를 쓴다는 점만 빼면 리스트 내포와 동일하다.

문자열 리스트에서 문자열 길이를 순차 자료형으로 저장해 보자

```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
lens = [len(s) for s in strings]
print(lens)

lens = {len(s) for s in strings}
print(lens)
```

4) 사전 Comprehension

문자열 리스트에서 문자열과 문자열 길이를 함께 dict로 저장 해 보자

```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
dicts = {s: len(s) for s in strings}
print(dicts)
```