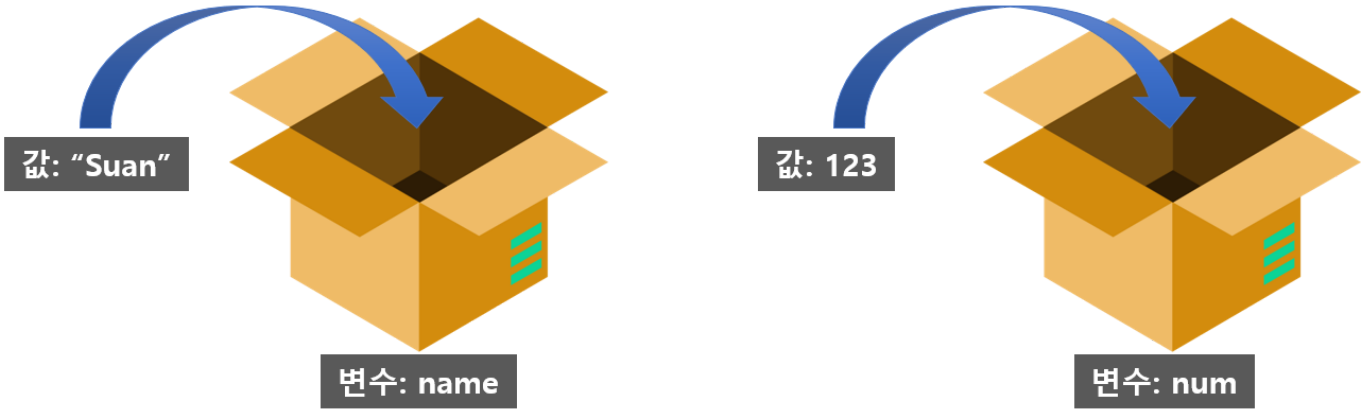
변수, 자료형, 연산자

변수(Variables)

변수 개념

- 어떠한 값을 저장하는 공간(메모리)
- 일반적으로 데이터의 저장 위치와 데이터 값으로 구분
- 변수에 값을 넣는 순간 메모리 저장 위치를 할당하고 그 위치는 메모리 주소로 관리

실제 변수에 대해서 실습하기 위해 그림과 같이 변수 `name`에 문자열 `"Suan"`를 넣고, 변수 `num`에 값 `123`을 넣기




```
name = "Suan"
print(name)
num = 123
print(num)
```

```
Suan
123
```

변수명 규칙

- 알파벳, 숫자, 언더바(_)로 선언 가능
- 의미 있는 단어로 표기하는 것이 좋음
- 대소문자가 구분
- 특별한 의미가 있는 예약어는 사용할 수 없음

예약어

예약어

예약어					
and	exec	not	assert	finally	or
break	for	pass	class	from	print
continue	global	raise	def	if	return
del	import	try	elif	in	while
else	is	with	except	lambda	yield

```
abc = "abc"
abc_123 = 123
Abc_123 = 123
print(abc)
print(abc_123)
print(Abc_123)
```

```
abc
123
123
```

자료형(Data Types)

- 파이썬에서 자료형은 유형에 따라서 논리형, 수치형, 문자형, 구조형으로 구분
- 일반적으로 숫자와 문자, 그리고 구조에 따른 저장을 위해서 여러가지 구분된 자료형을 제공

유형	자료형	설명	선언
논리형	불리언(boolean type)	참(True) 또는 거짓(False)을 표현할 때 사용	`b = True`
수치형	정수(integer type)	자연수를 포함해 값의 영역이 정수로 한정된 값	`i = 10`
수치형	2진수 정수(binary type)	2진수 자료형(숫자가 '0b'또는 '0B'로 시작)	`b = 0b010`
수치형	8진수 정수(octal type)	8진수 자료형(숫자가 `0o` 또는 `00` 로 시작)	`o = 0o130`
수치형	16진수 정수(hexadecimal type)	16진수 자료형(`0x` 로 시작)	`h = 0xABC`
수치형	실수(floating-point type)	소수점이 포함된 값	`f = 12.34`
수치형	복소수(complex type)	복소수 사용을 위한 자료형	`j = 1 + 23j`
문자형	문자열(string type)	값이 문자로 출력되는 자료형	`s = "Suan"`
시퀀스형	리스트(list type)	여러 요소를 묶어 하나의 변수로 사용	`l = [1, 2, 3]`
시퀀스형	튜플(tuple type)	리스트와 유사하지만 생성, 삭제, 수정 불가	`t = (1, 2, 3)`
시퀀스형	범위(range type)	숫자의 불변한 시퀀스 형태	`range(10)`
집합형	집합(set type)	중복과 순서가 없는 집합을 위한 자료형	`s = {1, 2, 3}`
집합형	불변 집합(fronzenset type)	불변한 집합을 위한 자료형	`fs = frozenset{1, 2, 3}`
매핑형	딕셔너리(dictionary type)	키(key)와 값(value)이 쌍(pair)으로 들어간 자료형	`d = {1:'One', 2:'Two'}`
바이트 시퀀스형	바이트(byte type)	바이트 값을 가지는 자료형	`byte = b'hello'`
바이트 시퀀스형	bytearray type	바이트의 시퀀스 형태로 수정 가능한 자료형	`ba = bytearray(b'hello')`
바이트 시퀀스형	memoryview type	바이트의 메모리 값으로 표현한 자료형	`mv = memoryview(b'hello'`

불리언형 자료형(Boolean Type)

- 불리언은 `True`와 `False` 값으로만 표현할 때 사용하는 자료형
- 예제 코드에서 변수 `b`에 `True` 값을 넣으면 불리언 자료형으로 결정
- 자료형을 반환하는 내장함수 `type()` 함수를 이용하여 자료형을 확인하면 `bool`로 반환된 결과를 볼 수 있음

```
b = True
print(b)
print(type(b))
```

```
True
<class 'bool'>
```

정수형 자료형(Integer Type)

10진수(Decimal)

- 10진수는 일반적인 숫자 표현을 그대로 사용
- 변수 `i`에 10진수로 숫자 10을 값으로 넣고, `type()` 함수를 통해 자료형을 확인하기
- 결과는 당연히 정수형(int) 자료형으로 출력

```
i = 10
print(i)
print(type(i))
```

```
10
<class 'int'>
```

2진수(Binary)

- 2진수는 앞에 `0b`를 붙여서 표현
- 변수 `b`에 2진수 표현으로 `0b010`을 넣고, `type()` 함수를 통해 자료형을 확인하기
- 2진수 010이 10진수로 변환되는 계산: $010_{(2)} = 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2_{(10)}$

```
b = 0b010
print(b)
print(type(b))
```

```
2
<class 'int'>
```

8진수(Octal)

- 8진수는 앞에 `0o`를 붙여서 표현
- 변수 `o`에 8진수 표현으로 `0o130`을 넣고 확인하기
- 8진수 130이 10진수로 변환되는 계산: $130_{(8)} = 1 \times 8^2 + 3 \times 8^1 + 0 \times 8^0 = 88_{(10)}$

```
o = 0o130
print(o)
print(type(o))
```

```
88
<class 'int'>
```

16진수(Hexadecimal)

- 16진수은 앞에 `0x`를 붙여서 표현
- 변수 `h`에 16진수 표현으로 `0xABC`를 넣고 확인하기
- 16진수는 10진수와 달리 10부터 A에서 F까지 알파벳을 사용하여 표현:
  $A_{(16)} = 10_{(10)}, B_{(16)} = 11_{(10)}, C_{(16)} = 12_{(10)}, D_{(16)} = 13_{(10)}, E_{(16)} = 14_{(10)}, F_{(16)}$
- 16진수 ABC가 10진수로 변환되는 계산:
  $ABC_{(16)} = A \times 16^2 + B \times 16^1 + C \times 16^0 = 2748_{(10)}$

```
h = 0xABC
print(h)
print(type(h))
```

```
2748
<class 'int'>
```

실수형 자료형(Floating-Point Type)

- 실수형 자료형 표현은 고정소수점과 부동소수점으로 구분

고정소수점(Fixed-Point)

- 기본적으로 고정된 자리수의 소수를 사용하는 고정소수점 방식
- 실수형 변수 `f`에 소수점이 포함된 `12.34` 값을 넣고 확인하기
- `type(f)`를 통해서 결과를 확인해보면 실수형(float) 자료형으로 출력
- 파이썬에서는 소수에서 정수부가 0인 경우에는 제외해서 표현 가능
- 코드에서 값 `.123`은 `0.123`을 의미

```
f = 12.34
print(f)
print(type(f))
```

```
12.34
<class 'float'>
```

```
f = .123
print(f)
```

```
0.123
```

부동소수점(Floating-Point)

- 컴퓨터에서 소수점의 위치가 고정되지 않고 넓은 범위의 값을 근사하여 표현하는 부동소수점 방식이 있음
- 부동소수점에 대한 개념을 모르는 경우에는 링크를 참고
- 파이썬에서는 e 기호를 이용하여 지수에 대한 표현을 사용
- `e` 변수에 할당한 `1234e-2` 라는 표현은 1234×10^{-2}를 의미하고 실제 결과 값은 12.34
- `type(e)` 함수를 통해 타입을 확인해보면 실수형
- 마찬가지로 `123e-3`은 123×10^{-3}를 의미하며 결과는 0.123

```
e = 1234e-2
print(e)
print(type(e))
```

```
12.34
<class 'float'>
```

```
e = 123e-3
print(e)
```

```
0.123
```

복소수형 자료형(Complex Type)

- 복소수(complex number)는 임의의 실수 a, b에 대해 $a + bi$의 꼴로 나타내는 수를 의미
- 여기서 a는 실수, b는 허수 부분
- i는 제곱하여 -1이 되는 수로 허수(imaginary number) 단위라고 함
- 복소수에 대한 개념을 모르는 경우에는 링크 참고
- 파이썬에서는 i 대신에 j 또는 J로 표현
- $1 + 23j$와 같이 복소수로 표현하기
- `print()` 함수를 이용해 출력한 결과 `(1+23j)`
- 실수부는 `c.real`로 허수부는 `c.imag`로 출력
- 복소수의 경우에는 `type()` 함수로 타입을 살펴보면 결과는 'complex'

```
c = 1 + 23j
print(c)
print(c.real)
print(c.imag)
print(type(c))
```

```
(1+23j)
1.0
23.0
<class 'complex'>
```

문자열 자료형(String Type)

- 문자열(string)은 문자들의 집합을 의미
- 문자열을 저장하는 변수를 문자열 변수라고 함

```
string = "Suan"
print(string)
print(type(string))
```

```
Suan
<class 'str'>
```

리스트 자료형(List Type)

- 리스트(list)는 여러 값을 하나의 변수에 담을 수 있는 자료형
- 대괄호 안에 여러 값을 쉽표를 구분자로 표현하여 리스트 변수 생성
- 간단하게 리스트 생성과 출력을 위해 변수 `l` 안에 값 1, 2, 3을 넣어보자
- 자료형을 출력해보면 'list'라는 결과 값을 가지는 것을 알 수 있음

```
l = [1, 2, 3]
print(l)
print(type(l))
```

```
[1, 2, 3]
<class 'list'>
```

- 리스트 자료형은 같은 자료형이 아니라 다른 자료형을 가지는 값들도 포함하여 구성 가능
- 숫자와 문자열이 포함된 리스트 변수 사용하기

```
l = [1, 'One', 2, 'Two', 3, 'Three']
print(l)
print(type(l))
```

```
[1, 'One', 2, 'Two', 3, 'Three']
<class 'list'>
```

튜플 자료형(Tuple Type)

- 튜플 자료형은 리스트 자료형과 유사하지만 생성, 삭제, 수정이 불가
- 튜플은 괄호 안에 여러 값을 쉼표로 구분하여 사용
- 변수 `t` 안에 값 1, 2, 3을 저장하고, 변수와 변수의 타입을 출력하기

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

```
(1, 2, 3)
<class 'tuple'>
```

범위 자료형(Range Type)

- 불변한 숫자 시퀀스 자료형

```
r = range(10)
print(r)
print(type(r))
```

```
range(0, 10)
<class 'range'>
```

집합 자료형(Set Type)

- 집합 자료형은 중복과 순서가 없다는 특징을 가지며 집합 처리를 쉽게할 수 있음
- 집합은 중괄호를 이용해서 값을 쉼표로 구분하여 넣음
- 변수 `s` 안에 값 2, 3, 1을 넣고 변수 `s`의 값과 타입을 출력한 예제
- 변경 불가능한 집합 자료형으로 fronzenset이 있음

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

fs = frozenset(s)
print(fs)
print(type(fs))
```

```
{1, 2, 3}
<class 'set'>
frozenset({1, 2, 3})
<class 'frozenset'>
```

딕셔너리 자료형(Dictionary Type)

- 딕셔너리 자료형은 다른 자료형과 달리 키(key)와 값(value)를 한 쌍으로 갖는 자료형
- 변수 `d`에 키와 값을 구분자 `:`를 이용해서 쌍으로 묶어서 `1: 'One', 2: 'Two'` 형태로 넣을 수 있음

```
d = {1: 'One', 2: 'Two'}
print(d)
print(type(d))
```

```
{1: 'One', 2: 'Two'}
<class 'dict'>
```

바이트 시퀀스형(Byte Sequence Type)

- 바이트 자료형은 1바이트 단위의 값을 연속적으로 저장하는 시퀀스 자료형

- 1바이트는 8비트에 해당하며, 정수값 0 ~ 255(0x00~0xFF) 사용
- 이진 데이터로 사용되어지거나 1바이트 문자로 고정을 위해 사용
- 유니코드가 아닌 문자열을 사용하는 것과 유사
- 바이트 자료형의 시퀀스 형태로 bytearray 자료형 사용
- 바이트 자료형의 메모리 값의 표현으로 memoryview 자료형 사용

```
byte = b'₩x00'
print(byte)
print(type(byte))

byte = b"Hello"
print(byte)
print(type(byte))

ba = bytearray(byte)
print(ba)
print(type(ba))

mv = memoryview(ba)
print(mv)
print(type(mv))
```

```
b'₩x00'
<class 'bytes'>
b'Hello'
<class 'bytes'>
bytearray(b'Hello')
<class 'bytearray'>
<memory at 0x7fa21661d4c8>
<class 'memoryview'>
```

> 동적 타이핑(dynamic typing)
>
> - 변수의 메모리 공간 확보가 실행 시점에서 발생
> - C나 자바는 int data = 4와 같이 data 변수를 정수형으로 사전 선언
> - 파이썬은 data = 4 형태로 선언하여 data라는 변수의 자료형이 정수(integer) 인지 실수(float)인지를 프로그래머가 아닌 인터프리터가 판단
> - 파이썬 언어가 실행 시점에 동적으로 자료형 결정
> - 다른 언어들과 달리 파이썬은 매우 유연한 언어로, 할당받는 메모리 공간도 저장되는 값의 크기에 따라 동적으로 다르게 할당받을 수 있음

자료형 변환

- 파이썬에는 유용한 자료형을 제공하며, 자료형들간의 변환이 가능

- 자료형 변환에 사용되는 내장 함수(Built-in Function) 모음

함수	설명
book([x])	x를 불리언형으로 변환
int(x[,base])	x를 정수로 변환, x가 문자열일 경우 base 지정
float(x)	x를 실수형(부동소수점 숫자)로 변환
complex(real, [,imag])	복소수 생성
str(x)	객체 x를 문자열 표현으로 변환
repr(x)	객체 x를 표현식 문자열로 변환
eval(str)	문자열을 평가하고 객체를 반환
tuple(s)	s를 튜플로 변환
list(s)	s를 리스트로 변환
set(s)	s를 집합으로 변환
dict(d)	딕셔너리 생성, d는 (키 값) 튜플의 시퀀스이여야 함
frozenset(s)	s를 고정 집합으로 변환
chr(x)	정수를 문자로 변환
ord(x)	단일 문자를 정수 값으로 변환
bin(x)	정수를 2진수 문자열로 변환
oct(x)	정수를 8진수 문자열로 변환
hex(x)	정수를 16진수 문자열로 변환
bytes(x)	자료형을 바이트형으로 변환
bytearray(x)	자료형을 변경가능한 바이트형으로 변환
memoryview(x)	바이트 자료형을 메모리의 이진 데이터로 변환

bool()

- `bool()` 함수는 다양한 자료형을 불리언형으로 변환
- `0`이나 `False` 값이 아닌 경우는 모두 True로 변환

```
print(bool(True))      # 불리언형
print(bool(False))     # 불리언형
print(bool(10))        # 정수형 (10진수)
print(bool(0b010))     # 정수형 (2진수)
print(bool(0o130))     # 정수형 (8진수)
print(bool(0xABC))     # 정수형 (16진수)
print(bool(12.34))     # 실수형 (고정소수점)
print(bool(1234e-2))   # 실수형 (부동소수점)
print(bool('10'))      # 문자형
```

```
True
False
True
True
True
True
True
True
True
```

int()

- `int()` 함수는 다양한 자료형을 정수형으로 변환
- 복소수형은 정수로 `int()` 함수를 통해 변환이 되지 않음
- 불리언 형은 값으로 `True`와 `False`만 가지고 `int()` 함수를 통해 정수로 변환되면 `True`는 1로 `False`는 0으로 변환
- 정수값 10과 2진수 `0b010`, 8진수 `0o130`, 16진수 `0xABC`는 모두 같은 정수형이라서 그대로 값을 유지하고, 10진수 형태로 변환되어 출력
- 실수형을 정수형으로 변환하는 경우에는 고정소수점이든 부동소수점이든지 소수점 이하를 제외하고 변환, 예를 들어, 12.34는 12로 소수점을 제외하고 변환
- 문자형은 기본적으로는 문자열을 10진수 변환, 예를 들어, `int('10')`은 기본 10진수로 문자열 '10'을 정수형 10으로 변환
- 2진수, 8진수, 16진수 등의 다른 진수로 된 문자열일 경우에는 몇 진수에 해당되는지를 입력해주면 그에 맞추어 변환
- `int('010', 2)`와 같이 문자열 '010'을 2진수라고 알려주고 변환하기 위해서 2라고 입력하고, 8진수과 16진수도 마찬가지

```
print(int(True))      # 불리언형
print(int(False))     # 불리언형
print(int(10))        # 정수형 (10진수)
print(int(0b010))     # 정수형 (2진수)
print(int(0o130))     # 정수형 (8진수)
print(int(0xABC))     # 정수형 (16진수)
print(int(12.34))     # 실수형 (고정소수점)
print(int(1234e-2))   # 실수형 (부동소수점)
print(int('10'))      # 문자형
print(int('010', 2))  # 문자형 (2진수)
print(int('130', 8))  # 문자형 (8진수)
print(int('ABC', 16)) # 문자형 (16진수)
```

```
1
0
10
2
88
2748
12
12
10
2
88
2748
```

float()

- `float()` 함수는 자료형을 실수형으로 변환
- 복소수형은 변환이 되지 않고, 문자열 형태의 2진수, 8진수, 16진수도 변환이 되지 않음
- `int()` 함수를 사용한 결과와 유사하지만, 소수점 이하 값이 포함

- 만약 소수점 이하 값이 없다면 .0으로 변환되고, 소수점 이하 값이 있는 실수형은 동일한 값으로 변환

```
print(float(True))    # 불리언형
print(float(False))   # 불리언형
print(float(10))      # 정수형 (10진수)
print(float(0b010))   # 정수형 (2진수)
print(float(0o130))   # 정수형 (8진수)
print(float(0xABC))   # 정수형 (16진수)
print(float(12.34))   # 실수형 (고정소수점)
print(float(1234e-2)) # 실수형 (부동소수점)
print(float('10'))    # 문자형
```

```
1.0
0.0
10.0
2.0
88.0
2748.0
12.34
12.34
10.0
```

complex()

- `complex()` 함수는 자료형을 복소수형으로 변환
- 불리언형, 정수형, 실수형, 복소수형, 문자형 모두 복소수형으로 변환 가능
- 허수가 있는 복소수형을 제외하고 다른 자료형은 허수 부분이 없어서 `0J`로 변환

```
print(complex(True))    # 불리언형
print(complex(False))   # 불리언형
print(complex(10))      # 정수형 (10진수)
print(complex(0b010))   # 정수형 (2진수)
print(complex(0o130))   # 정수형 (8진수)
print(complex(0xABC))   # 정수형 (16진수)
print(complex(12.34))   # 실수형 (고정소수점)
print(complex(1234e-2)) # 실수형 (부동소수점)
print(complex(1 + 23j)) # 복소수형
print(complex('10'))    # 문자형
```

```
(1+0j)
0j
(10+0j)
(2+0j)
(88+0j)
(2748+0j)
(12.34+0j)
(12.34+0j)
(1+23j)
(10+0j)
```

str()

- `str()` 함수는 파이썬의 모든 자료형을 문자열로 변환
- 자료형의 값을 그대로 문자열 형태로 변환

> `str()`와 `repr()` 비교
>
> - `str()` 함수와 유사하게 `repr()` 함수가 존재
> - `str()` 함수는 내부적으로 `__str__` 메소드를 호출
> - `__str__`의 경우에는 객체의 비공식적인 문자열 출력에 사용되며 사용자가 보기 쉬운 형태로 보여줄때 사용
> - `repr()` 함수는 내부적으로 `__repr__` 메소드를 호출
> - `__repr__`은 공식적인 문자열을 출력할때 사용되며, 주로 해당 객체를 인식할 수 있는 공식적인 문자열을 나타낼 때 사용

```
print(str(True))      # 불리언형
print(str(False))     # 불리언형
print(str(10))        # 정수형 (10진수)
print(str(0b010))     # 정수형 (2진수)
print(str(0o130))     # 정수형 (8진수)
print(str(0xABC))     # 정수형 (16진수)
print(str(12.34))     # 실수형 (고정소수점)
print(str(1234e-2))   # 실수형 (부동소수점)
print(str(1 + 23j))   # 복소수형
print(str('10'))      # 문자형
print(str([1, 2, 3])) # 리스트
print(str((1, 2, 3))) # 튜플
print(str({2, 3, 1})) # 집합
print(str({1: 'One', 2: 'Two'}))  # 딕셔너리
```

```
True
False
10
2
88
2748
12.34
12.34
(1+23j)
10
[1, 2, 3]
(1, 2, 3)
{1, 2, 3}
{1: 'One', 2: 'Two'}
```

eval()

- `eval()` 함수는 문자열에 포함된 수식을 계산하여 나온 결과를 반환
- '1 + 2'라는 문자열에 대해서 그대로 출력하는 것과 달리 계산된 결과 값은 3을 출력
- 그 밖에도 -, *, /와 같은 여러 수식들에 대해서도 계산 가능

```
print('1 + 2')
print(eval('1 + 2'))
print(eval('10 + 20 - 30'))
print(eval('10 * 10'))
print(eval('100 / 10'))
```

```
1 + 2
3
0
100
10.0
```

tuple()

- `tuple()` 함수는 문자열, 리스트, 튜플 등의 자료형을 튜플 자료형으로 변환
- 문자열 `'Suan'`을 문자 단위로 구분하여 튜플 자료형으로 변환
- 리스트 자료형 `[1, 2, 3]`이나 튜플 자료형 `(1, 2, 3)`도 튜플 자료형으로 `(1, 2, 3)`으로 변환

```
print(tuple('Suan'))
print(tuple([1, 2, 3]))
print(tuple((1, 2, 3)))
```

```
('S', 'u', 'a', 'n')
(1, 2, 3)
(1, 2, 3)
```

list()

- `list()` 함수는 문자열, 리스트, 튜플 등의 자료형을 리스트 자료형으로 변환
- 문자열 `'Suan'`, 리스트 `[1, 2, 3]`, 튜플 `(1, 2, 3)`에 대해서 리스트 자료형으로 변환

```
print(list('Suan'))
print(list([1, 2, 3]))
print(list((1, 2, 3)))
```

```
['S', 'u', 'a', 'n']
[1, 2, 3]
[1, 2, 3]
```

set()

- `set()` 함수는 문자열, 리스트, 튜플, 집합, 딕셔너리 자료형을 집합 자료형으로 변환
- 집합의 특성답게 요소에 대해서 순서를 고려하지 않고 변환
- 리스트, 튜플, 집합 자료형도 집합 형태로 변환이 되며, 딕셔너리 자료형도 집합으로 변환
- 딕셔너리 자료형은 키와 값으로 구성된 구조에서 키 값에 해당하는 것만 집합으로 변환

```
print(set('Suan'))
print(set([1, 2, 3]))
print(set((1, 2, 3)))
print(set({1, 2, 3}))
print(set({1:'One', 2:'Two'}))
```

```
{'n', 'a', 'u', 'S'}
{1, 2, 3}
{1, 2, 3}
{1, 2, 3}
{1, 2}
```

frozenset()

- `frozenset()` 함수는 문자열을 고정 집합 형태로 변환
- `set()` 함수는 집합의 값을 추가 및 제거 등이 가능
- `frozenset()` 함수는 수정 불가능한 집합 자료형

```
print(frozenset('Suan'))
print(frozenset([1, 2, 3]))
```

```
frozenset({'n', 'a', 'u', 'S'})
frozenset({1, 2, 3})
```

dict()

- `dict()` 함수는 딕셔너리 자료형으로 변환하는 역할
- (키: 값)으로 구성된 튜플의 시퀀스 형태로 값을 가짐

```
print(dict({1:'One', 2:'Two'}))
print(dict({'One':1, 'Two':2}))
```

```
{1: 'One', 2: 'Two'}
{'One': 1, 'Two': 2}
```

chr()

- `chr()` 함수는 정수를 문자로 변환
- 컴퓨터에서는 문자를 처리하기 위해 각 문자마다 특정 정수값으로 매핑: ASCII(American Standard Code for Information Interchange)

DEC	HEX	Char	DEC	HEX	Char	DEC	HEX	Char	DEC	HEX	Char
0	0x00	NULL	32	0x20	Space	64	0x40	@	96	0x70	`
1	0x01	Start of header	33	0x21	!	65	0x41	A	97	0x71	a
2	0x02	Start of text	34	0x22	"	66	0x42	B	98	0x72	b
3	0x03	End of text	35	0x23	#	67	0x43	C	99	0x73	c
4	0x04	End of transmission	36	0x24	$	68	0x44	D	100	0x74	d
5	0x05	Enquiry	37	0x25	%	69	0x45	E	101	0x75	e
6	0x06	Acknowledge	38	0x26	&	70	0x46	F	102	0x76	f
7	0x07	Bell	39	0x27	'	71	0x47	G	103	0x77	g
8	0x08	Backspace	40	0x28	(	72	0x48	H	104	0x78	h
9	0x09	Horizontal tab	41	0x29	)	73	0x49	I	105	0x79	i
10	0x0A	Line feed	42	0x2A	*	74	0x4A	J	106	0x7A	j
11	0x0B	Vertical tab	43	0x2B	+	75	0x4B	K	107	0x7B	k
12	0x0C	Form feed	44	0x2C	,	76	0x4C	L	108	0x7C	l
13	0x0D	Carriage return	45	0x2D	-	77	0x4D	M	109	0x7D	m
14	0x0E	Shift out	46	0x2E	.	78	0x4E	N	110	0x7E	n
15	0x0F	Shift in	47	0x2F	/	79	0x4F	O	111	0x7F	o
16	0x10	Data link escape	48	0x30	0	80	0x50	P	112	0x80	p
17	0x11	Device control 1	49	0x31	1	81	0x51	Q	113	0x81	q
18	0x12	Device control 2	50	0x32	2	82	0x52	R	114	0x82	r
19	0x13	Device control 3	51	0x33	3	83	0x53	S	115	0x83	s
20	0x14	Device control 4	52	0x34	4	84	0x54	T	116	0x84	t
21	0x15	Negative Ack.	53	0x35	5	85	0x55	U	117	0x85	u
22	0x16	Synchronous idle	54	0x36	6	86	0x56	V	118	0x86	v
23	0x17	End of trans. Block	55	0x37	7	87	0x57	W	119	0x87	w
24	0x18	Cancel	56	0x38	8	88	0x58	X	120	0x88	x
25	0x19	End of medium	57	0x39	9	89	0x59	Y	121	0x89	y
26	0x1A	Substitute	58	0x3A	:	90	0x5A	Z	122	0x9A	z
27	0x1B	Escape	59	0x3B	;	91	0x5B	[	123	0x9B	{
28	0x1C	File Separator	60	0x3C	<	92	0x6C	₩	124	0x9C	|
29	0x1D	Group Separator	61	0x3D	=	93	0x6D	]	125	0x9D	}
30	0x1E	Record Separator	62	0x3E	>	94	0x6E	^	126	0x9E	~
31	0x1F	Unit Separator	63	0x3F	?	95	0x6F	_	127	0x9F	Del

- `chr()` 함수는 ASCII 기준에 따라서 정수값에 해당하는 문자를 나타냄
- 예를 들어, ASCII에서 정수 값 97은 문자 a에 매핑되어 `chr(97)`의 결과 값은 문자 a

```
print(chr(97))
print(chr(65))
print(chr(122))
print(chr(90))
```

```
a
A
z
Z
```

ord()

- `ord()` 함수는 `chr()` 함수와 반대로 문자를 정수 값으로 변환

- ASCII에 맞춰서 문자 'a'에 해당하는 정수 값인 97을 출력하기

```
print(ord('a'))
print(ord('A'))
print(ord('z'))
print(ord('Z'))
```

```
97
65
122
90
```

bin()

- `bin()` 함수는 정수를 2진수 문자열로 변환
- 자료형을 2진수 표기인 '0b'가 붙은 형태의 문자열로 변환
- `bin()` 함수로 `True`는 `0b1`로 출력하고, `False`는 `0b0`으로 출력하기
- 10진수, 2진수, 8진수, 16진수 정수형도 2진수로 변환되어 출력

```
print(bin(True))      # 불리언형
print(bin(False))     # 불리언형
print(bin(10))        # 정수형 (10진수)
print(bin(0b010))     # 정수형 (2진수)
print(bin(0o130))     # 정수형 (8진수)
print(bin(0xABC))     # 정수형 (16진수)
```

```
0b1
0b0
0b1010
0b10
0b1011000
0b101010111100
```

oct()

- `oct()` 함수는 정수를 8진수 문자열로 변환
- 불리언형, 정수형 등의 자료형을 8진수 표기인 '0o'가 붙은 형태의 문자열로 변환
- `oct()` 함수를 이용해 `True`는 `0o1`로 출력되고, `False`는 `0o0`으로 출력하기
- 10진수, 2진수, 8진수, 16진수 정수형도 8진수로 변환되어 출력

```
print(oct(True))      # 불리언형
print(oct(False))     # 불리언형
print(oct(10))        # 정수형 (10진수)
print(oct(0b010))     # 정수형 (2진수)
print(oct(0o130))     # 정수형 (8진수)
print(oct(0xABC))     # 정수형 (16진수)
```

```
0o1
0o0
0o12
```

```
0o2
0o130
0o5274
```

hex()

- `hex()` 함수는 정수를 16진수 문자열로 변환
- 주어진 자료형을 16진수 표기인 '0x'가 붙은 16진수 형태의 문자열로 변환
- 불리언형과 정수형 모두 16진수 표기의 문자열로 변환하기

```
print(hex(True))      # 불리언형
print(hex(False))     # 불리언형
print(hex(10))        # 정수형 (10진수)
print(hex(0b010))     # 정수형 (2진수)
print(hex(0o130))     # 정수형 (8진수)
print(hex(0xABC))     # 정수형 (16진수)
```

```
0x1
0x0
0xa
0x2
0x58
0xabc
```

bytes()

- `bytes()` 함수는 자료형을 바이트형으로 변환
- 객체를 바이트 객체로 변환

```
print(bytes(True))      # 불리언형
print(bytes(False))     # 불리언형
print(bytes(10))        # 정수형
print(bytes(b"Hello"))  # 문자열형
print(bytes([1, 2, 3])) # 리스트형
```

```
b'₩x00'
b''
b'₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00'
b'Hello'
b'₩x01₩x02₩x03'
```

bytearray()

- `bytearray()` 함수는 자료형을 변경가능한 바이트형으로 변환

```
ba = bytearray(b'hello')
print(ba)
ba[0] = ord('H')
print(ba)
print(type(ba))
```

```
ba = bytearray([1, 2, 3])
print(ba)
ba[0] = 4
print(ba)
print(type(ba))
```

```
bytearray(b'hello')
bytearray(b'Hello')
<class 'bytearray'>
bytearray(b'₩x01₩x02₩x03')
bytearray(b'₩x04₩x02₩x03')
<class 'bytearray'>
```

memoryview()

- 바이트 자료형을 메모리의 이진 데이터로 변환한 자료형

```
ba = bytearray(b'hello')
mv = memoryview(ba)
ba[0] = 72
print(ba)
print(mv)
print(type(mv))
```

```
bytearray(b'Hello')
<memory at 0x7fa21661d588>
<class 'memoryview'>
```

자료형 계산

- 자료형들간의 계산이 가능하도록 몇 가지 유용한 내장 함수(Built-in Function) 제공

함수	설명
`min(iterable)`	두개 이상의 값에서 가장 작은 값을 반환
`max(iterable)`	두개 이상의 값에서 가장 큰 값을 반환
`sum(iterable)`	값들의 합을 반환
`divmod(a, b)`	`a`를 `b`로 나눈 값과 나머지를 쌍으로 반환
`abs(x)`	`x`의 절대값을 반환
`pow(a, b)`	`a`의 `b`승의 값을 반환
`len(s)`	시퀀스(문자열, 바이트, 튜플, 리스트 등)의 갯수를 반환
`round(x)`	소수점 뒤를 반올림한 값을 반환
`all(iterable)`	모든 값이 `True` 이거나 비어있을 때 `True` 반환
`any(iterable)`	어떤 값이 `True` 이면 `True` 반환, 값이 비어있을 때 `False` 반환

min()

- 입력된 값들 중에서 가장 작은 값을 반환하는 함수

```
print(min(1, 2, 3))
print(min([3, 4, 5]))
```

```
1
3
```

max()

- 입력된 값들 중에서 가장 큰 값을 반환하는 함수

```
print(max(1, 2, 3))
print(max([3, 4, 5]))
```

```
3
5
```

sum()

- 입력된 값들의 전체 합을 반환

```
print(sum((1, 2, 3)))
print(sum([3, 4, 5]))
```

```
6
12
```

divmod()

- 나눈 값과 나머지 값을 쌍으로 반환
- `(a // b, a % b)`와 동일한 값

```
print(divmod(3, 5))
print(divmod(10, 5))
print((10 // 5, 10 % 5))
```

```
(0, 3)
(2, 0)
(2, 0)
```

abs()

- 절대값 $|x|$으로 반환

```
print(abs(-4))
print(abs(5))
print(abs(-10))
```

```
4
```

```
5
10
```

pow()

- 지수승의 값 a^b으로 반환

```
print(pow(2, 3))
print(pow(10, 4))
```

```
8
10000
```

len(s)

- 다양한 시퀀스의 갯수를 반환

```
print(len("String"))
print(len((1, 2, 3)))
print(len([4, 5]))
```

```
6
3
2
```

round()

- 소수점 뒤를 반올림한 값을 반환

```
print(round(3.14))
print(round(3.141527, 2))
print(round(0.6))
print(round(0.4))
```

```
3
3.14
1
0
```

all()

- 모든 값이 `True` 이거나 비어있을 때 `True` 반환

```
print(all((1, 2, 3)))
print(all(()))
print(all((False, True, False)))
print(all((False, False, False)))
```

```
True
True
False
False
```

any()

- 어떤 값이 `True` 이면 `True` 반환
- 값이 비어있으면 `False` 반환

```
print(any((1, 2, 3)))
print(any(()))
print(any((False, True, False)))
print(any((False, False, False)))
```

```
True
False
True
False
```

연산자(Operators)

- 피연산자의 계산을 위해 다양한 연산자들이 존재
- 파이썬에서 제공하는 연산자들의 종류
  - 산술 연산자(Arithmetic Operators)
  - 비교 연산자(Comparison Operators)
  - 할당 연산자(Assignment Operators)
  - 비트 연산자(Bitwise Operators)
  - 논리 연산자(Logical Operators)
  - 멤버 연산자(Membership Operators)
  - 식별 연산자(Identity Operators)

산술 연산자(Arithmetic Operators)

연산자	설명	예제
+	덧셈	`c = a + b`
-	뺄셈	`c = a - b`
*	곱셈	`c = a * b`
/	나눗셈	`c = a / b`
%	나머지	`c = a % b`
**	제곱	`c = a ** b`
//	몫	`c = a // b`

```
a = 6
b = 4
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a % b)
print(a ** b)
print(a // b)
```

```
10
2
24
1.5
2
1296
1
```

비교 연산자(Comparison Operators)

연산자	설명	예제
==	두 피연산자의 값이 동일하면 True	`a == b`
!=	두 피연산자의 값이 다르면 True	`a != b`
>	왼쪽 피연산자가 오른쪽 피연산자보다 크면 True	`a > b`
<	왼쪽 피연산자가 오른쪽 피연산자보다 작으면 True	`a < b`
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같으면 True	`a >= b`
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같으면 True	`a <= b`

```
a, b = 6, 4
print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

```
False
True
True
False
True
False
```

할당 연산자(Assignment Operators)

연산자	설명	예제
=	오른쪽 피연산자의 값을 왼쪽 피연산자에 할당	`a, b = 20, 12`
+=	왼쪽 피연산자의 값을 오른쪽 피연산자에 더하고 그 결과를 왼쪽 피연산자에 대입	`a += b`
-=	왼쪽 피연산자의 값을 오른쪽 피연산자에 더하고 그 결과를 왼쪽 피연산자에 대입	`a -= b`

연산자	설명	예제
*=	왼쪽 피연산자의 값을 오른쪽 피연산자에 곱하고 그 결과를 왼쪽 피연산자에 대입	`a *= b`
/=	왼쪽 피연산자의 값을 오른쪽 피연산자에 나누고 그 결과를 왼쪽 피연산자에 대입	`a /= b`
%=	왼쪽 피연산자의 값을 오른쪽 피연산자에 나눈 나머지 값을 왼쪽 피연산자에 대입	`a %= b`
**=	왼쪽 피연산자의 값을 오른쪽 피연산자에 제곱한값을 왼쪽 피연산자에 대입	`a **= b`
//=	왼쪽 피연산자의 값을 오른쪽 피연산자에 나눈 몫을 왼쪽 피연산자에 대입	`a //= b`

```
a, b = 6, 4
a += b
print(a)
a -= b
print(a)
a *= b
print(a)
a /= b
print(a)
a %= b
print(a)
a **= b
print(a)
a //= b
print(a)
```

```
10
6
24
6.0
2.0
16.0
4.0
```

비트 연산자(Bitwise Operators)

연산자	설명	예제
&	두 피연산자의 비트를 AND 연산	`a & b`
|	두 피연산자의 비트를 OR 연산	`a | b`
^	두 피연산자의 비트를 XOR 연산	`a ^ b`
~	피연산자의 비트를 NOT 연산	`a ~ b`
<<	왼쪽 피연산자의 비트를 오른쪽 피연산자의 비트만큼 left shift 연산	`a << b`
>>	왼쪽 피연산자의 비트를 오른쪽 피연산자의 비트만큼 right shift 연산	`a >> b`

```
a, b = 6, 4
print(bin(a))
print(bin(b))
print(a & b, bin(a & b))
print(a | b, bin(a | b))
print(a ^ b, bin(a ^ b))
print(~a, bin(~a))
print(a << b, bin(a << b))
print(a >> b, bin(a >> b))
```

```
0b110
0b100
4 0b100
6 0b110
2 0b10
-7 -0b111
96 0b1100000
0 0b0
```

논리 연산자(Logical Operators)

연산자	설명	예제
and	두 피연산자가 모두 True이면 True	`True and True`
or	두 피연산자 중 하나라도 True이면 True	`True or False`
not	피연산자가 True이면 False, False이면 True	`not False`

```
print(True and True)
print(True and False)
print(True or False)
print(not True)
print(not False)
print(not (True and False))
```

```
True
False
True
False
True
True
```

멤버 연산자(Membership Operators)

연산자	설명	예제
in	멤버로 포함되어 있으면 True, 포함되어 있지 않으면 False	`a in l`
not in	멤버로 포함되어 있지 않으면 True, 포함되어 있으면 False	`a not in l`

```
a, b = 6, 4
l = [2, 4, 8]
print(a in l)
print(b in l)
print(a not in l)
print(b not in l)
```

```
False
True
True
False
```

식별 연산자(Identity Operators)

연산자	설명	예제
is	피연산자가 동일한 객체를 가리키면 True, 동일한 객체를 가리키고 있지 않으면 False	`a is b`
is not	피연산자가 동일한 객체를 가리키고 있지 않으면 True, 동일한 객체를 가리키면 False	`a is not b`

```
a, b =  6, 4
print(a is b)
print(a is not b)
a, b = 5, 5
print(a is b)
print(a is not b)
```

```
False
True
True
False
```

연산자 우선순위(Operators Precedence)

- 여러 연산자들이 사용되면 어떤 연산자들이 우선되는지를 결정하기 위해 연산자의 우선순위가 존재
- 연산자 중에서 괄호가 가장 높은 연산 순위를 가지며, 논리 연산자가 가장 낮은 연산 순위를 가짐

연산자	설명
()	괄호
**	지수(승수)
~, +, -	보수, 단항 덧셈과 뺄셈
*, /, %, //	곱셈, 나눗셈, 나머지, 몫
+, -	덧셈과 뺄셈
>>, <<	좌우 비트 시프트
&	비트 AND
^, |	비트 XOR, 비트 OR
<=, <>, >=	비교 연산자
==, !=	동등 연산자
=, %=, /=, //=, -=, +=, =, *=	할당 연산자
is, is not	식별 연산자
in, not in	멤버 연산자
not, or, and	논리 연산자