

Python Programming

Fundamental

Python 시작하기

소개

파이썬이란?

- ▶ 1991년 귀도 반 로섬(Guido Van Rossum)이 개발한 고급 프로그래밍 언어
- ▶ 플랫폼 독립적, 인터프리터 방식, 객체지향적, 동적 타이핑 대화형 언어
- ▶ 많은 상용 응용 프로그램에서 스크립트 언어로 채용
- ▶ 과학 기술 컴퓨팅, 공학 분야에서도 널리 이용
 - ▶ Pyrex, Psyco, Numpy 등 관련 패키지 이용



파이썬의 특징

- ▶ 대화형 인터프리터 언어
- ▶ 동적타이핑(동적인 데이터 타입 결정) 지원
- ▶ 플랫폼 독립적 언어
- ▶ 간단하고 쉬운 문법
- ▶ 높은 가독성
- ▶ 비교적 짧은 개발 시간
- ▶ 고수준 내장 객체 자료형(List, Dictionary, Tuple 등 자료 구조)
- ▶ 메모리 자동 관리
- ▶ 풍부한 라이브러리
- ▶ 높은 확장성 (Glue Language)
- ▶ 유니코드
- ▶ 무료 (파이썬 재단이 관리하는 개방형, 공동체 기반 개발 모델)

간단하고 쉬운 문법, 높은 가독성

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '%s [%label=%s]' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s';' % ast[1]
        else:
            print ''
    else:
        print '';
    children = []
    for n, child in enumerate(ast[1:]):
        children.append(dotwrite(child))
    print '%s -> {' % nodename
    for name in children:
        print '%s' % name,
```

높은 확장성: Glue Language

- ▶ 언어 자신의 기능은 작게 유지
 - ▶ 사용자가 언제나 필요로 하는 최소한의 기능만을 제공하도록 설계
- ▶ 속도나 성능이 필요한 기능은 타 언어(C, C++ 등)로 구현,
 - ▶ 파이썬에서는 전반적인 뼈대만 구성

파이썬의 종류: 구현체

| 명칭 | 설명 |
|------------|------------------------------|
| CPython | C로 작성된 파이썬 인터프리터 (*) |
| Jython | Java로 작성된 파이썬 인터프리터 |
| IronPython | .NET 플랫폼용 파이썬 인터프리터. C#으로 구현 |
| PyPy | Python으로 작성된 파이썬 인터프리터 |

파이썬의 종류 : 버전

- ▶ 2.x
 - ▶ 2000년 10월 16일 배포
 - ▶ 2017년 현재 2.7.14
 - ▶ 기 개발된 것들이 많아 현재도 많이 사용중
 - ▶ 2.8 버전은 배포 예정이 없으며, 버전 2는 2020년까지만 지원할 예정
- ▶ 3.x
 - ▶ 2008년 12월 3일 배포 -> 현재 최신 버전
 - ▶ 2.x 버전과의 차이
 - ▶ 사전형, 문자열형 등 내장 자료형의 변화
 - ▶ 구 버전의 비효율적 구성 요소 제거
 - ▶ 표준라이브러리 재배치
 - ▶ Unicode 체계 변경

파이썬 활용분야

- ▶ 시스템 유틸리티
 - ▶ 운영체제의 시스템 명령어들을 이용할 수 있는 각종 도구를 갖춤
- ▶ GUI
 - ▶ Tcl/tk를 이용한 UI, wxPython(Windows 인터페이스)
- ▶ 웹 프로그래밍
 - ▶ Django, Flask
- ▶ 데이터베이스 프로그래밍
 - ▶ SQLite 내장, Oracle, DB2, Sybase, MySQL 등 DB 시스템 인터페이스 제공
- ▶ 텍스트 처리
 - ▶ 뛰어난 문자열 처리, 정규식, XML 처리

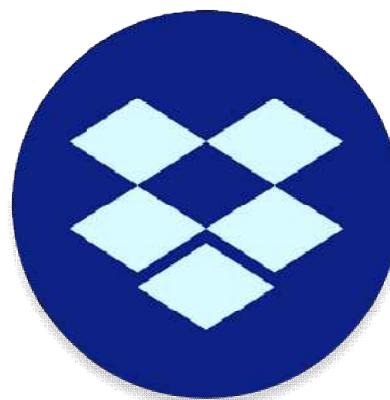
파이썬 활용분야

- ▶ 데이터 분석
 - ▶ Numpy, Pandas 라이브러리를 활용한 데이터 분석
 - ▶ Matplotlib 라이브러리를 활용한 그래프, 또는 2차원 Data Visualization
 - ▶ SciPy를 활용한 과학/공학 계산
- ▶ 병렬 연산
 - ▶ IPython을 이용한 병렬 연산
- ▶ 사물 인터넷
 - ▶ 라즈베리 파이를 이용한 사물 인터넷 프로토타이핑

주요 프로젝트

- ▶ BitTorrent, Trac, Yum
- ▶ Flask, CherryPy, Django
- ▶ GIMP, Maya, Paint Shop Pro
- ▶ Youtube, Google Groups, Google maps, Gmail 등

Google



Polyglot

TIOBE Index: 2017 November

| Nov 2017 | Nov 2016 | Change | Programming Language | Ratings | Change |
|----------|----------|--------|----------------------|---------|--------|
| 1 | 1 | | Java | 13.231% | -5.52% |
| 2 | 2 | | C | 9.293% | +0.09% |
| 3 | 3 | | C++ | 5.343% | -0.07% |
| 4 | 5 | ▲ | Python | 4.482% | +0.91% |
| 5 | 4 | ▼ | C# | 3.012% | -0.65% |
| 6 | 8 | ▲ | JavaScript | 2.972% | +0.27% |
| 7 | 6 | ▼ | Visual Basic .NET | 2.909% | -0.26% |
| 8 | 7 | ▼ | PHP | 1.897% | -1.23% |
| 9 | 16 | ▲ | Delphi/Object Pascal | 1.744% | -0.21% |
| 10 | 9 | ▼ | Assembly language | 1.722% | -0.72% |
| 11 | 19 | ▲ | R | 1.605% | -0.11% |
| 12 | 15 | ▲ | MATLAB | 1.604% | -0.36% |
| 13 | 14 | ▲ | Ruby | 1.593% | -0.39% |
| 14 | 13 | ▼ | Go | 1.570% | -0.43% |
| 15 | 10 | ▼ | Perl | 1.562% | -0.80% |
| 16 | 26 | ▲ | Scratch | 1.550% | +0.47% |
| 17 | 17 | | Visual Basic | 1.489% | -0.43% |
| 18 | 20 | ▲ | PL/SQL | 1.453% | -0.06% |
| 19 | 11 | ▼ | Objective-C | 1.412% | -0.83% |
| 20 | 12 | ▼ | Swift | 1.389% | -0.65% |

”Life is too short, You need Python”

Python 시작하기

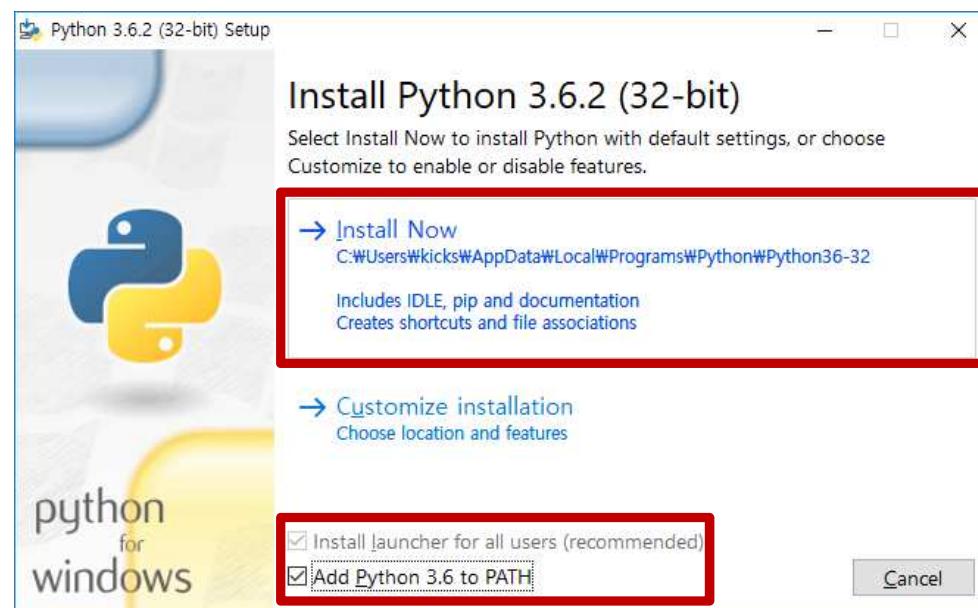
설치

파이썬 다운로드/설치

맥, 리눅스 사용자라면 직접 설치보다
pyenv 등 환경 구성 도구를 이용하면 편리

▶ <https://www.python.org/downloads/>

- ▶ 컴퓨터 환경에 맞는 버전 다운로드
- ▶ 하단 Add Python 3.6 to PATH 반드시 체크



파이썬 설치 요소

▶ Python (Command Line Interface)

- ▶ cmd.exe 를 실행하고 python을 실행
- ▶ ^D(Ctrl+D) 혹은 quit() 입력하면 인터페이스 종료

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\kicks>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print( "Hello World" )
Hello World
>>> 10 + 20
30
>>> 2**10
1024
>>>
```

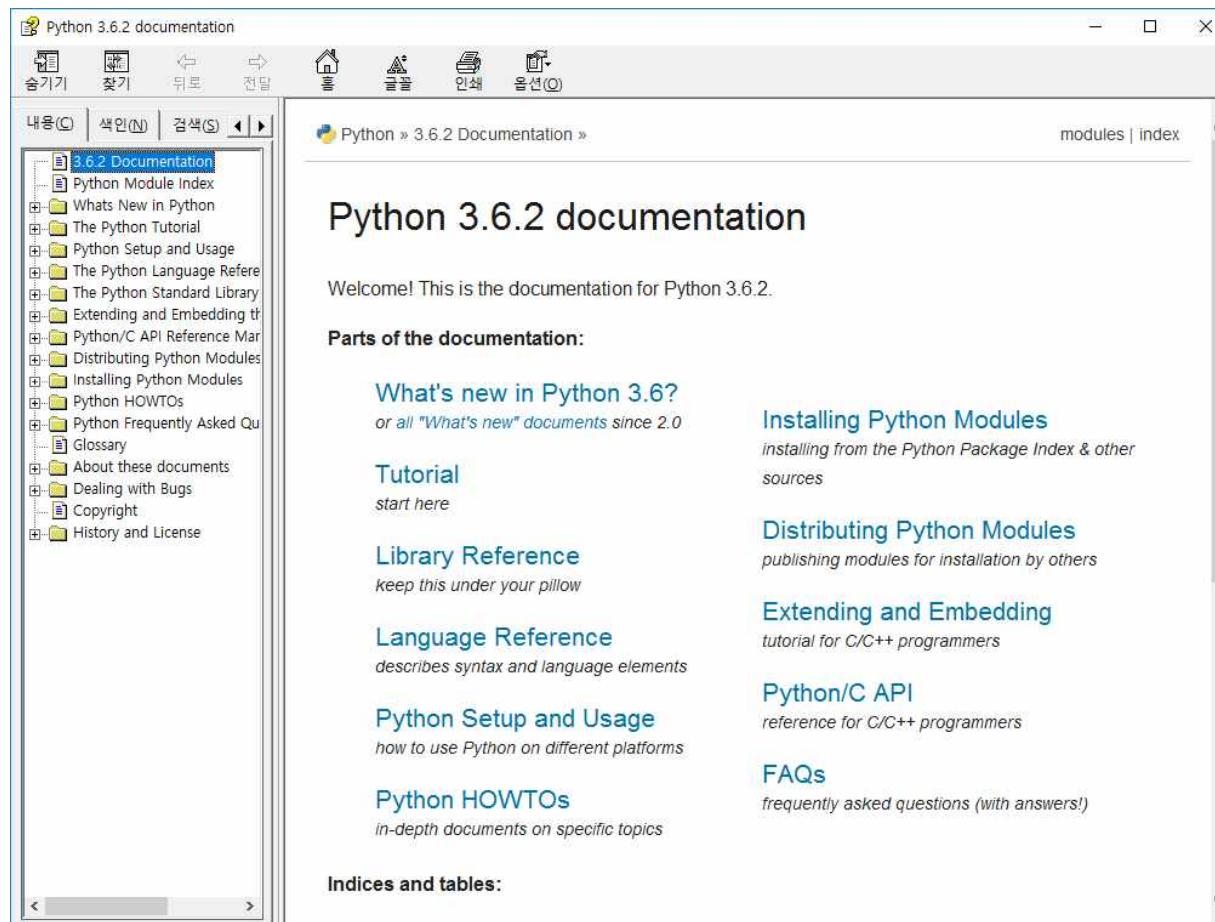
파이썬 설치 요소

▶ Module Docs



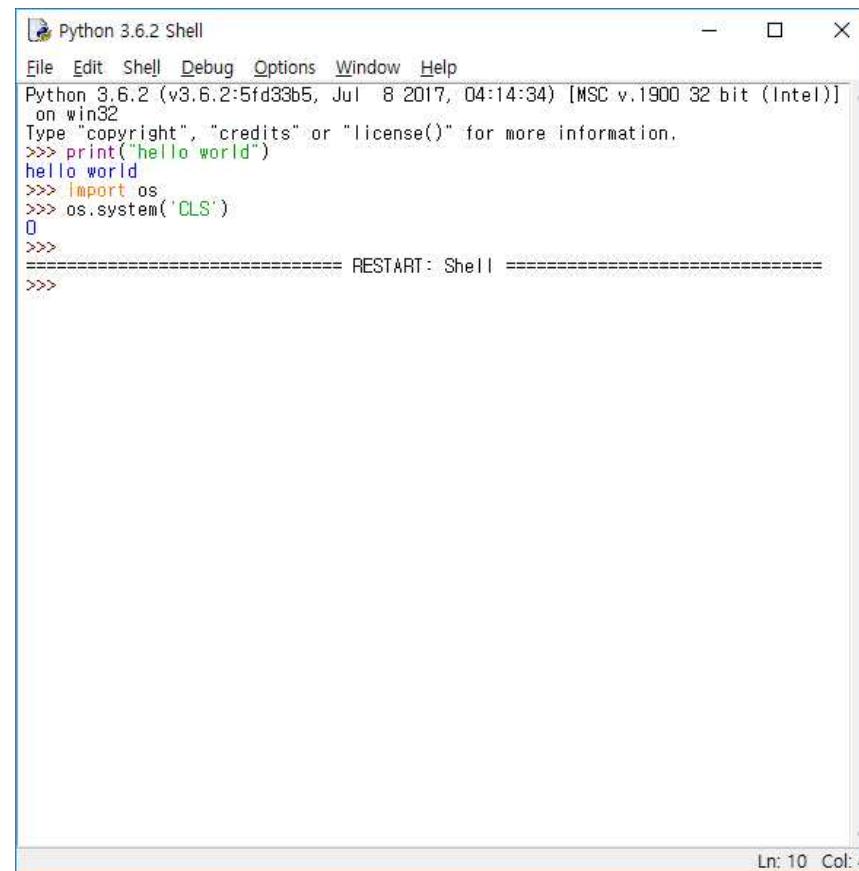
파이썬 설치 요소

▶ Python Manuals



파이썬 설치 요소

▶ IDLE (Python GUI)



```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("hello world")
hello world
>>> import os
>>> os.system('CLS')
0
>>>
===== RESTART: Shell =====
>>>
```

Ln: 10 Col: 4

파이썬 설치 확인

- ▶ 실습: 파이썬 버전 확인
 - ▶ cmd.exe 혹은 powershell에서 Python 버전 확인 (--version 옵션)
- ▶ 실습: 간단한 산술 연산 실행
 - ▶ IDLE을 이용, 간단한 산술 계산 실행
- ▶ 실습: Python의 규약을 살펴보자
 - ▶ Python CLI에 다음 라인을 입력

```
>>> import this
```

import문은 모듈편에서 자세히 설명

Python 입출력 (I)

Console 입출력

Console 출력

: print 함수

- ▶ 콘솔 화면 출력을 위해서는 `print()` 함수를 사용
- ▶ 인자의 개수, 형식도 제한이 없음
- ▶ 내부적으로는 인자 객체의 `__str__` 메서드를 실행

```
>>> print(1)
1
>>> print("hello", "python")
hello python
>>> x = 0.2
>>> s = "hello"
>>> print(x, s) # 다른 타입의 인자도 함께 전달 가능
0.2 hello
```

Console 출력

: print 함수

- ▶ 공백 대신 두 파라미터를 +로 연결하는 방법이 있으나 객체 내에 + 연산자가 오버라이딩되어 있어야 한다
- ▶ 오류 발생시 캐스팅(형 변환)으로 해결 가능(str)

```
>>> x = 0.2
>>> s = "Hello"
>>> print(x + s)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'float' and
'str'
>>> print(str(x) + " " + s) # 수치형을 문자형으로 변환(캐스팅)
0.2 Hello
```

Console 출력

: print 함수 - sep, end

| 파라미터 | 용례 | 기본값 |
|------|------------|-----------|
| sep | sep = ',' | ' ' (공백) |
| end | end = '\n' | '\n' (개행) |

- ▶ sep : 출력 객체 사이에 표시할 문자
- ▶ end : 출력의 마지막에 출력할 문자

```
>>> x = 0.2
>>> s = "Hello"
>>> print(x, s, sep = ', ', end = '\n')
0.2,Hello
```

Console 입력

: input 함수

- ▶ `Input()` 함수를 이용, 사용자의 키보드 입력을 받을 수 있음
- ▶ 화면에 출력할 프롬프트를 `input` 함수의 인자로 줄 수 있다
- ▶ 결과값은 문자열 객체를 반환

```
>>> name = input("What is your name?: ")  
What is your name?: Nam  
>>> print("Hello", name)  
Hello Nam
```

Console 입출력

▶ 실습: Hello Python

- ▶ IDLE에서 다음 코드를 작성하고 실행
- ▶ 에러가 없으면 `hello.py`로 저장하고 Command Line에서 파일을 실행해 봅시다

```
>>> print("Hello Python")
```

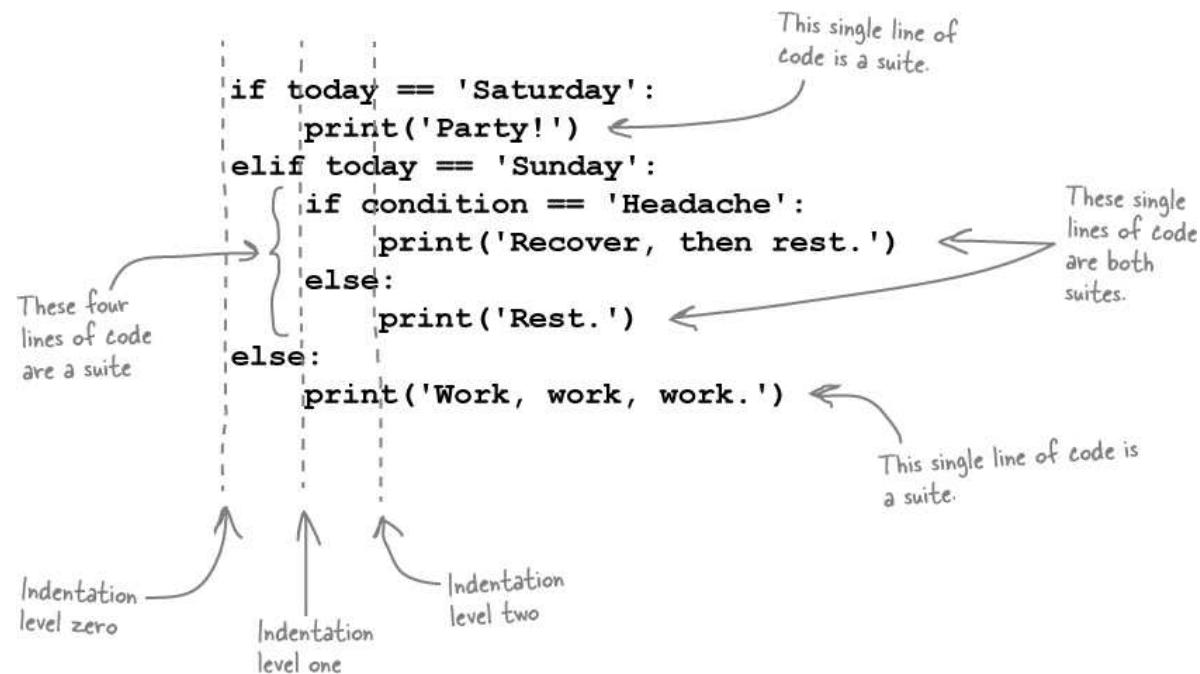
Python 시작하기

프로그램의 작성과 실행



들여쓰기(Indent)

- ▶ 파이썬 프로그램 작성시 가장 주의해야 할 사항
- ▶ 들여쓰기를 잘못하면 `IndentationError`를 발생한다



들여쓰기 규칙

- ▶ 가장 바깥쪽에 있는 블록의 코드는 1열부터 시작한다

```
>>> a = 1
>>> a = 1
File "<stdin>", line 1
    a = 1
^
IndentationError: unexpected indent
```

들여쓰기 규칙

- ▶ 내부 블록은 같은 거리만큼 들여쓰기 해야 한다

```
>>> if (a > 1):  
...     print("big")  
...         print("really?")  
File "<stdin>", line 3  
     print("really?")  
     ^  
IndentationError: unexpected indent
```

들여쓰기 규칙

- ▶ 블록은 들여쓰기로 결정된다
- ▶ 탭과 공백을 함께 쓰는 것은 권장하지 않는다
- ▶ 들여쓰기 간격은 일정하기만 하면 된다(4 spaces 추천)

파이썬 실행: 대화식 모드

- ▶ 대화식 모드
 - ▶ 커맨드 라인에 `python` 타이핑
 - ▶ 명령을 입력하고 바로 결과를 확인할 수 있다

```
>>> import sys
>>> sys.version
'3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]'
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=2, releaselevel='final', serial=0)
>>>
```

파이썬 실행: 스크립트 실행 모드

- ▶ 스크립트 실행 모드
 - ▶ 파이썬 명령 모음 파일을 작성한 후 .py 확장자로 저장
 - ▶ `python {파일명}` 커맨드를 이용하여 실행

```
python hello.py
```

파이썬 실행: 스크립트 실행모드

- ▶ 다음 코드를 에디터에 작성하고 `cal.py` 이름으로 저장

```
# file: cal.py
import calendar
print(calendar.month(2022, 2))
```

- ▶ 작성한 파일을 실행

```
python cal.py
```

| February 2022 | | | | | | |
|---------------|----|----|----|----|----|----|
| Mo | Tu | We | Th | Fr | Sa | Su |
| | | | | | 1 | 2 |
| | | | | | 3 | 4 |
| | | | | | 5 | 6 |
| | | | | | 7 | 8 |
| | | | | | 9 | 10 |
| | | | | | 11 | 12 |
| | | | | | 13 | |
| | | | | | 14 | 15 |
| | | | | | 16 | 17 |
| | | | | | 18 | 19 |
| | | | | | 20 | |
| | | | | | 21 | 22 |
| | | | | | 23 | 24 |
| | | | | | 25 | 26 |
| | | | | | 27 | |
| | | | | | 28 | |

주석 (Comment) :

- ▶ # 이후의 내용은 인터프리터가 해석하지 않는다

```
>>> # 이것은 주석입니다
```

```
>>> 3 + 5 # 이 표시 뒤의 내용은 해석하지 않고 건너뜁니다
```

파이썬은 공식적으로 여러 줄 주석을 허용하지 않는다.

Python 프로그래밍 기초

산술연산자

산술연산자

| 연산자 | 사용예 | 기능 |
|-----|----------|----------------|
| + | $1 + 2$ | 덧셈 |
| - | $3 - 1$ | 뺄셈 |
| * | $4 * 3$ | 곱셈 |
| / | $4 / 3$ | 나눗셈 |
| // | $4 // 3$ | 나눗셈의 몫 (정수나누기) |
| % | $4 \% 3$ | 나눗셈의 나머지 |
| ** | $2 ** 3$ | 제곱 |

/는 2.* 과 3.*에서 다소 다르게 작동함에 유의

산술 연산자

```
>>> 1 + 2 # 덧셈
3
>>> 3 - 1 # 뺄셈
2
>>> 4 * 3 # 곱셈
12
>>> 4 / 3 # 나눗셈
1.333333333333333
>>> 4 // 3 # 나눗셈의 몫
1
>>> 4 % 3 # 나눗셈의 나머지
1
>>> 2 ** 3 # 제곱
8
```

복소수

- ▶ 실수부 + 허수부로 구성
- ▶ 허수부는 j 혹은 J 로 표기

```
>>> type(3 - 4j)
<class 'complex'>
>>> (3 - 4j).real # 실수부 반환
3.0
>>> (3 - 4j).imag # 허수부 반환
-4.0
>>> (3 - 4j).conjugate() # 콤팩트복소수 반환
(3+4j)
```

Python 프로그래밍 기초

변수

변수

- ▶ 숫자나 문자열 등 데이터에 이름을 붙여 기억하도록 하는 기능
- ▶ 변수를 사용하면 데이터나 결과값을 반복하여 사용할 수 있다
- ▶ Python에서는 변수를 선언하는 과정이 없다
 - ▶ 변수에 값을 할당하는 순간 자동으로 선언된다
- ▶ 할당 연산자 : =
 - ▶ ‘같다’는 의미가 아니라 변수에 내용을 담으라는 의미

파이썬은 변수의 타입이 고정되지 않은 동적 타입(**dynamically typed**) 언어

변수

- ▶ 변수명 = 할당값

```
>>> price = 120000
>>> vat = 0.1 # 부가가치세율
>>> final_price = price + (price * vat)
>>> print(final_price)
132000.0
```

변수

: 다양한 할당방법

- ▶ 여러 개를 한꺼번에 치환

```
>>> e, f = 3.5, 5.3
```

- ▶ 여러 개를 같은 값으로 치환

```
>>> x = y = z = 10
```

변수

: 다양한 할당방법

- ▶ 값 교환 (swap)

```
>>> e, f = 3.5, 5.3
>>> e, f = f, e
>>> print(e, f)
```

변수명 작성 규칙

- ▶ 문자, 숫자, _(언더바)의 조합으로 구성
- ▶ 숫자로 시작할 수 없다
- ▶ 예약어를 사용할 수 없다
- ▶ 변수가 가지는 의미를 나타내는 영어 단어를 조합하여 사용하기를 추천
- ▶ 변수명은 대소문자를 구분한다

```
>>> value = 100
>>> _value2 = 200
>>> 3value = 300 # 변수명은 숫자로 시작할 수 없다
File "<stdin>", line 1
  3value = 300
          ^
SyntaxError: invalid syntax
```

예약어

- ▶ Python이 이미 사용하기로 지정해 둔 문자열

```
>>> global = 10 # global은 키워드이다 : 변수명으로 사용 x
      File "<stdin>", line 1
          global = 10
          ^
SyntaxError: invalid syntax
>>> # 키워드 목록 확인하기
>>> import keyword
>>> 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']
```

Python 프로그래밍 기초

비교연산자

비교연산자

| 비교연산자 | 사용예 | 설명 |
|--------------------|------------------------|---|
| <code>></code> | <code>x > y</code> | <code>x</code> 는 <code>y</code> 보다 크다 |
| <code>>=</code> | <code>x >= y</code> | <code>x</code> 는 <code>y</code> 보다 크거나 같다 |
| <code><</code> | <code>x < y</code> | <code>x</code> 는 <code>y</code> 보다 작다 |
| <code><=</code> | <code>x <= y</code> | <code>x</code> 는 <code>y</code> 보다 작거나 같다 |
| <code>==</code> | <code>x == y</code> | <code>x</code> 는 <code>y</code> 와 같다 |
| <code>!=</code> | <code>x != y</code> | <code>x</code> 는 <code>y</code> 와 같지 않다 |

비교연산자와 bool

- ▶ 비교연산자의 비교 결과는 **bool** 값(True or False)을 반환한다
- ▶ 같음(equal)을 비교하는 연산자는 == 이다(= 는 할당연산자) : 주의
- ▶ 비교연산자와 **bool**은 조건 분기와 긴밀히 연결되어 있다

```
>>> 12 > 34
False
>>> 12 >= 34
False
>>> 12 == 34
False
>>> 12 <= 34
True
>>> 12 < 34
True
>>> 12 != 34
True
```

내가 누구~게: type

- ▶ 변수 혹은 값이 어떤 형식인지 알아낼 수 있다

```
>>> type(1)
<class 'int'>
>>> type(1.234)
<class 'float'>
>>> type(1+2j)
<class 'complex'>
>>> type(True)
<class 'bool'>
>>> type("Hello World")
<class 'str'>
```

Python 프로그래밍 기초

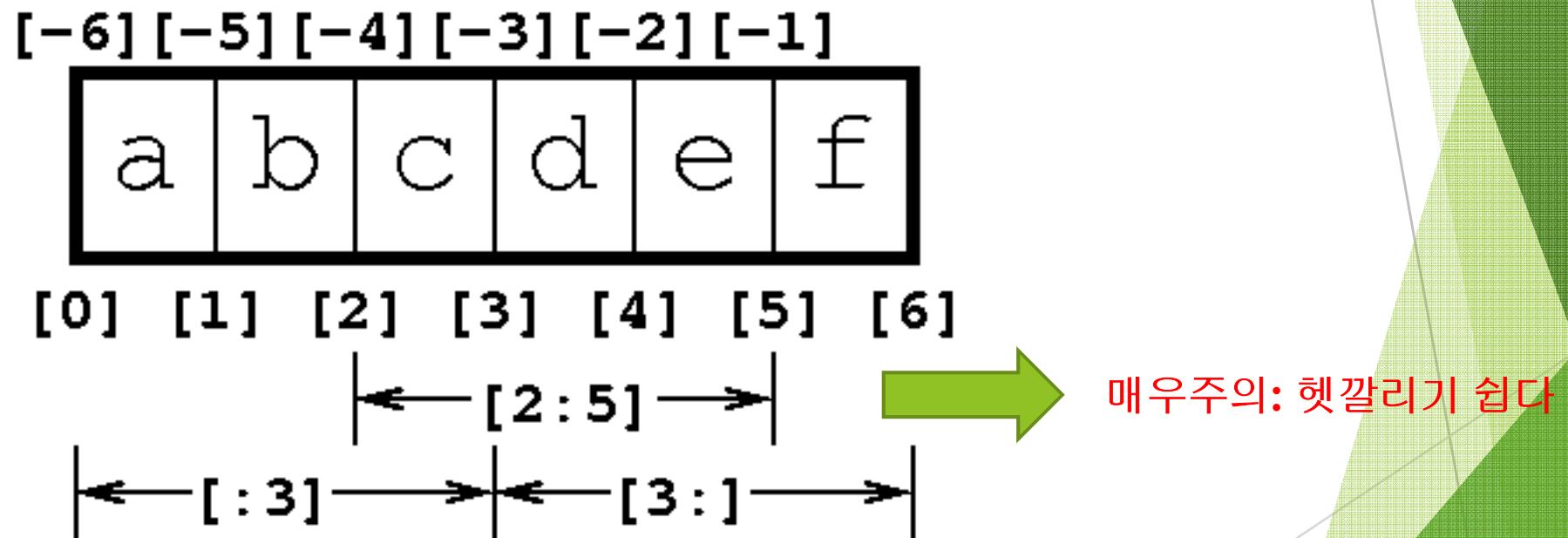
기초 자료형

자료형의 분류

- ▶ 접근방법
 - ▶ 직접(Direct) : int, float, complex, bool
 - ▶ 시퀀스(Sequence) : bytes, str, list, tuple
 - ▶ 매팅(Mapping) : dict
- ▶ 변경가능성
 - ▶ 변경 가능(Mutable) : list, set, dict
 - ▶ 변경 불가능(Immutable) : int, float, complex, bool, bytes, str, tuple
- ▶ 저장 모델
 - ▶ 리터럴(Literal) : int, float, complex, bool, bytes, str
 - ▶ 저장(Container) : list, tuple, dict, set

시퀀스 모델

: 인덱싱(indexing)과 슬라이싱(slicing)



시퀀스 모델에서 인덱싱과 슬라이싱은 매우 중요하다
충분히 반복 연습하여 원하는 데이터를 추출해낼 수 있도록

시퀀스 모델의 주요 연산

| 연산 | 설명 | 사용예 |
|--------|------------|---------------------|
| + | 연결 | "Pyt" + "hon" |
| * | 반복 | "Python" * 2 |
| len() | 길이 반환 | len("Python") |
| in | 포함 여부 | "P" in "Python" |
| not in | 포함되지 않음 여부 | "r" not in "Python" |

논리형

: bool

사실상 **False** 는 0 값을 갖고
그 이외의 값은 모두 **True**로 판정한다

- ▶ 참이나 거짓을 나타내는 **True**, **False** 두 상수를 갖는다

```
>>> a = 1
>>> a > 10
False
>>> a < 10
True
>>>
>>> b = a == 1 # = 우변의 비교값의 결과는 논리형으로 저장(*)
>>> type(b)
<class 'bool'>
>>> b + 10
11
>>> True + True
2
```

논리형

: 논리연산자

| 연산자 | 용례 | 설명 |
|------|---------------------|----------------------|
| 논리합 | {expr1} or {expr2} | 두 값 중 하나만 True면 True |
| 논리곱 | {expr1} and {expr2} | 두 값 모두 True여야 True |
| 논리부정 | not {expr} | expr의 논리값을 반대로 |

논리 연산자와 비교 연산자를 적절히 조합하면,
다양한 조건의 논리값을 만들어 낼 수 있다

논리형

: 논리연산자 - 논리합(or)

- ▶ or 연산자를 이용, 논리합을 구한다
- ▶ 두 값 중 하나만 True면 True

| 값1 | 값2 | 결과 |
|-------|-------|-------|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

논리형

: 논리연산자 - 논리곱(and)

- ▶ and 연산자를 이용, 논리곱을 구한다
- ▶ 두 값 모두 True일때만 True

| 값1 | 값2 | 결과 |
|-------|-------|-------|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

논리형

: 논리연산자 - 논리부정(not)

- ▶ not 연산자를 이용, 논리 결과값을 부정한다

| 값 | 부정 | 결과 |
|-------|-----------|-------|
| True | not True | False |
| False | not False | True |

수치형

- ▶ 숫자를 다루는 데이터형
- ▶ 수치형 데이터끼리는 더하기, 빼기 등의 산술연산을 할 수 있다
- ▶ 수치형의 종류
 - ▶ 정수(Integer) : int
 - ▶ 실수형, 부동소수점(소수) : float
 - ▶ 복소수 : complex

수치형

: 정수형(int)

- ▶ 10진, 2진, 8진, 16진 정수를 표현
- ▶ 파이썬 3.x에서는 `long`형이 없어지고 모두 `int` 형으로 처리된다

```
>>> a = 23
>>> print(type(a)) # 형식 점검
<class 'int'>
>>> print(isinstance(a, int))
True
```

```
>>> b = 0b1101 # 2진수 0b로 시작
>>> c = 0o23 # 8진수 0o로 시작
>>> d = 0x23 # 16진수 0x로 시작
>>> print(a, b, c, d)
23 13 19 35
```

수치형

: 정수형(int)

- ▶ bin, oct, hex 함수를 이용, 2진, 8진, 16진 문자열로 변환할 수 있다

```
>>> a = 2017
>>> bin(a)
'0b11111100001'
>>> oct(a)
'0o3741'
>>> hex(a)
'0x7e1'
```

수치형

: 실수형(float)

- ▶ 소수점을 포함하거나 e나 E 지수로 표현

```
>>> a = 1.2
>>> type(a) # 형식 점검
<class 'float'>
>>> isinstance(a, float)
True
>>> a.is_integer() # 타입 판별이 아니라 실수의 값이 정수인지 판별
False

>>> b = 3e3 # e의 지수로 표현
>>> c = -0.2E-4 # 대문자 E로도 표현 가능
>>> print(a, b, c)
1.2 3000.0 -2e-05
```

수치형

: 실수형(float)

- ▶ `Is_integer()`는 타입 판별이 아니라 값이 정수인지를 판별

```
>>> a = 1.234
>>> type(a)
<class 'float'>
>>> a.is_integer()
False

>>> b = 2.0
>>> type(b)
<class 'float'>
>>> b.is_integer()
True
```

수치형

: 복소수(complex)

- ▶ 실수부 + 허수부로 구분, 허수부에는 j 또는 J를 숫자 뒤에 붙인다

```
>>> cpx = 4 + 5j
>>> type(a)
<class 'float'>
>>> type(cpx)
<class 'complex'>
>>> isinstance(cpx, complex)
True
```

수치형

: 복소수(complex)

- ▶ 실수부와 허수부 값만 따로 참조할 수 있음(real, imag)

```
>>> b = 7 - 2j
>>> b.real, b.imag
(7.0, -2.0)
```

- ▶ complex 함수를 이용, 복소수 타입의 객체를 만들 수 있음

```
>>> cpx = complex(7, -2) # Usage: complex({실수부}, {허수부})
>>> type(cpx)
<class 'complex'>
>>> cpx.real, cpx.imag
(7.0, -2.0)
```

수치형

: 내장 수치 함수

| 함수명 | 사용예 | 설명 |
|---------|----------------|------------|
| abs | abs (-3) | 절대값 |
| int | int (3.141592) | 정수변환 |
| float | float (3) | 실수변환 |
| complex | complex (1, 2) | 허수 생성 |
| divmod | divmod (5, 3) | 나눗셈 몫과 나머지 |
| pow | pow (2, 10) | 제곱 |

비트 연산자

- ▶ 정수 자료형에만 적용
 - ▶ 비트 단위로 수치를 다룰 수 있다

| | | | |
|-------|---------|---------|--------|
| flag | 0 0 0 0 | 0 1 1 1 | 비트 AND |
| ~mask | 1 1 1 1 | 1 1 0 1 | |
| flag | 0 0 0 0 | 0 1 0 1 | |

| 연산자 | 기능 | 문법 | 설명 |
|-----------------------|------------|---------------------------|---|
| <code><<</code> | 비트 왼쪽 시프트 | <code>a << b</code> | <code>a</code> 의 비트를 <code>b</code> 번 왼쪽으로 이동 |
| <code>>></code> | 비트 오른쪽 시프트 | <code>a >> b</code> | <code>a</code> 의 비트를 <code>b</code> 번 오른쪽으로 이동 |
| <code>&</code> | 비트 AND | <code>a & b</code> | <code>a</code> 와 <code>b</code> 의 비트를 AND 연산 |
| <code> </code> | 비트 OR | <code>a b</code> | <code>a</code> 와 <code>b</code> 의 비트를 OR 연산 |
| <code>^</code> | 비트 XOR | <code>a ^ b</code> | <code>a</code> 와 <code>b</code> 의 비트를 XOR(베타적 OR) 연산 |
| <code>~</code> | 비트 NOT | <code>~x</code> | <code>x</code> 의 비트를 뒤집음 (<code>0 <-> 1</code>) |

정수의 왼쪽 시프트는 $\ast 2$, 정수의 오른쪽 시프트는 $\text{}/2$ 한 것과 동일하다

비트 연산자

```
>>> bin(0b0001 << 2) # 왼쪽으로 2비트 이동
'0b100'
>>> bin(0b1000 >> 2) # 오른쪽으로 2비트 이동
'0b10'
>>> bin(0b00001000 & 0b11111111) # 비트 AND를 이용한 필터링
'0b1000'
>>> bin(8 | 2) # 비트 OR 연산
'0b1010'
```

확장 치환문

- ▶ 산술, 비교 연산자 등을 치환문과 함께 사용할 수 있다

```
x op= y    # => x = x op y
```

- ▶ 확장 치환 연산자 종류
 - ▶ `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=`, `<<=`, `>>=`, `&=`, `|=`, `^=`

문자열

: str

- ▶ 쌍따옴표(“) 혹은 홑따옴표(‘)로 묶인 문자들의 모임

```
>>> s = ""  
>>> str1 = "hello world"  
>>> str2 = "life is too short, you need python"  
>>> type(s), type(str1), type(str2)  
(<class 'str'>, <class 'str'>, <class 'str'>)  
>>> isinstance(str1, str)  
True
```

문자열

: 여러 줄의 문자열 정의

- ▶ “” 혹은 “”을 이용, 여러 줄의 문자열을 정의할 수 있음

```
>>> str3 = """ABCDEFG
... abcdef
... 가나다라마바사아
... 1234567890"""
>>> str3
'ABCDEFG\nabcdef\n가나다라마바사아\n1234567890'
```

문자열의 연산

: 연결(+)과 반복(*)

- ▶ 문자열은 시퀀스형: 연결(+), 반복(*) 연산이 가능

```
first_name = "Seung Kyun"  
last_name = "Nam"  
full_name = first_name + " " + last_name # 문자열 연결은 +로  
print(full_name)  
print(first_name, last_name)  
  
laugh = "Ha"  
print(laugh * 3) # laugh를 3번 반복하여 연결
```

- ▶ 문자열 객체와 수치형 객체는 + 연산을 할 수 없다

```
>>> "Python" + 3  
TypeError: Can't convert 'int' object to str implicitly
```

문자열의 연산

: 인덱싱, 슬라이싱, len

- ▶ 문자열은 시퀀스형: 인덱싱, 슬라이싱, len, in, not in 연산 가능

```
str = "Life is too short, You need Python!"

print(len(str)) # len() : 시퀀스형의 길이를 반환

print(str[2]) # 2번 인덱스의 문자를 반환
print(str[8:11]) # 인덱스 8 ~ 10 사이의 문자열을 반환
print(str[-7:-1]) # 음수 인덱스는 뒤로부터 계산
print(str[5:]) # 인덱스는 필요에 따라 생략 가능
```

- ▶ 문자열은 변경 불가(immutable) 자료형이다

```
>>> str[0] = "l"
...
TypeError: 'str' object does not support item assignment
```

문자열 메서드

: 대소문자 관련

| 메서드 | 설명 |
|--------------|-------------------------|
| upper() | 문자열을 대문자로 변환 |
| lower() | 문자열을 소문자로 변환 |
| swapcase() | 대 <-> 소문자를 전환 |
| capitalize() | 문자열의 첫 글자를 대문자로 변환 |
| title() | 문자열의 각 단어의 첫글자를 대문자로 변환 |

문자열 메서드

: 대소문자 관련

```
s = "i like Python"

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

I LIKE PYTHON # upper
i like python # lower
I LIKE pYTHON # swapcase
I like python # capitalize
I Like Python # title
```

문자열 메서드

: 검색 관련

| 메서드 | 설명 |
|--------------|-------------------------------|
| count() | 문자열 내 검색어 개수를 반환 |
| find() | 문자열 내 첫번째 검색된 위치의 인덱스를 반환 |
| index() | 문자열 내 검색된 위치의 인덱스를 반환 |
| rindex() | 문자열 내 오른쪽으로부터 검색된 위치의 인덱스를 반환 |
| startswith() | 문자열이 지정된 검색어로 시작하는지 여부 반환 |
| endswith() | 문자열이 지정된 검색어로 끝나는지 여부 반환 |

문자열 메서드

: 검색 관련

```
s = 'I Like Python. I Like Java Also'  
print(s.count('Like'))  
  
print(s.find('Like'))  
print(s.find('Like', 5)) # 인덱스 5부터 검색  
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))
```

문자열 메서드

: 편집, 치환 관련

| 메서드 | 설명 |
|-----------|--|
| strip() | 문자열 내 좌우 공백문자를 삭제 좌우 삭제할 문자열을 지정 가능 |
| lstrip() | 문자열 내 왼쪽의 공백문자를 제거 |
| rstrip() | 문자열 내 오른쪽 공백문자를 제거 |
| replace() | 문자열 내 지정된 검색어를 다른 문자열로 치환 |

문자열 메서드

: 편집, 치환 관련

```
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'))
```

문자열 메서드

: 정렬 관련

| 메서드 | 설명 |
|----------|-----------------------|
| center() | 문자열을 가운데로 정렬 |
| ljust() | 문자열을 왼쪽으로 정렬 |
| rjust() | 문자열을 오른쪽으로 정렬 |
| zfill() | 자리수를 지정하고 빈 공간을 0로 채움 |

문자열 메서드

: 정렬 관련

```
s = 'Alice and the Heart Queen'

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

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

문자열 메서드

: 분리, 결합 관련

| 메서드 | 설명 |
|--------------|-------------------------------------|
| split() | 문자열을 공백문자(혹은 지정된 문자)를 기준으로 분리 |
| rsplit() | 문자열을 공백문자(혹은 지정된 문자)를 기준으로 오른쪽부터 분리 |
| join() | 문자열을 지정된 기호로 합침 |
| splitlines() | 문자열을 개행문자를 기준으로 분리 |

문자열 메서드

: 분리, 결합 관련

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

문자열 메서드

: 판별 관련

| 메서드 | 설명 |
|-----------|----------------------------|
| isdigit() | 문자열이 숫자로 구성되어 있는가 여부를 반환 |
| isalpha() | 문자열이 알파벳으로 구성되어 있는가 여부를 반환 |
| islower() | 문자열이 소문자로 구성되어 있는가 여부를 반환 |
| isupper() | 문자열이 대문자로 구성되어 있는가 여부를 반환 |
| isspace() | 문자열이 공백문자로 구성되어 있는가 여부를 반환 |

문자열 메서드

: 판별 관련

```
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())
```

문자열 메서드

: 서식 메서드 - 문자열 포맷 코드

| 코드 | 설명 |
|----|------------------------|
| %s | 문자열 (string) |
| %c | 문자 1개 (character) |
| %d | 정수 (integer) |
| %f | 부동 소수 (floating point) |
| %o | 8진수 |
| %x | 16진수 |
| %% | Literal % |

서식 메서드에 출력 포맷을 추가 지정하는 것도 가능:

예) %2.4f -> 정수부 2자리, 소수부 4자리

문자열 메서드

: 서식 메서드 - 문자열 포맷 코드

```
>>> "I have %d apples" % 5
'I have 5 apples'
>>> "interest rate is %f" % 1.24
'interest rate is 1.240000'

>>> "interest rate is %2.4f" % 1.24
'interest rate is 1.2400'
```

문자열 메서드

: 고급 문자열 포매팅 - `.format()` 메서드

- ▶ 문자열의 `format` 메서드를 이용하면 좀 더 편리한 방식으로 문자열 포맷을 지정 할 수 있다
- ▶ `format_map` 메서드를 이용하면 이름 기반으로 `map`의 데이터 형식을 이용 포맷 을 지정할 수 있다

```
>>> "I have {} apples, and I ate {} apples.".format(5, 3)
'I have 5 apples, and I ate 3 apples.'
>>> "I have {total} apples, and I ate {num} apples.".format(total = 5, num = 3)
'I have 5 apples, and I ate 3 apples.'
>>> "I have {total} apples, and i ate {num} apples.".format_map({"total": 5, "num": 3})
'I have 5 apples, and i ate 3 apples.'
```

리스트

- ▶ 순서를 가지는 객체들의 집합, 파이썬 자료형들 중 가장 많이 사용
- ▶ 리스트 생성과 연산
 - ▶ 시퀀스 자료형 : 시퀀스 연산(인덱싱, 슬라이싱, 연결, 반복, `len`, `in`, `not in`) 가능
 - ▶ 변경 가능(mutable) 자료형이므로 항목의 추가, 변경, 삭제 모두 가능

```
l = [1, 2, 'python'] # 리스트는 [ ] 기호를 이용하여 생성
```

```
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)
```

리스트

: 항목의 변경 및 슬라이스를 이용한 치환

```
a = ['apple', 'banana', 10, 20]
a[2] = a[2] + 90 # mutable 자료형 -> 항목 변경 가능
print(a)
```

```
# 슬라이스를 이용한 치환의 예
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)
```

리스트

: 슬라이스를 이용한 삭제와 삽입

```
# 슬라이스를 이용한 삭제
```

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

```
# 슬라이스를 이용한 삽입
```

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

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

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

리스트

: 리스트의 메서드

| 함수 | 설명 |
|---------------|--------------------------------|
| append (x) | 리스트의 마지막에 x를 추가 |
| insert (i, x) | 리스트 인덱스 i 위치에 x를 추가 |
| reverse () | 리스트를 역순으로 뒤집음 |
| sort () | 리스트 요소를 순서대로 정렬 |
| remove (i) | 리스트 인덱스 i에 있는 요소를 제거 |
| extend (l) | 리스트 마지막에 리스트 l을 추가 |
| index (x) | 인덱스 내에 x가 있으면 인덱스값을 반환. 없으면 -1 |
| count (x) | 리스트 내에 x가 몇 개 있는지 그 개수를 반환 |

리스트

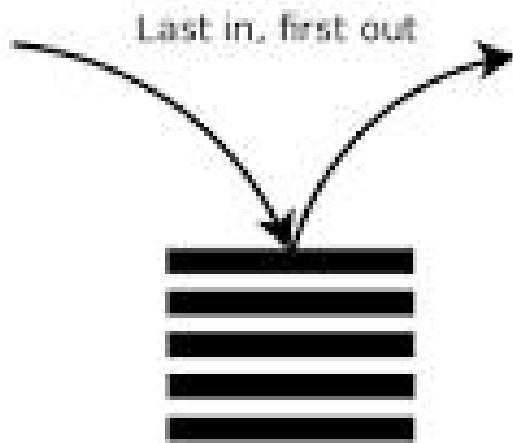
: 리스트의 메서드

```
a = [1, 2, 3]
print(a)
a.append(5)
print(a)
a.insert(3, 4) # 인덱스 3에 요소 4를 추가
print(a)
print(a.count(2)) # 리스트 내 요소 2의 개수를 반환
a.reverse()
print(a)
a.sort()
print(a)
a.remove(3) # 내부에 있는 요소 3을 제거
print(a)
a.extend([6, 7, 8])
print(a)
```

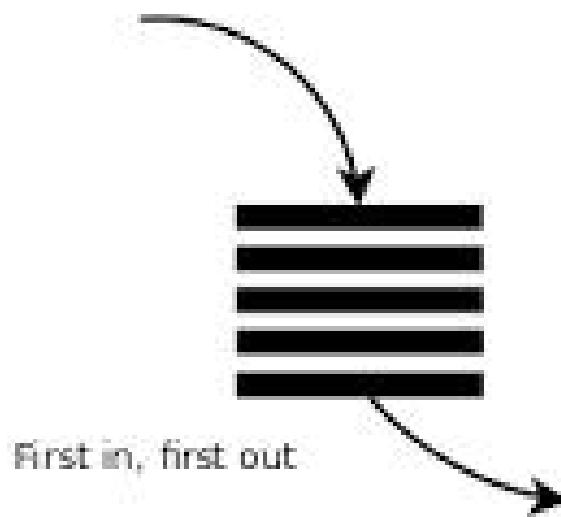
리스트

: 리스트를 Stack과 Queue로 사용하기

Stack:



Queue:



리스트

: 리스트를 Stack으로 사용하기

- ▶ 리스트의 `append`와 `pop` 메서드를 이용하여 스택을 구현할 수 있다

```
stack = [ ]  
  
stack.append(10)  
stack.append(20)  
stack.append(30)  
  
print(stack)  
  
print(stack.pop())  
print(stack.pop())  
print(stack)
```

리스트

: 리스트를 Queue로 사용하기

- ▶ 리스트의 `append`와 `pop` 메서드를 이용하여 스택을 구현할 수 있다

```
queue = [ ]  
  
queue.append(100)  
queue.append(200)  
queue.append(300)  
  
print(queue)  
  
print(queue.pop(0)) # 가장 앞쪽 인덱스의 요소를 pop  
print(queue.pop(0))  
  
print(queue)
```

리스트

: sort 메서드의 활용

- ▶ sort 메서드의 reverse 를 True로 설정하면 역순으로 정렬할 수 있다

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

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

리스트

: sort 메서드의 활용

- ▶ 키값 기반의 사용자 정의 정렬

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

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

세트(Set)

- ▶ 순서가 없고 중복이 없는 객체들의 집합 (non sequence). {} 기호로 정의
 - ▶ len(), in, not in 정도만 활용 가능
- ▶ 수정이 가능한(**mutable**) 자료형
- ▶ 수학의 집합을 표현할 때 사용한다

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

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

세트(Set)

: 세트의 메서드

| 메서드 | 설명 |
|----------------|------------------------------|
| add (x) | 세트에 x를 추가 |
| remove (x) | 세트에서 x를 제거. x가 세트에 없으면 오류 발생 |
| discard (x) | 세트에서 x를 제거. x가 세트에 없으면 무시 |
| update ({set}) | 세트에 여러 개의 값을 추가 |
| clear () | 세트를 비움 |

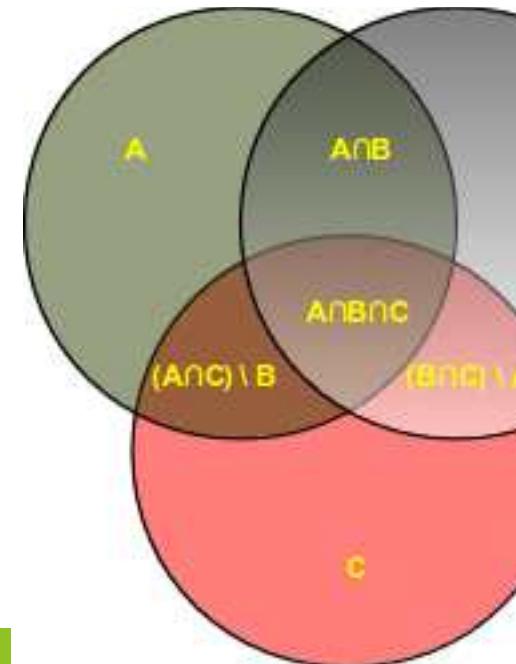
```
s = {1, 2, 3}
```

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

세트(Set)

: 교집합, 합집합, 차집합

- ▶ 세트(Set)는 교집합, 합집합, 차집합을 구하는데 유용하게 사용



| | 연산자 | 메서드 |
|-------------|----------|--------------------------------|
| 교집합 (set) | $a \& b$ | <code>a.intersection(b)</code> |
| 합집합 (set) | $a b$ | <code>a.union(b)</code> |
| 차집합 (set) | $a - b$ | <code>a.difference(b)</code> |
| 모집합 (bool) | | <code>a.issuperset(b)</code> |
| 부분집합 (bool) | | <code>a.issubset(b)</code> |

세트(Set)

: 교집합, 합집합, 차집합

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

```
s3 = s1.union(s2) # 합집합  
print(s3)
```

```
s4 = s1.intersection(s2) # 교집합  
print(s4)
```

```
s4 = s1.difference(s2) # 차집합  
print(s4)
```

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

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

튜플(Tuple)

- ▶ 리스트와 거의 비슷하지만 다름 : 시퀀스 자료형
 - ▶ 튜플은 () 기호로 생성하며 그 값을 바꿀 수 없다(**immutable**)
 - ▶ 하나의 요소만을 가질 때는 요소 뒤에 콤마(,)를 반드시 붙임
 - ▶ 괄호를 생략해도 튜플로 인식

튜플(Tuple)

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

t = 1, 2, 'python' # ()를 생략해도 튜플을 생성할 수 있다
print(t, type(t))

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

print(t * 2) # 반복(*)
print(t + (3, 4, 5)) # 연결(+)
print(len(t)) # 요소 개수 반환
print(5 in t) # 요소 5가 내부에 있는지 확인
```

튜플(Tuple)

: packing과 unpacking

- ▶ **Packing** : 나열된 객체를 Tuple로 저장하는 것
- ▶ **Unpacking** : 튜플, 리스트 안의 객체를 변수로 할당하는 것

```
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)
```

튜플(Tuple)

: 확장 unpacking

- ▶ Unpacking 시 왼쪽 변수가 부족한 경우, 에러가 발생한다(**ValueError**)
- ▶ 확장 Unpacking에서는 왼쪽 변수가 적은 경우에도 적용할 수 있다 (*)

```
a, b = (10, 20, 30, 40, 50) # ValueError 발생
```

```
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)
```

사전(dict)

- ▶ 순서를 가지지 않는 객체의 집합
- ▶ Key 기반으로 값을 저장하고 참조하는 매팅형 자료형
- ▶ 시퀀스 자료형이 아니므로 len(), in, not in 정도만 가능

```
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)
```

사전(dict)

: 다양한 사전 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)) # 키와 값을 별도로 선언 후 합침
print(d)
```

사전(dict)

: 사전의 키(Key)

- ▶ 사전의 키는 해싱해야 하기 때문에 수정 불가능한 객체여야 한다
 - ▶ 예) 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' # TypeError 발생
```

사전(dict)

: 사전의 메서드

| 메서드 | 설명 |
|-----------------------|--|
| keys () | 사전내 키 목록을 dict_keys 객체로 반환 |
| values () | 사전내 값 목록을 dict_values 객체로 반환 |
| items () | 사전내 키-값 쌍을 튜플로 묶은 dict_items 객체로 반환 |
| get (key {, default}) | 사전내 key 에 대응하는 값을 반환 default 를 지정하면 key 에 대응하는 값이 없을 때 default 를 반환 |
| del dic[key] | dic 사전 내 key 에 대응하는 객체를 삭제 |
| clear () | 사전을 비움 |

dict_keys, **dict_values**, **dict_items** 를 리스트로 사용하려면 **list()** 함수를 활용

사전(dict)

: 사전의 메서드

```
d = {'basketball': 5, 'soccer': 11, 'baseball': 9}
d['volleyball'] = 6 # 새로운 값 할당

print(d.keys()) # key 목록 가져오기
=> dict_keys(['volleyball', 'baseball', 'soccer', 'basketball'])
print(d.values()) # value 목록 가져오기
=> dict_values([6, 9, 11, 5])
print(d.items()) # (key, value) 튜플 목록 가져오기
=> dict_items([('volleyball', 6), ('baseball', 9), ('soccer', 11),
('basketball', 5)])
```

사전(dict)

: 사전의 메서드

```
d = {'basketball': 5, 'soccer': 11, 'baseball': 9}
d['volleyball'] = 6

print(d.keys())
print(d.values())
print(d.items())

# x = d['handball'] # KeyError
x = d.get('handball') # None 반환
print(x)

del d['soccer']
print(d)

d.clear()
print(d)
```

사전(dict)

: 사전 순회

```
d = {'basketball': 5, 'soccer': 11, 'baseball': 9}

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

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

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

순차 자료형(Sequence) 내장 함수

: range

```
range({start = 0,} end {, step = 1})  
# start부터 end까지의 순차적 리스트를 step 간격으로 생성
```

```
seq = range(10) # 0이상 10 미만의 순차적 정수 목록  
print(seq, type(seq))  
print(seq[0:])  
print(len(seq))  
  
for i in seq:  
    print(i)  
  
seq2 = range(5, 15) # 5 이상 15 미만의 순차적 정수 목록  
for i in seq2:  
    print(i)  
  
seq3 = range(0, -10, -1) # 0 이하 -10 초과의 순차적 정수 목록  
for i in seq3:  
    print(i)
```

순차 자료형(Sequence) 내장 함수

: enumerate

- ▶ 순차 자료형에서 현재 아이템의 색인과 함께 처리하고자 할 때 흔히 사용

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

비교 : **enumerate** 함수를 사용했을 때

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

Python 프로그래밍 기초

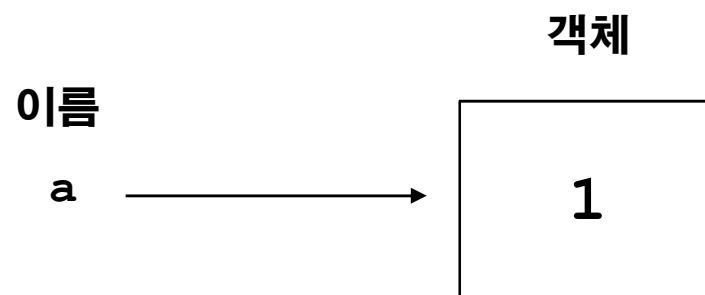
객체

객체

: 이름과 객체

- ▶ 파이썬에서 모든 자료(데이터)들은 객체의 형태로 저장된다
- ▶ 파이썬의 변수는 컴파일러 언어처럼 변수에 할당된 값을 저장하는 저장공간(메모리)의 주소 심볼릭이 아니다
- ▶ 변수는 단지 객체의 이름(심볼)일 뿐이다
- ▶ 파이썬의 객체 이름(변수)과 객체의 ID(주소)는 심볼 테이블에 함께 저장되어 관계를 갖게 된다

a = 1



객체

: 심볼테이블

- ▶ 변수의 이름과 저장된 데이터의 주소를 저장하는 테이블
- ▶ 심볼테이블의 내용을 살펴보기 위해 `globals()`, `locals()` 내장 함수를 이용
- ▶ 두 함수는 해당 스코프 내 심볼 테이블의 내용을 `dict` 타입의 객체로 반환한다

```
>>> # 글로벌 변수 선언
>>> g_a = 1
>>> g_b = "symbol"
>>>
>>> def f(): # 로컬 변수 확인을 위한 함수 선언
...     l_a = 2
...     l_b = "table"
...     print(locals()) # 로컬 심볼테이블 확인
>>> f()
{'l_a': 2, 'l_b': 'table'}
>>> globals() # 글로벌 심볼테이블 확인
{'__doc__': None, '__loader__': <class
'__frozen_importlib.BuiltinImporter'>, '__package__': None,
'f': <function f at 0x104d06bf8>, 'g_a': 1, 'g_b': 'symbol'
 '__spec__': None, '__name__': '__main__', '__builtins__': <module 'builtins' (built-in)>}
```

객체

: 레퍼런스 카운트와 쓰레기 수집

- ▶ 레퍼런스 카운트(Reference Count) : 객체를 참조하는 참조 수
- ▶ 레퍼런스 카운트가 0이 되면 사용하지 않는 객체로 판단, 자동으로 사라짐
- ▶ 이러한 작업을 쓰레기 수집(Garbage Collection)이라 함

```
>>> import sys
>>> x = object()
>>> sys.getrefcount(x)
2
>>> y = x
>>> sys.getrefcount(x)
3
>>> sys.getrefcount(y)
3
>>> del(x) # 레퍼런스 값이 줄어든다
>>> sys.getrefcount(y)
2
```

객체

: 객체 ID

- ▶ `id()` 함수를 이용하면 객체의 주소를 식별할 수 있다
 - ▶ 만일 두 객체의 ID가 동일하면, 같은 객체를 참조하고 있는 것

```
>>> i1 = 10
>>> i2 = 10
>>> print(hex(id(i1)), hex(id(i2)))
0x1067fb710 0x1067fb710
>>> l1 = [1, 2, 3]
>>> l2 = [1, 2, 3]
>>> print(hex(id(l1)), hex(id(l2)))
0x106a0be88 0x106b36b08
>>> s1 = "hello"
>>> s2 = "hello"
>>> print(hex(id(s1)), hex(id(s2)))
0x106f39110 0x106f39110
>>> i1 is i2
True
>>> l1 is l2
False
>>> s1 is s2
True
```

객체

: 객체의 복사

- ▶ 레퍼런스 복사
 - ▶ 객체를 참조하는 주소만 복사하는 것

```
>>> x = [1, 2, 3]
>>> y = x # 객체 참조 주소만 복사된다
>>> y
[1, 2, 3]
>>> hex(id(x)), hex(id(y))
('0x106b36d48', '0x106b36d48')
>>> x[1] = 4
>>> y
[1, 4, 3]
```

객체

: 객체의 복사

- ▶ [:] 이용한 복사
 - ▶ 객체 전체를 가리키는 [:]를 이용하여 복사한다

```
>>> x = [1, 2, 3]
>>> y = x[:]
>>> y
[1, 2, 3]
>>> x is y
False
>>> x[1] = 4
>>> y
[1, 2, 3]
```

객체

: 객체의 복사

- ▶ `copy` 함수 이용
 - ▶ `copy` 모듈의 `copy` 함수를 사용하여 복사한다

```
>>> import copy
>>> x = [1, 2, 3]
>>> y = copy.copy(x)
>>> x is y
False
>>> x[1] = 4
>>> y
[1, 2, 3]
```

객체

: 객체의 복사

- ▶ `deepcopy` 함수 이용
 - ▶ `copy` 모듈의 `deepcopy` 함수를 사용하여 복사한다
 - ▶ `deepcopy`는 복합객체를 재귀적으로 생성하고 복사한다

```
>>> a = [1, 2, 3]
>>> b = [4, 5, a]
>>> x = [a, b, 100]
>>> import copy
>>> y = copy.deepcopy(x)
```

Python 프로그래밍 기초

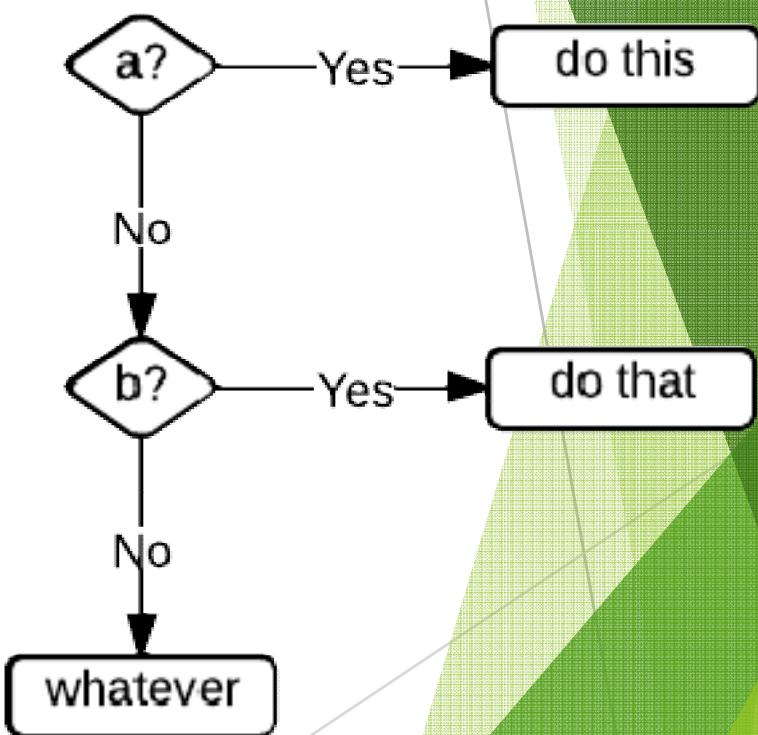
제어문 (조건문과 반복문)

조건문

: if - elif - else

```
if( 조건식1 ) :  
    구문1  
    구문2  
  
elif( 조건식2 ) :  
    구문3  
    구문4  
  
else :  
    구문 5
```

```
if a:  
    do this  
elif b:  
    do that  
else:  
    whatever
```



조건문

: if - elif - else

```
>>> n = -2
>>> if n > 0:
...     print("양수")
... elif n < 0:
...     print("음수")
... else:
...     print("0")
...
음수
```

조건문

: 조건 표현식(Conditional Expression)

- ▶ C 또는 Java의 3항 연산자와 같은 역할을 한다

```
value = {true-expr} if {condition} else {false-expr}
```

```
>>> money = 8500
>>> print("by taxi" if money > 10000 else "by bus")
by bus
```

조건문

: in, not in

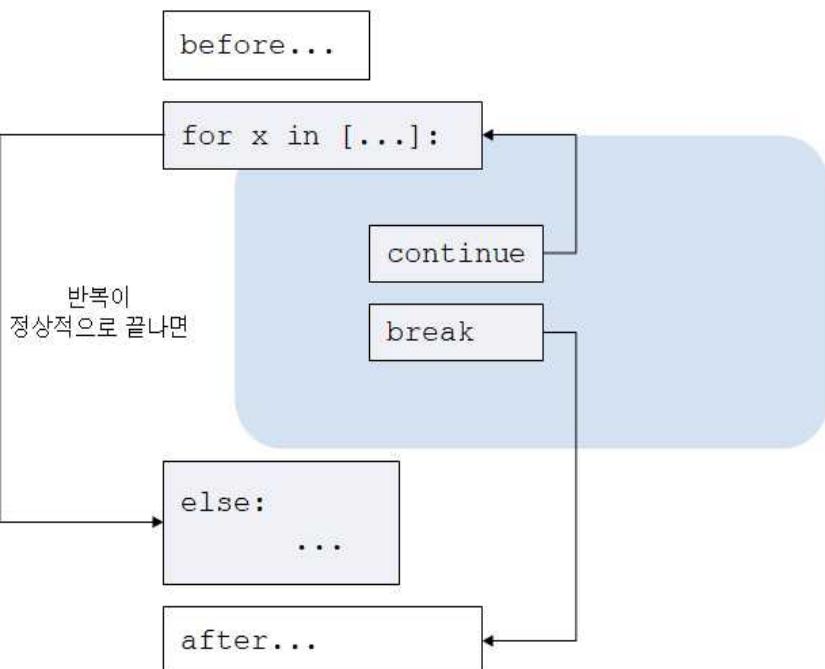
- ▶ 파이썬은 리스트, 튜플, 문자열에 in, not in 등 편리한 조건식을 제공

| in | not in |
|----------|--------------|
| x in 리스트 | x not in 리스트 |
| x in 튜플 | x not in 튜플 |
| x in 문자열 | x not in 문자열 |

```
>>> 1 in [1, 2, 3]
True
>>> 1 not in [1, 2, 3]
False
>>> "y" in "Python"
True
>>> "y" not in "Python"
False
```

반복문

: for



```
for {타킷} in {객체}:
    구문1
    구문2
else:
    구문 3
```

- ▶ {객체}는 list, str, tuple, bytes, bytearray, range 등 시퀀스 자료형
- ▶ 반복횟수는 {객체}의 크기
- ▶ {객체} 안의 객체를 하나씩 순차적으로 꺼내어 구문1,2가 실행됨
- ▶ 반복이 정상적으로 끝나면 else 블록이 실행
- ▶ for문 내에서 break로 빠져나오면 else 블록은 실행되지 않음

반복문

: for

```
>>> # list 객체를 이용한 for 문
>>> animals = ['cat', 'cow', 'tiger']
>>> for animal in animals:
...     print(animal, end = " ")
cat cow tiger

>>> # range 객체를 이용한 for 문
>>> for x in range(1, 10, 3):
...     print(x, end = " ")
1 4 7

>>> # for ~ else 문의 활용
>>> for x in data:
...     if x > 10:
...         break
... else:
...     print("10보다 큰 수 없음")
```

반복문

: enumerate

- ▶ 요소의 값은 물론 인덱스가 필요할 경우 `enumerate()` 함수를 이용한다

```
>>> colors = ['red', 'orange', 'yellow', 'green', 'pink', 'blue']
>>> for index, color in enumerate(colors):
...     print(index, color)
...
0 red
1 orange
2 yellow
3 green
4 pink
5 blue
```

반복문

: break

- ▶ 어떤 조건에서 반복을 중지하고 빠져나가야 하는 경우 break문

```
>>> l = [1, 3, 5, 7, 9, 11, 12, 13, 15, 17]
>>> for x in l:
...     if x % 2 == 0:
...         break;
...     print(x)
1
3
5
7
9
11
```

반복문

: continue

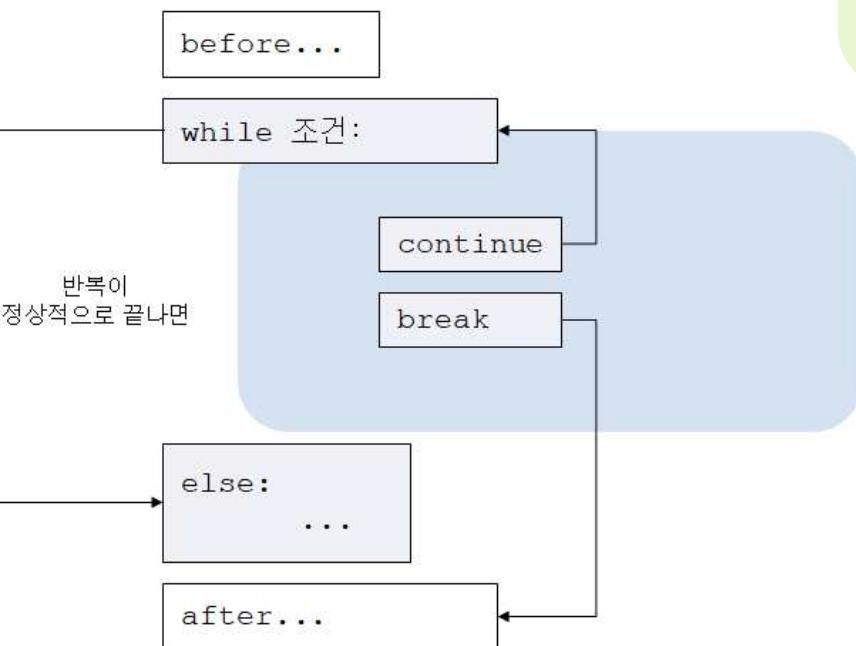
- ▶ Continue문을 만나면, 이후 구문은 실행하지 않고 처음으로 이동한다

```
>>> for x in range(10):  
...     if x % 2 == 0:  
...         continue  
...     print(x, end = " ")
```

```
1 3 5 7 9
```

반복문

: while



```
while {조건식}:
    구문1
    구문2
else:
    구문 3
```

- ▶ {조건}이 참인 동안 구문1, 2가 반복 실행
- ▶ Else 블록은 while문을 빠져나올 때 실행
- ▶ 단, break로 while문을 빠져나올 경우 else 블록은 실행되지 않는다
- ▶ 무한루프(실행이 종료되지 않고 계속 실행되는 반복)에 빠지지 않도록 유의
- ▶ 경우에 따라서는 의도적으로 무한루프를 돌리기도 한다

반복문

: List 내포

- ▶ 리스트 내포(List Comprehension)를 이용하면 좀더 직관적인 프로그램을 만들 수 있다

```
[{표현식} for {항목} in {객체} if {조건}]
```

```
>>> a = [1, 6, 4, 13, 8, 3]
>>> a 리스트 항목 중, 짝수인 것만 2배 하여 result에 저장
>>> result = [num * 2 for num in a if num % 2 == 0]
>>> result
[12, 8, 16]
```

반복문

: while

```
>>> counter = 1
>>> while counter < 11:
...     print(counter, end = " ")
...     counter += 1
... else:
...     print("")
...
1 2 3 4 5 6 7 8 9 10
```

```
>>> sum, i = 0, 1
>>> while i <= 100: # 1~100까지의 정수 합 구하기
...     sum += i
...     i += 1
...
>>> sum
5050
```

반복문

: while 내부에서 break, continue, else 사용하기

```
>>> i = 0
>>> while i < 100:
...     i += 1
...     if i < 5:
...         continue
...     print(i, end = " ")
...     if i > 10:
...         break
... else:
...     print("else block") # 이 블록은 실행되지 않을 것임. WHY?
...
5 6 7 8 9 10 11
```

반복문

: 무한루프

- ▶ 조건을 `True`로 주면 무한루프를 구성할 수 있다
- ▶ `break` 문으로 루프를 탈출할 수 있는 조건이 있어야 한다

```
>>> while True:  
...     print("Ctrl+C를 눌러 루프를 종료하십시오.")  
...  
Ctrl+C를 눌러 루프를 종료하십시오.  
Ctrl+C를 눌러 루프를 종료하십시오.  
...
```

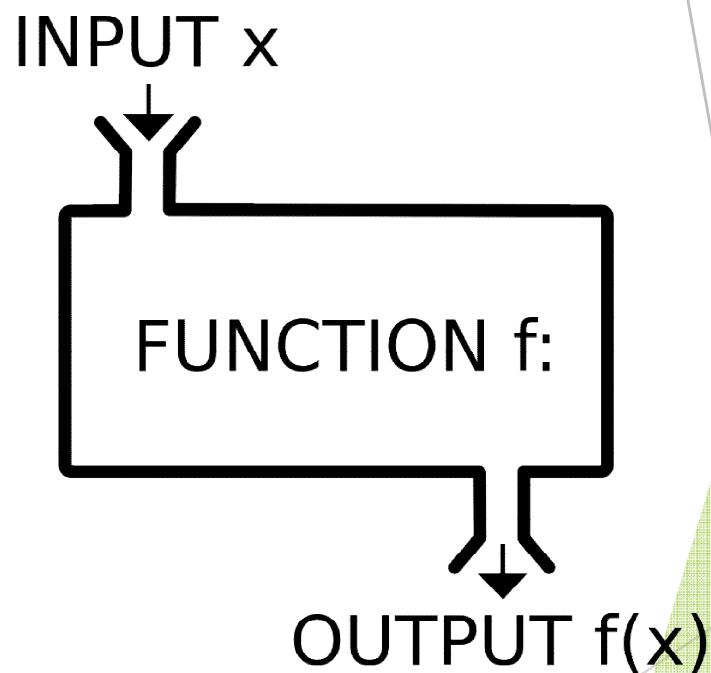
Python 프로그래밍 기초

함수(Function)

함수

: 함수란

- ▶ 입력값을 가지고 어떤 일을 수행한 다음 그 결과물을 내 놓는 것
- ▶ 함수를 사용하는 이유
 - ▶ 반복되는 부분이 있을 경우 재활용을 위해
 - ▶ 프로그램의 흐름을 일목요연하게 볼 수 있다



함수 정의하기

- ▶ **Def** 키워드를 이용하여 정의
- ▶ 함수 이름과 인수들이 기술
- ▶ 함수 선언부는 `:`로 끝난다
- ▶ 들여쓰기 규칙이 적용
- ▶ 함수의 끝은 들여쓰기가 적용 안되는 라인에서 끝난다

함수 정의하기

```
def dummy():
    pass # 실행할 내용이 없을때는 pass

def my_function():
    print("Hello World")

def times(a, b):
    return a * b # 결과값을 돌려줘야 할 때는 return 문으로 반환

def do_nothing():
    return # return 문만 썼을 경우, None이 반환

dummy()
my_function()
print(times(10, 10))
print(do_nothing())
```

함수

: 함수도 객체다

- ▶ 함수도 객체이므로 다음과 같은 호출도 가능하다

```
t = times
print(t(100, 100))
print(t, times, sep = ",")
```

함수

: return

- ▶ 함수를 종료시키고, 해당 함수를 호출한 곳으로 되돌아 가게 한다
- ▶ 파이썬에서는 어떤 종류의 객체도 반환할 수 있다
- ▶ 여러 객체를 `return`하면 튜플로 반환한다
- ▶ `return`문을 만나면 함수는 종료한다
- ▶ `return`문만 사용하면 `None`을 반환한다
- ▶ 함수가 끝날 때까지 종료할 필요가 없고 반환할 값이 없을 때는 `return`문이 없어도 된다

함수

: return문 활용

```
# 인수 없이 반환하기
def do_nothing():
    return # None을 반환

# return 문이 필요없는 경우도 있다
def say_hello():
    print("Life is too short, You need Python")

# 한 개의 값을 반환
def max_value(a, b):
    if a > b:
        return a
    else:
        return b
```

함수

: return문 활용

```
# 여러 값을 반환할 때
def swap(a, b):
    return b, a

print(swap(10, 20))

# 결과값은 튜플로 반환된다
```

함수

: 인수의 전달 방법

- ▶ 기본적으로 참조에 의한 호출(Call-by-reference)이다
- ▶ 하지만 인수의 타입이 변경가능(mutable), 변경불가(immutable)에 따라 처리 방식이 달라진다

```
# 변경 가능 객체를 인수로 전달할 경우
def g(t):
    t[0] = 0

a = [1, 2, 3]
g(a)
print(a)
```

함수

: 인수의 전달 방법

- ▶ 기본적으로 참조에 의한 호출(Call-by-reference)이다
- ▶ 하지만 인수의 타입이 변경가능(mutable), 변경불가(immutable)에 따라 처리 방식이 달라진다

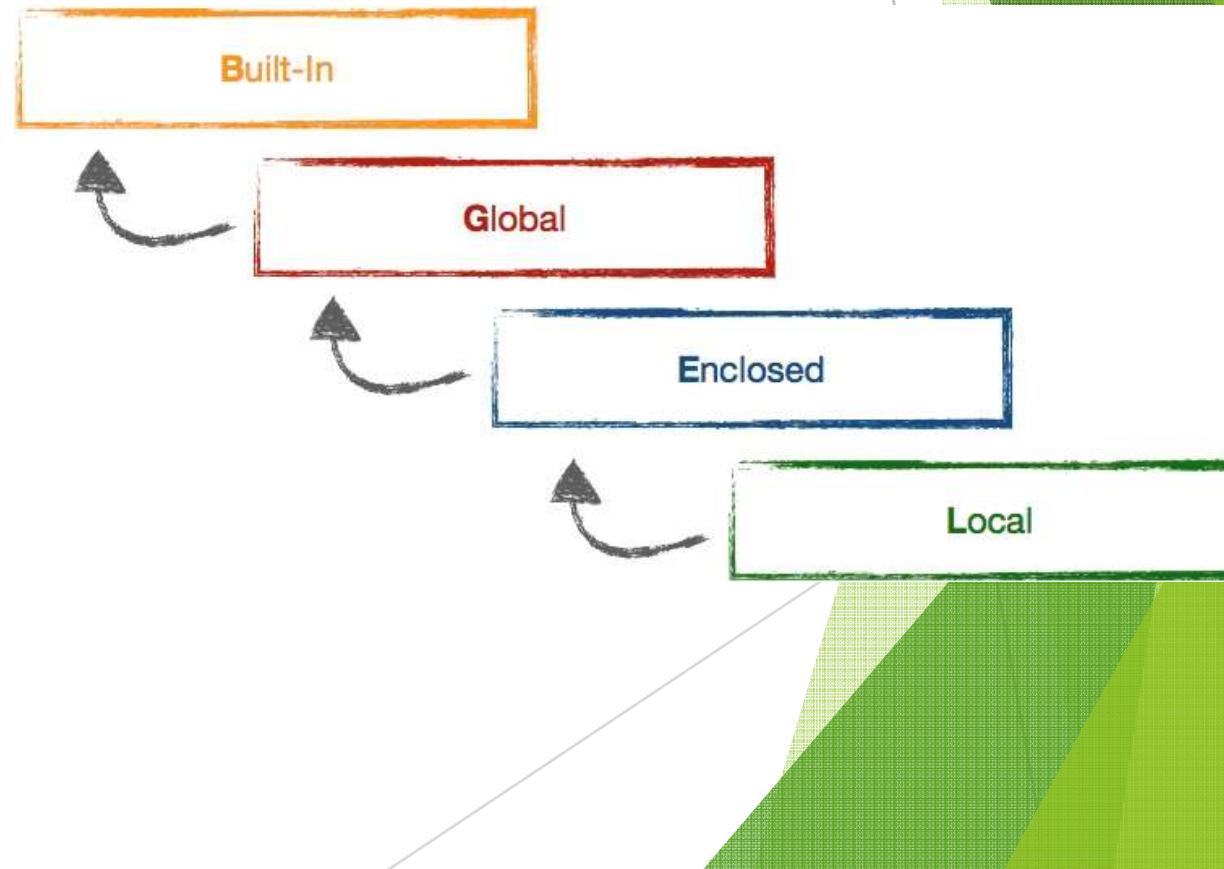
```
# 변경 불가 객체를 인수로 전달했을 때
def h(t):
    t = (10, 20, 30)

a = (1, 2, 3)
h(a)
print(a) # a 객체는 변경되지 않는다
```

함수

: 스코핑 룰(Scope)

- ▶ 이름공간(Namespace)
 - ▶ 프로그램에서 사용되는 이름들이 저장되는 공간
- ▶ 이름은 값을 치환할 때 해당 값의 객체와 함께 생겨나고 이름공간에 저장
- ▶ 이름공간에 저장된 이름을 통해 객체를 참조
- ▶ 이름공간의 종류
 - ▶ Local Scope: 함수 내부
 - ▶ Enclosed: function in function
 - ▶ Global: 모듈 내부
 - ▶ Built-in: 내장 영역
- ▶ 동일한 이름이 여러 영역에 있다면 LEBG 순으로 찾는다



함수

: 스코핑 룰 예제

```
x = 1
def func(a):
    return a + x # Local 스코프에 x 가 없으므로 Global x를 사용한다

def func2(a):
    x = 2
    return a + x # Local 스코프에 x가 있으므로 Local x를 사용한다

print(func(10))
print(func2(10))
print(x)
```

함수

: 스코핑 룰 예제

- ▶ 함수 내부에서 전역 객체를 사용해야 하는 경우 **global** 선언문을 이용한다
- ▶ 가능하면 함수 내부에서 글로벌 객체를 직접 사용하는 것은 피한다

```
g = 1
def func3(a):
    global g
    g = 3 # 본 객체는 글로벌 객체이다
    return a + g

print(func3(10))
print(g)
```

함수

: 함수의 인수

- ▶ 인수는 필요한 개수만큼 선언할 수 있다
- ▶ 기본값이 필요하면 함수 선언시 지정할 수 있다

```
def sum(a, b):  
    return a + b  
  
def incr(a, step = 1): # 두 번째 인자의 기본값은 1  
    return a + step  
  
print(sum(2, 3))  
print(incr(10)) # 두 번째 인자의 기본값을 사용한다  
print(incr(10, 2)) # 두 번째 인자를 직접 지정한다
```

함수

: 함수의 인수 - 키워드 인수

- ▶ 인수값을 인수 이름으로 전달할 수 있다(함수의 정의에 따른다)

```
def area(width, height):  
    return width * height  
  
print(area(10, 12))  
print(area(height = 4, width = 3)) # == area(3, 4)
```

함수

: 함수의 인수 - 가변인수

- ▶ 정해지지 않은 개수의 인수값을 받을 때 사용한다
- ▶ 선언시 인수명 앞에 *를 붙여 선언한다

```
def get_total(*args):  
    sum = 0  
    for x in args:  
        sum += x  
    return sum  
  
print(get_total(1, 3, 5, 7, 9))
```

함수

: 함수의 인수 - 사전 키워드 전달

- ▶ 정해지지 않은 키워드 인수는 모두 `dict`로 받을 수 있다
- ▶ 선언시 인수명 앞에 `**`를 붙인다
- ▶ 사전 키워드 인수는 선언의 맨 마지막에 있어야 한다

```
def f(a, b, *args, **kwd):  
    print(a, b)  
    print(args)  
    print(kwd)  
  
f(10, 20, 30, 40, depth = 10, dimension = 3)
```

함수

: 함수 객체를 인수로 전달

- ▶ 파이썬에서는 함수도 객체이다
- ▶ 따라서 인수로 함수를 전달하는 것도 가능하다

```
states = ['Alabama', ' Georgia', 'Georgia ', 'georgia', 'FlOrIda',
'south carolina ', 'West virginia']

def clean_strings(strings, *funcs): # 함수 목록을 가변인수로 전달
    results = []
    for string in strings:
        for func in funcs: # 전달된 함수들을 순차적으로 적용
            string = func(string)
        results.append(string)
    return results

states = clean_strings(states, str.strip, str.title)
print(states)
```

함수

: 익명 함수(Lambda)

- ▶ 이름이 정의되지 않은 ‘익명 함수’를 선언
- ▶ 데이터 분석/변형 함수에서 파라미터로 처리 함수를 인자로 받는 경우가 많다

```
def square(x):  
    return x * 2  
  
for i in range(10):  
    print("{0}:{1}".format(i, square(i)), end = " ")  
else:  
    print()  
  
# Same as above with Lambda  
for i in range(10):  
    print("{0}:{1}".format(i, (lambda x: x * 2)(i)), end = " ")  
else:  
    print()
```

함수

: Lambda를 이용한 정렬

- ▶ 정렬할 때, key 함수로 정의하기에도 편리한 경우가 많다

```
strings = ['foo', 'card', 'bar', 'abab', 'aaaa', 'abab', 'foo']

strings.sort(key=lambda x: len(x))
print(strings)

strings.sort(key=lambda x: strings.count(x))
print(strings)
```

파이썬 프로그래밍 기초

예외처리 (Handling Exceptions)

예외 처리

- ▶ 오류는 프로그램이 잘못 작동되는 것을 막기 위한 파이썬의 처리
- ▶ 이 오류를 회피하기 위한 동작을 예외 처리라 한다

```
>>> list = []
>>> list[0] # 내부에 0 인덱스에 매칭되는 값이 없으므로 IndexError를 발생
IndexError: list index out of range

>>> text = "Try convert me to int"
>>> number = int(text) # 정수 형식의 문자열이 아니어서 변환이 불가: ValueError
ValueError: invalid literal for int() with base 10: 'Try convert me to int'

>>> result = 4/0 # 0으로는 나눗셈을 할 수 없다. ZeroDivisionError 발생
ZeroDivisionError: division by zero
```

예외 처리

: 오류의 회피 - try, except

```
try:  
    # Do Something  
except:  
    # Do Something when Error occurred
```

```
>>> try:  
...     4 / 0  
... except:  
...     print("오류 발생")
```

오류 발생

예외 처리

: 특정 오류의 회피 - try, except

```
try:  
    # Do Something  
except {발생오류}:  
    # Do Something when {발생오류} occurred
```

```
>>> try:  
...     4 / 0  
... except ZeroDivisionError:  
...     print("오류 발생")
```

오류 발생

예외 처리

: 특정 오류의 회피 - try, except

```
try:  
    # Do Something  
except {발생오류} as {오류변수}:  
    # Do Something when {발생오류} occurred
```

```
>>> try:  
...     4 / 0  
... except ZeroDivisionError as e: # 발생한 오류의 정보를 e에 담아 사용한다  
...     print(e)  
  
division by zero
```

예외 처리

: - try, except, else

```
try:  
    # Do Something  
except {발생오류} as {오류변수}:  
    # Do Something when {발생오류} occurred  
else:  
    # 예외가 발생하지 않은 경우 실행
```

```
>>> try:  
...     f = open("notexists.txt", "r")  
... except FileNotFoundError as e:  
...     print(e)  
... else:  
...     data = f.read()  
...     f.close()
```

예외 처리

: try ~ finally

```
try:  
    # Do Something  
except {발생오류} as {오류변수}:  
    # Do Something when {발생오류} occurred  
finally:  
    # 예외 발생 여부와 상관 없이 마지막에 항상 수행
```

```
>>> f = open("result.txt", "r")  
>>> try:  
...     print(f.read())  
... finally:  
...     f.close() # 오류 여부와 상관 없이 파일 close 작업을 수행한다
```

Python 프로그래밍 기초

파일 입출력

파일 입출력 개요

: 파일의 생성과 파일 모드

```
파일객체 = open({파일명}, {파일모드}[, encoding='인코딩'])
```

| 파일 모드 | 설명 |
|-------------|-----------------------------------|
| r (default) | 읽기 모드 - 파일을 읽기만 할 때 사용 |
| w | 쓰기 모드 - 파일에 내용을 기록할 때 사용 |
| a | 추가 모드 - 파일의 마지막에 새로운 내용을 추가할 때 사용 |

| 파일 모드 | 설명 |
|-------------|---------|
| t (default) | 텍스트 모드 |
| b | 바이너리 모드 |

파일 입출력 개요

: 파일 제어 기본 함수

| 함수명 | 설명 |
|-------|---|
| open | 파일을 생성한다 |
| write | 파일에 내용을 기록한다 |
| read | 파일에서 내용을 읽어온다 |
| close | 파일 사용을 끝낸다. 파일을 열었으면(open) 반드시 사용후 닫아주도록 한다 |

```
>>> # File Write Sample
>>>
>>> f = open('text.txt', 'w', encoding='utf-8') # text.txt, 쓰기모드
>>> write_size = f.write("Life is too short, You need Python")
>>> print(write_size)
33
>>> f.close() # 반드시 닫아주자
```

파일 입출력 개요

: 텍스트 파일 예제

```
# File Write

f = open('test.txt', 'w', encoding='utf-8')
for i in range(1, 10):
    f.write("%d: Life is too short, You need Python\n" % i)
f.close()

# File Read

f = open('test.txt', 'r', encoding='utf-8')
text = f.read()
print(text)
f.close()
```

파일 입출력 개요

: 텍스트 파일 예제 - write and read

```
# File Write

f = open('multilines.txt', 'w', encoding='utf-8')
for i in range(1, 10):
    f.write("%d: Life is too short, You need Python\n" % i)
f.close()

# File Read

f = open('multilines.txt', 'r', encoding='utf-8')
text = f.read()
print(text)
f.close()
```

파일 입출력 개요

: `readline` 함수를 이용한 텍스트 파일 읽기

- ▶ `readline` 함수를 이용하면 텍스트 파일을 줄 단위로 읽어올 수 있다

```
f = open('multilines.txt', 'r')

while True:
    line = f.readline()
    if not line:
        break # 무한루프 탈출
    print(line)

f.close()
```

파일 입출력 개요

: `readlines` 함수를 이용한 텍스트 파일 읽기

- ▶ `readlines` 함수를 이용하면 모든 라인을 불러 리스트로 제공한다

```
f = open('multilines.txt', 'r')

lines = f.readlines()
# print(lines)

for line in lines:
    print(line)

f.close()
```

파일 입출력 개요

: 바이너리(Binary) 파일 다루기

- ▶ 바이너리 파일을 다루려면 모드를 바이너리로 지정해야 한다

```
>>> f = open('python.png')
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/namsk/.pyenv/versions/3.4.3/lib/python3.4/codecs.py",
line 319, in decode
    (result, consumed) = self._buffer_decode(data, self.errors,
final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x89 in position
0: invalid start byte
```

파일 입출력 개요

: 바이너리(Binary) 파일 다루기

- ▶ 바이너리 파일을 다루려면 모드를 바이너리로 지정해야 한다

```
>>> # copy binary sample
>>>
>>> f_src = open('python.png', 'rb') # 바이너리 읽기 모드
>>> data = f_src.read()
>>> f_src.close()
>>>
>>> f_dest = open('python_copy.png', 'wb') # 바이너리 쓰기 모드
>>> f_dest.write(data)
11155
>>> f_dest.close()
```

파일 입출력 개요

: 그 외 파일 관련 함수

| 함수명 | 설명 |
|------|----------------------------|
| seek | 사용자가 원하는 위치로 파일 포인터 이동 |
| tell | 현재 파일에서 어디까지 읽고 썼는지 위치를 반환 |

```
f = open('multilines.txt', 'r', encoding='utf-8')

text = f.readline()
print(text)
pos = f.tell() # 현재의 파일 포인터를 얻어옴
print(pos)

f.seek(16)
text = f.read()
print(text)
```

파일 입출력 개요

: with ~ as - 자동 자원 정리

- ▶ with ~ as 를 이용, 파일 입출력을 수행하면 수동으로 파일을 close 하지 않아도 된다

```
with open('multilines.txt', 'r') as f_as:  
    for line in f_as.readlines():  
        print(line, end = "")  
  
print(f_as.closed) # 파일이 close 되었는지 점검
```

Using Pickle

- ▶ 객체의 내용을 파일에 저장하거나 복원해야 할 경우에 **Pickle** 모듈을 사용하면 편리
- ▶ **Pickle** 모듈은 객체를 파일에 썼다가 나중에 복원할 수 있도록 객체를 바이트 스트림으로 직렬화
 - ▶ 모든 파이썬의 객체를 저장하고 읽을 수 있음
 - ▶ 원하는 객체를 형태 변환 없이 쉽게 쓰고 읽을 수 있다
- ▶ **Pickle** 모듈을 사용하려면 `import pickle` 을 이용, 모듈을 로드해야 한다
- ▶ **Pickle** 모듈 주요 메서드

| 메서드 | 설명 |
|--|--|
| <code>dump(data, file [, protocol])</code> | <code>data</code> 객체를 <code>[protocol 을 이용해]</code> <code>file</code> 에 저장 |
| <code>load(file)</code> | <code>File</code> 로부터 저장된 객체를 불러옴 |

Using Pickle

: 객체의 저장:Pickling - dump

- ▶ file에 객체를 저장하고자 할 때에는 dump 메서드를 이용한다

```
import pickle
f = open("players.bin", "wb")
data = {"baseball": 9}
pickle.dump(data, f)
f.close()
```

- ▶ dump 메서드에 프로토콜 버전을 정의해 줄 수 있다

- ▶ 최신 프로토콜 버전을 확인하려면 pickle.HIGHEST_PROTOCOL로 확인

```
...
pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
...
print(pickle.HIGHEST_PROTOCOL)
```

Using Pickle

: 객체의 복원:Unpickling - load

- ▶ file에 객체로부터 객체를 불러올 때에는 `load` 메서드를 이용한다

```
import pickle
f = open("players.bin", "rb")
data = pickle.load(f)
f.close()
print(data)
```

- ▶ `dump`시에 `PROTOCOL`을 지정했다 하더라도 `load`할 때는 지정해주지 않아도 된다
 - ▶ `pickle` 파일에 `PROTOCOL` 버전이 저장되어 있음

Using Pickle

: 복수 객체의 저장

- ▶ 기본적으로 **Pickle**은 단일 객체를 저장하는 포맷이지만, **dump** 메서드를 중복하여 사용하면 복수 개의 객체를 저장할 수 있다

```
import pickle
with open("players.bin", "wb") as f:
    pickle.dump({"baseball": 9}, f)
    pickle.dump({"soccer": 11}, f)
    pickle.dump({"basketball": 5}, f)
```

Using Pickle

: 복수 객체의 복원

- ▶ 저장된 객체를 복원하려면 `load` 메서드를 이용
 - ▶ `load`가 수행될 때마다 한줄씩 불러들이며 더 이상 불러올 객체가 없을 때 `EOFError` 발생
 - ▶ 다음 코드를 수행해 보고 무엇이 문제인지 확인해 봅니다

```
>>> import pickle
>>> with open("players.bin", "rb") as f:
...     print(pickle.load(f))
...     print(pickle.load(f))
...     print(pickle.load(f))
...     print(pickle.load(f))
...
{'baseball': 9}
{'soccer': 11}
{'basketball': 5}
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
EOFError: Ran out of input
```

Using Pickle

: 복수 객체의 복원

▶ [Solution]

```
>>> import pickle
>>> with open("players.bin", "rb") as f:
...     data_list = []
...     while True:
...         try:
...             data = pickle.load(f)
...         except EOFError:
...             break
...     data_list.append(data)
...
>>> print(data_list)
[{'baseball': 9}, {'soccer': 11}, {'basketball': 5}]
```

Using Pickle

▶ Pickle 사용시 유의사항

- ▶ Pickle은 단순 텍스트 저장이 아닌 바이트 스트림 직렬화를 이용한 것이므로 파일 모드는 반드시 "b" 모드 (wb / rb)로 지정해야 한다
- ▶ Pickle에 사용되는 데이터 포맷은 파이썬에 특화되어 있기 때문에 다른 언어로 작성된 응용프로그램과의 데이터 교환에는 사용하지 않는 것이 좋다
- ▶ 저장된 데이터에 대한 보안을 제공하지 않는 점에 유의하여 사용

파이썬 프로그래밍 기초

날짜와 시간



날짜와 시간

- ▶ 날짜와 시간은 파이썬에서 기본으로 제공하는 자료형에는 포함되어 있지 않지만 중요한 자료형
- ▶ 날짜와 시간은 각각 **date** 객체, **time** 객체를 이용하며 이들 객체는 **datetime** 모듈에 포함되어 있다
 - ▶ `import datetime` 을 이용, 해당 모듈을 불러올 수 있다
 - ▶ 주요 클래스
 - ▶ **datetime** : 날짜와 시간을 모두 포함하는 클래스
 - ▶ **date** : 날짜 관련 클래스
 - ▶ **time** : 시간 관련 클래스

날짜와 시간

: 날짜와 시간의 획득

- ▶ `datetime` 모듈 내 `datetime` 클래스의 `now()` 메서드를 이용하면 현재 날짜와 시간을 획득할 수 있다

```
>>> import datetime      # datetime 모듈 불러오기
>>> dt = datetime.datetime.now()
>>> dt
datetime.datetime(2017, 5, 2, 14, 59, 56, 411440)
(year, month, day, hour, minute, second, microsecond)
```

- ▶ `datetime` 클래스 생성자로 직접 날짜를 지정할 수 있다
 - ▶ 년, 월, 일, 시, 분, 초, 마이크로초 순으로 매개변수를 전달하면 되며, 최소 년월일은 지정해 주어야 한다

```
>>> import datetime      # datetime 모듈 불러오기
>>> dt = datetime.datetime(1999, 12, 24)
>>> dt
datetime.datetime(1999, 12, 24, 0, 0)
```

날짜와 시간

: datetime 클래스

▶ datetime 클래스의 주요 속성

| 속성 | 설명 |
|-------------|-------|
| year | 년 |
| month | 월 |
| day | 일 |
| hour | 시 |
| minute | 분 |
| second | 초 |
| microsecond | 마이크로초 |

```
>>> import datetime
>>> dt = datetime.datetime.now()
>>> dt.year, dt.month, dt.day, dt.hour, dt.minute, dt.second,
dt.microsecond
(2017, 5, 15, 15, 22, 625369)
```

날짜와 시간

: datetime 클래스

▶ datetime 클래스의 주요 메서드

| 메서드 | 설명 |
|------------|-------------------------|
| weekday() | 요일을 반환 |
| strftime() | datetime 을 문자형으로 포맷 |
| date() | 날짜 정보만 가지는 date 클래스로 변환 |
| time() | 시간 정보만 가지는 time 클래스로 변환 |

```
>>> import datetime
>>> dt = datetime.datetime.now()
>>> dt.weekday() # 월:0, 화:1, 수:2, 목:3, 금:4, 토:5, 일:6
2
>>> dt.date()
datetime.date(2018, 5, 2)
>>> dt.time()
datetime.time(15, 29, 42, 821062)
```

- ▶ date는 datetime 클래스의 날짜 관련 속성과 요일 메서드를 이용 가능
- ▶ time은 datetime 클래스의 시간 관련 속성 이용 가능

날짜와 시간

: datetime의 비교와 두 datetime의 차이 구하기

- ▶ 두 개의 날짜는 대소 비교, 차이 값을 구할 수 있다

```
>>> import datetime
>>> current = datetime.datetime.now()
>>> past = datetime.datetime(1999, 12, 31)
>>> current > past # 현재 날짜가 과거보다 큰가?
True
>>> diff = current - past #
>>> diff
datetime.timedelta(6697, 56169, 742005)
>>> diff.days, diff.seconds, diff.microseconds
(6697, 56169, 742005)
```

- ▶ 두 날짜의 차이는 datetime.timedelta 클래스로 반환, 날짜의 차이를 구할 수 있다
 - ▶ days, seconds, microseconds

날짜와 시간

: timedelta

- ▶ timedelta 클래스의 주요 속성
 - ▶ days : 일수
 - ▶ seconds: 초 (0 ~ 863999)
 - ▶ microseconds: 마이크로초 (0 ~ 999999)
- ▶ timedelta 클래스의 메서드
 - ▶ total_seconds() : 모든 속성을 초단위로 모아서 변환
- ▶ 두 datetime의 차이가 timedelta로 반환되었던 것과 반대로 datetime 클래스에 timedelta 값을 더해 새로운 날짜를 만들 수도 있다

날짜와 시간

: timedelta

▶ datetime과 timedelta의 합산

```
>>> import datetime
>>> current = datetime.datetime.now()
>>> current
datetime.datetime(2018, 5, 2, 15, 55, 19, 406471)
>>> diff = datetime.timedelta(days=30, seconds=0, microseconds=0)
>>> current + diff
datetime.datetime(2018, 6, 1, 15, 55, 19, 406471)
```

날짜와 시간

: datetime을 문자열로 변환

- ▶ datetime의 strftime 메서드를 이용하면 datetime 형식을 문자열로 변환할 수 있다
- ▶ strftime의 출력 형식 기호

| 기호 | 표시되는 형식 |
|----|---------------|
| %Y | 서기 년도 4자리 표시 |
| %y | 서기 년도 2자리 표시 |
| %m | 달을 두 자리로 표시 |
| %d | 일을 두 자리로 표시 |
| %B | 영어로 달을 표시 |
| %b | 영어로 달을 단축 표시 |
| %A | 영어로 요일을 표시 |
| %a | 영어로 요일을 단축 표시 |

| 기호 | 표시되는 형식 |
|----|--------------|
| %H | 시간 표시(24시간) |
| %I | 시간 표시(12시간) |
| %p | AM/PM 표시 |
| %M | 분을 두 자리로 표시 |
| %S | 초를 두 자리로 표시 |
| %f | 마이크로초를 6자리수로 |

날짜와 시간

: datetime을 문자열로 변환

- ▶ 다양한 형식으로 datetime을 문자열로 포맷하기

```
>>> import datetime
>>> current = datetime.datetime.now()
>>> current.strftime('%Y/%m/%d')
'2018/05/02'
>>> current.strftime(' %Y년 %m월 %d일 ') # 하단 참고
'2018년 05월 02일'
>>> current.strftime(' %Y-%m-%d %H-%M-%S ')
'2018-05-02 16-09-31'
```

- ▶ 참고 : Windows에서는 Locale로 인해 UnicodeEncodeError가 발생할 수 있다
다음과 같이 조치한다

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'ko_KR.UTF-8')
'ko_KR.UTF-8'
>>> current.strftime(' %Y년 %m월 %d일 ')
'2018년 05월 02일'
```

날짜와 시간

: 문자열을 datetime으로 변환

- ▶ `datetime` 클래스의 `strptime` 메서드를 이용하면 문자열로부터 날짜와 시간 정보를 읽어서 `datetime` 클래스로 변환할 수 있다
 - ▶ `strptime`의 인수
 - ▶ 첫 번째 인수 : 날짜와 시간 정보를 가진 문자열
 - ▶ 두 번째 인수 : 문자열 해독을 위한 형식 문자열

```
>>> import datetime
>>> str = '2017-12-24 23:59'
>>> datetime.datetime.strptime(str, '%Y-%m-%d %H:%M')
datetime.datetime(2017, 12, 24, 23, 59)
```

파이썬 프로그래밍 기초

모듈(Module)

파이썬 모듈

- ▶ 함수나 변수 등을 정의해 놓은 파이썬 프로그램 파일
- ▶ 모듈 내에서는 어떤 코드도 작성 가능하다 (변수, 함수, 클래스 등)
- ▶ 다른 모듈에 의해 호출되고 사용된다
- ▶ 모듈의 종류
 - ▶ 표준 모듈
 - ▶ 사용자 생성 모듈
 - ▶ 서드 파티 모듈

파이썬 모듈

: 간단한 모듈 만들기

```
# mymod.py

Pi = 3.14159

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    return a / b
```

파이썬 모듈

: 모듈 불러오기 - import

```
import {모듈명}
```

```
# test-mymod.py

import mymod

print(mymod.add(10, 20))
print(mymod.subtract(10, 20))
print(mymod.multiply(10, 20))
print(mymod.divide(10, 20))
```

파이썬 모듈

: 네임스페이스

- ▶ 네임스페이스는 모듈 내부의 이름(변수, 함수, 클래스)를 구분하는 역할 수행
- ▶ 네임스페이스가 주어지지 않은 변수나 함수는 **LEGB** 규칙에 따라 찾게 된다

```
import math
import mymath

print(mymath.pi) # mymath 모듈 내의 pi 이용
print(math.pi) # math 모듈 내부의 pi 이용
```

파이썬 모듈

: from ~ import

```
from {모듈명} import {모듈객체} # 모듈명 없이 객체명만으로 접근
```

```
from math import pi, sin, cos, tan  
print(sin(pi/6), cos(pi/3), tan(pi/4))
```

- ▶ 현재 모듈에 특정 이름이 중복되는 경우 맨 마지막에 import된 객체가 적용

```
from math import pi, sin, cos, tan  
from mymod import pi  
print(sin(pi/6), cos(pi/3), tan(pi/4))
```

- ▶ 모듈 내에 정의된 모든 이름을 현재 모듈로 가져온다(*)

```
from math import * # math 모듈 내 모든 이름 가져오기  
print(sin(pi/6), cos(pi/3), tan(pi/4))
```

파이썬 모듈

: from ~ import

- ▶ 모듈 이름을 다른 이름으로 바꾸는 것도 가능

```
from math import *
import mymod as m # mymod의 이름을 m으로 변경
print(sin(pi/6), cos(pi/3), tan(pi/4)) # math 모듈 내의 객체들을 이용
print(m.pi) # mymod의 pi 객체를 이용
```

- ▶ 모듈 내에 정의된 객체의 이름을 변경하는 것도 가능

```
from math import sin as msin, cos as mcos, tan as mtan
import mymod as m # mymod의 이름을 m으로 변경
print(msin(m.pi/6), mcos(m.pi/3), mtan(m.pi/4))
```

모듈 지원 함수 목록 보기

: dir 함수

- ▶ dir 함수 인자에 객체를 넣어주면 해당 객체가 어떤 변수와 메서드를 갖고 있는지 반환해준다

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>> a = [1, "Python", 3.14159]
>>> dir(a)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

파이썬 모듈

: `__name__` 과 “`__main__`”

- ▶ 모든 모듈은 모듈의 이름을 저장하고 있는 내장 변수 `__name__`을 갖고 있다
- ▶ 최상위 모듈(인터프리터에서 실행되는 모듈)의 `__name__`은 “`__main__`”이다
- ▶ 그 외 `import` 되는 모듈의 이름은 파일명이다

```
# mymod.py
print("mymod.py의 모듈이름:" + __name__)
```

```
# moduleimport.py
import mymod
print("moduleimport.py의 모듈이름:" + __name__)
```

```
> python moduleimport.py
mymod.py의 모듈이름:mymod
moduleimport.py의 모듈이름:__main__
```

파이썬 모듈

: `__name__` 과 “`__main__`”

- ▶ 내장 변수 `__name__` 과 최상위 실행 모듈의 이름이 “`__main__`”인 점을 응용하면
 - ▶ `import` 되는 모듈로 사용하면서
 - ▶ 자신이 최상위 모듈로 실행될 때만 특정 기능을 수행하는 코드를 만들 수 있다

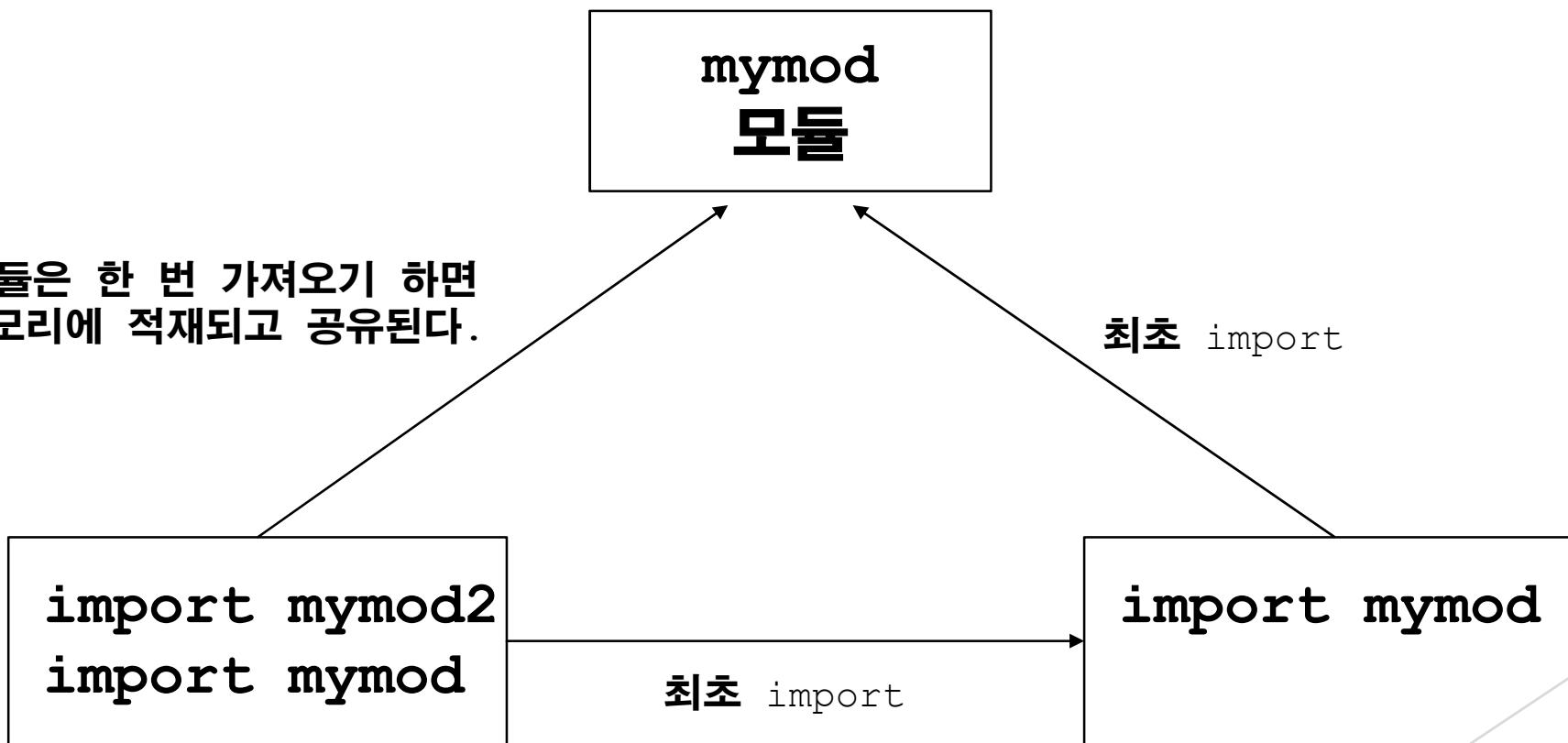
```
# mymod.py
def main():
    print("mymod.py를 최상위 모듈로 실행했습니다.")

if __name__ == "__main__":
    main()
else:
    print("mymod.py의 모듈이름:" + __name__)
```

파이썬 모듈

: 모듈의 공유

모듈은 한 번 가져오기 하면
메모리에 적재되고 공유된다.



test-mymod2.py

mymod2.py

파이썬 모듈

: import한 module 이름의 열거

- ▶ 한번이라도 import한 모듈은 dict 타입인 sys.modules 변수에 저장된다

```
import mod_a
import mod_b
import mymod
import mymod2
import sys

for key in sys.modules.keys():
    print(key)
```

파이썬 모듈

: 이름(변수, 함수, 클래스)이 속한 모듈 알아내기

- ▶ 파이썬 변수, 함수, 클래스는 각각 자신이 정의된 모듈의 이름이 저장된 `__module__` 속성을 가지고 있다

```
>>> from math import sin
>>> from cmd import Cmd
>>> sin.__module__
'math'
>>> Cmd.__module__
'cmd'
```

여러가지 내장 모듈

: sys 모듈의 argv

```
import sys
```

- ▶ 파이썬 인터프리터와 관련된 정보와 기능 제공
- ▶ argv : 명령행에서 넘어온 인수(arguments)를 처리할 수 있다

```
# args.py
import sys
args = sys.argv[1:] # arg[0]는 스크립트명 자체

for x in args:
    print(x, end = " ")
else:
    print()
```

```
python args.py arg1 arg2 arg3
arg1 arg2 arg3
```

여러가지 내장 모듈

: math 모듈 - 파이와 자연상수

```
import math
```

▶ 파이와 자연상수

```
>>> math.pi # 파이값  
3.141592653589793  
>>> math.e # 자연상수  
2.718281828459045
```

여러가지 내장 모듈

: math 모듈 - 절대값, 반올림, 버림계산

- ▶ abs, round는 내장 수치함수

```
>>> abs(10) # 절대값
10
>>> abs(-10) # 절대값
10
>>> round(1.2345, 2) # 소수점 셋째자리에서 반올림
1.23
>>> round(1.5213) # 소수점 반올림
2
>>> math.trunc(1.7) # 소수점 이하 버림
1
```

여러가지 내장 모듈

: math 모듈 - 팩토리얼, 제곱과 제곱근

```
>>> math.factorial(10) # 10! = 10 * 9 * 8 * ... * 1  
3628800
```

```
>>> math.pow(3, 3) # 3의 3승 = 3 * 3 * 3  
27.0
```

```
>>> math.sqrt(4) # 제곱근  
2.0
```

여러가지 내장 모듈

: math 모듈 - 로그 함수

- ▶ 첫 번째 매개변수의 로그를 반환, 두 번째 매개변수는 밑수
 - ▶ 두 번째 매개변수가 생략되면 밑수는 자연지수 e 로 간주한다
 - ▶ 밑수가 10인 로그를 위한 `log10` 함수도 별도로 제공

```
>>> math.log(2)
0.6931471805599453
>>> math.log(4, 2)
2.0
>>> math.log10(1000)
3.0
```

여러가지 내장 모듈

: re 모듈 - 정규식(Regular Expression)

```
import re
```

- ▶ `string` 보다 더 전문적으로 문자열을 다룰 수 있는 모듈
- ▶ 문자열 내에서 패턴에 매칭되는 문자열을 추출

```
import re
```

```
list = """
Primary email : skyun.nam@gmail.com
Secondary email : litmuscube@gmail.com
"""

result_list = re.findall(r"(\w+[\w\.\.]*@\w+[\w\.\.]*\.\w+)", list)
print (result_list)
for result in result_list:
    print (result)
```

여러가지 내장 모듈

: random 모듈 - 난수

```
import random
```

- ▶ 임의로 특정 값을 선택해 제공하는 기능 - 주사위의 예
- ▶ random 모듈은 단순히 난수를 발생하는 것 이외에 다양한 기능을 제공

| 함수 | 설명 |
|----------------------------------|--------------------------------------|
| random() | 0 ~ 1 사이의 난수를 발생 |
| randint(s, e) | s ~ e 사이의 정수 난수를 발생 |
| randrange([start,] stop[, step]) | start, stop 간격 step 사이의 수 중에서 난수를 발생 |
| shuffle(seqvar) | 시퀀스 자료형을 섞음 |
| choice(seqvar) | 시퀀스 자료형에서 아무거나 하나를 뽑아줌 |
| sample(seqvar, size) | 시퀀스 자료형에서 size만큼의 값을 임의로 뽑아옴 |

여러가지 내장 모듈

: random 모듈 - 난수

```
>>> import random
>>> random.random() # 0 ~ 1 사이의 난수를 발생
0.05414977057567982

>>> random.randint(1, 6) # 1 ~ 6 사이의 수 중에서 정수 난수를 발생
1

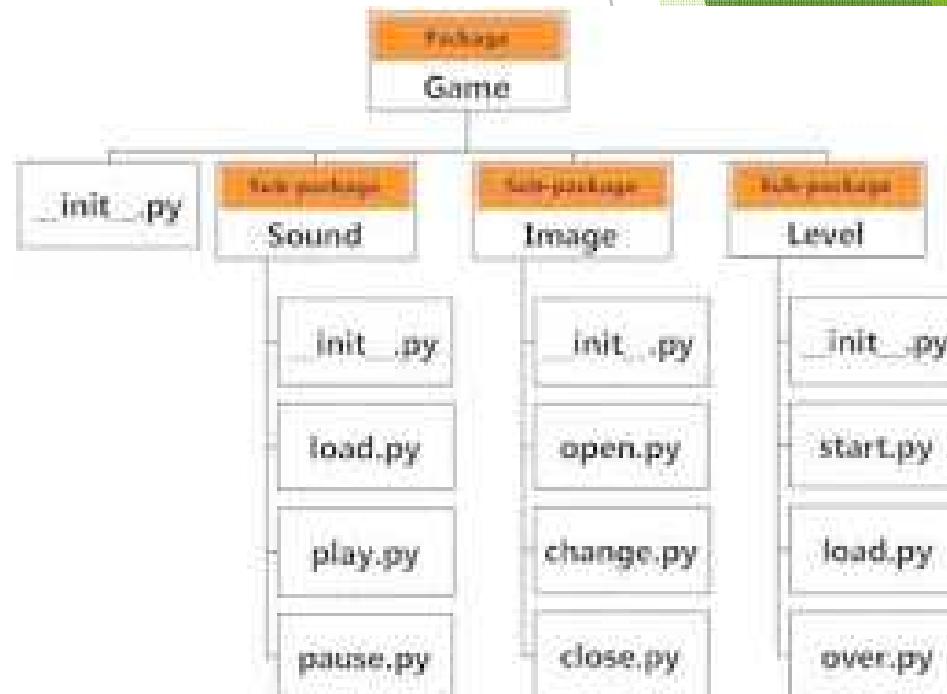
>>> random.randrange(1, 100, 3) # 1 ~ 100 사이 3간격의 수 중에서 난수 발생
40

>>> seqvar = ["짬뽕", "짜장면", "짬짜면"]
>>> random.shuffle(seqvar) # 시퀀스 자료형 섞기
>>> seqvar
['짬짜면', '짜장면', '짬뽕']

>>> random.choice(seqvar) # 시퀀스 자료형 중 임의로 한 개의 값 반환
'짜장면'
```

패키지

- ▶ 모듈을 모아놓은 단위
- ▶ 관련된 여러 개의 모듈을 계층적인 디렉토리로 분류해서 저장하고 관리한다
- ▶ 닷(.)을 이용하여 관리할 수 있다
- ▶ 오른쪽 예제에서 **Game**, **Sound**, **Image**, **Level**은 디렉토리이고 **.py** 파일이 파이썬 모듈이다
- ▶ **__init__.py**는 패키지를 인식시켜주는 역할을 수행
-> 특정 디렉토리가 패키지로 인식되기 위해 필요한 파일



파이썬 프로그래밍 기초

클래스

파이썬 클래스

- ▶ 새로운 이름 공간을 지원하는 단위: 데이터의 설계도
- ▶ 데이터와 데이터를 변경하는 함수(메서드)를 같은 공간 내에 작성
- ▶ 클래스를 정의하는 것은 새로운 자료형을 정의하는 것이고, 인스턴스는 이 자료형의 객체를 생성하는 것이다
- ▶ 클래스와 인스턴스는 각자의 이름공간을 가지게 되며 유기적인 관계로 연결

```
class MyString(str):  
    pass  
  
s = MyString()  
print(type(s))  
print(MyString.__bases__)  
  
s2 = str()  
print(type(s2))  
print(str.__bases__)
```

파이썬 클래스

: 용어 정리

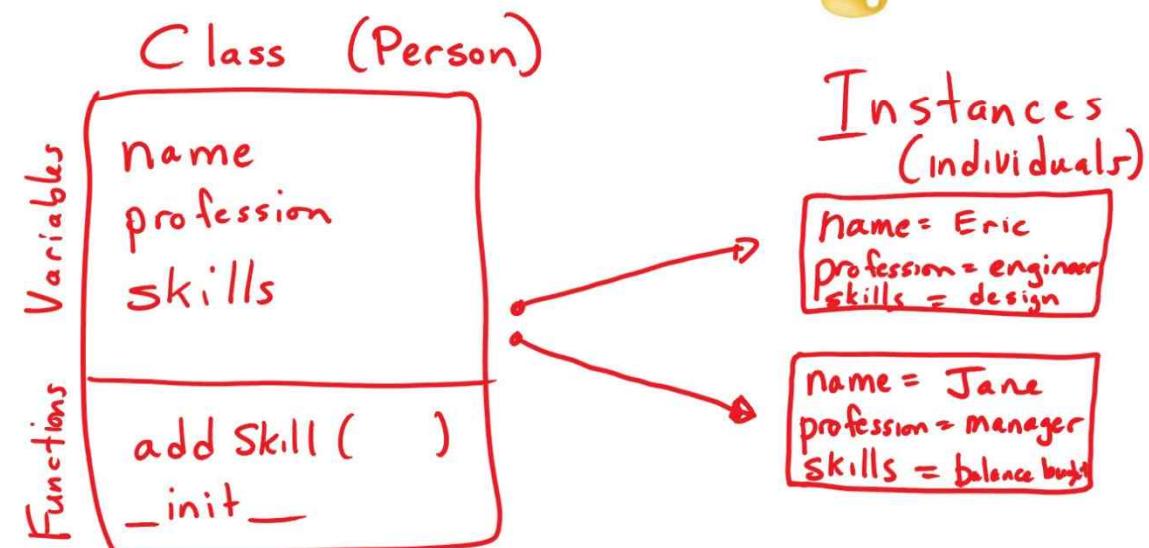
| 용어 | 설명 |
|------------|--|
| 클래스(Class) | <code>class</code> 문으로 정의하며, 멤버와 메서드를 가지는 객체 |
| 클래스 객체 | 어떤 클래스를 구체적으로 가리킬 때 사용 |
| 인스턴스 | 클래스를 호출하여 만들어지는 객체 |
| 인스턴스 객체 | 인스턴스화 된 객체 |
| 멤버 | 클래스가 갖는 변수 |
| 메서드 | 클래스 내에 정의된 함수 |
| 속성 | 멤버 + 메서드 |
| 상위클래스 | 기반 클래스. 어떤 클래스의 상위에 있으며 여러 속성을 상속해준다 |
| 하위클래스 | 파생 클래스. 상위 클래스로부터 여러 속성을 상속 받는다 |

파이썬 클래스

: 클래스 이용의 장점

- ▶ 프로그램의 규모가 커졌을 때 의미 있는 집합체 단위로 프로그램을 정리할 수 있다
- ▶ 설계도(Class)가 있으므로 인스턴스를 양산할 수 있다

Introduction to Python Classes



파이썬 클래스

: 클래스의 생성과 메서드의 정의

```
# point.py
class Point:

    def set_x(self, x):
        self.x = x

    def get_x(self):
        return self.x

    def set_y(self, y):
        self.y = y

    def get_y(self):
        return self.y
```

* 인스턴스 메서드의 첫 번째 인자는 항상 **self**

파이썬 클래스

: 인스턴스 객체 생성과 메서드 호출

▶ Bound Instance Method 호출

```
# paint.py
from point import Point

def main():
    bound_class_method()

def bound_class_method():
    p = Point() # Point 인스턴스 객체 생성
    p.set_x(10)
    p.set_y(10)
    print(p.get_x(), p.get_y(), sep = ', ')

if __name__ == '__main__':
    main()
```



파이썬 클래스

: 인스턴스 객체 생성과 메서드 호출

▶ Unbound Instance Method 호출(참고)

```
# paint.py
from point import Point

def main():
    unbound_class_method()

def unbound_class_method():
    p = Point()
    Point.set_x(p, 10) # 객체를 첫 번째 인자에 할당하고 있음에 유의
    Point.set_y(p, 10) # 객체를 첫 번째 인자에 할당하고 있음에 유의
    print(Point.get_x(p), Point.get_y(p), sep = ', ')

if __name__ == '__main__':
    main()
```

파이썬 클래스

: 정적 메서드(static method)와 클래스 메서드(class method)

- ▶ 인스턴스 객체의 멤버에 접근할 필요가 없는 메서드
- ▶ 첫 번째 인자로 인스턴스 객체 참조값을 받지 않는 클래스 내에 정의된 메서드
- ▶ Class 메서드의 첫 번째 인자는 클래스 객체 참조를 위한 객체 참조값
- ▶ @staticmethod, @classmethod 데코레이터로 손쉽게 구현 가능

```
# in point.py

# ...
@staticmethod
def static_method():
    return "static_method() 호출"

@classmethod
def class_method(cls): # => 클래스 참조를 위한 객체 참조값
    return "class_method() 호출"

# ...
```

파이썬 클래스

: 클래스 멤버와 인스턴스 멤버

| 종류 | 이름 공간 | 공유 범위 |
|---------|--------------|------------------|
| 클래스 멤버 | 클래스 이름 공간 내 | 모든 인스턴스 객체들에 공유 |
| 인스턴스 멤버 | 인스턴스 이름 공간 내 | 개별 인스턴스 객체에서만 참조 |

```
# in point.py

class Point:
    count_of_instance = 0

    def set_x(self, x):
        self.x = x
```

클래스 멤버 정의

인스턴스 멤버 정의

파이썬 클래스

: 클래스 멤버와 인스턴스 멤버 접근

- ▶ 인스턴스 객체에서 참조하는 멤버의 객체를 찾는 순서는 아래와 같다
 - ▶ 인스턴스 멤버
 - ▶ 인스턴스 멤버가 없다면 클래스 멤버를 찾음

```
# in paint.py
```

```
def test_member():
    p = Point()
    Point.set_x(p, 10)
    Point.set_y(p, 10)
    print('x={0}, y={1}, count_of_instance={2}'.format(p.x, p.y, p.count_of_instance))
```

파이썬 클래스

: 생성자와 소멸자

- ▶ 생성자 : 클래스가 인스턴스화 될 때 실행되는 내용. 초기화.
 - ▶ `__init__` 메서드 내에 작성
- ▶ 소멸자 : 클래스 인스턴스가 제거될 때 실행되는 내용.
 - ▶ `__del__` 메서드 내에 작성

```
# in point.py

# ...
def __init__(self, x=0, y=0):
    self.x, self.y = x, y
    Point.count_of_instance += 1

def __del__(self):
    Point.count_of_instance -= 1
# ...
```

파이썬 클래스

: __str__ 메서드

- ▶ 객체를 문자열로 반환하는 함수

```
# in point.py

# ...
def __str__(self):
    return "Point({0}, {1})".format(self.x, self.y)
# ...
```

```
# in paint.py

def test_to_string():
    p = Point()
    print(p)

# => Point(0, 0)
```

파이썬 클래스

: __repr__ 메서드

- ▶ __str__ 과 비슷하지만 "문자열로 객체를 다시 생성할 수 있기 위해" 사용
- ▶ Eval을 수행하면 다시 그 해당 객체가 생성될 수 있어야 한다

```
# in point.py

# ...
def __repr__(self):
    return "\"Point({0}, {1})\"".format(self.x, self.y)
# ...
```

```
# in paint.py

def test_to_string():
    p = Point()
    print(p)
    print(repr(p))

p2 = eval(repr(p))
print(p2)
```

파이썬 클래스

: __str__ vs __repr__

| | <u>__str__</u> | <u>__repr__</u> |
|----|----------------|-----------------------|
| 구분 | 비공식적 문자열 출력 | 공식적 문자열 출력 |
| 목적 | 사용자가 보기 쉽게 | 문자열로 객체를 다시 생성할 수 있도록 |
| 대상 | 사용자(End User) | 개발자(Developer) |

파이썬 클래스

: 연산자 재정의(Operator Overloading)

- ▶ 연산자에 대해 클래스에 새로운 동작을 정의하는 것
- ▶ 파이썬의 클래스는 새로운 데이터 형을 정의하는 것이므로 그에 상응하는 연산자의 재정의가 필요할 수 있다
- ▶ 연산자가 정의되어 있지 않으면 `TypeError`가 발생
- ▶ 파이썬에서는 사용하는 거의 모든 연산에 대해 새롭게 정의할 수 있다
 - ▶ 수치 연산자 오버로딩
 - ▶ 역이항 연산자 오버로딩
 - ▶ 확장 산술 연산자 오버로딩
 - ▶ 비교 연산자 오버로딩

파이썬 클래스

: 수치 연산자 오버로딩

| 연산자 | 연산자 메서드 |
|-----------|---------------------------|
| + | <code>__add__</code> |
| - | <code>__sub__</code> |
| * | <code>__mul__</code> |
| / | <code>__truediv__</code> |
| // | <code>__floormod__</code> |
| % | <code>__mod__</code> |
| divmod() | <code>__divmod__</code> |
| pow(), ** | <code>__pow__</code> |
| << | <code>__lshift__</code> |
| >> | <code>__rshift__</code> |
| & | <code>__and__</code> |
| ^ | <code>__xor__</code> |
| | <code>__or__</code> |

▶ + 연산자 오버로딩 예제

```
class MyString:

    def __init__(self, s):
        self.s = s

    def __add__(self, other):
        return self.s + other

print(MyString("Life is too short, ") + "You need Python!")
```

파이썬 클래스

: 역이행 연산자 오버로딩

| 연산자 | 연산자 메서드 |
|------------------------|----------------------------|
| + | <code>__radd__</code> |
| - | <code>__rsub__</code> |
| * | <code>__rmul__</code> |
| // | <code>__rfloormod__</code> |
| % | <code>__rmod__</code> |
| <code>divmod()</code> | <code>__rdivmod__</code> |
| <code>pow(), **</code> | <code>__rpow__</code> |
| <code><<</code> | <code>__rlshift__</code> |
| <code>>></code> | <code>__rrshift__</code> |
| <code>&</code> | <code>__rand__</code> |
| <code>^</code> | <code>__rxor__</code> |
| <code> </code> | <code>__ror__</code> |

▶ + 역이행 연산자 오버로딩 예제

```
class MyString:

    def __init__(self, s):
        self.s = s

    def __add__(self, other):
        return self.s + other

    def __radd__(self, other):
        return other + self.s

print(MyString("Life is too short, ")
    + MyString("You need Python!"))
```

파이썬 클래스

: 확장 산술 연산자 오버로딩

| 연산자 | 연산자 메서드 |
|------------------------|----------------------------|
| <code>+=</code> | <code>__iadd__</code> |
| <code>-=</code> | <code>__isub__</code> |
| <code>*=</code> | <code>__imul__</code> |
| <code>//=</code> | <code>__ifloormod__</code> |
| <code>/=</code> | <code>__idiv__</code> |
| <code>%=</code> | <code>__imod__</code> |
| <code>**=</code> | <code>__ipow__</code> |
| <code><<=</code> | <code>__ilshift__</code> |
| <code>>>=</code> | <code>__irshift__</code> |
| <code>&=</code> | <code>__iand__</code> |
| <code>^=</code> | <code>__ixor__</code> |
| <code> =</code> | <code>__ior__</code> |

▶ `+=, -=` 연산자 오버로딩 예제

```
# in point.py
def __iadd__(self, other):
    return Point(self.x + other.x, self.y + other.y)

def __isub__(self, other):
    return Point(self.x - other.x, self.y - other.y)
```

```
>>> from point import Point
>>> p = Point(10, 10)
>>> print(p += Point(20, 15))
>>> p += Point(20, 15) # __iadd__ 메서드로 구현된 += 연산자
>>> print(p) Point(30, 25)
```

파이썬 클래스

: 비교 연산자 오버로딩

| 연산자 | 연산자 메서드 |
|-----|---------------------|
| < | <code>__lt__</code> |
| <= | <code>__le__</code> |
| > | <code>__gt__</code> |
| >= | <code>__ge__</code> |
| == | <code>__eq__</code> |
| != | <code>__ne__</code> |

- ▶ `==`, `<`, `>` 연산자 오버로딩 예제

```
# in point.py
class Rect:

    def __eq__(self, other):
        return self.area() == other.area()

    def __lt__(self, other):
        return self.area() < other.area()

    def __gt__(self, other):
        return self.area() > other.area()
```