Python

Kim Hye Kyung
topickim@naver.com

구축

Ⅰ Python 개요 및 특징

Ⅱ 설치 및 개발 환경 구축

Ⅲ Python 기초 문법

Ⅳ Python 프로그래밍

Ⅴ 객체 지향 프로그래밍

Python 개요 및 특징

Python 개요

- 1991년 귀도 반 로썸(Guido van Rossum)이 발표한 인터프리터 언어
- 구글의 3대 개발 언어 중에 하나로 채택되면서 사용자층이 늘어남

Python의 특징

생산성이 좋음	"Life is too short, You need python."
풍부한 라이브러리	광범위한 라이브러리가 내장되어 있고 확장성이 뛰어남 외부 라이브러라가 풍부하며 확장성이 좋음
가독성	간결하고 가독성이 좋음 코드 블록의 들여쓰기(indentation)으로 구분
접착성	쉽게 라이브러리 추가가 가능
유니코드	문자열을 유니코드로 처리함으로 한글, 중국어, 영어 문제없이 처리
동적 타이핑	런타임 시에 타입 체크를 하는 동시에 자동으로 메모리 관리

Python 종류

Cpython	c로 작성된 Python
Jpython	자바로 구현된 Python 자바 가상 머신에서 작동
IronPython	.NET과 Mono용으로 개발된 Python C#으로 구현되어 있음
PyPy	Python으로 구현된 Python

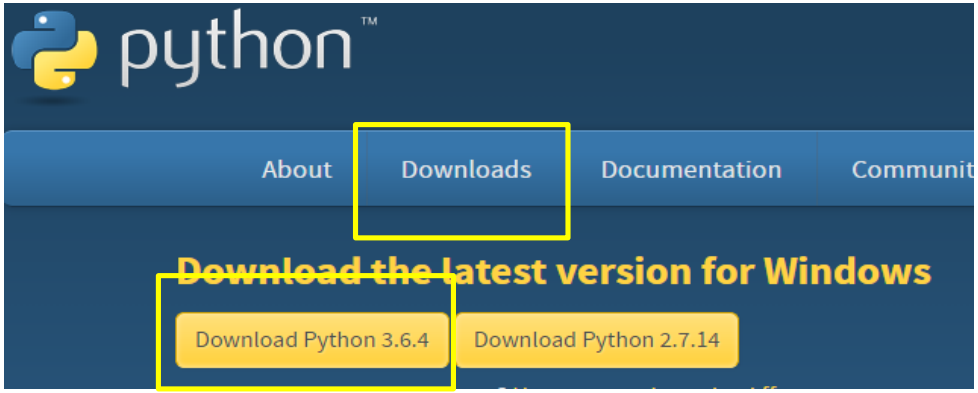
Python으로 구현된 프로젝트들

- 유투브, Google Maps, Gmail등
- Django, Yum
- 구글의 Tensorflow 머신러닝 프레임워크등

설치 및 개발 환경 구축

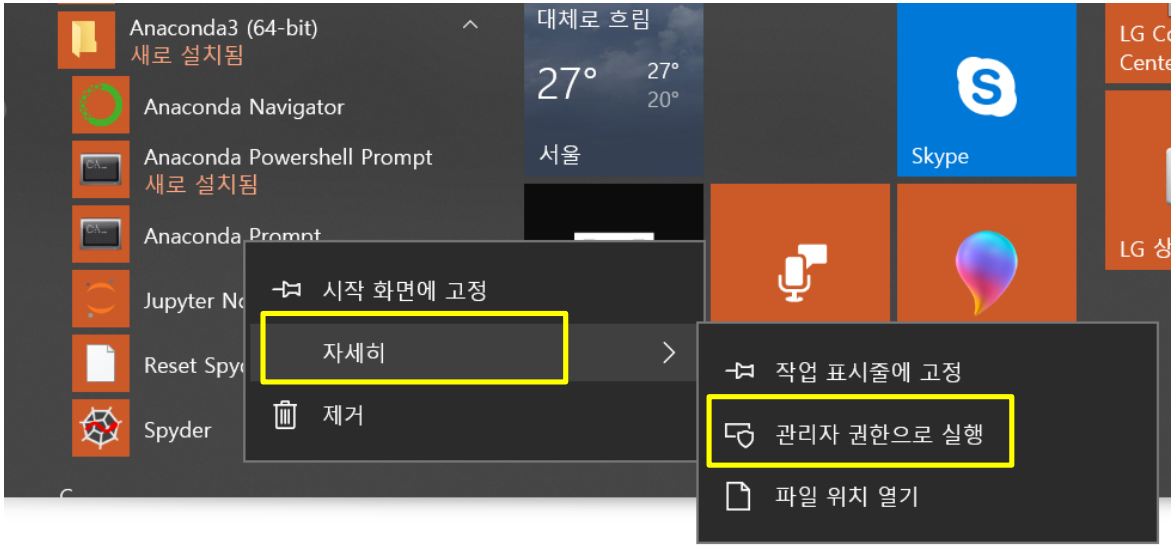
설치 및 개발 환경 구축

- 개발 환경 셋팅
  - python
    - https://www.python.org/
  - 아나콘다
    - https://www.anaconda.com
  - 개발 tool
    - Jupyter notebook
    - Visual Studio code
  - 참고 사이트
    - http://www.pythontutor.com/



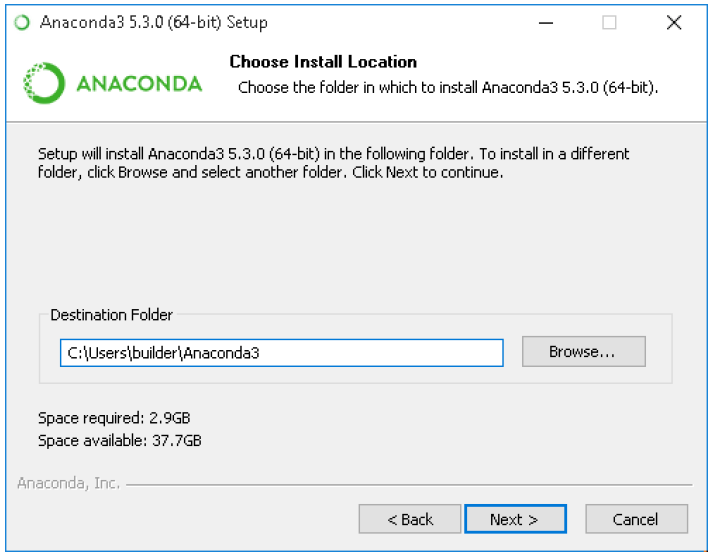
설치 및 개발 환경 구축

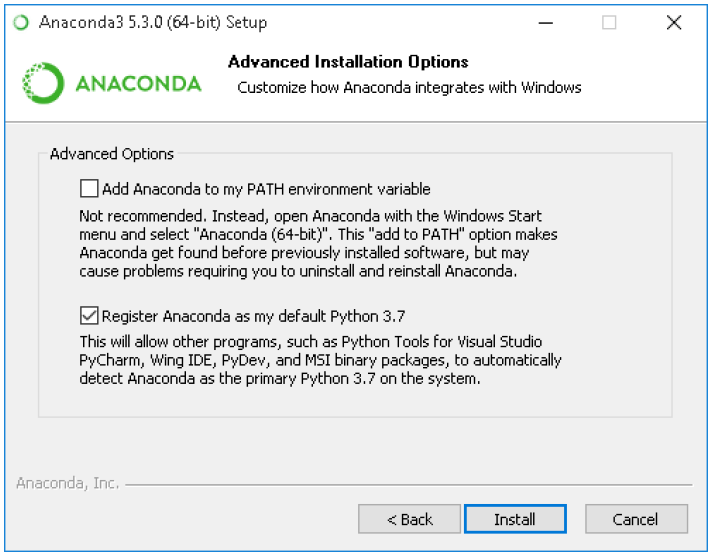
- Anaconda 설치
  - window10 환경에서의 주의 사항
    - 관리자 원한으로 prompt 실행해야 제대로 패키지 설치
    - python package를 내려 받아 저장하려면 관리자 권한이어야 함

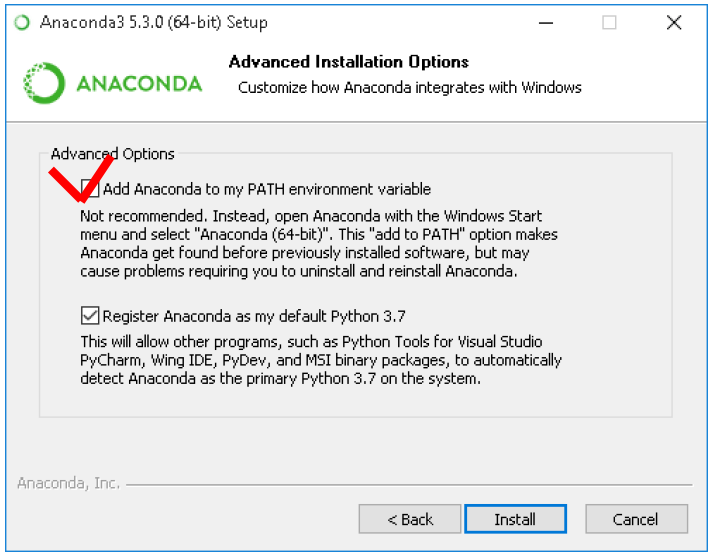


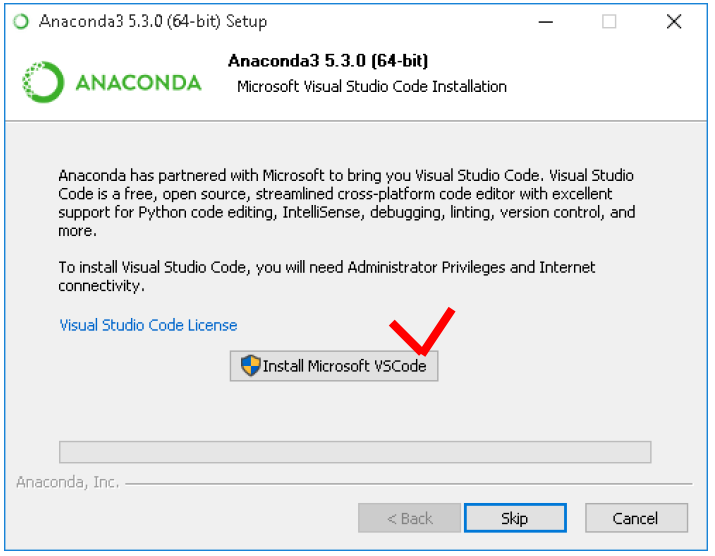

```
(base) C:₩Windows₩system32>python -V
Python 3.7.3
```

설치 및 개발 환경 구축

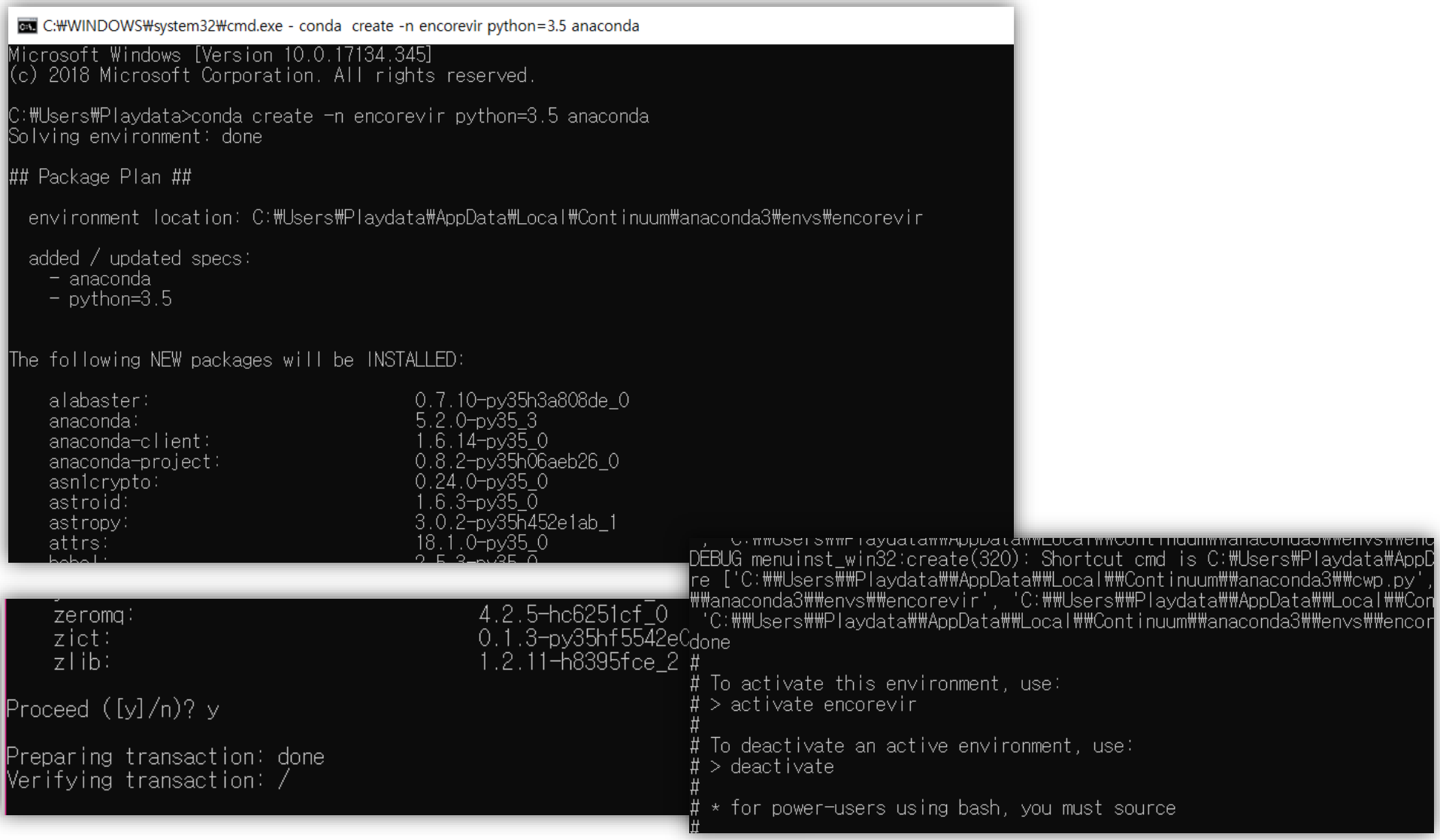

Anaconda3 5.3.0 (64-bit) Setup
ANACONDA
Choose Install Location
Choose the folder in which to install Anaconda3 5.3.0 (64-bit).
Setup will install Anaconda3 5.3.0 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Users\builder\Anaconda3
Browse...
Space required: 2.9GB
Space available: 37.7GB
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 5.3.0 (64-bit) Setup
ANACONDA
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda as my default Python 3.7
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.
Anaconda, Inc.
< Back
Install
Cancel


Anaconda3 5.3.0 (64-bit) Setup
ANACONDA
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda as my default Python 3.7
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.
Anaconda, Inc.
< Back
Install
Cancel


Anaconda3 5.3.0 (64-bit) Setup
ANACONDA
Anaconda3 5.3.0 (64-bit)
Microsoft Visual Studio Code Installation
Anaconda has partnered with Microsoft to bring you Visual Studio Code. Visual Studio Code is a free, open source, streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more.
To install Visual Studio Code, you will need Administrator Privileges and Internet connectivity.
Visual Studio Code License
Install Microsoft VSCode
Anaconda, Inc.
< Back
Skip
Cancel

가상 환경 구축


C:₩WINDOWS₩system32₩cmd.exe - conda create -n encorevir python=3.5 anaconda
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. All rights reserved.
C:₩Users₩Playdata>conda create -n encorevir python=3.5 anaconda
Solving environment: done
Package Plan
environment location: C:₩Users₩Playdata₩AppData₩Local₩Continuum₩anaconda3₩envs₩encorevir
added / updated specs:
- anaconda
- python=3.5
The following NEW packages will be INSTALLED:
alabaster: 0.7.10-py35h3a808de_0
anaconda: 5.2.0-py35_3
anaconda-client: 1.6.14-py35_0
anaconda-project: 0.8.2-py35h06aeb26_0
asn1crypto: 0.24.0-py35_0
astroid: 1.6.3-py35_0
astropy: 3.0.2-py35h452e1ab_1
attrs: 18.1.0-py35_0
zeromq: 4.2.5-hc6251cf_0
zict: 0.1.3-py35hf5542eC
zlib: 1.2.11-h8395fce_2
Proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: /
DEBUG menuinst_win32:create(320): Shortcut cmd is C:₩Users₩Playdata₩AppD
re ['C:₩₩Users₩₩Playdata₩₩AppData₩₩Local₩₩Continuum₩₩anaconda3₩₩cwp.py',
₩₩anaconda3₩₩envs₩₩encorevir', 'C:₩₩Users₩₩Playdata₩₩AppData₩₩Local₩₩Con
'C:₩₩Users₩₩Playdata₩₩AppData₩₩Local₩₩Continuum₩₩anaconda3₩₩envs₩₩encor
done
#
To activate this environment, use:
> activate encorevir
#
To deactivate an active environment, use:
> deactivate
#
* for power-users using bash, you must source
#

프로그램 용어

1. 쉘(shell) 프로그램
    1. 터미널이나 터미널 창을 사용하면 입력한 문자를 읽고 실행하여 결과를 보여주는 프로그램 의미
    2. 윈도우의 쉘
        1. cmd(command의 약어)
        2. .bat 확장자의 배치 파일 실행

2. python shell
    1. 한 번에 하나의 명령어가 실행되고 실행 결과가 화면에 나타남
    2. >>> : 프롬프트라 함

```
C:\Users\khk>python
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

python 특징

1. 대화식 인터프리터
    1. 기계어 코드로 변환하는 컴파일 단계가 없음
    2. 기계어 (machine language) : 컴퓨터가 알아듣는 유일한 언어
        - 기계어는 0과 1로 구성
    - 코드 입력 후 결과를 바로 확인 가능

```
C:\Users\khk>python
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

2. text 파일로 저장 및 실행 가능
    1. 확장자 : *.py
3. python2 버전과 python3 버전은 호환되지 않음
4. 개발 생산성이 좋음
5. python 철학

Python 환경 및 인코딩

- 버전 확인
  - >python –version

```
C:₩Users₩Playdata>python --version
Python 3.6.6 :: Anaconda custom (64-bit)
```

- 소스 코드 인코딩
  - 기본적으로, UTF-8로 처리
  - 기본값 외의 것으로 선언하려 할 경우

```
# -*- coding: encoding -*-
```

Python 기초 문법

주석

- 소스에 붙이는 설명글

```
# 한줄 주석

'''
    다중 라인 주석
'''

"""
    다중 라인 주석
"""
```

수식과 연산자

- 수식(expression)이란?
  - 피연산자와 연산자의 조합

```
3 * 6
- 연산자 : 어떤 연산을 나타내는 기호 : *
- 피연산자 : 연산의 대상이 되는 값들 : 3과 6
```

- 연산자

```
+       -       *       **      /       //      %       @
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=
```

- 참고
  - / : Python에서 나누기는 항상 실수로 계산됨
  - // : 정수로 계산하고자 할 경우

구분자

```
(       )       [       ]       {       }
,       :       .       ;       @       =       ->
+=      -=      *=      /=      //=     %=      @=
&=      |=      ^=      >>=     <<=     **=
```

키워드

- 식별자들은 예약어, 또는 언어의 키워드로 사용
- 일반적인 식별자로 사용될 수 없음

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

연산자

```
+       -       *       **      /       //      %       @
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=
```

```
>>> 1+2
3
>>> 'a'
'a'
>>> 'a' + 'b'
'ab'
>>> type(1)
<class 'int'>
>>> type('a')
<class 'str'>
>>> a = 95
>>> a -= 3
>>> a
92
>>> print(a)
92
>>> int(a)
92
>>>
```

문자열 슬라이스

0부터 시작, 끝부분에서는 해당 위치는 제외

```
>>> a = 'python'
>>>a[0:1]
'p'
>>>a[1:4]
'yth'
>>>a[:2]
'py'
>>>a[-2:]
'on'
>>>a[:]
'python'
>>>a[::2] ➔ 2칸씩 이동하며 글자 추출
'pto'
>>>a[-1]
'n'
```

0	1	2	3	4	5
p	y	t	h	o	n
-6	-5	-4	-3	-2	-1

기타 제어 흐름

- If 문

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

- for 문

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

반복문

```
for x in 시퀀스 :
        반복구문
```

```
초기식;
while 조건식:
    true인 경우 실행
print("반복이 종료되었습니다.")
```

```
for x in range(5) :
        반복구문
```

range() 함수를 이용하면 특정 구간의 정수 표현 가능

range(10)하면 0부터 9까지의 정수가 생성

for 문 - 정해진 횟수만큼 반복하는 구조

while 문 - 어떤 조건이 만족되는 동안, 반복을 계속하는 구조

기타 제어 흐름

- range() 함수
  - 내장 함수로 수열 생성
  - List인듯 동작 하나 list 아님, 실제로 list를 만들지 않아 공간이 절약됨

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

```
range(5, 10)
   5, 6, 7, 8, 9

range(0, 10, 3)
   0, 3, 6, 9

range(-10, -100, -30)
  -10, -40, -70
```

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

루프의 break와 continue문

- break
  - 반복문 종료
    - 소수를 찾는 예시

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
```

- continue
  - 루프의 다음 반복을 수행

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
```

데이터 타입

- 데이터 타입
  - 부울, 정수, 실수, 문자열

- python에선 모든 것이 객체
  - 모든 것(부울, 정수, 실수, 문자열, 데이터 구조, 함수, 프로그램)이 객체로 구현되어 있음
  - 다른 언어에는 결여된 언어 일관성과 유용한 기능을 제공

데이터 타입

자료형	설명	사용 예
숫자형	정수,실수	200, 12345L, 7.23
문자열	문자들의 시퀀스	‘lee’, “kim”, “park”
리스트	순서를 가지는 Python 임의 객체의 집합	[“lee”,“kim”]
사전	순서를 가지지 않는 객체의 집합. 키로 데이터 사용	{“lee”:4, “kim”:5}
튜플	순서를 가지는 Python 임의 객체의 집합으로 내용 변경 불가	(“lee”, “kim”)
세트	집합 형태로 사용 합집합, 교집합, 차집합	{1,2,3,4}

python 자료구조

리스트	[값1, 값2, 값3,...] 데이터를 순차적으로 파악하는데 유용 값 변경 가능, [] or list()로 생성
튜플	(값1, 값2, 값3) 값 변경 불가, () 표기로 생성
딕셔너리	{key1:value1, key2:value2, ...} 리스트와 흡사, 단 0과 1등의 index 값으로 데이터 사용 불가, {} 표기로 생성
셋	{value1, value2, ...} 중복 불허, 키만 버린 딕셔너리와 흡사, set()으로 생성

튜플 대입 연산

```
>>> student1 = ("철수", 19, "CS")
>>> (name, age, major) = student1
>>> name
'철수'
>>> age
19
>>> major
'CS'
```

딕셔터리

```
>>> contacts = {'Kim':0100000000',  'Park':0102222222' }

>>> contacts['Kim']
'0100000000'

>>> contacts.get('Kim')
'0102222222'

>>> contacts.pop("Kim")
' 0102222222'
```

```
>>> if "Kim" in contacts:
        print("키가 딕셔너리에 있음")
```

```
>>> contacts
{'Park': 0102222222'}
```

셋

- 수학에서의 집합
- 중복되지 않은 항목들이 모인 것
- 세트의 항목 간에는 순서가 없음

```
>>> numbers = {2, 1, 3}
>>> numbers
{1, 2, 3}

>>> len(numbers)
3

>>> fruits = { "Apple", "Banana", "Pineapple" }
>>> mySet = { 1.0, 2.0, "Hello World", (1, 2, 3) }
```

set in, 요소 추가하기

```
>>> numbers = {2, 1, 3}
>>> if 1 in numbers:
        print("집합 안에 1이 있습니다.")

집합 안에 1이 있습니다.
```

```
>>> numbers = {2, 1, 3}
>>> for x in numbers:
        print(x, end=" ")

1 2 3
```

```
>>> numbers = { 2, 1, 3 }
>>> numbers[0]
...
TypeError: 'set' object does not support indexing

>>> numbers.add(4)
>>> numbers
{1, 2, 3, 4}
```

부분 및 집합 연산

```
>>> A = {1, 2, 3}
>>> B = {1, 2, 3}
>>> A == B
True

>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B < A
True

>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B.issubset(A)
True
```

```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5}

>>> A | B
{1, 2, 3, 4, 5}



>>> A & B
{3}

>>> A - B
{1, 2}
```

Python Programming

함수

- def 키워드 사용

```
def  함수이름 ( [매개변수1, 매개변수2, ...]):
        문장1
        문장2
```

```
>>>def say_hello(name):
        print("안녕, ", name);
```

```
>>>say_hello("철수")
안녕, 철수
```

```
>>>def say_hello(name, msg):
        print("안녕, ", name, "야, ", msg);
```

```
>>>name="철수"
>>>msg="가 운동을 합니다 "
>>>say_hello(name, msg)
철수가 운동을 합니다
```

반환값이 있는 함수

```
>>>def getSum(start, end) :
        sum=0
        for i in range(start, end+1) :
                sum += i
        return sum
```

```
>>>value = getSum(1, 10)
>>>print(value)
55
```

키워드 인자

- 함수는 "변수명=기본값" 형식의 키워드 인자 를 사용해서 호출될 수 있음

```
1  def persons(name1, name2='신사임당'):
2      print(name1)
3      print(name2)
```

```
1  persons("율곡")
2  print("-----------------------")
3  persons(name1 = '율곡')
4  print("-----------------------")
5  persons(name1 = '율곡', name2='이이')
```

```
율곡
신사임당
-----------------------
율곡
신사임당
-----------------------
율곡
이이
```

키워드 인자

- 인수 이름으로 값을 전달하는 방식
- 변수의 이름으로 특정 인수를 전달할 수 있음

```
1  def shopping(kind, *arguments, **keywords):
2      print("구매 희망 품목 : ", kind)
3
4      for arg in arguments:
5          print(arg)
6
7      print("-" * 40)
8
9      for kw in keywords:
10         print(kw, ":", keywords[kw])
```

```
1  shopping("비누", "사과비누", "자몽비누", 물비누 = "오렌지향",고체비누="딸기향")
```

```
구매 희망 품목 :  비누
사과비누
자몽비누
----------------------------------------
물비누 : 오렌지향
고체비누 : 딸기향
```

임의의 인자 목록

- 인자의 개수가 정해지지 않은 가변 인자를 전달
- * 를 사용하며 인수는 튜플 형식으로 전달됨

```
>>> def concat(*args, sep="/"):
...     return sep.join(args)
...
>>> concat("earth", "mars", "venus")
'earth/mars/venus'
>>> concat("earth", "mars", "venus", sep=".")
'earth.mars.venus'
```

람다 표현식

- 이름이 없는 한 줄짜리 함수
- 한 줄의 간단한 함수가 필요한 경우
  - lambda a, b: a+b. 함수 객체가 있어야 하는 곳이면 어디나 람다 함수가 사용 될 수 있음

lambda 인수 : <구문>
(lambda <인수> : <인수 활용 수행 코드>)(<인자>)

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

자료구조

리스트	[값1, 값2, 값3,...] 데이터를 순차적으로 파악하는데 유용 값 변경 가능, [] or list()로 생성
튜플	(값1, 값2, 값3) 값 변경 불가, () 표기로 생성
딕셔너리	{key1:value1, key2:value2, ...} 리스트와 흡사, 단 0과 1등의 index 값으로 데이터 사용 불가, {} 표기로 생성
셋	{value1, value2, ...} 중복 불허, Key만 버린 딕셔너리와 흡사, set()으로 생성

리스트

- 리스트 함수

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4)  # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

리스트를 스택으로 사용하기

- append() : 리스트를 스택으로 사용하기 쉽게 구성("last-in, first-out")
- pop() : 스택의 꼭대기에서 값을 꺼내려면 명시적인 인덱스 없이

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

리스트를 큐로 사용하기

- 큐를 구현하려면, 양 끝에서의 덧붙이기와 꺼내기가 모두 빠르도록 설계된 collections.deque 를 사용

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

리스트 컴프리헨션

- 리스트를 만드는 간결한 방법을 제공

[<반복 실행문> for <반복 변수> in <반복 범위>]

조건문 포함

[<반복 실행문> for <반복 변수> in <반복 범위> if <조건문>]

리스트 컴프리헨션

- 리스트를 만드는 간결한 방법을 제공

```
squares = []
for x in range(10):
    squares.append(x**2)

squares
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
squares = list(map(lambda x: x**2, range(10)))
squares
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
squares = [x**2 for x in range(10)]
squares
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

리스트 컴프리헨션

- 리스트 컴프리헨션의 예제

```
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
combs = []
for x in [1,2,3]:
    for y in [3,1,4]:
        if x != y:
            combs.append((x, y))

combs
```

```
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

del 문

- 리스트에서 값 대신에 인덱스를 사용해서 항목을 삭제

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

튜플

- 불변 데이터
- 쉼표로 구분되는 여러 값으로 구성
- 괄호로 표현
- 리스트와 같은 가변 객체를 포함해서 튜플 생성 가능

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

튜플 패킹

튜플 언패킹

```
t = 1.2, 5, 'string'
x, y, z = t
```

```
x
```

```
1.2
```

```
y
```

```
5
```

```
z
```

```
'string'
```

집합

- 중복되는 요소가 없는 순서 없는 컬렉션
- 합집합, 교집합, 차집합, 대칭 차집합과 같은 수학적인 연산들도 지원
- 중괄호나 set() 함수를 사용

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership testing
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

딕셔너리

- 키: 값 쌍의 순서 없는 집합

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

딕셔너리에 존재하는 모든 키의 list를 return
순서는 정해져 있지 않음

딕셔너리에 존재하는 모든 키의 list를 return
정력 가능

딕셔너리

- dict() 생성자는 키-값 쌍들의 시퀀스로 부터 직접 딕셔너리를 구성

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

- 컴프리헨션은 임의의 키와 값 표현식들로 부터 딕셔너리를 만드는데 사용

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

- 키가 간단한 문자열일 때, 때로 키워드 인자들을 사용해서 쌍을 지정하기 쉬움

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

루프 테크닉

- items() : 딕셔너리로 반식시에 Key와 대응하는 Valu를 획득 가능

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

- enumerate() : 위치 인덱스와 대응하는 값을 동시에 획득 가능

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

루프 테크닉

- zip() : 둘이나 그 이상의 시퀀스를 동시에 반복시 엔트리들의 쌍을 구성할 수 있음

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

- reversed() : 시퀀스를 거꾸로 루핑하려면, 정방향으로 시퀀스를 지정한 후 호출 가능

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

루프 테크닉

- sorted() : 정렬된 순서로 시퀀스를 루핑

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for f in sorted(set(basket)):
...     print(f)
...
apple
banana
orange
pear
```

- 새로운 리스트 구성

```
>>> import math
>>> raw_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]
>>> filtered_data = []
>>> for value in raw_data:
...     if not math.isnan(value):
...         filtered_data.append(value)
...
>>> filtered_data
[56.2, 51.7, 55.3, 52.5, 47.8]
```

출력 포매팅

- 제곱수와 세제곱수의 표를 쓰는 두 가지 방법

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     # Note use of 'end' on previous line
...     print(repr(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000

>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

왼쪽에 스페이스를 채워서 주어진 폭으로 문자열을 우측 줄에 맞춤

출력 포매팅

- str.zfill() 숫자 문자열의 왼쪽에 0을 채움
- 플러스와 마이너스 부호도 이해

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
```

- str.format() 메서드의 기본적인 사용법

```
>>> print('We are the {} who say "{}!"'.format('knights', 'Ni'))
We are the knights who say "Ni!"
```

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

출력 포매팅

- ':' 뒤에 정수를 전달하면 해당 필드의 최소 문자 폭,
- 표를 보기 좋게 만들게 좋음

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print('{0:10} ==> {1:10d}'.format(name, phone))
...
Jack       ==>       4098
Dcab       ==>       7678
Sjoerd     ==>       4127
```

- '**' 표기법을 사용해서 table을 키워드 인자로 전달해도 같은 결과를 얻음

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab: {Dcab:d}'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

파일 읽고 쓰기

- open() 은 파일 객체 반환
- 두 개의 인자를 주는 방식이 가장 많이 사용
  - open(filename, mode).

```
>>> f = open('workfile', 'w')
>>> f.close()
```

```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

에러와 예외

- 문법 에러(파싱 에러)

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

- 예외
  - 문장이나 표현식이 문법적으로 올바르다 할지라도, 실행하려고 하면 에러를 일으킬 수 있음

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

예외 처리하기

- 예외를 처리

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

예외 발생하기

- raise :
  - 프로그래머가 지정한 예외를 발생 시킬 수 있음

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
```

사용자 정의 예외

- 새 예외 클래스 개발 가능, 자신의 예외에 이름을 붙일 수 있음

```
class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class InputError(Error):
    """Exception raised for errors in the input.

    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """

    def __init__(self, expression, message):
        self.expression = expression
        self.message = message

class TransitionError(Error):
    """Raised when an operation attempts a state transition that's not
    allowed.

    Attributes:
        previous -- state at beginning of transition
        next -- attempted new state
        message -- explanation of why the specific transition is not allowed
    """

    def __init__(self, previous, next, message):
        self.previous = previous
        self.next = next
        self.message = message
```

필수로 실행되는 문장

- finally

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
...
Goodbye, world!
KeyboardInterrupt
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
```

객체 지향 프로그래밍

객체와 클래스

- Python에선 모든 것이 객체

- 객체지향 프로그래밍
  - 실제 세계가 객체들로 구성되어 있는 것과 흡사하게, SW도 객체로 구성하는 방법

- 객체
  - 상태와 동작으로 구성
  - 상태 = 변수
  - 동작 = 메소드

클래스

- 객체 지향 프로그래밍

Python에선 모든것이 객체

  - 상태와 동작 보유
  - 속성(변수) – 객체의 상태
  - 기능(메소드) – 객체의 동작
  - 클래스란? 객체에 대한 설계도
  - 인스턴스란? 클래스로부터 만들어진 각각의 객체

Python에서의 변수 종류

지역 변수	함수 안에 선언된 변수
클래스(전역) 변수	클래스(전역) 변수 – 함수 외부에서 선언된 변수
인스턴스 변수	클래스 안에 선언된 변수, self. 적용

정보 은닉을 위한 private 적용하기

- 구현의 세부 사항을 클래스 안에 감추는 것
- private으로 정의하는 방법
  - 변수명 앞단에 __ 적용하기

```
class Student:
        def __init__(self, name=None, age=0):
                self.__name = name
                self.__age = age


obj=Student()
print(obj.__age)
```

```
...
AttributeError: 'Student' object has no attribute '__age'
```

정보 은닉

- 인스턴스 변수를 private으로 정의하기
  - 변수 선언시 __(두개의 under line) 사용
  - 유효한 값 대입을 위한 setter/활용을 위한 getter 메소드 필요

```
1  #정보 은닉
2  class Student:
3      def __init__(self, name=None, age=0):
4          self.__name = name
5          self.__age = age
6
7  obj = Student()
8  print(obj.__age)
9
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-15-cebcaa9fa125> in <module>
      6
      7 obj = Student()
----> 8 print(obj.__age)

AttributeError: 'Student' object has no attribute '__age'
```

정보 은닉

```
1  class Student:
2      def __init__(self, name=None, age=0):
3          self.__name = name
4          if(age < 1):
5              self.__age = 0
6          else:
7              self.__age=age
8
9      def getAge(self):
10         return self.__age
11
12     def getName(self):
13         return self.__name
14
15     def setAge(self, age):
16         if(age < 1):
17             self.__age = 0
18         else:
19             self.__age=age
20
21     def setName(self, name):
22         self.__name=name
```

```
24 obj = Student("Lee", 10)
25 print( obj.getName(), ' ', obj.getAge())
26 print('----------------------------')
27 obj.setAge(-11)
28 print( obj.getName(), ' ', obj.getAge())
```

```
Lee   10
----------------------------
Lee   0
```

클래스 변수

- 모든 객체가 공유하는 멤버 변수

```python
# 클래스 변수 이해를 위한 적용 전 예제
class Television:
    def __init__(self, channel, volume, on):
        self.channel = channel
        self.volume = volume
        self.on = on

    def setChannel(self, channel):
        self.channel = channel

    def getChannel(self):
        return self.channel

    def __str__(self):
        return "채널:%s , 볼륨:%s, 전원:%s " % (self.channel, self.volume, self.on)
```

```python
t = Television(9, 10, True)
print(t)

print("--------------------------")
t = Television(9, 10, True)
print(t)
```

```
채널:9 , 볼륨:10, 전원:True
--------------------------
채널:9 , 볼륨:10, 전원:True
```

클래스 변수

- 클래스 변수 호출
  - 클래스명으로 호출 해야 함

```python
# 클래스 변수 이해를 위한 적용 예제 - 클래스 이르므로 호출
class Television:
    seralNo = 0
    def __init__(self, channel, volume, on):
        Television.seralNo += 1
        self.channel = channel
        self.volume = volume
        self.on = on

    def setChannel(self, channel):
        self.channel = channel

    def getChannel(self):
        return self.channel

    def __str__(self):
        return "SeralNo:%d, 채널:%s , 볼륨:%s, 전원:%s " % \
    (Television.seralNo, self.channel, self.volume, self.on)

```

```python
t = Television(9, 10, True)
print(t)

print("--------------------------------")
t = Television(9, 10, True)
print(t)

```

```
SeralNo:1, 채널:9 , 볼륨:10, 전원:True
--------------------------------
SeralNo:2, 채널:9 , 볼륨:10, 전원:True
```

클래스 작성하기

```
1  class PythonClass:
2      def reset(self):
3          self.instanceVar = ''
4
5      def setVar(self):
6          self.instanceVar = '인스턴스 변수'
7
8      def getVar(self):
9          return self.instanceVar
10
11 instance = PythonClass()
12 instance.reset()
13
14 instance.setVar()
15 print("변수 값 : ", instance.getVar())
```

- 모든 메소드의 첫번째 매개변수
  - self
  - 현재 객체임을 의미
  - 이 self 를 통해서만 멤버 변수에 접근 가능

- 인스턴스 변수
  - 메소드 내에 self. 문법으로 선언

생성자

- **생성자(constructor)**는 객체가 생성될 때 객체를 기본값으로 초기화하는 특수한 메소드
- Python에서는 생성자의 이름으로 __init__() 사용

생성자로 객체 초기화 담당

```
class Counter:

    def __init__(self) :
        self.count = 0

    def reset(self) :
        self.count = 0

    def increment(self):
        self.count += 1

    def get(self):
        return self.count
```

생성자 및 메소드

- 객체가 생성될 때 호출되며, 객체를 기본 값으로 초기화 하는 특수 메소드
- 클래스당 하나의 생성자만 허용

```
1  class PythonClass:
2      def __init__(self):
3          self.instanceVar = ''
4          print('난 생성자')
5
6      def reset(self):
7          self.instanceVar = ''
8
9      def setVar(self):
10         self.instanceVar = '인스턴스 변수'
11
12     def getVar(self):
13         return self.instanceVar
14
15 instance = PythonClass()
16
17 instance.setVar()
18 print("변수 값 : ", instance.getVar())
```

생성자

객체의 초기화 담당

생성자

- 생성자 매개 변수 초기화

```
1  class PythonClass:
2      def __init__(self, intValue=10):
3          self.instanceVar = ''
4          self.intValue = intValue;
5          print('난 생성자')
6
7      def setVar(self):
8          self.instanceVar = '인스턴스 변수'
9
10     def getVar(self):
11         return self.instanceVar
12
13     def getIntValue(self):
14         return self.intValue;
15
16 instance = PythonClass()
17
18 instance.setVar()
19
20 print("instanceVar 변수 값 : ", instance.getVar())
21 print("intValue 변수 값 : ", instance.getIntValue())
```

생성자

자동 초기화

```
난 생성자
instanceVar 변수 값 :  인스턴스 변수
intValue 변수 값 :  10
```

클래스의 특수 메소드

- __str__ & __len__ 메소드

```
1  class Book:
2      def __init__(self, title, author, pages):
3          self.title = title
4          self.author = author
5          self.pages = pages
6
7      # 객체의 참조 변수 출력시 자동 호출되는 함수
8      def __str__(self):
9          return "제목:%s , 저자:%s, 페이지:%s " % ₩
10                 (self.title, self.author, self.pages)
11
12     def __len__(self):
13         return self.pages
14
15 book = Book("Data 구조 이해하기", "나작가", 375)
16 print(book)
17 print(len(book))
18
```

```
제목:Data 구조 이해하기 , 저자:나작가, 페이지:375
375
```

클래스의 특수 메소드

- 연산자 관련 특수 메소드 재정의
- 객체에 대하여 +, -, #, / 와 같은 연산을 적용하면 자동으로 호출

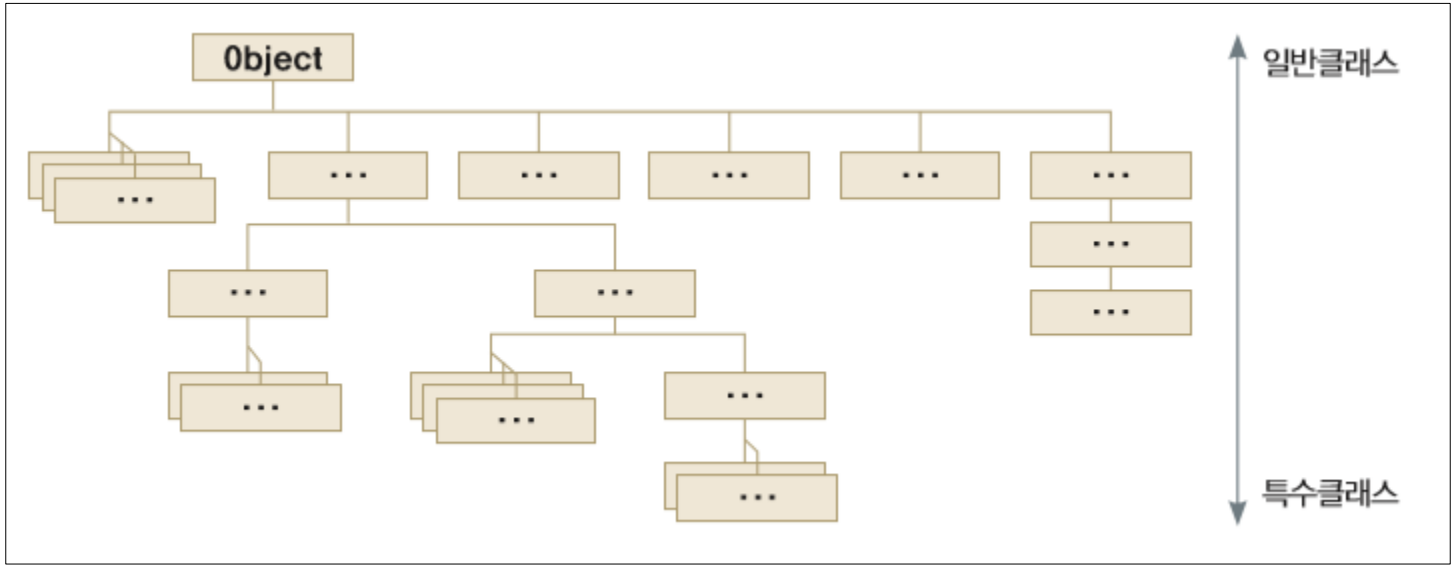
```
# 특별한 메소드 재정의, 객체의 연산시 자동 호출
class SpecialMethod :
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return SpecialMethod( self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return SpecialMethod(self.x - other.x, self.y -

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%d, %d)' % (self.x, self.y)
```

```
s1 = SpecialMethod(0,1)
s2 = SpecialMethod(2,3)

no1 = s1 + s2
print(no1)
print("--------------------------------")

no2 = s2 - s1
print(no2)
```

```
(2, 4)
--------------------------------
(2, 2)
```

상속과 다형성

- 필요성
  - 재사용성
- Object 클래스 상속
  - __str__(문자열 표현 반환) , __cmp__(객체 비교), __repr__(객체 표현 문자열 반환)등의 메소드 보유



상속과 다형성

- 문법

```
class 클래스명(부모 클래스명):
        생성자
        메소드
```

```
class ChildClass(ParentClass) :
        def __init__(self):
                super().__init__()
                ...
```

모듈 & 패키지

- Python은 풍부한 표준 library 제공
- 모듈
  - Python library
    - 함수나 클래스들을 포함
  - 확장자 : 파일명.py
  - Python에는 많은 모듈 포함되어 있음
    - 프로그래밍에서 중요한 하나의 원칙은 이전에 개발된 코드를 적극적으로 재활용 하자는 것
  - 사용 방법
    - import 문장으로 파일명 선언하기
    - ex : import file명[,..]

      from file명 import 파일보유한 함수명등
- 패키지
  - 파일 계층구조(폴더 구조)
  - 사용방법
    - from 폴더명 import 파일명[,..]

모듈

fibo.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

fibo 모듈 사용

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

모듈 내에서, 모듈의 이름은 전역 변수 __name__ 으로 제공

함수를 자주 사용할 경우 지역 이름으로 대입

```
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

패키지

```
sound/
      __init__.py
      formats/
            __init__.py
            wavread.py
            wavwrite.py
            aiffread.py
            aiffwrite.py
            auread.py
            auwrite.py
            ...
      effects/
            __init__.py
            echo.py
            surround.py
            reverse.py
            ...
      filters/
            __init__.py
            equalizer.py
            vocoder.py
            karaoke.py
            ...
```

- "점으로 구분된 모듈 이름" 를 써서 Python의 모듈 이름 공간을 구조화하는 방법
- 모듈 이름 A.B 는 A 라는 이름의 패키지에 있는 B 라는 이름의 서브 모듈
- Python이 디렉터리를 패키지로 취급하게 만들기 위해서 __init__.py 파일이 필요

패키지로부터 개별 모듈을 import

```
import sound.effects.echo
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

서브 모듈 import

```
from sound.effects import echo
echo.echofilter(input, output, delay=0.7, atten=4)
```

원하는 함수나 변수를 직접 import

```
from sound.effects.echo import echofilter
echofilter(input, output, delay=0.7, atten=4)
```

주요 모듈

모듈 이름	설명
re	정규 표현식
datetime	날짜와 시간
collections	여러가지 콜렉션 자료형
math	수학 관련 함수
random	무작위 처리
itertools	반복 가능한 객체에 대한 조작
sqlite3	SQLite 데이터베이스
csv	CSV 조작
json	JSON 조작
os	OS 관련 조작
os.path	파일 또는 디렉터리 등의 경로와 관련된 조작
multiprocessing	멀티 프로세스를 사용한 병렬 처리
subprocess	다른 프로세스 실행

주요 모듈

모듈 이름	설명
urllib	URL 관련 조작
unittest	단위 테스트
Sys	Python 인터프리터 관련 변수와 함수

유용한 모듈1 - copy 모듈

```
import copy
colors = ["red", "blue", "green"]
clone = copy.deepcopy(colors)

clone[0] = "white"
print(colors)
print(clone)
```

```
['red', 'blue', 'green']
['white', 'blue', 'green']
```

유용한 모듈2 – random 모듈

- 난수 발생 모듈

- randint()
  - 정수 범위의 난수 생성
- random()
  - 0~1 사이의 난수
- choice()
  - 주워진 시퀀스 항목을 랜덤

```
>>> import random
>>> print(random.randint(1, 6))
6
>>> print(random.randint(1, 6))
3

>>> import random
>>> print(random.random()*100)
81.16185158804 31

>>> myList = [ "red", "green", "blue" ]
>>> random.choice(myList)
'blue'
```

유용한 모듈2 – random 모듈

- shuffle()
  - 리스트의 항목을 랜덤하게 혼합
- random.randrange(start, stop, step)
  - 구간으로 부터 랜덤하게 요소를 생성

```
>>> myList = [ [x] for x in range(10) ]
>>> random.shuffle(myList)
>>> myList
[[3], [2], [7], [9], [8], [1], [4], [6], [0], [5]]

>>> for i in range(3):
        print(random.randrange(0, 101, 3))
81
21
57
```

유용한 모듈3 – os 모듈

- Python이 실행되는 운영 체제에 관계에 없이 운영 체제의 기본적인 기능을 다룰수 있는 모듈

```
>>> import os
>>> os.system("calc")


>>> os.listdir(".")
현 디렉토리 구조가 list로 출력됨
```