Python 기초

개요

본 강의 자료는 Python 공식 Tutorial에 근거하여 만들어졌으며, Python 3.8 버전에 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

기초 문법(Syntax)

주석(Comment)

- 한 줄 주석은 `#` 으로 표현한다.
- 여러 줄의 주석은
    1. 한 줄 씩 `#` 을 사용해서 표현하거나,
    2. `"""` 또는 `'''` (여러줄 문자열, multiline string)으로 표현할 수 있습니다. (multiline은 주로 함수/클래스를 설명(docstring)하기 위해 활용됩니다.)

In [ ]:
```
# b : 아래에 하나씩 추가됨
# ctrl+enter : 출력결과나옴
# a : 위에 하나 추가됨
# H  : 단축키 볼 수 있음
# M : 마크다운 문법 넣을 수 있음
# 왼쪽 파란색  : 명령모드
# 초록색으로 바뀌면 : 수정모드
# 명형 > 수정 : enter
# 수정 > 명령 : esc
```

In [ ]:
```
# 주석을 연습해봅시다.
# 3 + 2
```

In [ ]:
```
3+2
```

In [ ]:
```
#
# print('hello') <- 이 줄은 실행되지 않습니다.
```

In [ ]:
```
# 여러줄 주석을 multiline string을 활용하여 연습해봅시다.
```

In [ ]:
```
#
"""
이것은
여러줄에 걸친
주석을 만드는 코드입니다.
"""
```

코드 라인

- 파이썬 코드는 '1줄에 1문장(statment)'이 원칙이다.
- 문장(statement)은 파이썬이 실행 가능(executable)한 최소한의 코드 단위이다.
- 기본적으로 파이썬에서는 ; 을 작성하지 않는다.
- 한 줄로 표기할때는 ; 을 작성하여 표기할 수 있다.

In [ ]:
```
# print문을 두번 써봅시다.
```

In [ ]:
```
print('hello')
print('world')
```

In [ ]:
```
# print문을 한줄로 이어서 써봅시다. 오류 메시지를 확인해주세요.
```

In [ ]:
```
print('hello')print('world')
```

In [ ]:
```
# ;을 통해 오류를 해결해봅시다.
```

In [ ]:
```
print('hello');print('world')
```

In [ ]:
```
# 하지만 위와 같이 잘 쓰지는 않습니다.
```

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

In [ ]:
```
# print문을 통해 안되는 코드 예시 작성해봅시다.
```

In [ ]:
```
print('hello
world')
```

In [ ]:
```
# print문을 통해 되는 코드 예시 작성해봅시다.
```

In [ ]:
```
# 아래와 같은 동작이 가능 하지만 잘 쓰이지는 않습니다.
```

In [ ]:
```
```

```
print('hello\
world')     # 백슬래시를 넣어서 실행한다.
```

In [ ]:
```
## PEP-8 가이드에 따르면 여러줄 문자열은 아래와 같이 쓰는 게 관례(convention)입니다.
```

In [ ]:
```
##
print("""hello
world""")
```

- [] {} () 는 \ 없이도 가능하다.

In [ ]:
```
# list를 여러 줄에 걸쳐서 만들어봅시다.
```

In [ ]:
```
lunch = [
    '짜장면', '짬뽕', '탕수육',
    '군만두', '물만두', '왕만두',
]
print(lunch)
```

In [ ]:
```
## PEP-8 가이드에 따르면 여러 줄의 생성자의 닫히는 괄호(소, 중, 대)는
## (1) 첫번째 문자(요소) 위치에 오거나 (2) 마지막 줄에서 생성자가 시작되는 첫번째 열에 위치합
## 저희는 (1) 방식으로 사용할 것입니다.
```

변수(Variable)

변수

박스를 떠올리세요

= 60

dust 는 60이다 (X)

dust 에 60을 저장한다 (O)

할당 연산자(Assignment Operator): =

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

In [ ]:
```
# 변수에 값을 할당해봅시다.
```

In [ ]:
```
dust = 60
```

In [ ]:
```
dust2 = 60+10
```

In [ ]:
```
print(dust,dust2)
```

In [ ]:
```
# type()을 사용해봅시다.
```

In [ ]:
```
type(dust)
```

In [ ]:
```
name = 'Kim'
type(name)
```

In [ ]:
```
print(dust,type(dust))
```

In [ ]:
```
# id()를 사용해봅시다.
```

In [ ]:
```
id(dust)   # 컴퓨터에 특정한 공간에 들어있다. 주소처럼
```

- 같은 값을 동시에 할당할 수 있다.

In [ ]:
```
# 같은 값을 동시에 할당해봅시다.
```

In [ ]:
```
x = y = 10
print(x,y)
```

- 다른 값을 동시에 할당 가능하다.

In [ ]:
```
# 동시에 두개의 변수에 값 두개를 할당해봅시다.
```

In [ ]:
```
x, y = 1, 10
print(x,y)
```

In [ ]:
```
# 변수의 개수가 더 많을 때 오류를 알아봅시다.
```

In [ ]:
```
x, y, z = 1, 2
```

In [ ]:
```
# 변수의 개수가 더 적을 때 오류를 알아봅시다.
```

In [ ]:
```
x, y = 1
```

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

In [ ]:
```
# 변수 x와 y의 값을 바꿔봅시다. # 중요!!!!!!!!
```

In [ ]:
```
x = 10
y = 100
```

In [ ]:
```
# 임시 변수 활용
temp = x
x = y
y = temp
print(x, y)
```

In [ ]:
```
x = 10
y = 100
x, y = y, x
print(x,y)
```

식별자(Identifiers)

파이썬에서 식별자는 변수, 함수, 모듈, 클래스 등을 식별하는데 사용되는 이름(name)입니다.

- 식별자의 이름은 영문알파벳(대문자와 소문자), 밑줄(_), 숫자로 구성된다.
- 첫 글자에 숫자가 올 수 없다.
- 길이에 제한이 없다.
- 대소문자(case)를 구별한다.
- 아래의 키워드는 사용할 수 없다. 파이썬 문서

```
False, None, True, and, as, assert, async, await, 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
```

In [ ]:
```
# 키워드들을 직접 확인해봅시다.
```

In [ ]:
```
# import 구문은 모듈파트에서 다시 알아봅시다.
import keyword
print(keyword.kwlist)
```

- 내장함수나 모듈 등의 이름으로도 만들면 안된다.

In [ ]:
```
# 내장함수의 이름을 사용하면 어떤일이 일어나는지 확인해봅시다.
```

In [ ]:
```
# print는 값을 출력해주는 내장함수(Built-in function)입니다.
print(5)
```

In [ ]:
```
# 예시로 print에 값을 할당해보고, 오류를 확인해봅시다.
# print은 이제 'hi'라는 값으로 인식되기 때문에 이전의 기능을 수행하지 못합니다.
print = 'hi'
print(5)
```

In [ ]:
```
# 뒤에서 진행될 코드에 영향이 갈 수 있기 때문에 방금 생성한 print를 삭제합니다.
del print
```

In [ ]:
```
print('hi')
```

데이터 타입(Data Type)

- **숫자**(Number) 타입
- **글자**(String) 타입
- **참/거짓**(Boolean) 타입

숫자(Number) 타입

파이썬 문서

(1) `int` (정수, ingteger)

모든 정수는 `int` 로 표현됩니다.

파이썬 3.x 버전에서는 `long` 타입은 없고 모두 `int` 타입으로 표기 됩니다.

- python 3.x에서 long은 없어졌다.
- 보통 프로그래밍 언어 및 파이썬 2.x에서의 long은 OS 기준 32/64비트이다.
- 파이썬 3.x에서는 모두 int로 통합되었다.

8진수 : `0o` / 2진수 : `0b` / 16진수: `0x` 로도 표현 가능합니다.

In [ ]:
```
# 변수에 정수를 넣고 해당 변수의 type을 알아봅시다.
```

In [ ]:
```
a = 100
print(a,type(a))
```

In [ ]:
```
b = 100**100
print(b,type(b))
```

파이썬에서 표현할 수 있는 가장 큰 수

- 파이썬에서 가장 큰 숫자를 활용하기 위해 sys 모듈을 불러온다.
- 파이썬은 기존 C 계열 프로그래밍 언어와 다르게 정수 자료형(integer)에서 오버플로우가 없다.
- 임의 정밀도 산술(arbitrary-precision arithmetic)을 사용하기 때문이다.

> **오버플로우(overflow)**
>
> - 데이터 타입 별로 사용할 수 있는 메모리의 크기가 제한되어 있다.
> - 표현할 수 있는 수의 범위를 넘어가는 연산을 하게 되면, 기대했던 값이 출력되지 않는 현상, 즉 메모리가 차고 넘쳐 흐르는 현상
>
> **임의 정밀도 산술(arbitrary-precision arithmetic)**
>
> - 사용할 수 있는 메모리양이 정해져 있는 기존의 방식과 달리, 현재 남아있는 만큼의 가용 메모리를 모두 수 표현에 끌어다 쓸 수 있는 형태
> - 특정 값을 나타내는데 4바이트가 부족하다면 5바이트, 더 부족하면 6바이트까지 사용할 수 있게 유동적으로 운용

In [ ]:
```
import sys
max_int = sys.maxsize
# sys.maxsize 의 값은 2**63 - 1 => 64비트에서 부호비트를 뺀 63개의 최대치
print(max_int)
super_max = sys.maxsize * sys.maxsize
print(super_max)
print(type(super_max))
```

In [ ]:
```
# n진수를 만들어보고, 출력해봅시다.
```

In [ ]:
```

```

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

(2) `float` (부동소수점, 실수, floating point number)

실수는 `float` 로 표현됩니다.

다만, 실수를 컴퓨터가 표현하는 과정에서 부동소수점을 사용하며, 항상 같은 값으로 일치되지 않습니다. (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 있습니다.

In [ ]:
```
# 변수에 실수를 넣고 해당 변수의 type을 알아봅시다.
```

In [ ]:
```
c = 3.5
print(c,type(c))
```

컴퓨터식 지수 표현 방식

- e를 사용할 수도 있다. (e와 E 둘 중 어느 것을 사용해도 무방)

In [ ]:
```
# 컴퓨터식 지수 표현 방식을 사용해봅시다.
```

In [ ]:
```
pi = 314e-2
print(pi,type(pi))
```

실수의 연산

- 실수의 경우 실제로 값을 처리하기 위해서는 조심할 필요가 있다.

In [ ]:
```
# 실수의 덧셈을 해봅시다.
```

In [ ]:
```
3.5 + 3.2
```

In [ ]:
```
# 실수의 뺄셈을 해봅시다.
```

In [ ]:
```
3.5 - 3.12
```

In [ ]:
```
# 우리가 원하는대로 반올림을 해봅시다.
# round() 는 0~4는 내림, 5는 동일하게 작동하지 않고 반올림 방식에 따라 다릅니다.
```

```
# 짝수에서 5는 내림 / 홀수에서 5는 올림
```

In [ ]:
```
round(3.5-3.12,2)
```

In [ ]:
```
# 두 개의 값이 같은지 확인해봅시다.
```

In [ ]:
```
3.5 - 3.12 == 0.38
```

In [ ]:
```

```

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

In [ ]:
```
# 1. 기본적인 처리방법을 알아봅시다.
```

In [ ]:
```
a = 3.5 - 3.12
b = 0.38

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

In [ ]:
```
# 2. sys 모듈을 통해 처리하는 방법을 알아봅시다.
# `epsilon` 은 부동소수점 연산에서 반올림을 함으로써 발생하는 오차 상환
```

In [ ]:
```
import sys
abs(a-b) <= sys.float_info.epsilon
```

In [ ]:
```
# 3. python 3.5부터 활용 가능한 math 모듈을 통해 처리하는 법을 알아봅시다.
```

In [ ]:
```
import math
math.isclose(a,b)
```

(3) complex (복소수, complex number)

각각 실수로 표현되는 실수부와 허수부를 가집니다.

복소수는 허수부를 `j` 로 표현합니다.

In [ ]:
```
# 변수에 복소수를 넣고 해당 변수의 type을 알아봅시다.
```

In [ ]:
```
a = 3 - 4j   # j로 하는 것은 복소수
type(a)
```

In [ ]:
```
# 문자열을 복소수로 변환해봅시다.
```

In [ ]:
```

```

```
complex('1+2j')
```

In [ ]:
```
# 문자열을 변환할 때, 문자열은 중앙의 + 또는 - 연산자 주위에 공백을 포함해서는 안 됩니다.
```

In [ ]:
```
complex('1 + 2j') # ValueError
```

문자(String) 타입

기본 활용법

- 문자열은 Single quotes(')나 Double quotes(")을 활용하여 표현 가능하다.
  - 작은따옴표: '"큰" 따옴표를 담을 수 있습니다'
  - 큰따옴표: "'작은' 따옴표를 담을 수 있습니다"
  - 삼중 따옴표: '''세 개의 작은따옴표''' , """세 개의 큰따옴표"""
- 단, 문자열을 묶을 때 동일한 문장부호를 활용해야하며, PEP-8 에서는 **하나의 문장부호를 선택**하여 유지하도록 하고 있다. (Pick a rule and Stick to it)

In [ ]:
```
# 변수에 문자열을 넣고 출력해봅시다.
```

In [ ]:
```
name = 'ssafy'
name2 = 1
print(name, type(name))
print(name2,type(name2))
```

In [ ]:
```
# 사용자에게 받은 입력은 기본적으로 str입니다.
```

In [ ]:
```
name = input()
print(name,type(name))
```

따옴표 사용

문자열 안에 문장부호(' , ")가 사용될 경우 이스케이프 문자(\ )를 활용 가능 합니다.

In [ ]:
```
# 문자열 안에 문장부호를 활용해서 오류를 확인해봅시다.
```

In [ ]:
```
print('철수가 말했다. '안녕?'')
```

In [ ]:
```
# 오류를 이스케이프 문자와 서로 다른 문장부호를 통해 해결해봅시다.
```

In [ ]:
```
print('철수가 말했다. "안녕?"')
print('철수가 말했다. \'안녕?\'')
```

여러줄에 걸쳐있는 문장은 다음과 같이 표현 가능합니다.

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

In [ ]:
```
# 여러줄을 출력해봅시다.
```

In [ ]:
```
print("""
개행문자 없이
여러 줄을 그대로 출력 가능합니다.
""")
```

In [ ]:
```
# 문자열은 + 연산자로 이어붙이고, * 연산자로 반복시킬 수 있습니다.
```

In [ ]:
```
'hi' * 10
```

In [ ]:
```
'hi' + 'hong'
```

In [ ]:
```
# 당연히 변수화해서도 사용가능합니다.
```

In [ ]:
```
a = 'hi'
b = ',hong'
a+b
```

이스케이프 시퀀스

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

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

In [ ]:
```
# 이스케이프 문자열을 조합하여 프린트해봅시다.
```

In [ ]:
```
print('이 다음은 엔터.\n그리고 탭\t탭 \\n')
```

In [ ]:
```
# print를 하는 과정에서도 이스케이프 문자열을 활용 가능합니다.
```

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

In [ ]:
```
# 물론, end 옵션은 이스케이프 문자열이 아닌 다른 것도 가능합니다.
```

In [ ]:
```
print('개행 문자 말고도 가능합니다', end='!')
print('진짜로', end='!')
print('알고보면 print는 기본이 \\n', end='!')
```

In [ ]:
```
print('hi')
print('hong')
```

String interpolation

- %-formatting
  - %d : 정수
  - %f : 실수
  - %s : 문자열
- str.format()
- f-strings : 파이썬 3.6 이후 버전에서 지원

In [ ]:
```
# 문자열의 변수 값을 넣는 방법
```

In [ ]:
```
# name 변수에 이름을 입력해봅시다.
```

In [ ]:
```
name = 'Kim'
score = 4.5
```

In [ ]:
```
# %-formatting을 활용해봅시다.   # 첫번째 방법
```

In [ ]:
```
print('Hello, %s' % name)   # string
print('내 성적은 %d' % score)  # 정수
print('내 성적은 %f' % score)  # 실수
```

In [ ]:
```
# str.format()을 활용해봅시다.   # 두번째 방법
```

In [ ]:
```
print('Hello, {}. 내 성적은 {}'.format(name,score))
```

In [ ]:
```
# f-string을 활용해봅시다.       # 세번째 방법 3.6+
```

In [ ]:
```
print(f'Hello, {name}. 내 성적은 {score}. {score}지롱!')
```

In [ ]:
```
# 여러줄 문자열에서도 사용 가능합니다.
```

In [ ]:
```
#
print(f"""
Hello,
{name}
""")
```

- f-strings에서는 형식을 지정할 수 있다.

In [ ]:
```
# 다양한 형식을 활용하기 위해 datetime 모듈로 오늘을 표현해봅시다.
```

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

In [ ]:
```
# interpolation에서 출력형식을 지정할 수 있습니다.
# https://docs.python.org/ko/3/library/datetime.html#strftime-and-strptime-format-codes
```

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

- f-strings에서는 연산과 출력형식 지정도 가능하다.

In [ ]:
```
# string interpolation을 통해 출력형식 지정 뿐만 아니라, 연산도 가능합니다.
```

In [ ]:
```
pi = 3.141592
f'원주율은 {pi:.5}! 반지름이 2일때 원의 넓이는 {pi*2*2}'
```

참/거짓(Boolean) 타입

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

비교/논리 연산을 수행 등에서 활용됩니다.

다음은 `False` 로 변환됩니다.

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

In [ ]:
```
# True와 False의 타입들을 알아봅시다.
```

In [ ]:
```
print(type(True))
print(type(False))
```

In [ ]:
```
# 다양한 True, False 상황들을 확인해봅시다.
# 형변환(Type Conversion)에서 추가적으로 다루는 내용입니다.
```

In [ ]:
```
bool(0)
```

In [ ]:
```
bool(0.1)
```

In [ ]:
```
bool([])
```

In [ ]:
```
bool({})
```

In [ ]:
```
bool(None)
```

In [ ]:
```
bool('')
```

In [ ]:
```
bool('0')
```

In [ ]:
```

```

None 타입

파이썬에서는 값이 없음을 표현하기 위해 None 타입이 존재합니다.

In [ ]:
```
# None의 타입을 알아봅시다.
```

In [ ]:
```
print(type(None))
```

In [ ]:
```
# 변수에 저장해서 확인해봅시다.
```

In [ ]:
```
a = 3
a = None
print(a)
```

형변환(Type conversion, Typecasting)

파이썬에서 데이터타입은 서로 변환할 수 있습니다.

- 암시적 형변환
- 명시적 형변환

암시적 형변환(Implicit Type Conversion)

사용자가 의도하지 않았지만, 파이썬 내부적으로 자동으로 형변환 하는 경우입니다. 아래의 상황에서만 가능합니다.

- bool
- Numbers (int, float, complex)

bool형으로 자동형변환 가능 : 0 (False), 1 (True) bool형으로 자동형변환 불가 : 나머지 숫자들 .. bool(3) 이런식으로 해줘야함

In [ ]:
```
# 자동으로 변환
```

In [ ]:
```
# boolean과 integer는 더할 수 있을까요?
```

In [ ]:
```
True + 5    # True는 integer로 바꿨을 때 1이다.
```

In [ ]:
```
# int, float, complex를 각각 변수에 대입해봅시다.
```

In [ ]:
```
int_number = 3
float_number = 5.0
complex_number = 3+5j
```

In [ ]:
```
# int와 float를 더해봅시다. 그 결과의 type은 무엇일까요?
```

In [ ]:
```
result = int_number + float_number
print(result, type(result))
```

In [ ]:
```
# int와 complex를 더해봅시다. 그 결과의 type은 무엇일까요?
```

In [ ]:
```
result = int_number + complex_number
print(result, type(result))
```

명시적 형변환(Explicit Type Conversion)

위의 상황을 제외하고는 모두 명시적으로 형 변환을 해주어야합니다.

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

암시적 형변환이 되는 모든 경우도 명시적으로 형변환이 가능합니다.

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

`list()`, `tuple()` 등은 다음 챕터에서 배울 예정입니다.

In [ ]:
```
# 사용자가 의도적으로 형변환
```

In [ ]:
```
# integer와 string 사이의 관계는 명시적으로 형변환을 해줘야만 합니다.
```

In [ ]:
```
str(1) + '등'
```

In [ ]:
```

```

In [ ]:
```
# string 3을 integer로 변환해봅시다.
```

In [ ]:
```
a = '3'
int(a)
```

In [ ]:
```
# string 3.5를 float로 변환해봅시다.
```

In [ ]:
```
a = '3.5'
float(a)
```

In [ ]:
```
# string은 글자가 숫자일때만 형변환이 가능합니다.
```

In [ ]:
```
a = 'hi'
int(a)
```

In [ ]:
```
# string 3.5를 int로 변환할 수는 없습니다.
```

In [ ]:
```
a = '3.0'  # 3.5리는 문자열은 float으로만 변환 가능
int(a)
```

In [ ]:
```
a = '3.5'  # 3.5리는 문자열은 float으로만 변환 가능
float(a)
```

In [ ]:
```
# float 3.5는 int로 변환이 가능합니다.
```

In [ ]:
```
a = 3.7
int(a)
```

In [ ]:
```
# 모든 거는 문자열표현가능
# float는 int로 변환 가능
```

연산자(Operator)

- 산술 연산자
- 비교 연산자
- 논리 연산자
- 복합 연산자
- 기타 연산자

산술 연산자

Python에서는 기본적인 사칙연산이 가능합니다.

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

- 나눗셈 (/) 은 항상 float를 돌려준다.
- 정수 나눗셈 으로 (소수부 없이) 정수 결과를 얻으려면 // 연산자를 사용한다.

In [ ]:
```
# 2의 1000승을 확인해봅시다.
```

In [ ]:
```
2 ** 1000
```

In [ ]:
```
# 나눗셈과 관련된 산술연산자를 활용해봅시다.
```

In [ ]:
```
print(5 / 2)
print(5 // 2)
print(int(5 / 2))
print(5 % 2)
```

In [ ]:
```
# divmod는 나눗셈과 관련된 함수입니다.
```

In [ ]:
```
quotient, remainder = divmod(5, 2)
print(f'몫은 {quotient}, 나머지는 {remainder}')
```

In [ ]:
```
# 음수 양수 표현도 해봅시다.
```

In [ ]:
```
positive_num = 4
print(-positive_num)

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

비교 연산자

우리가 수학에서 배운 연산자와 동일하게 값을 비교할 수 있습니다.

연산자	내용

연산자	내용
<	미만
<=	이하
>	초과
>=	이상
==	같음
!=	같지않음
is	객체 아이덴티티
is not	부정된 객체 아이덴티티

In [ ]:
```
# 숫자의 대소관계를 비교해봅시다.
```

In [ ]:
```
3 < 2
```

In [ ]:
```
# 다른 숫자인지 확인해봅시다.
```

In [ ]:
```
3 != 3
```

In [ ]:
```
# 같은 숫자인지 확인해봅시다.
```

In [ ]:
```
3 == 3
```

In [ ]:
```
# 문자열도 같은지 확인해봅시다.
```

In [ ]:
```
'hi' == 'Hi'
```

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 연산자입니다.

In [ ]:
```
# and과 관련해서 모든 case를 출력해봅시다.
```

In [ ]:
```
print(True and True)
print(True and False)
```

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

In [ ]:
```
# or과 관련해서 모든 case를 출력해봅시다.
```

In [ ]:
```
print(True or True)
print(True or False)
print(False or True)
print(False or False)
```

In [ ]:
```
# not을 활용해봅시다.
```

In [ ]:
```
print(not True)
print(not 0)
```

- 파이썬에서 and는 a가 거짓이면 a를 리턴하고, 참이면 b를 리턴한다.
- 파이썬에서 or은 a가 참이면 a를 리턴하고, 거짓이면 b를 리턴한다.

단축평가

- 첫 번째 값이 확실할 때, 두 번째 값은 확인 하지 않음
- 조건문에서 뒷 부분을 판단하지 않아도 되기 때문에 속도 향상

In [ ]:
```
'a' and 'b'
```

In [ ]:
```
'a' or 'b'
```

In [ ]:
```
vowels = 'aeiou'
```

In [ ]:
```
('a' and 'b') in vowels    # 모음
```

In [ ]:
```
('a' and 'b')
```

In [ ]:
```
('b' and 'a') in vowels   # 모음
```

In [ ]:
```
('b' and 'a')
```

- and 는 둘 다 True일 경우만 True이기 때문에 첫번째 값이 True라도 두번째 값을 확인해야 하기 때문에 'b'가 반환된다.
- or 는 하나만 True라도 True이기 때문에 True를 만나면 해당 값을 바로 반환한다.

In [ ]:
```
# and의 단축평가(short-circuit evaluation)에 대해서 알아봅시다.  # and는 둘다 True여야 True
```

In [ ]:
```
```

```
# and : 둘다 True여야 True
# 첫번째 True, 두 번째 것도 확인해야함
print(3 and 5)   # 3이True니까 5까지 검사해서 출력이 5가 된 것
print(3 and 0)
print(0 and 3)  # 이미 False가 나왔으니까 3까지 확인할 필요는 없으니까 0이 나옴
print(0 and 0)
```

In [ ]:
```
# or의 단축평가(short-circuit evaluation)에 대해서 알아봅시다.  # or는 둘 중 하나만 True이면
```

In [ ]:
```
print(3 or 5)
print(3 or 0)
print(0 or 3)   # 0은 False이다.
print(0 or 0)
```

복합 연산자

복합 연산자는 연산과 대입이 함께 이뤄집니다.

가장 많이 활용되는 경우는 반복문을 통해서 개수를 카운트하거나 할 때 활용됩니다.

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
a **= b	a = a ** b

In [ ]:
```
# 복합연산자는 이럴 때 사용됩니다.
```

In [ ]:
```
cnt = 0
while cnt < 5:
    print(cnt)
    cnt += 1  # cnt = cnt + 1
```

기타 주요 연산자

Concatenation

숫자가 아닌 자료형은 + 연산자를 통해 합칠 수 있습니다.

In [ ]:
```
# 문자열을 더해봅시다.(합쳐봅시다.)
```

In [ ]:
```
'abc' + 'efg'
```

In [ ]:
```
# list를 더해봅시다.(합쳐봅시다.)
```

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

Containment Test

`in` 연산자를 통해 요소가 속해있는지 여부를 확인할 수 있습니다.

In [ ]:
```
# 문자열안에 특정한 문자가 있는지 확인해봅시다.
```

In [ ]:
```
'a' in 'apple'
```

In [ ]:
```
# list안에 특정한 원소가 있는지 확인해봅시다.
```

In [ ]:
```
1 in [1,2,3]
```

In [ ]:
```

```

In [ ]:
```
# range안에 특정한 원소가 있는지 확인해봅시다.
```

In [ ]:
```
1 in range(1,5)
```

Identity

`is` 연산자를 통해 동일한 object인지 확인할 수 있습니다. (OOP 파트에서 다시 학습하게 됩니다.)

In [ ]:
```
# 특정 문자열과 특정 범위의 숫자는 '조금 더 빨리 메모리에서 찾기 위해서' 같게끔 설정하였다.
# == : '값' 자체에 집중, 값 자체가 같은지 물어보는 것
# is : '오브젝트' 가 같은지 물어보는 것
"""
-5 is -5
10 is 10
이라는것
"""
```

In [ ]:
```
# 파이썬에서 -5 부터 256 까지의 id는 동일합니다.
```

In [ ]:
```
id(5)
```

In [ ]:
```
# 느낌표 때문에 같지 않다..
a = 'hi!'
b = 'hi!'
a is b
```

In [ ]:
```
# 의도적으로.. 공백없는 알파벳 문자열도 같게끔 해놨다.
a = 'hi'
b = 'hi'
a is b
```

In [ ]:
```
# 버그나 오류가 아니라 특정 범위의 숫자를 id를 같게끔 해놨다.
a = 1
b = 1
print(a is b)
print(id(a), id(b))
```

In [ ]:
```
a = 1004
b = 1004
print(a is b)
print(id(a), id(b))
```

In [ ]:
```

```

In [ ]:
```
# 257 이루의 id 는 다릅니다.
```

In [ ]:
```
a = []
b = []
# 값을 비교하면, 빈 리스트라서 같은데
# is를 통해서 object, 다르다.. 즉, 값은 같은데 위치해있는 곳이 다르다!
print(a == b, a is b)
print(id(a),id(b))
```

In [ ]:
```

```

Indexing/Slicing

`[]` 를 통한 값을 접근하고, `[:]` 을 통해 리스트를 슬라이싱할 수 있습니다. (Container 파트에서 자세하게 학습합니다.)

In [ ]:
```
# 문자열을 인덱싱을 통해 값에 접근해봅시다.
```

In [ ]:
```
a = 'samsung'
a[0]
```

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`

파이썬 문서

In [ ]:
```
# 우선순위를 확인해봅시다.
```

In [ ]:
```
-3 ** 6
```

In [ ]:
```
(-3) ** 6
```

[참고] 표현식(Expression) & 문장(Statement)

표현식(Expression)

> 표현식 => evaluate => 값

- 하나의 값(value)으로 환원(reduce)될 수 있는 문장
- 식별자, 값 (리터럴), 연산자 로 구성됩니다.
- 표현식을 만드는 문법(syntax)은 일반적인 (중위표기) 수식의 규칙과 유사합니다.

파이썬 문서

In [ ]:
```
# 표현식에 대해 알아봅시다.
```

In [ ]:
```
# 하나의 값(value)도 표현식(expression)이 될 수 있습니다.
'hello'
```

In [ ]:
```
# 표현식은 하나의 값으로 평가(evaluate)될 수 있어야 합니다. 그러면 할당문(assignment stateme
radius = 10
```

In [ ]:
```
# 식별자가 값이 할당되어 있는 경우 수식의 일부가 될 수 있습니다.
3.14 * (radius - 5) ** 2
```

In [ ]:
```
# 표현식을 만드는 문법(syntax)은 일반적인 (중위표기) 수식의 규칙과 유사합니다. 아래와 같은 문
4 +
```

문장(Statement)

- 파이썬이 실행 가능한 최소한의 코드 단위 (a syntatic unit of programming)

In [ ]:
```
# 문장에 대해 알아봅시다.
```

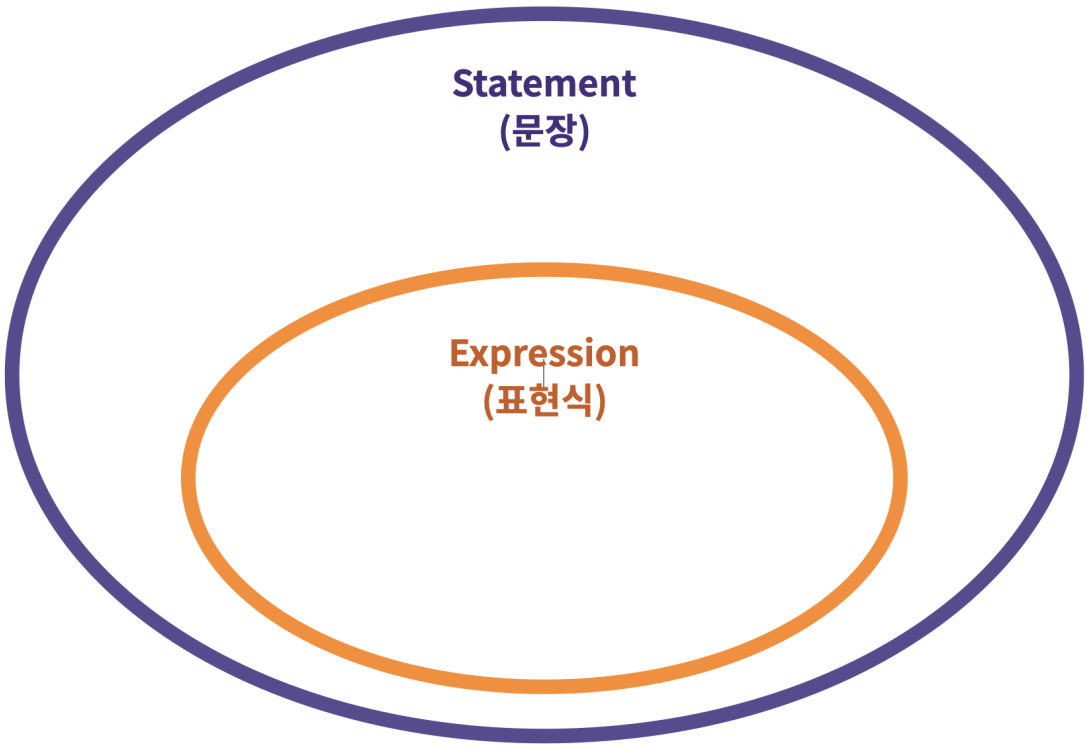
In [ ]:
```
# 하나의 값(value)도 문장이 될 수 있습니다.
'ssafy'
```

In [ ]:
```
# 표현식(expression)도 문장이 될 수 있습니다.
5 * 21 - 4
```

In [ ]:
```
# 실행 가능(executable)해야 하기 때문에 아래의 코드는 문장이 될 수 없습니다.
```

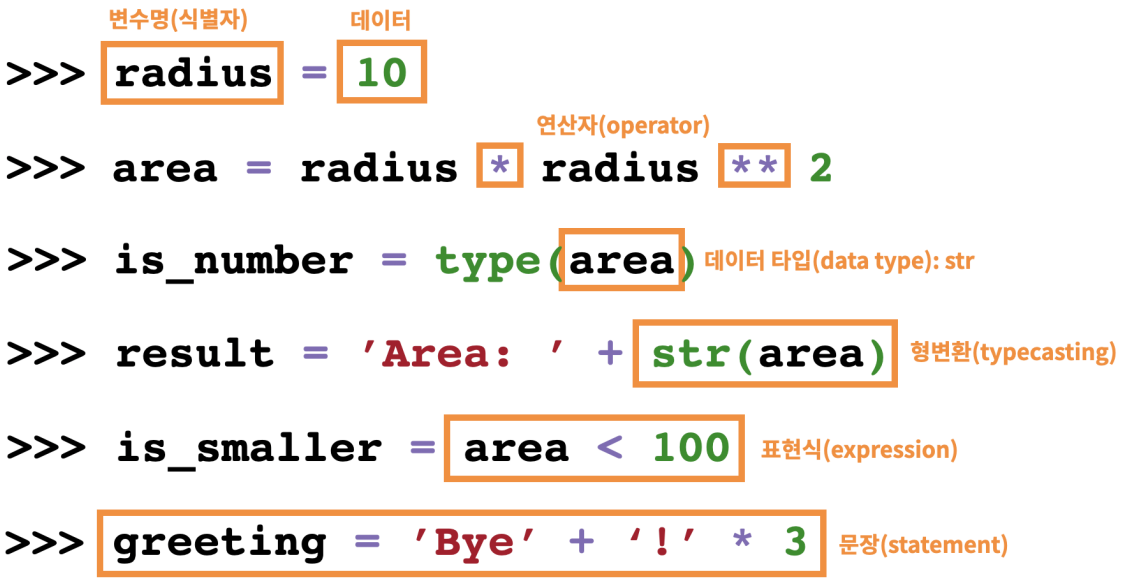
In [ ]:
```
name = '   # 문장이 아니다.
```

문장과 표현식의 관계



정리

변수(Variable)와 자료형(Data Type)


변수명(식별자)
데이터
>>> radius = 10
연산자(operator)
>>> area = radius * radius ** 2
>>> is_number = type(area)
데이터 타입(data type): str
>>> result = 'Area: ' + str(area)
형변환(typecasting)
>>> is_smaller = area < 100
표현식(expression)
>>> greeting = 'Bye' + '!' * 3
문장(statement)