Table of Contents

Python 기초

개요

본 강의 자료는 Python 공식 Tutorial에 근거하여 만들어졌으며, Python 3.6버전에 해당하는 내용을 담고 있습니다.

또한, 파이썬에서 제공하는 스타일 가이드인 `PEP-8` 내용을 반영하였습니다.

파이썬을 활용하는 다양한 IT기업들은 대내외적으로 본인들의 스타일 가이드를 제공하고 있습니다.

- 구글 스타일 가이드
- Tensorflow 스타일 가이드

식별자

파이썬에서 식별자는 변수, 함수, 모듈, 클래스 등을 식별하는데 사용되는 이름이다.

- 식별자의 이름은 영문알파벳, _, 숫자로 구성된다.
- 첫 글자에 숫자가 올 수 없다.
- 대소문자를 구별한다.
- 아래의 예약어는 사용할 수 없다.

```
False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except,
finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, ret
urn, try, while, with, yield
```

In [1]:

```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif
', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

- 내장함수나 모듈 등의 이름으로도 만들면 안된다.

In [2]:

```
str(5)
```

Out[2]:

```
'5'
```

In [3]:

```
# str() 형변환 함수로 정해진 식별자로 변수를 할당해버리면, 함수호출이 되지 않음.
str = 'hi'
str(5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-0ff51483eb31> in <module>
      1 # str() 형변환 함수로 정해진 식별자로 변수를 할당해버리면, 함수호출이 되지 않음.
      2 str = 'hi'
----> 3 str(5)

TypeError: 'str' object is not callable
```

In [4]:

```
# 뒤에 코드에 영향이 가니까 변수를 메모리에서 지워줍시다!
del str
```

기초 문법

인코딩 선언

인코딩은 선언하지 않더라도 `UTF-8` 로 기본 설정이 되어 있다.

만약, 인코딩을 설정하려면 코드 상단에 아래와 같이 선언한다. 주석으로 보이지만, Python `parser` 에 의해 읽혀진다.

In [ ]:

```
# -*- coding: <encoding-name> -*-
```

주석(Comment)

- 주석은 `#` 으로 표현한다.
- `docstring` 은 `"""` 으로 표현한다.

  : 여러 줄의 주석을 작성할 수 있으며, 보통 함수/클래스 선언 다음에 해당하는 설명을 위해 활용한다.

- 예시 : flask 공식 문서 일부 발췌

In [5]:

```
# 이 줄은 실행되지 않습니다.
def mysum(a,b):
    """이것은 덧셈 함수입니다.
    이 줄은 실행되지 않아요.
    그런데 docstring인 이유가 있습니다.
    """
    print(a+b)
```

In [6]:

```
mysum.__doc__
```

Out[6]:

```
'이것은 덧셈 함수입니다.\n    이 줄은 실행되지 않아요.\n    그런데 docstring인 이유가 있습니다.\n    '
```

코드 라인

- 기본적으로 파이썬에서는 `;` 을 작성하지 않는다.
- 한 줄로 표기할 때는 `;` 를 작성하여 표기할 수 있다.

In [7]:

```
print("hello")
print("ssafy")
```

```
hello
ssafy
```

In [8]:

```
print("hello") print("ssafy")
```

```
  File "<ipython-input-8-74a709ae5a00>", line 1
    print("hello") print("ssafy")
                   ^
SyntaxError: invalid syntax
```

In [9]:

```
print("hello");print("ssafy")
```

```
hello
ssafy
```

- 줄을 여러줄 작성할 때는 역슬래시 `\` 를 사용하여 아래와 같이 할 수 있다.

In [10]:

```
a = 0
```

```
if a
== 0:
    print(a)
```

```
  File "<ipython-input-10-8ea600b3db2f>", line 2
    if a
       ^
SyntaxError: invalid syntax
```

In [11]:

```
a = 0
if a \
== 0:
    print(a)
```

```
0
```

- `[]` `{}` `()` 는 `\` 없이도 가능하다.

In [12]:

```
lunch = ["떡국", "부침개", "샐러드",
         "군만두", "물만두"]
```

변수(variable) 및 자료형

- 변수는 `=`을 통해 할당(assignment) 된다.
- 해당 자료형을 확인하기 위해서는 `type()` 을 활용한다.
- 해당 변수의 메모리 주소를 확인하기 위해서는 `id()` 를 활용한다.

In [13]:

```
x = 1004
print(x)
print(type(x))
print(id(x))

x = 10004
print(type(x))
print(id(x))
```

```
1004
<class 'int'>
2732738185072
<class 'int'>
2732738185104
```

- 같은 값을 동시에 할당할 수 있다.

In [14]:

```
x = y = 1004
print(x, y)
```

```
1004 1004
```

- 다른 값을 동시에 할당 가능하다.

In [15]:

```
# not good
x, y = 1, 2
print(x, y)

# good
x = 1
y = 2
```

```
1 2
```

In [16]:

```
x, y = 1, 2
```

In [17]:

```
x, y = 1, 2, 3
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-10edbbb6d673> in <module>
----> 1 x, y = 1, 2, 3

ValueError: too many values to unpack (expected 2)
```

- 이를 활용하면 서로 값을 바꾸고 싶은 경우 아래와 같이 활용 가능하다.

In [18]:

```
# 쉽게 변수 값을 swap 가능함.
x, y = y, x
print(x, y)
```

```
2 1
```

수치형(Numbers)

`int` (정수)

모든 정수는 `int` 로 표현된다.

파이썬 3.x 버전에서는 `long` 타입은 없고 모두 `int` 형으로 표기 된다.

10진수가 아닌 8진수 : `0o` /2진수 : `0b` /16진수: `0x` 로도 표현 가능하다.

In [19]:

```
a = 3
type(a)
```

Out[19]:

```
int
```

In [20]:

```
# 보통 프로그래밍 언어 및 파이썬 2.x에서의 long은 OS 기준 32/64비트이다.
# 파이썬 3.x에서는 모두 int로 통합되었다.
a = 2**64
print(a)
type(a)
```

```
18446744073709551616
```

Out[20]:

```
int
```

In [21]:

```
# 파이썬은 기존 C 계열 프로그래밍 언어와 다르게 정수 자료형에서 오버플로우가 없다.
# arbitrary-precision arithmetic를 사용하기 때문이다. (메모리양이 정해져 있는 기존의 fixed-precision과 달리
, 현재 남아있는 만큼의 가용 메모리를 모두 수 표현에 끌어다 쓸 수 있는 형태)
import sys
max_int = sys.maxsize
print(max_int)
a = sys.maxsize * sys.maxsize
print(a)
```

```
9223372036854775807
85070591730234615847396907784232501249
```

In [22]:

```
# n진수
binary_number = 0b10
octal_number = 0o10
decimal_number = 10
hexadecimal_number = 0x10
print(f'''
2진수 : {binary_number}
8진수 : {octal_number}
10진수 : {decimal_number}
16진수 : {hexadecimal_number}
''')
```

```
2진수 : 2
8진수 : 8
10진수 : 10
16진수 : 16
```

`float`(부동소수점, 실수)

실수는 `float`로 표현된다.

다만, 실수를 컴퓨터가 표현하는 과정에서 부동소수점을 사용하며, 항상 같은 값으로 일치되지 않는다. (floating point rounding error)

이는 컴퓨터가 2진수(비트)를 통해 숫자를 표현하는 과정에서 생기는 오류이며, 대부분의 경우는 중요하지 않으나 값을 같은지 비교하는 과정에서 문제가 발생할 수 있다.

In [23]:

```
a = 3.5
type(a)
```

Out[23]:

```
float
```

In [24]:

```
b = 314e-2
type(b)
```

Out[24]:

```
float
```

- 실수의 경우 실제로 값을 처리하기 위해서는 조심할 필요가 있다.

In [39]:

```
3.5 + 3.2
```

Out[39]:

```
6.7
```

In [40]:

```
3.5 - 3.12
```

Out[40]:

```
0.3799999999999999
```

In [41]:

```
round(3.5 - 3.12, 2)
```

Out[41]:

```
0.38
```

In [42]:

```
0.1 * 3 == 0.3
```

Out[42]:

```
False
```

In [43]:

```
print(0.1*3)
```

```
0.30000000000000004
```

- 따라서 다음과 같은 방법으로 처리 할 수 있다. 이외에 다양한 방법이 있음

In [44]:

```
# 처리방법 1-1. 절대값을 비교
a = 0.1 * 3
b = 0.3

abs(a - b) <= 1e-10
```

Out[44]:

```
True
```

In [45]:

```
# 처리방법 1-2. 절대값 비교를 내장된 float epsilon값과 비교
import sys
abs(a-b) <= sys.float_info.epsilon
```

Out[45]:

```
True
```

In [46]:

```
# 처리방법 2. math 모듈을 통해 근사한 값인지 비교
# python 3.5부터는 math 모듈을 활용할 수 있다.
import math
math.isclose(a, b)
```

Out[46]:

```
True
```

`complex` (복소수)

복소수는 허수부를 `j` 로 표현한다.

In [47]:

```
a = 3 + 4j
```

In [50]:

```
# 허수 성분 얻기 (float)
print(a.imag)
# 실수 성분 얻기 (float)
print(a.real)
# 켤레 복소수, 공액복소수 (허수 부분인 4j가 -4j가 된 수)
print(a.conjugate())
```

```
4.0
3.0
(3-4j)
```

Bool

파이썬에는 `True` 와 `False` 로 이뤄진 `bool` 타입이 있다.

비교/논리 연산을 수행 등에서 활용된다.

다음은 `False` 로 변환됩니다.

```
0, 0.0, (), [], {}, '', None
```

In [65]:

```
print(type(True))
print(type(False))
```

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

- 형변환(Type Conversion)에서 추가적으로 다루는 내용입니다.

In [55]:

```
bool(0)
```

Out[55]:

```
False
```

In [56]:

```
bool(1)
```

Out[56]:

```
True
```

In [57]:

```
bool(None)
```

Out[57]:

```
False
```

In [58]:

```
bool([])
```

Out[58]:

```
False
```

In [59]:

```
bool('')
```

Out[59]:

```
False
```

In [60]:

```
bool(['hi'])
```

Out[60]:

```
True
```

In [61]:

```
bool('hi')
```

Out[61]:

```
True
```

None

파이썬에서는 값이 없음을 표현하기 위해 `None` 타입이 존재합니다.

In [62]:

```
type(None)
```

Out[62]:

```
NoneType
```

In [63]:

```
a = None
print(a)
```

```
None
```

문자형(String)

기본 활용법

기본 활용법

문자열은 Single quotes(`'`)나 Double quotes(`"`)을 활용하여 표현 가능하다.

단, 문자열을 묶을 때 동일한 문장부호를 활용해야하며, `PEP-8` 에서는 **하나의 문장부호를 선택**하여 유지하도록 하고 있습니다. (Pick a rule and Stick to it)

In [66]:

```
greeting = 'hi'
name = 'hs'
print(greeting, name)
```

```
hi hs
```

- 다만 문자열 안에 문장부호(`'` , `"`)가 활용될 경우 이스케이프 문자(`\`)를 사용하는 것 대신 활용 가능 합니다.

In [67]:

```
print('철수가 말했다. '안녕?'')
```

```
  File "<ipython-input-67-4258189cde38>", line 1
    print('철수가 말했다. '안녕?'')
                          ^
SyntaxError: invalid syntax
```

In [72]:

```
print('철수가 말했다. "안녕?"')
print('철수가 말했다. \'안녕?\'')
print("철수가 말했다. '안녕?'")
```

```
철수가 말했다. "안녕?"
철수가 말했다. '안녕?'
철수가 말했다. '안녕?'
```

- 여러줄에 걸쳐있는 문장은 다음과 같이 표현 가능합니다.

`PEP-8` 에 따르면 이 경우에는 반드시 `"""` 를 사용하도록 되어 있습니다.

In [73]:

```
print("""
개행문자없이
여러 줄을
그대로 출력
가능합니다.
""")
```

```
개행문자없이
여러 줄을
그대로 출력
가능합니다.
```

In [75]:

```
# string interpolation 도 가능합니다.
a = True
print(f"""
물론,
스트링 인터폴레이션도 가능합니다! {a}!!!
""")
```

```
물론,
스트링 인터폴레이션도 가능합니다! True!!!
```

이스케이프 문자열

문자열을 활용하는 경우 특수문자 혹은 조작을 하기 위하여 사용되는 것으로 `\`를 활용하여 이를 구분한다.

예약문자	내용(의미)
\n	줄바꿈
\t	탭
\r	캐리지리턴
\0	널(Null)
\\	\
\'	단일인용부호(')
\"	이중인용부호(")

In [77]:

```
print('이 다음은 엔터. \n그리고 탭\t 탭')
```

이 다음은 엔터.
그리고 탭 탭

- 이를 출력할 때 활용할 수가 있다.

In [78]:

```
print('내용을 띄워서 출력하고 싶으면?', end='\t')
print('옆으로 띄워짐')
```

내용을 띄워서 출력하고 싶으면? 옆으로 띄워짐

In [81]:

```
print('개행 문자 말고도 가능합니다.', end='!')
print('진짜로', end='!')
print('print는 기본이 \\n')
```

개행 문자 말고도 가능합니다.!진짜로!print는 기본이 \n

깜짝 퀴즈

다음의 문장을 출력해보세요.

- `"""` 사용 금지
- `print` 여러번 사용 금지

"파일은 C:\Windows\Users\내문서\Python에 저장이 되어있습니다."
나는 생각했다. 'cd를 써서 git bash로 들어가봐야지'

In [92]:

```
# 여기에 코드를 작성해주세요.
print('"파일은 C:\\Windows\\Users\\내문서\\Python에 저장이 되어있습니다." \n나는 생각했다. \'cd를 써서 git bash로 들어가봐야지\'')
```

"파일은 C:\Windows\Users\내문서\Python에 저장이 되어있습니다."
나는 생각했다. 'cd를 써서 git bash로 들어가봐야지'

String interpolation

1) % formatting

1) `%-formatting`

2) `str.format()`

3) `f-strings` : 파이썬 3.6 버전 이후에 지원 되는 사항입니다.

본 슬라이드에서는 `f-strings`의 기본적인 활용법만 알려드리고 나머지 `.format()`는 해당 링크에서 확인바랍니다.

In [96]:
```
name = "kim"
```

In [102]:
```
'hello, %s' % name
```

Out[102]:
```
'hello, kim'
```

In [98]:
```
'hello, {}'.format(name)
```

Out[98]:
```
'hello, kim'
```

In [99]:
```
f'hello, {name}'
```

Out[99]:
```
'hello, kim'
```

- f-strings에서는 형식을 지정할 수 있으며,

In [103]:
```
import datetime
today = datetime.datetime.now()
print(today)
```

```
2019-01-02 15:41:38.474542
```

In [114]:
```
f'오늘은 {today:%Y}년 {today:%m}월 {today:%d}일 {today:%A}'
```

Out[114]:
```
'오늘은 2019년 01월 02일 Wednesday'
```

- 연산도 가능합니다.

In [115]:
```
pi = 3.141592
f'원주율은 {pi:.3}. 반지름이 2일때 원의 넓이는 {pi*2*2}'
```

Out[115]:
```
'원주율은 3.14. 반지름이 2일때 원의 넓이는 12.566368'
```

연산자

산술 연산자

Python에서는 기본적인 사칙연산이 가능합니다.

연산자	내용
+	덧셈
-	뺄셈
*	곱셈
/	나눗셈
//	몫
%	나머지(modulo)
**	거듭제곱

In [116]:

```
2 ** 1000
```

Out[116]:

```
107150860718626732094842504906000181056140481170553360744375038837035105112493612249319837881569585
594672917553146825187145285692314043598457757469857480393456777482423098542107460506237114187795418
046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376
```

In [117]:

```
print(5/2)
print(5//2)
print(int(5/2))
print(5%2)
```

```
2.5
2
2
1
```

In [120]:

```
# 내장함수 divmod()
print(divmod(5, 2))
quotient, remainder = divmod(5, 2)
print(f'몫은 {quotient}, 나머지는 {remainder}')
```

```
(2, 1)
몫은 2, 나머지는 1
```

- 양수/음수도 표현 가능합니다.

In [121]:

```
positive_num = 4
print(-positive_num)
negative_num = -4
print(+negative_num)
print(-negative_num)
```

```
-4
-4
4
```

비교 연산자

우리가 수학에서 배운 연산자와 동일하게 값을 비교할 수 있습니다.

연산자	내용
a > b	초과
a < b	미만
a >= b	이상
a <= b	이하
a == b	같음
a != b	같지않음

In [122]:

```
3 > 6
```

Out[122]:

```
False
```

In [123]:

```
3 != 3
```

Out[123]:

```
False
```

In [124]:

```
3.0 == 3
```

Out[124]:

```
True
```

In [125]:

```
'HI' == 'hi'
```

Out[125]:

```
False
```

논리 연산자

연산자	내용
a and b	a와 b 모두 True시만 True
a or b	a 와 b 모두 False시만 False
not a	True -> False, False -> True

우리가 보통 알고 있는 `&` `|` 은 파이썬에서 비트 연산자이다.

In [126]:

```
print(True and True)
print(True and False)
print(False and True)
print(False and False)
```

```
True
False
False
False
```

In [127]:

```
print(True or True)
print(True or False)
print(False or True)
print(False or False)
```

```
True
True
True
False
```

In [128]:

```
print(not True)
print(not 0)
```

```
False
True
```

In [129]:

```
# quiz! 답을 적어보고 실행하세요.
print(3 and 5)
print(0 and 3)
print(3 and 0)
print(0 and 0)
```

```
5
0
0
0
```

In [130]:

```
# quiz! 답을 적어보고 실행하세요.
print(3 or 5)
print(0 or 3)
print(3 or 0)
print(0 or 0)
```

```
3
3
3
0
```

- 파이썬에서 and 는 a 가 거짓이면 a 를 리턴하고, 참이면 b 를 리턴한다.
- 파이썬에서 or 은 a 가 참이면 a 를 리턴하고, 거짓이면 b 를 리턴한다.

복합 연산자

복합 연산자는 연산과 대입이 함께 이뤄진다.

가장 많이 활용되는 경우는 반복문을 통해서 갯수를 카운트하거나 할 때 활용된다.

연산자	내용
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a //= b	a = a // b
a %= b	a = a % b

In [131]:

```
count = 0
while count < 5:
    print(count)
    count += 1
```

```
0
1
2
3
4
```

기타 연산자

Concatenation

숫자가 아닌 자료형은 `+` 연산자를 통해 합칠 수 있다.

Containment Test

`in` 연산자를 통해 속해있는지 여부를 확인할 수 있다.

Identity

`is` 연산자를 통해 동일한 object인지 확인할 수 있다.

(나중에 Class를 배우고 다시 학습)

Indexing/Slicing

`[]`를 통한 값 접근 및 `[:]`을 통한 슬라이싱

(다음 챕터를 배우면서 추가 학습)

In [133]:

```
'hi ' + 'bye!'
```

Out[133]:

```
'hi bye!'
```

In [136]:

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

Out[136]:

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

In [137]:

```
'a' in 'apple'
```

Out[137]:

```
True
```

In [138]:

```
1 in [2, 3, 4]
```

Out[138]:

```
False
```

In [139]:

```
5 in range(5)
```

Out[139]:

```
False
```

In [144]:

```
a = 1000000
b = 1000000
a is b
print(id(a))
print(id(b))
```

```
1933620525392
1933625537488
```

In [145]:

```
a = 3
b = 3
a is b
print(id(a))
print(id(b))
```

```
1425697600
1425697600
```

In [146]:

```
'hello'[1]
```

Out[146]:

```
'e'
```

연산자 우선순위

1. `()`을 통한 grouping
2. Slicing
3. Indexing
4. 제곱연산자 **
5. 단항연산자 +, - (음수/양수 부호)
6. 산술연산자 *, /, %
7. 산술연산자 +, -
8. 비교연산자, `in`, `is`
9. `not`
10. `and`
11. `or`

In [147]:

```
3**2/2+8*3/2
```

Out[147]:

```
16.5
```

기초 형변환(Type conversion, Typecasting)

파이썬에서 데이터타입은 서로 변환할 수 있다.

암시적 형변환(Implicit Type Conversion)

사용자가 의도하지 않았지만, 파이썬 내부적으로 자동으로 형변환 하는 경우이다. 아래의 상황에서만 가능하다.

- bool
- Numbers (int, float, complex)

In [148]:

```
True + 3
```

Out[148]:

```
4
```

In [149]:

```
int_num = 3
float_num = 5.0
complex_num = 3 + 5j
```

In [150]:

```
print(int_num + float_num)
print(type(int_num + float_num))
```

```
8.0
<class 'float'>
```

In [151]:

```
print(int_num + complex_num)
print(type(int_num + complex_num))
```

```
(6+5j)
<class 'complex'>
```

명시적 형변환(Explicit Type Conversion)

위의 상황을 제외하고는 모두 명시적으로 형 변환을 해주어야한다.

- string -> intger : 형식에 맞는 숫자만 가능
- integer -> string : 모두 가능

암시적 형변환이 되는 모든 경우도 명시적으로 형변환이 가능하다.

- `int()` : string, float를 int로 변환
- `float()` : string, int를 float로 변환
- `str()` : int, float, list, tuple, dictionary를 문자열로 변환

`list(), tuple()` 등은 다음 챕터에서 배울 예정이다.

In [152]:

```
# integer와 string 사이의 관계는 명시적으로 형변환을 해줘야만 한다.
1 + '등'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-152-7bb9f06cc914> in <module>
      1 # integer와 string 사이의 관계는 명시적으로 형변환을 해줘야만 한다.
----> 2 1 + '등'
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [154]:

```
str(1) + '등'
```

Out[154]:

```
'1등'
```

In [159]:

```
a = '3'
int(a)
```

Out[159]:

```
3
```

In [160]:

```
a = '3.5'
float(a)
```

Out[160]:

```
3.5
```

In [161]:

```
# string은 글씨가 숫자일때만 형변환이 가능하다.
a = 'hi'
int(a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-161-388a321899ce> in <module>
      1 # string은 글씨가 숫자일때만 형변환이 가능하다.
      2 a = 'hi'
----> 3 int(a)

ValueError: invalid literal for int() with base 10: 'hi'
```

In [163]:

```
# string 3.5를 int로 변환할 수는 없다.
a = '3.5'
int(a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-163-13fb0a6b7c00> in <module>
      1 # string 3.5를 int로 변환할 수는 없다.
      2 a = '3.5'
----> 3 int(a)

ValueError: invalid literal for int() with base 10: '3.5'
```

In [164]:

```
# float 3.5는 int로 변환이 가능하다.
a = 3.5
int(a)
```

Out[164]:

```
3
```

In [167]:

```
print(int('55')+3)
print('55' + str(3))
```

```
58
553
```

시퀀스(sequence) 자료형

`시퀀스`는 데이터의 순서대로 나열된 형식을 나타낸다.

주의! 순서대로 나열된 것이 정렬되었다라는 뜻은 아니다.

파이썬에서 기본적인 시퀀스 타입은 다음과 같다.

1. 리스트(list)
2. 튜플(tuple)
3. 레인지(range)
4. 문자열(string)
5. 바이너리(binary) : 따로 다루지는 않습니다.

`list`

활용법

```
    [value1, value2, value3]
```

리스트는 대괄호`[]`를 통해 만들 수 있습니다.

값에 대한 접근은 `list[i]`를 통해 합니다.

In [168]:

```
L = []
print(type(L))
```

```
<class 'list'>
```

In [169]:

```
location = ['서울', '대전', '광주', '구미']
print(location)
```

```
['서울', '대전', '광주', '구미']
```

In [170]:

```
location[1]
```

Out[170]:

```
'대전'
```

`tuple`

활용법

```
    (value1, value2)
```

튜플은 리스트와 유사하지만, `()`로 묶어서 표현합니다.

그리고 tuple은 수정 불가능(immutable)하고, 읽을 수 밖에 없습니다.

직접 사용하는 것보다는 파이썬 내부에서 사용하고 있습니다.

In [174]:

```
tuple_ex = (1, 2)
print(type(tuple_ex))

# 아래와 같이 만들 수 있습니다.
is_tuple = 1, 2
print(is_tuple)
print(type(is_tuple))
```

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

In [175]:

```
# 파이썬 내부에서는 다음과 같이 활용됩니다.
# 앞선 2. 변수 및 자료형 예제에서 사용된 코드입니다.
x, y = 1, 2
print(x)
print(y)
```

```
1
2
```

In [179]:

```
x, y = (1, 2)
print(x)
print(y)
```

```
1
2
```

In [177]:

```
# 아래의 변수의 값을 swap하는 코드 역시 tuple을 활용하고 있습니다.
x, y = y, x
```

range()

레인지는 숫자의 시퀀스를 나타내기 위해 사용됩니다.

기본형 : `range(n)`

> 0부터 n-1까지 값을 가짐

범위 지정 : `range(n, m)`

> n부터 m-1까지 값을 가짐

범위 및 스텝 지정 : `range(n, m, s)`

> n부터 m-1까지 +s만큼 증가한다

In [180]:

```
type(range(1))
```

Out[180]:

Out[180]:

range

In [181]:

```
list(range(10))
```

Out[181]:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [182]:

```
list(range(4,9))
```

Out[182]:

[4, 5, 6, 7, 8]

In [183]:

```
list(range(0, -10, -1))
```

Out[183]:

[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

시퀀스에서 활용할 수 있는 연산자/함수

operation	설명
x in s	containment test
x not in s	containment test
s1 + s2	concatenation
s * n	n번만큼 반복하여 더하기
s[i]	indexing
s[i:j]	slicing
s[i:j:k]	k간격으로 slicing
len(s)	길이
min(s)	최솟값
max(s)	최댓값
s.count(x)	x의 갯수

In [186]:

```
# contain test
s = 'string'
print('a' in s)
L = [1, 2, 3, 6, 7]
print(3 in L)
```

False
True

In [187]:

```
# concatenation
print('안녕' + '하세요.')
print([1,2] + [4,5])
```

안녕하세요.
[1, 2, 4, 5]

In [188]:

```
[0]*6
```

Out[188]:

```
[0, 0, 0, 0, 0, 0]
```

In [189]:

```
# indexing과 slicing
location = ['서울', '대전', '광주', '구미']
location[1]
```

Out[189]:

```
'대전'
```

In [190]:

```
location[1:3]
```

Out[190]:

```
['대전', '광주']
```

In [199]:

```
r = list(range(30))
r[0:len(r):3]
```

Out[199]:

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

In [200]:

```
# 기타 내장함수
len(r)
```

Out[200]:

```
30
```

In [201]:

```
min(r)
```

Out[201]:

```
0
```

In [202]:

```
max(r)
```

Out[202]:

```
29
```

In [203]:

```
L = [1, 2, 1, 1, 2, 3]
L.count(1)
```

Out[203]:

3

set, dictionary

- `set`과 `dictionary`는 기본적으로 순서가 없습니다.

`set`

세트는 수학에서의 집합과 동일하게 처리됩니다.

세트는 중괄호 `{}`를 통해 만들며, 순서가 없고 중복된 값이 없습니다.

활용법

```
{value1, value2, value3}
```

연산자/함수	설명
a | b	합집합
a & b	교집합
a - b	차집합
a.union(b)	합집합
a.intersection(b)	교집합
a.difference(b)	차집합

In [205]:

```
set_a = {1, 2, 3}
set_b = {3, 6, 9}
print(set_a | set_b)
print(set_a & set_b)
print(set_a - set_b)
```

```
{1, 2, 3, 6, 9}
{3}
{1, 2}
```

In [207]:

```
set_c = set((1, 2, 3))
print(type(set_c))
```

```
<class 'set'>
```

- `set`을 활용하면 `list`의 중복된 값을 손쉽게 제거할 수 있습니다.

In [208]:

```
list_a = [1, 2, 3, 1, 2, 3]
set(list_a)
```

Out[208]:

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

In [209]:

```
list(set(list_a))
```

Out[209]:

```
[1, 2, 3]
```

`dictionary`

활용법

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

- 딕셔너리는 `key`와 `value`가 쌍으로 이뤄져있으며, 궁극의 자료구조입니다.
- `{}`를 통해 만들며, `dict()`로 만들 수도 있습니다.
- `key`는 immutable한 모든 것이 가능하다. (불변값 : string, integer, float, boolean, tuple, range)
- `value`는 `list`, `dictionary`를 포함한 모든 것이 가능하다.

In [211]:

```
dict_a = {}
print(type(dict_a))
dict_b = dict()
print(type(dict_b))
```

```
<class 'dict'>
<class 'dict'>
```

In [212]:

```
phone_book = {"서울": "02", "대전": "042"}
phone_book["서울"]
```

Out[212]:

```
'02'
```

In [214]:

```
# 중복된 key는 존재할 수가 없습니다.
dict_a = {1: 1, 2: 2, 3: 3, 1: 3, 1: 4}
print(dict_a)
```

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

In [217]:

```
# 딕셔너리의 메소드를 활용하여 key와 value를 확인 해볼 수 있습니다.
phone_book.keys()
```

Out[217]:

```
dict_keys(['서울', '대전'])
```

In [218]:

```
phone_book.values()
```

Out[218]:

```
dict_values(['02', '042'])
```

In [ ]:

```
# 실습! 친구 전화번호부 딕셔너리를 작성해주세요.
```

정리

데이터 타입

데이터 타입

Type Conversion