2. Python : 기초 자료형 (Data Type)

자료형이란 프로그래밍을 할 때 쓰이는 숫자, 문자열 등 자료 형태로 사용하는 모든 것을 뜻한다.

프로그램의 기본이자 핵심 단위가 바로 자료형이다. 파이썬의 기초 자료형에 대해서 익혀보자.

Table of Contents

1. 주석

주석은 주로 소스 코드를 읽는 사람들을 위해 설명을 남기는 용도로 빈번하게 사용된다.

1) # 문자 뒤에 따라오는 짧은 문장

In [3]:
```
print("Hello, world")  # Note that print is a statement
```

```
Hello, world
```

2) 따옴표 세 개로 문장을 감싸기(""" """ or ''' ''')

In [4]:
```
"""
Note that print is a statement
"""
print("Hello World")
```

```
Hello World
```

2. 숫자형 자료형 (Numeric)

- 숫자형 (Numeric) 이란 숫자 형태로 이루어진 자료형으로, 형태에 따라서 1, 2, 3 과 같은 정수(interger), 12.34 와 같은 실수(float), 드물게 사용하는 복소수(complex)로 구분하여 생각할 수 있다.
- 이들을 활용한 다양한 연산이 가능하다.
- type(변수명) 함수는 type을 출력하는 함수로, 해당 변수의 자료형을 확인할 수 있다.

정수형 (Integer)

- 정수형 (Integer) 이란 말 그대로 1, 2, 3 과 같은 정수를 뜻하는 자료형을 말한다.

In [7]:
```
x = 123
x
```

Out[7]:
```
123
```

In [8]:
```
type(x)
```

Out[8]:
```
int
```

In [ ]:
```
a=0 ; type(a)
```

In [ ]:
```
b= -178 ; type(b)
```

In [14]:
```
'''
8진수와 16진수
드물게 사용하는 8진수와 16진수 역시 만들수 있다. 잘 사용하지 않는 숫자 자료형이니 눈으로 간단히 익히자
'''
a = 0o177   #8진수 : 8진수(Octal)를 만들기 위해서는 숫자가 0o (숫자 0 + 알파벳 소문자 o) 또는 0O(숫자 0 + 대문자 O)로 시작
```

```
b = 0x8ff   #16진수 :16진수(Hexadecimal)를 만들기 위해서는 0x로 시작
```

In [15]:
```
a   # (7*1)+(7*8)+(1*64)
```
Out[15]: 127

In [16]:
```
b   # (15*1) +(15*16) +(8*256)
```
Out[16]: 2303

In [17]:
```
type(a)
```
Out[17]: int

실수형 (Float)

- 파이썬에서 실수형 (Floating-point) 은 12.34 와 같은 소수점이 포함된 숫자를 말한다.
- 실수형은 ._E 표기법을 활용한 scientific notation으로도 표현 가능하다.
  - e.g) 52.3E-4 : E 표기법은 E뒤의 값이 10의 지수임을 나타냄.
    즉, 52.3E-4 는 $52.3 * 10^{-4}$ 을 의미

In [19]:
```
x=3.2
x
```
Out[19]: 3.2

In [20]:
```
type(x)
```
Out[20]: float

In [21]:
```
a = 3.4e-23   # 3.4*10^-23
b = 3.4E-23   # 위와 동일
print(a,  b)
```
```
3.4e-23 3.4e-23
```

In [22]:
```
type(b)
```
Out[22]: float

복소수형 (complex)

1) complex 함수를 활용

In [23]:
```
x = complex(1, 3)
x
```
Out[23]: (1+3j)

In [24]:
```
type(x)
```

Out[24]: complex

2) j 를 활용

In [25]:
```
x = 3-4j
x
```

Out[25]: (3-4j)

숫자형을 활용한 연산자

- 연산자

Operation	Description
x+y	sum of x and y
x-y	difference of x and y
x* y	product of x and y
x/y	quotient of x and y
x//y	floored quotient of x and y
x%y	remainder of x/y
pow(x,y)	x to the power y
x** y	x to the power y
-x	x negated
+x	x unchanged
abs(x)	absolute value or magnitude of x
int(x)	x converted to interger
float(x)	x converted to floating point
complex(re,im)	a complex number with real part re, imaginary part im. im defaults to zero
c.conjugate()	conjugate of the complex number c
divmod(x,y)	the pair (x//y, x%y)

In [54]:
```
# ex1) x 의 y 제곱을 나타내는 ** 연산자
a=2
b=3
a**b
```

Out[54]: 8

In [26]:
```
# ex2) 나눗셈 후 나머지를 반환하는 % 연산자
7%2
```

Out[26]: 1

In [28]:
```
# ex3) 나눗셈 후 몫을 반환하는 // 연산자
7//2
```

Out[28]: 3

- x = -23, y=3 일때 다음의 연산자의 결과를 예측해 보자

In [37]:
```
x=-23
y=3
```

In [38]:
```
x
```

Out[38]: -23

In [ ]:
```
+x
```

In [40]:
```
abs(x)
```

Out[40]: 23

In [42]:
```
float(x)
```

Out[42]: -23.0

In [43]:
```
divmod(-x,y)   # 몫과 나머지
```

Out[43]: (7, 2)

In [44]:
```
pow(-x,y)
```

Out[44]: 12167

In [45]:
```
c= complex(x,y)
c
```

Out[45]: (-23+3j)

In [46]:
```
c.conjugate()   # 켤레 복소수
```

Out[46]: (-23-3j)

3. 문자열 자료형 (String)

- 문자열(String)이란 문자, 단어 등으로 구성된 문자들의 집합을 의미한다. 예를 들어 아래의 예와 같은 것들이 문자열이다.
- 문자열 예문을 보면 모두 큰따옴표(" ")로 둘러싸여 있다. "123은 숫자인데 왜 문자열이지?"라는 의문이 들수도 있으나 따옴표로 둘러싸여 있으면 모두 문자열 자료형이라고 보면 된다.

"Life is too short, You need Python"

"ab"

"123"

문자형 자료 만들기

- 파이썬에서 문자열 자료형을 만드는 방법은 총 4 가지이다.

1) 큰 따옴표(") 로 양쪽 둘러싸기

```
In [47]: x = "this is also a string"
         x
Out[47]: 'this is also a string'
```

2) 작은따옴표(')로 양쪽 둘러싸기

```
In [48]: 'it is a string'
Out[48]: 'it is a string'
```

3) 큰따옴표 3개를 연속(""")으로 써서 양쪽 둘러싸기

```
In [49]: """this is a string"""
Out[49]: 'this is a string'
```

4) 작은따옴표 3개를 연속(''')으로 써서 양쪽 둘러싸기

```
In [50]: '''this is also a string'''
Out[50]: 'this is also a string'
```

문자열 안에 따옴표를 포함시키고 싶을 때

- 문자열을 만들어 주는 주인공은 작은따옴표(')와 큰따옴표(")이다. 그런데 문자열 안에도 작은 따옴표와 큰따옴표가 들어 있어야 할 경우가 있다. 이때는 좀 더 특별한 기술이 필요하다.
- "this is also a pyton" string" --> 이런 형태의 문자열은 python 이 인식하지 못한다.

```
In [59]: "this is also a pyton's string" # 작은 따옴표(') 를 문자열 안에 사용하고자 할때는 큰 따옴표("")로 둘러싼다
Out[59]: "this is also a pyton's string"
```

```
In [58]: '"Python is very easy." he says.' # 큰 따옴표(") 를 문자열 안에 사용하고자 할때는 작은 따옴표(')로 둘러싼다
Out[58]: '"Python is very easy." he says.'
```

In [57]:
```
'this is also a pyton\'s string' # 백슬래시(\)를 사용해서 작은따옴표(')와 큰따옴표(")를 문자열에 포함시키기
```

Out[57]:
```
"this is also a pyton's string"
```

In [56]:
```
"\"Python is very easy.\" he says." # 백슬래시(\)를 사용해서 작은따옴표(')와 큰따옴표(")를 문자열에 포함시키기
```

Out[56]:
```
'"Python is very easy." he says.'
```

문자열 안에서 줄바꿈을 하고 싶을 때

표현하고 싶은 문자열이 항상 한 줄짜리만 있는 것은 아니다. 여러 줄의 문자열을 변수에 대입하려면 어떻게 처리해야 할까?

Hello world!

Nice to meet you!

(1) 줄 바꿈을 위한 이스케이프 코드 \n 삽입하기 (백슬래시(\) 활용)

In [62]:
```
s = "Hello World! \nNice to meet you!"
print(s)
```

```
Hello World!
Nice to meet you!
```

(2) 줄 바꿈을 위해 연속된 작은따옴표 3개(''') 또는 큰따옴표 3개(""") 사용하기

In [ ]:
```
# error code
s = "Hello World!
Nice to meet you!"
s
```

In [64]:
```
s = """Hello World!
Nice to meet you!"""
s
```

Out[64]:
```
'Hello World!\nNice to meet you!'
```

In [65]:
```
print(s)
```

```
Hello World!
Nice to meet you!
```

Tip) 이스케이프 코드

- 이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"이다. 주로 출력물을 보기 좋게 정렬하는 용도로 사용한다.
- 자주 사용하는 이스케이프 코드를 정리하면 다음과 같다.

코드	설 명
\n	문자열 안에서 줄을 바꿀 때 사용
\t	문자열 사이에 탭 간격을 줄 때 사용
\\	문자 \를 그대로 표현할 때 사용
\'	작은따옴표(')를 그대로 표현할 때 사용
\"	큰따옴표(")를 그대로 표현할 때 사용

In [66]:
```
s = "Hello World! \tNice to meet you!"
print(s)
```

```
Hello World!    Nice to meet you!
```

In [ ]:
```
#다음은 어떻게 출력 될지 생각해보자.
s = "Hello World! \
Nice to meet you!."
print(s)
```

문자열 연산하기

- 파이썬에서는 문자열을 더하거나 곱할 수 있다.

1) 문자열 합치기 (Concatenation)

In [83]:
```
a= "R"
b= " & "
c= "Python"
a+b+c
```

Out[83]:
```
'R & Python'
```

2) 문자열 반복

In [1]:
```
a="Python is fun!"
a*2
```

Out[1]:
```
'Python is fun!Python is fun!'
```

문자열 인덱싱과 슬라이싱

- 인덱싱(Indexing)이란 무엇인가를 "가리킨다"는 의미이고, 슬라이싱(Slicing)은 무엇인가를 "잘라낸다"는 의미이다.
- 인덱싱 기법과 슬라이싱 기법은 뒤에서 배울 자료형인 리스트나 튜플에서도 사용할 수 있다.

인덱싱 (Indexing)

- 인덱싱은 문자열의 위치에 번호를 매겨 출력한 것이다.
- 주의할 점은 파이썬은 0 부터 번호를 매긴다는 점이다.
- 아래의 예제에서 x[번호]는 문자열 안의 특정한 값을 뽑아내는 역할을 하며, 이러한 작업을 인덱싱이라고 한다.

```
In [86]: x = "Life is too short, You need Python"
```

```
In [87]: x[0] #0번째 값 출력
Out[87]: 'L'
```

```
In [88]: x[4] #4번째 값 출력
Out[88]: ' '
```

```
In [89]: x[-1] #뒤에서 1번째 값 출력
Out[89]: 'n'
```

```
In [90]: x[-2]
Out[90]: 'o'
```

슬라이싱 (Slicing)

- "Life is too short, You need Python" 문자열에서 단순히 한 문자만을 뽑아내는 것이 아니라 'Life' 또는 'You' 같은 단어를 뽑아내는 방법이 슬라이싱이다.
- 아래는 "Life" 단어를 뽑아내는 다양한 슬라이싱 방법을 보여주고 있다

```
In [134]: a = "Life is too short, You need Python"
          a[0:4]
Out[134]: 'Life'
```

```
In [136]: a[0] + a[1] + a[2]+ a[3] + a[4]
Out[136]: 'Life '
```

```
In [126]: x[0:4] #0번째에서 3번째 값까지 출력 (0 <= x < 4)
Out[126]: 'Life'
```

```
In [127]: x[:4]  #처음부터 3번째 값까지 출력
Out[127]: 'Life'
```

```
In [133]: x[-6:] #뒤에서 6번째 값부터 마지막까지 출력
Out[133]: 'Python'
```

```
In [91]: x[19:-7] #19번째 부터 a[-8]번째 값까지 출력
```

```
Out[91]: 'You need'
```

```
In [138]: x[:] #문자열의 처음부터 끝까지 출력
```

```
Out[138]: 'Life is too short, You need Python'
```

문자열 포맷팅 (formatting)

- 문자열에서 특정 값을 변경하려면 포맷팅 (formatting)을 사용한다.
- 아래 예제와 같이 문자열 안의 특정한 값 만을 바꿔야 할 경우가 있을 때 이것을 가능하게 해주는 것이 바로 문자열 포매팅 기법이다.

"Current temperature is 20 Celsius"

"Current temperature is 23 Celsius"

"Current temperature is 18 Celsius"

- **문자열 포맷 코드**

Operation	Description
%d	정수(Integer)
%f	부동소수(floating-point)
%o	8진수
%x	16진수
%c	문자 1개(character)
%s	문자열(String)
%%	Literal % (문자 % 자체)

숫자 대입

```
In [6]: "Current temperature is %d Celsius" % 20
```

```
Out[6]: 'Current temperature is 20 Celsius'
```

```
In [7]: #8진수
        "Current temperature is %o Celsius" % 20
```

```
Out[7]: 'Current temperature is 24 Celsius'
```

```
In [8]: #16진수
        "Current temperature is %x Celsius" % 20
```

```
Out[8]: 'Current temperature is 14 Celsius'
```

```
In [163]: "Current temperature is %d Celsius" % 20.3
```

```
Out[163]: 'Current temperature is 20 Celsius'
```

```
In [9]: #소수점 4째자리 까지의 소수
        "Current temperature is %.4f Celsius" % 20.125456325
Out[9]: 'Current temperature is 20.1255 Celsius'
```

```
In [10]: "Current temperature is %.f Celsius" % 20.125456325
Out[10]: 'Current temperature is 20 Celsius'
```

```
In [14]: #문자 % 자체를 사용
         "Error is %d%%." % 98
Out[14]: 'Error is 98%.'
```

문자열 대입

```
In [159]: "I am %c." % "H"
Out[159]: 'I am H.'
```

```
In [160]: "I am %s." % "SJ"
Out[160]: 'I am SJ.'
```

```
In [161]: "I am %s." % "H"
Out[161]: 'I am H.'
```

```
In [ ]: "I am %c." % "SJ"
```

2개 이상의 값 대입

```
In [165]: "This is %d, and that is %d" % (10,15)
Out[165]: 'This is 10, and that is 15'
```

```
In [12]: number = 15
         day = "two"
         "I ate %d apples. so I was sick for %s days." % (number, day)
Out[12]: 'I ate 15 apples. so I was sick for two days.'
```

format 함수를 사용한 포맷팅

- format 함수를 사용한 포맷팅도 가능하다. 앞에서 살펴본 문자열 포매팅 예제를 format 함수를 사용해서 바꾸면 다음과 같다.

1) 숫자 바로 대입

In [19]:
```
# format() 함수 사용
"Current temperature is {0} Celsius".format(20)
```

Out[19]:
```
'Current temperature is 20 Celsius'
```

In [20]:
```
#f문자열 포맷팅 : 문자열 앞에 f를 쓰고 문자열에서 변수를 사용하려는 곳에 {변수명}을 쓰면 변수 값을 사용
temperature = 20
f"Current temperature is {temperature} Celsius"
```

Out[20]:
```
'Current temperature is 20 Celsius'
```

2) 문자열 바로 대입

In [21]:
```
# format() 함수 사용
"I am {0}".format("SJ")
```

Out[21]:
```
'I am SJ'
```

In [22]:
```
# f문자열 포맷팅 : 문자열 앞에 f를 쓰고 문자열에서 변수를 사용하려는 곳에 {변수명}을 쓰면 변수 값을 사용
name="SJ"
f"I am {name}"
```

Out[22]:
```
'I am SJ'
```

3) 2개 이상의 값 대입

In [25]:
```
#인덱스 위치 사용
number = 10
day = "two"
"I ate {0} apples. so I was sick for {1} days.".format(number, day)
```

Out[25]:
```
'I ate 10 apples. so I was sick for two days.'
```

In [24]:
```
#인덱스 위치 대신 이름 사용
"I ate {number} apples. so I was sick for {day} days.".format(number = 10, day = "two")
```

Out[24]:
```
'I ate 10 apples. so I was sick for two days.'
```

문자열 관련 함수

- 문자열 자료형은 자체적으로 내장 함수를 가지고 있다. 이 내장 함수를 사용하려면 문자열 변수 이름 뒤에 '.'를 붙인 다음에 함수 이름을 사용하면 된다.

- **문자열 관련 함수**

Operation	Description
len()	문자열 길이

x.count("a")	갯수 세기 : 문자열 x 안의 문자 a의 갯수
x.find("a")	위치 찾기 : 문자열 X 중 문자 a가 처음으로 나온 위치 (-1 if not found)
x.index("a")	위치 찾기 : 문자열 X 중 문자 a가 처음으로 나온 위치 (Error if not found)
"a".join(x)	문자열 삽입 : 문자열 x의 각각의 문자 사이에 "a"를 삽입
x.upper()	소문자를 대문자로 바꾸기
x.lower()	대문자를 소문자로 바꾸기
x.replace("a", "b")	문자열 바꾸기 : a를 b로 바꿈
x.lstrip() / x.rstrip() / x.strip()	공백 지우기 : x 문자열의 왼쪽 / 오른쪽 / 양쪽 공백 지우기
x.split()	문자열 나누기 : x 문자열을 공백을 기준으로 나누어 list로 반환

In [27]:
```
x = "learning python is fun!"
```

In [28]:
```
#문자열 길이
len(x)
```

Out[28]: 23

In [29]:
```
#갯수 세기
x.count("n")
```

Out[29]: 4

In [30]:
```
#위치 찾기 (파이썬은 숫자를 0부터 센다!)
x.find("r")
```

Out[30]: 3

In [32]:
```
#위치 찾기
x.find("python")
```

Out[32]: 9

In [33]:
```
x.find("k")
```

Out[33]: -1

x.index("k")

In [35]:
```
#문자열 삽입 : join 함수는 문자열뿐만 아니라 앞으로 배울 리스트나 튜플도 입력으로 사용할 수 있음
"/".join(x)
```

Out[35]: 'l/e/a/r/n/i/n/g/ /p/y/t/h/o/n/ /i/s/ /f/u/n/!'

In [36]:
```
" ".join(x)
```

Out[36]: 'l e a r n i n g   p y t h o n   i s   f u n !'

In [37]:
```
#소문자를 대문자로 바꾸기
x.upper()
```

Out[37]: 'LEARNING PYTHON IS FUN!'

In [38]:
```
#대문자를 소문자로 바꾸기
x.lower()
```

Out[38]: 'learning python is fun!'

In [39]:
```
#문자열 바꾸기
x.replace("fun", 'easy')
```

Out[39]: 'learning python is easy!'

In [40]:
```
#왼쪽 공백 지우기
a = " Hello "
a.lstrip()
```

Out[40]: 'Hello '

In [41]:
```
#오른쪽 공백 지우기
a.rstrip()
```

Out[41]: ' Hello'

In [42]:
```
#양쪽 공백 지우기
a.strip()
```

Out[42]: 'Hello'

In [43]:
```
#문자열 나누기
x.split()
```

Out[43]: ['learning', 'python', 'is', 'fun!']

In [44]:
```
#문자열 나누기
x = "a:b:c:d"
x.split(":")
```

Out[44]: ['a', 'b', 'c', 'd']

4. 불 자료형 (Bool)

- 불(bool) 자료형이란 참과 거짓을 나타내는 논리 유형을 나타내는 자료형이다.
- True (참) / False (거짓)
- 비교 연산자는 bool 값을 반환한다.

In [45]:
```
#비교 연산자
a =  3 < 7
a
```

Out[45]: True

In [46]:
```
type(a)
```

Out[46]:
```
bool
```

In [47]:
```
# True / False 의 첫 문자는 항상 대문자로 사용 !
b = False
type(b)
```

Out[47]:
```
bool
```

5. 리스트 자료형(List)

- 다양한 요소의 데이터의 집합을 표현하기 위한 자료형
- 여러 요소를 그룹화하여 단일 변수로 사용하는데 사용
- 정수, 실수, 문자열, 리스트 및 튜플과 같은 모든 데이터 유형을 요소로 사용 가능
- 순서가 있는 자료형으로 Indexing과 slicing 가능
- Mutable한 자료형 (삽입, 삭제, 정렬 가능)
- 덧셈 및 곱셈 연산 가능

리스트 (List) 만들기

- 각 요소 값은 쉼표 (,)로 구분한 뒤 대괄호 ([ ]) 를 사용하여 감싸 표현할 수 있다.
- 리스트명 = [요소1, 요소2, 요소 3, ...]

In [248]:
```
even = [2,4,6,8]
```

In [48]:
```
# 리스트는 다양한 유형의 요소를 지닐 수 있음 (비어있는 리스트, 숫자형, 문자열, 리스트 등...)
a = [2,4,6,8]
b = [1, "I", 30, "Love"]
c = [1, "I", [2, "Love"]]
d = [1, "I", [2, "Love", [3, "You"]]]
e = []
e = list()
```

리스트의 인덱싱과 슬라이싱

리스트의 인덱싱

In [49]:
```
print("List Index")
print(a[0])
print(a[1])
print(a[-1])
print(a[-2])
print(b[2])
print(c[2])
print(c[2][1])
print(d[-1])
```

```
List Index
2
4
8
6
30
[2, 'Love']
Love
[2, 'Love', [3, 'You']]
```

In [268]:
```
#인덱싱과 연산
print(a[0]+a[1])
print(b[1]+b[3])
```

```
6
ILove
```

In [ ]:
```
#아래의 출력된 결과를 예측해 보자
print(d[2][2][1])
print(d[-1][-1][1])
```

리스트의 슬라이싱

In [278]:
```
print("List Slicing")
print(a[0:2])
print(c[1:2])
print(c[1:-1])
print(d[1:])
```

```
List Slicing
[2, 4]
['I']
['I']
['I', [2, 'Love', [3, 'You']]]
```

- **리스트 요소 변경**

In [284]:
```
a[2] = 100     #2번째 요소 변경
print(a)
a[1:4] = 'x'   #1에서 3번째 요소 삭제하고 'x'로 변경
print(a)
```

```
[2, 4, 100, 8]
[2, 'x']
```

리스트 연산하기

리스트 합치기 (+)

In [51]:
```
a = [ 1, 2, 3]
```

```
b = ["you", "and" , "I"  ]
```

In [289]:
```
a+b
```

Out[289]:
```
[1, 2, 3, 'you', 'and', 'I']
```

In [ ]:
```
# tip : 아래 결과를 출력해보자 (문자형과 숫자형은 더할 수 없다. str() 함수 활용)
a[0] + b[0]
```

리스트 반복하기 (*)

In [290]:
```
a = [1, 2, 3]
a*3
```

Out[290]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

리스트 길이 구하기

In [292]:
```
len(a)
```

Out[292]:
```
3
```

리스트의 값 변경과 삭제

- 리스트 값을 삭제하는 방법은 del 함수 사용 외에도 remove 와 pop함수를 사용하는 방법이 있는데, 이는 5.5 리스트 관련 함수 에서 확인하자

1) 요소 값 변경

In [300]:
```
a = [1, 2, 3]
a[2] = 4
a
```

Out[300]:
```
[1, 2, 4]
```

2) del 함수를 이용한 요소 삭제

In [302]:
```
a = [1, 2, 3]
del a[2]
a
```

Out[302]:
```
[1, 2]
```

In [303]:
```
a = [1, 2, 3, 4, 5]
del a[2:]
a
```

Out[303]:
```
[1, 2]
```

리스트 관련 함수

- 문자열 자료형과 동일하게 리스트 변수 이름 뒤에 '.'를 붙여서 여러 가지 리스트 관련 함수를 사용할 수 있다.

- **함수**

Operation	Description
x.append()	요소 추가 : 리스트 x의 마지막 요소 추가
x.sort()	리스트 정렬 : 리스트 x 의 요소를 순서대로 정렬
x.reverse()	리스트 뒤집기 : 현재 리스트 x의 요소를 거꾸로 뒤집기
x.index(a)	위치 반환 : 리스트 x에 a 값이 있으면 위치를 반환
x.insert(a,b)	요소 삽입 : 리스트 x의 a 번째 위치에 b를 삽입
x.remove(a)	요소 제거 : 리스트 x에서 첫번째로 나타나는 a를 삭제
x.pop()	요소 끄집어내기 : 리스트 x의 마지막 요소(혹은 위치 지정가능)를 반환하고 그 요소를 삭제
x.count(a)	갯수세기 : 리스트 x 안에 a가 몇개 있는지 갯수 반환
x.extend(a)	리스트 확장: 리스트 x에 리스트 a를 더함
x.copy()	리스트 복사: 리스트 x를 복사

In [56]:
```
List = ["A", "B", "C", "D"]
```

In [57]:
```
# append: 요소 값 추가
List.append("E")
print(List)

# extend : 리스트 추가(확장)
List.extend(["G","H"])
print(List)

b= ["I", "J"]
List.extend(b)
print(List)

# insert : 요소 삽입
List.insert(5, "F") # 3번째에 "F" 삽입
print(List)
```

```
['A', 'B', 'C', 'D', 'E']
['A', 'B', 'C', 'D', 'E', 'G', 'H']
['A', 'B', 'C', 'D', 'E', 'G', 'H', 'I', 'J']
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

In [58]:
```
# 아래 두가지의 차이는 ?
List.append(100)
print ( List )
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 100]
```

In [ ]:
```
List.extend(100)
print (List)
```

In [63]:
```
#pop : 요소 끄집어 내기
print(List.pop()) #마지막 요소 출력
print(List)

print(List.pop(2)) #2번째 요소 출력
print(List)
```

```
C
['A', 'B', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

In [65]:
```
#copy : 리스트 복사
List1= List.copy()
print(List1)
```

```
['A', 'B', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

In [66]:
```
#remove : 요소 삭제
x = ["I", "My", "Me", "Mine"]
x.remove("My")
print(x)

a = [1,2,3,2,3]
a.remove(2)    #해당 값이 2개 이상이면 가장 먼저 위치한 값만 삭제
print(a)
```

```
['I', 'Me', 'Mine']
[1, 3, 2, 3]
```

In [341]:
```
#또 다른 요소 삭제 방법
del a[0]
print(a)
```

```
[3, 2, 3]
```

In [67]:
```
#또 다른 요소 삭제 방법
x[0:2] =[]   #0에서 1번째 요소 삭제
print(x)
```

```
['Mine']
```

In [354]:
```
#sort : 정렬
ListNum = [7, 4, 3, 6, 5, 1, 2, 2]
ListChr = ['c', 'b', 'a', 'd']

#오름차순 정렬
ListNum.sort()
print(ListNum)
ListChr.sort()
print(ListChr)

#내림차순 정렬
```

```
ListNum.sort(reverse=True)
print(ListNum)
ListChr.sort(reverse=True)
print(ListChr)
```

```
[1, 2, 2, 3, 4, 5, 6, 7]
['a', 'b', 'c', 'd']
[7, 6, 5, 4, 3, 2, 2, 1]
['d', 'c', 'b', 'a']
```

In [355]:
```
# reverse : 요소 뒤집기
ListNum.reverse()
print(ListNum)
```

```
[1, 2, 2, 3, 4, 5, 6, 7]
```

In [357]:
```
# index : 위치 반환
ListNum.index(1)
```

Out[357]: 0

In [358]:
```
# count : 갯수 세기
ListNum.count(2)
```

Out[358]: 2

6. 튜플 자료형 (Tuple)

- 튜플(tuple)은 리스트와 유사하게 여러 개의 객체를 모아 담는 데 사용되지만 리스트와는 몇가지 다른 점을 갖고 있는 자료형이다.
- 리스트와 튜플의 차이점은 아래와 같다.
  - 리스트는 대괄호 [ ]으로 둘러싸지만 튜플은 소괄호 ( )으로 둘러싼다.리스트는
  - 요소의 생성, 삭제, 변경이 가능하지만 튜플은 그 값을 바꿀 수 없다.

- 튜플은 리스트와 동일하게 정수, 실수, 문자열, 리스트 등 모든 자료형을 요소로 사용 가능하고
- 순서가 있는 자료형으로 indexing과 slicing이 가능하며
- 덧셈과 곱셈 연산 가능하다.
- 속도가 빨라서 Python에서 내부적으로 사용하는 일이 많다.

In [365]:
```
# 튜플의 모습은 다음과 같다
t1 = (1, 2, 3)
t2 = 1, 2, 3
t3 = 1,
t4 = (1,)
t5= ()
t6= ('a', 'b',('ab','cd'))
```

리스트와 모습은 거의 비슷하지만 튜플에서는 리스트와 다른 2가지 차이점을 찾아볼 수 있다.

t24 처럼 단지 1개의 요소만을 가질 때는 요소 뒤에 콤마(,)를 반드시 붙여야 한다는 것과 t2 처럼

괄호( )를 생략해도 무방하다는 점이다.

튜플과 리스트의 가장 큰 차이는 값을 변화시킬 수 있는가 여부이다.

프로그램이 실행되는 동안 그 값이 항상 변하지 않기를 바란다거나 값이 바뀔까 걱정하고 싶지 않다면 주저하지 말고 튜플을 사용하면 된다.

튜플(tuple) 요소의 값을 삭제 혹은 변경을 시도

In [ ]:
```
#값 변경을 시도해보자 !
t1 = (1, 2, 3, 4, 5)
t1[2] = 10
```

In [ ]:
```
#값 삭제를 시도해보자
del t1[0]
```

튜플의 인덱싱과 슬라이싱

In [70]:
```
#튜플의 인덱싱 (indexing)
t1 = 1, 2, 'a', 'b'
t1[0]
```

Out[70]:
```
1
```

In [71]:
```
#튜플의 슬라이싱 (slicing)
t1[1:]
```

Out[71]:
```
(2, 'a', 'b')
```

튜플 관련 함수

In [381]:
```
#튜플 더하기
t1 = 1, 2, 'a', 'b'
t2 = (3, 4)
t1 + t2
```

Out[381]:
```
(1, 2, 'a', 'b', 3, 4)
```

In [382]:
```
#튜플 곱하기
t2*2
```

Out[382]:
```
(3, 4, 3, 4)
```

In [383]:
```
#튜플 길이 구하기
len(t1)
```

Out[383]:
```
4
```

In [387]:
```
#튜플을 리스트로 변경
t1 = (1,2,3,4,5)
```

```
l1 = list(t1)
print (l1, type(l1))
t2 = tuple(l1)
print (t2, type(t2))
```

```
[1, 2, 3, 4, 5] <class 'list'>
(1, 2, 3, 4, 5) <class 'tuple'>
```

7. 딕셔너리 자료형 (Dictionary)

- 딕셔너리는 사전 처럼 대응관계를 나태날 수 있는 자료형이다. "apple" 라는 단어에 사과, "love" 라는 단어에 "사랑" 이라는 뜻이 대응되듯 딕셔러니 자료는 key 와 value를 한 쌍으로 갖는 자료형이다. key 가 love 라고 한다면, value는 "사랑' 이 될 것이다.

- 리스트나 튜플과의 차별점
  - 리스트와 튜플은 순차적으로 요소 값을 모두 확인하지만, 딕셔너리는 Key를 통해서 바로 value를 확인한다. 예를 들어, love의 뜻을 찾기 위해서 사전의 처음 부터 순차적으로 모두 검색하는 것이 아니라, love 라는 단어가 있는 곳만 확인하는 것이다.

- 따라서, key는 고유한 값(중복되지 않아야 함)이어야 하며, 문자열 또는 단일 숫자, 튜플과 같이 변하지 않는 값인 unhashable type이어야 한다. 리스트와 같이 변경이 가능한 값은 Key로 사용이 불가능하다.
- Value는 중복된 값이 사용 될 수 있으며, 변하는 값이든 변하지 않는 값이든 아무 값이나 넣을수 있다.

- 즉, 딕셔너리는 Index가 아닌 Key를 사용하여 데이터 탐색하고
- 따라서, index number를 이용한 Indexing과 slicing이 불가하다.

- 딕셔너리는 아래와 같이 중괄호 { }로 묶어서 표현 한다.

{Key1:Value1, Key2:Value2, Key3:Value3, ...}

In [77]:
```
# example
dic = {'Name':'James', 'Job':'Student', 'Phone':[82,10,9994,323]} # Key (Name , job , Phone) , value
print(dic)
print(dic['Name'] + ' is a ' + dic['Job'])
```

```
{'Name': 'James', 'Job': 'Student', 'Phone': [82, 10, 9994, 323]}
James is a Student
```

In [75]:
```
type(dic)
```

Out[75]: dict

딕셔너리 추가와 삭제

1) 딕셔너리 **key & value** 추가

In [408]:
```
dic = {'Name':'James', 'Job':'Student'}
dic['Phone'] = [82,10,9994,323]  # key & value 삽입
print(dic)
```

```
{'Name': 'James', 'Job': 'Student', 'Phone': [82, 10, 9994, 323]}
```

In [78]:
```
dic[1] ="a"
print(dic)
```

```
{'Name': 'James', 'Job': 'Student', 'Phone': [82, 10, 9994, 323], 1:
'a' }
```

2) 딕셔너리 삭제

In [410]:
```
del dic['Phone'] # key & value 삭제
print(dic)

del dic[1]  # Index 가 아닌 key 를 찾아 삭제한다
print(dic)
```

```
{'Name': 'James', 'Job': 'Student', 1: 'a'}
{'Name': 'James', 'Job': 'Student'}
```

딕셔너리 관련 함수

- 함수

Operation	Description
x.keys()	Key 리스트 만들기
x.values()	Value 리스트 만들기
x.items()	Key, Value 쌍 얻기
x.clear()	Key: Value 쌍 모두 지우기
x.get(a)	Key a에 해당 하는 Value얻기
a in x	key a가 딕셔너리 x 안에 있는지 확인 (True/ False)

In [83]:
```
dic = {'Name':'James', 'Job':'Student',  'Phone': [82, 10, 9994, 323]}
print(dic.keys()) # key를 리스트로 묶어 dict_key 객체로 돌려줌
print(dic.values()) # value를 리스트로 묶어 dict_values 객체로 돌려줌
print(dic.items()) # key & value 쌍을 튜플로 묶어 dict_items로 돌려줌
print(dic.get('Name')) # key 값을 이용해 value 출력
print(dic['Name']) # 위와 동일
print('Job' in dic) # 해당 값을 지닌 key 존재 여부 확인
print('ID' in dic)
dic.clear() # 전체 삭제
print(dic)
```

```
dict_keys(['Name', 'Job', 'Phone'])
dict_values(['James', 'Student', [82, 10, 9994, 323]])
dict_items([('Name', 'James'), ('Job', 'Student'), ('Phone', [82, 10, 99
```

```
94, 323])])
James
James
True
False
{}
```

- Tip : Key 가 중복된 경우 어떻게 될까?
  - 이 경우 중복된 key가 사용된 경우 하나만 사용하고 나머지는 버린다.
  - 어떤 value를 사용하고, 어떤 value를 버릴지 알 수 없음(random하게 선택 됨)

In [416]:
```
dic = {'Name':'James', 'Name':'SJ', 'Job':'Student'}
print(dic)
```

```
{'Name': 'SJ', 'Job': 'Student'}
```

8. 집합 자료형 (Set)

- 집합(set) 자료형은 파이썬 2.3부터 지원하기 시작한 자료형으로, 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형이다.
- 수학에서 집합과 유사한 기능을 하며, set 키워드를 사용해 만들 수 있다.

In [418]:
```
s1 = set([1,2,3])
s1
```

Out[418]:
```
{1, 2, 3}
```

In [424]:
```
s2 = set("python")
print(type(s2))
print(s2)
```

```
<class 'set'>
{'h', 'n', 't', 'y', 'p', 'o'}
```

In [425]:
```
s3=set([1, 2, 2, 3, 4, 4, 4, 4])
print(s3)
```

```
{1, 2, 3, 4}
```

In [426]:
```
s4 = set('hello123')
print(s4)
```

```
{'h', '3', '1', 'l', '2', 'e', 'o'}
```

In [429]:
```
#비어있는 집합 자료형
s5=set()
print( s5)
```

```
set()
```

집합 자료형의 특징은 아래와 같다.

- 중복을 허용하지 않는다.
  -> 이 특성을 활용하여 리스트에서 중복을 제거하는데 유용하게 사용 할 수 있다.
- 순서가 없기 때문에 인덱싱이나 슬라이싱이 불가능한다.
  -> 만약 set 자료 형에서 저장된 값을인덱싱으로 접근하려면 리스트나 튜플로 변환하면 된다.

In [435]:
```
#리스트를 집합 자료형으로 바꾸어 중복 제거
l1 = [1, 2, 3, 2, 3]
s1 = set(l1)
print(s1, type(s1))
l2 = list(s1)
print( l2[0:2])    #리스트로 바꾸어 인덱싱
print(l2, type(l2))
```

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

교집합, 합집합, 차집합

- set 자료형을 정말 유용하게 사용하는 경우는 교집합, 합집합, 차집합을 구할 때이다.

In [438]:
```
set1 = set([1, 2, 3])
set2 = set([2, 4, 5, 6])
```

1) 교집합

- set1 과 set2 의 교집합 구하기 : & 혹은 .intersection() 이용

In [439]:
```
set1 & set2
```

Out[439]:
```
{2}
```

In [440]:
```
set1.intersection(set2)
```

Out[440]:
```
{2}
```

2) 합집합

- set1 과 set2 의 합집합 구하기 : | 혹은 .union() 이용

In [442]:
```
set1|set2
```

Out[442]:
```
{1, 2, 3, 4, 5, 6}
```

In [443]:
```
set1.union(set2)
```

Out[443]:
```
{1, 2, 3, 4, 5, 6}
```

3) 차집합

- set1과 set2의 차집합 구하기 : - 혹은 .difference() 이용

```
In [446]: set1 - set2
```

```
Out[446]: {1, 3}
```

```
In [447]: set2 - set1
```

```
Out[447]: {4, 5, 6}
```

```
In [448]: set1.difference(set2)
```

```
Out[448]: {1, 3}
```

```
In [449]: set2.difference(set1)
```

```
Out[449]: {4, 5, 6}
```

집합 자료형 관련 함수

- **함수**

Operation	Description
x.add()	값 1 개 추가
x.update()	값 여러개 추가
x.remove()	특정 값 제거하기

```
In [91]: set1 = set([1, 2, 3])
```

```
In [450]: # add : 값 1개 추가
          set1.add(4)
          print(set1)
```

```
          {1, 2, 3, 4}
```

```
In [451]: # update() : 값 여러개 추가
          set1.update([5, 6, 7, 8, 9, 10])
          print(set1)
```

```
          {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In [452]: # remove : 특정 값 제거
          set1.remove(10) # 값 제거 : remove()
          print(set1)
```

```
          {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
In [92]: # 집합 제거
         del set1 # 집합 set1 제거 --> 이후 set1을 불어오면 error 발생
```

9. Examples

Q1) 다음의 고유한 파이썬의 특징을 확인해 보자.

파이썬은 아래와 같이 동일한 값에 여러개의 변수를 선언 할 수 있다.

a, b 변수를 한번에 동일한 값으로 선언한 뒤 a의 요소값을 변경한다면, b의 값은 어떻게 될까?

In [ ]:
```
a = b = [1, 2, 3]
a[1] = 4
print(b)
```

Q2) 위에서 a와 동일한 값을 갖지만, 동시에 참조되지 않는 별개의 변수를 만들고자 한다면 어떤 방법이 있을까?

Q3) 아래 리스트 x 에서 중복을 제거해 보자

In [468]:
```
x = [1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5]
```

Q4) (1,2,3) 튜플에 값 4를 추가하여 (1,2,3,4)를 만들어 출력해 보자.

Q5) ['Python', 'is', 'so', 'fun'] 리스트를 Python is so fun 문자열로 만들어 출력해 보자.

In [521]:
```
x=['Python', 'is', 'so', 'fun']
```