

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, Psycho, 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 n, child in enumerate(children):  
            print '%s' % child,
```

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 재배포
 - ▶ Unicode 체계 변경

파이썬 활용분야

- ▶ 시스템 유틸리티
 - ▶ 운영체제의 시스템 명령어들을 이용할 수 있는 각종 도구를 갖추
- ▶ 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, Seaborn 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
- ▶ 머신러닝/딥러닝
 - ▶ Tensorflow, SKLearn, PyTorch 등을 이용한 머신러닝/딥러닝

주요 프로젝트

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



Polyglot



TIOBE Index: 2017 November

Feb 2019	Feb 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.876%	+0.89%
2	2		C	12.424%	+0.57%
3	4	▲	Python	7.574%	+2.41%
4	3	▼	C++	7.444%	+1.72%
5	6	▲	Visual Basic .NET	7.095%	+3.02%
6	8	▲	JavaScript	2.848%	-0.32%
7	5	▼	C#	2.846%	-1.61%
8	7	▼	PHP	2.271%	-1.15%
9	11	▲	SQL	1.900%	-0.46%
10	20	▲▲	Objective-C	1.447%	+0.32%
11	15	▲▲	Assembly language	1.377%	-0.46%
12	19	▲▲	MATLAB	1.196%	-0.03%
13	17	▲▲	Perl	1.102%	-0.66%
14	9	▼▼	Delphi/Object Pascal	1.066%	-1.52%
15	13	▼	R	1.043%	-1.04%
16	10	▼▼	Ruby	1.037%	-1.50%
17	12	▼▼	Visual Basic	0.991%	-1.19%
18	18		Go	0.960%	-0.46%
19	49	▲▲	Groovy	0.936%	+0.75%
20	16	▼▼	Swift	0.918%	-0.88%

”Life is too short, You need Python”

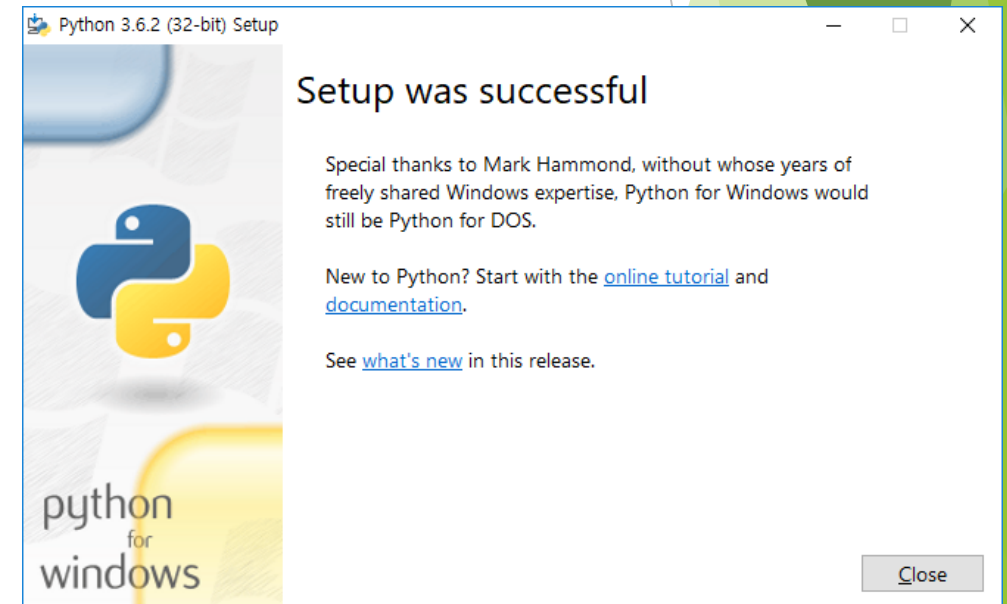
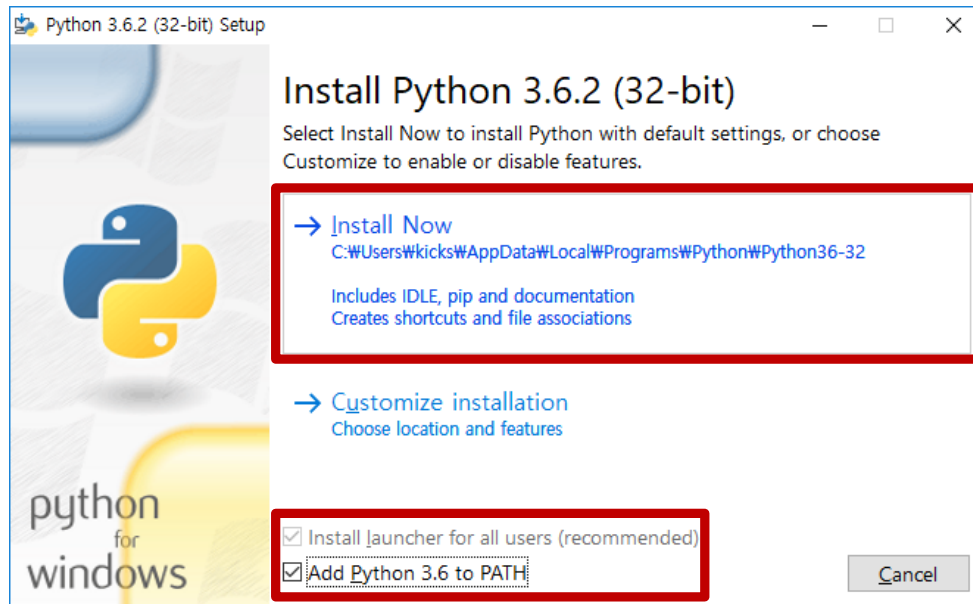
Python 시작하기

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

Read
Eval
Print
Loop

▶ Python (Command Line Interface)

- ▶ cmd.exe 를 실행하고 python을 실행
- ▶ ^D(Ctrl+D: Unix 계열), ^Z(Ctrl+Z: 윈도우 계열) 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

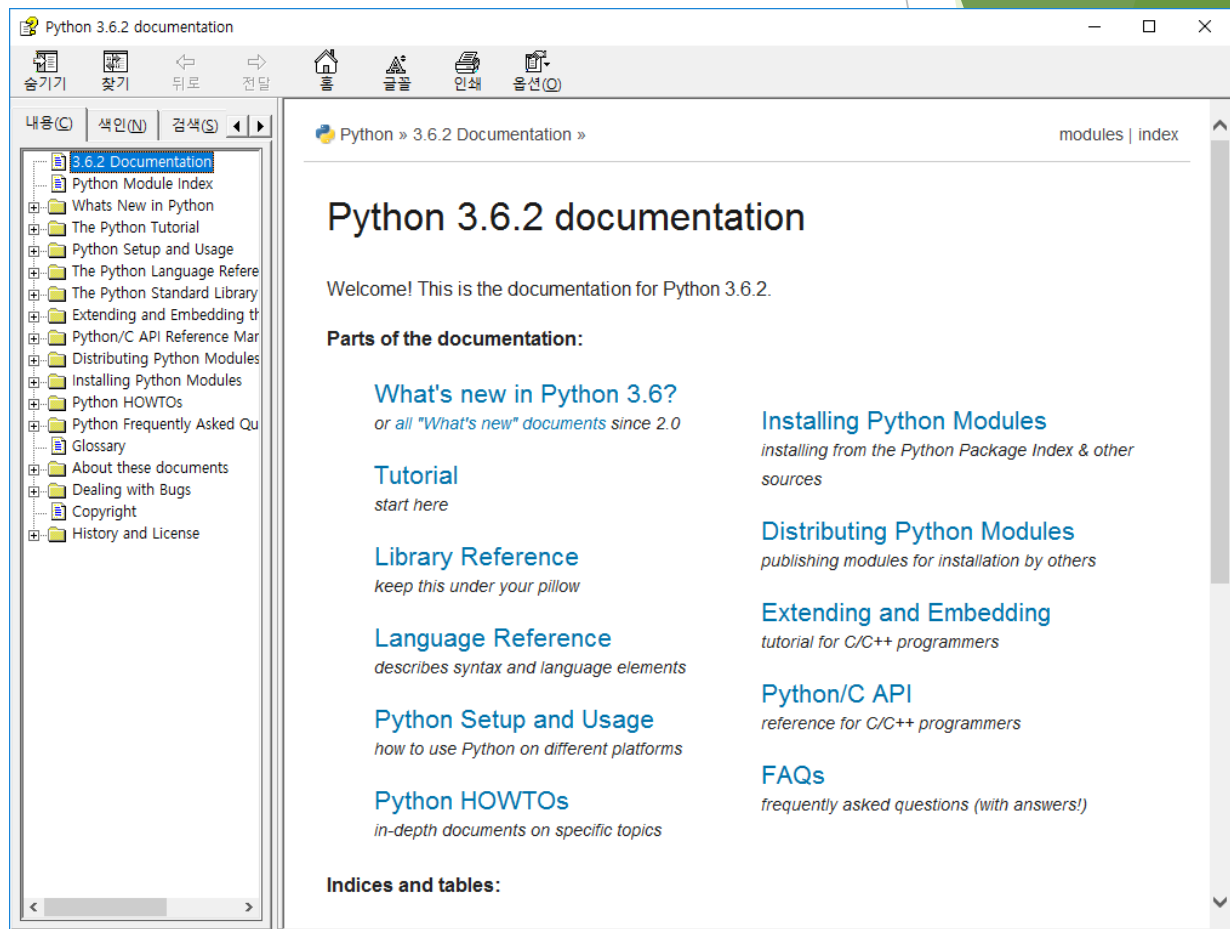
- ▶ 파이썬 모듈 도움말을 **WEB PAGE** 형태로 보여줌
- ▶ 로컬 컴퓨터에서 내장 서버 형식으로 실행
 - ▶ **b** : 모듈 문서를 브라우저에서 보가
 - ▶ **q** : 도움말 서버 종료



파이썬 설치 요소

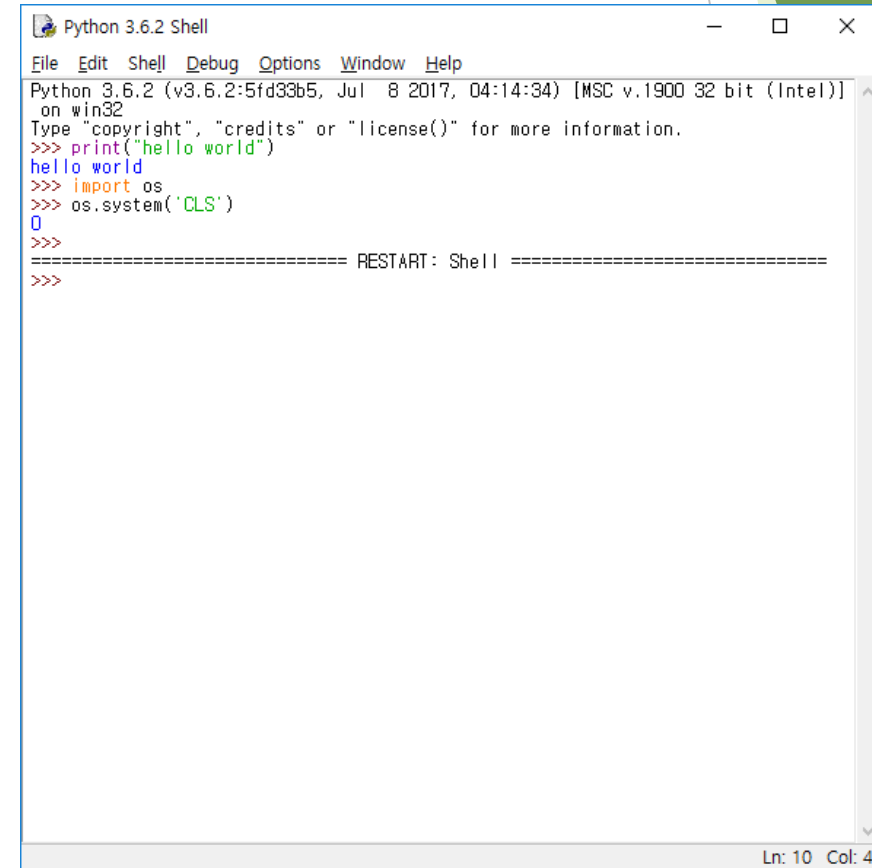
▶ Python Manuals

- ▶ 파이썬 문서를 윈도 도움말 형태로 제공



파이썬 설치 요소

- ▶ IDLE (Python GUI)
 - ▶ 간단한 파이썬용 내장 IDE
 - ▶ 간단한 수준의 **Code Assistant** 기능을 수행
 - ▶ 콘솔에서 직접 입력 이외에도 파일 작성 및 저장 기능을 제공



```
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 오버라이딩 되어 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 Line**에서 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

The diagram illustrates Python code indentation with the following code and annotations:

```
if today == 'Saturday':  
    print('Party!')  
elif today == 'Sunday':  
    if condition == 'Headache':  
        print('Recover, then rest.')  
    else:  
        print('Rest.')  
else:  
    print('Work, work, work.')
```

Annotations:

- These four lines of code are a suite:** Points to the four lines of code under the `elif today == 'Sunday':` condition.
- This single line of code is a suite:** Points to the `print('Party!')` line under the `if today == 'Saturday':` condition.
- These single lines of code are both suites:** Points to the `print('Recover, then rest.')` and `print('Rest.')` lines under the `if condition == 'Headache':` and `else:` conditions.
- This single line of code is a suite:** Points to the `print('Work, work, work.')` line under the `else:` condition.
- Indentation level zero:** Points to the first line of the code (`if today == 'Saturday':`).
- Indentation level one:** Points to the first line of the `elif` block (`elif today == 'Sunday':`).
- Indentation level two:** Points to the first line of the `if` block inside the `elif` block (`if condition == 'Headache':`).

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

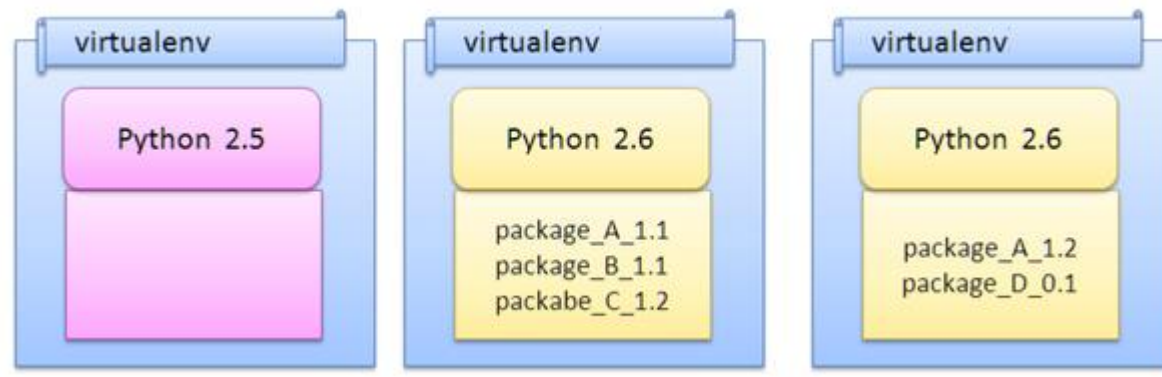
```
>>> # 이것은 주석입니다
```

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

- ▶ 파이썬은 공식적으로 여러 줄 주석을 허용하지 않는다
- ▶ 여러 줄 주석을 사용하려면 파이썬 여러 줄 문자 리터럴("""")을 활용할 수 있으며 실제로 소스코드 문서화 작업 등에 널리 활용

파이썬 가상환경

- ▶ 시스템에 설치한 파이썬은 한 라이브러리에 대해 하나의 버전만 설치 가능
- ▶ 기존 프로젝트를 유지보수하거나 여러 개의 프로젝트를 하나의 시스템에서 개발하고자 한다면 문제가 된다
 - ▶ 작업 프로젝트를 변경 시 해당 프로젝트에 적합한 버전의 라이브러리를 재설치해야 함
 - ▶ 각 프로젝트에 필요한 라이브러리 버전이 맞지 않을 시 오류 발생 가능성 있음
- ▶ 이러한 상황을 방지하기 위, 각 프로젝트를 독립된 가상환경으로 격리



파이썬 가상환경의 설정

: pyenv - Version Manager

- ▶ 시스템에 여러 버전의 Python을 설치할 수 있는 Version Manager
- ▶ 전역 혹은 지역적으로 다른 파이썬의 버전과 버전별 모듈을 분리하여 설치 가능
- ▶ Linux, Unix에서만 지원, Windows OS 환경에서는 지원하지 않음
- ▶ <https://github.com/pyenv/pyenv>

파이썬 가상환경의 설정

: 가상환경 설정 도구 - venv, virtualenv, anaconda

- ▶ PIP : Python Default Package Manager
- ▶ venv : Python 3.3 이상에 기본 모듈로 탑재된 가상 환경 모듈
- ▶ virtualenv : Python 2 버전부터 사용해 오던 가상 환경 라이브러리
- ▶ conda : Anaconda Python에서 지원하는 가상환경 모듈
 - ▶ 단순히 파이썬 가상 환경 모듈 혹은 라이브러리가 아님
 - ▶ PIP 처럼 패키지 매니저로서의 역할
 - ▶ Python 이외의 다양한 개발 환경을 지원함

파이썬 가상환경의 설정

: venv를 이용한 가상환경 설정 연습

▶ 가상 환경의 설정

```
cd {path/to/rootdir}  
python -m venv {name_of_venv}
```

▶ 가상 환경 수행

```
cd {path/to/rootdir}  
{name_of_venv}\Scripts\activate.bat
```

▶ 가상 환경 종료

```
cd {path/to/rootdir}  
{name_of_venv}\Scripts\deactivate.bat
```


파이썬 IDE 설치 및 사용

: PyCharm Community, Visual Studio Code

- ▶ 표준 개발 환경 **IDLE**을 사용해서도 파이썬 프로그래밍을 할 수 있으나 보다 효율적이고 편리한 기능들을 포함한 통합 개발 도구(**IDE**)를 활용할 것을 추천
- ▶ Visual Studio Code
 - ▶ Microsoft에서 만든 코드 편집기 겸 통합 개발 도구
 - ▶ 가볍고 단순하면서도 다양한 기능(디버깅, 버전 관리)과 플러그인을 이용한 확장이 용이
- ▶ PyCharm Community
 - ▶ JetBrains에서 만든 파이썬 개발 **IDE**
 - ▶ **virtualenv**, **conda**를 이용한 가상환경 구성을 자동 지원하며 다양한 개발 편의 기능을 지원
 - ▶ **Professional** 버전에 비해 기능 제약이 있고, 라이선스 제한이 있어 상용 소프트웨어는 개발할 수 없음

Python 프로그래밍 기초

산술연산자

산술 연산자

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
3
>>> 4 * 3 # 곱셈
12
>>> 4 / 3 # 나눗셈
1.3333333333333333
>>> 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() # 켤레복소수 반환
(3+4j)
```

Python 프로그래밍 기초

비교연산자

비교연산자

비교연산자	사용예	설명
>	$x > y$	x는 y 보다 크다
>=	$x \geq y$	x는 y 보다 크거나 같다
<	$x < y$	x는 y 보다 작다
<=	$x \leq y$	x는 y 보다 작거나 같다
==	$x == y$	x는 y 와 같다
!=	$x != y$	x는 y 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 bool은 조건 분기와 긴밀히 연결되어 있다

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


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']
```

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
- ▶ 매핑(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

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 비교 값의 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 long형이 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)
```

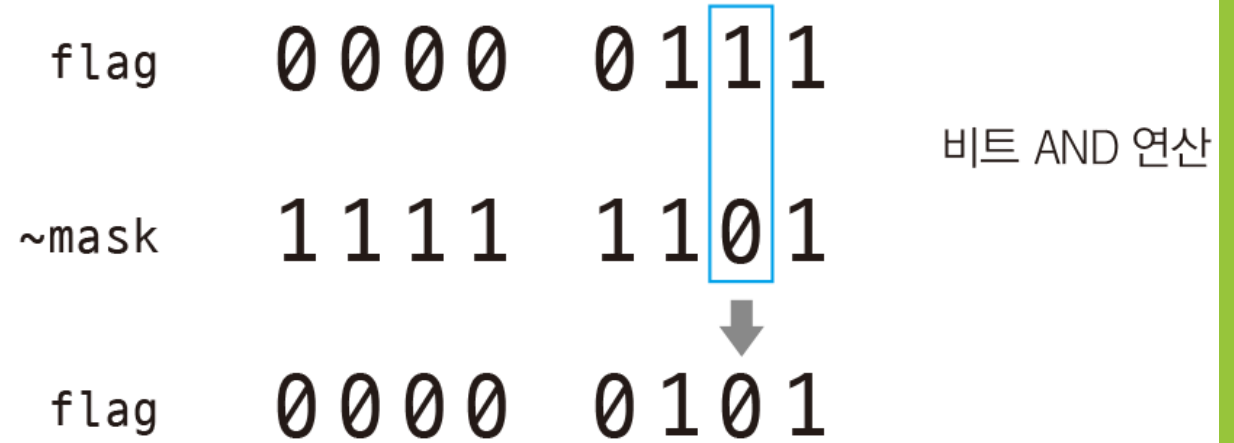
수치형

: 내장 수치 함수

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

비트 연산자

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



연산자	기능	문법	설명
<<	비트 왼쪽 시프트	$a \ll b$	a의 비트를 b번 왼쪽으로 이동
>>	비트 오른쪽 시프트	$a \gg b$	a의 비트를 b번 오른쪽으로 이동
&	비트 AND	$a \& b$	a와 b의 비트를 AND 연산
	비트 OR	$a b$	a와 b의 비트를 OR 연산
^	비트 XOR	$a \wedge b$	a와 b의 비트를 XOR(배타적 OR) 연산
~	비트 NOT	$\sim x$	x의 비트를 뒤집음 (0 <-> 1)

정수의 왼쪽 시프트는 * 2, 정수의 오른쪽 시프트는 // 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

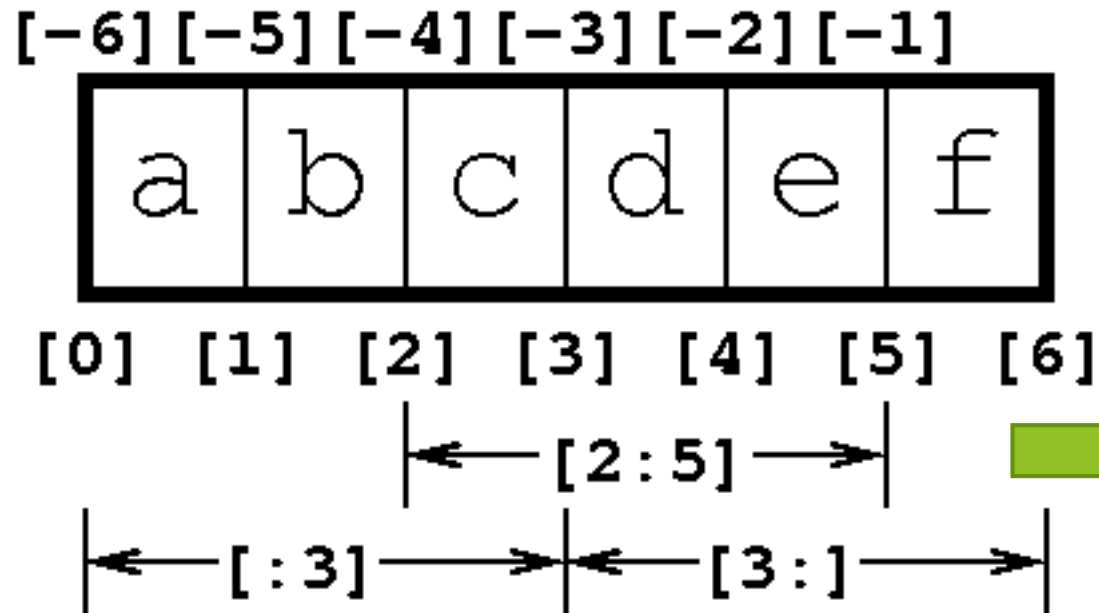
- ▶ 산술, ~~비교 연산자~~ 비트 연산자 등을 치환문과 함께 사용할 수 있다

$x \text{ op} = y \quad \# \Rightarrow x = x \text{ op } y$

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

시퀀스 모델

: 인덱싱(indexing)과 슬라이싱(slicing)



매우주의: 헛갈리기 쉽다

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

시퀀스 모델의 주요 연산

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

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 = """ABCDEFGF  
... abcdef  
... 가나다라마바사아  
... 1234567890"""  
>>> str3  
'ABCDEFGF\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] = "1"
...
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
<code>count()</code>	문자열 내 검색어 개수를 반환
<code>find()</code>	문자열 내 첫번째 검색된 위치의 인덱스를 반환
<code>index()</code>	문자열 내 검색된 위치의 인덱스를 반환
<code>rindex()</code>	문자열 내 오른쪽으로부터 검색된 위치의 인덱스를 반환
<code>startswith()</code>	문자열이 지정된 검색어로 시작하는지 여부 반환
<code>endswith()</code>	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
<code>strip()</code>	문자열 내 좌우 공백문자를 삭제 좌우 삭제할 문자열을 지정 가능
<code>lstrip()</code>	문자열 내 왼쪽의 공백문자를 제거
<code>rstrip()</code>	문자열 내 오른쪽 공백문자를 제거
<code>replace()</code>	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
<code>center()</code>	문자열을 가운데로 정렬
<code>ljust()</code>	문자열을 왼쪽으로 정렬
<code>rjust()</code>	문자열을 오른쪽으로 정렬
<code>zfill()</code>	자리수를 지정하고 빈 공간을 0 로 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
<code>split()</code>	문자열을 공백문자(혹은 지정된 문자)를 기준으로 분리
<code>rsplit()</code>	문자열을 공백문자(혹은 지정된 문자)를 기준으로 오른쪽부터 분리
<code>join()</code>	문자열을 지정된 기호로 합침
<code>splitlines()</code>	문자열을 개행문자를 기준으로 분리

문자열 메서드

: 분리, 결합 관련

```
s = 'spam and ham'
t = s.split();
print(t)
t = s.split(' and ');
print(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
<code>isdigit()</code>	문자열이 숫자로 구성되어 있는가 여부를 반환
<code>isalpha()</code>	문자열이 알파벳으로 구성되어 있는가 여부를 반환
<code>islower()</code>	문자열이 소문자로 구성되어 있는가 여부를 반환
<code>isupper()</code>	문자열이 대문자로 구성되어 있는가 여부를 반환
<code>isspace()</code>	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 지정할 수 있다
- ▶ **format_map** 메서드를 이용하면 이름 기반으로 **map**의 데이터 형식을 이용 포맷을 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
<code>append(x)</code>	리스트의 마지막에 <code>x</code> 를 추가
<code>insert(i, x)</code>	리스트 인덱스 <code>i</code> 위치에 <code>x</code> 를 추가
<code>reverse()</code>	리스트를 역순으로 뒤집음
<code>sort()</code>	리스트 요소를 순서대로 정렬
<code>remove(i)</code>	리스트 인덱스 <code>i</code> 에 있는 요소를 제거
<code>extend(l)</code>	리스트 마지막에 리스트 <code>l</code> 을 추가
<code>index(x)</code>	인덱스 내에 <code>x</code> 가 있으면 인덱스값을 반환. 없으면 에러
<code>count(x)</code>	리스트 내에 <code>x</code> 가 몇 개 있는지 그 개수를 반환

리스트

: 리스트의 메서드

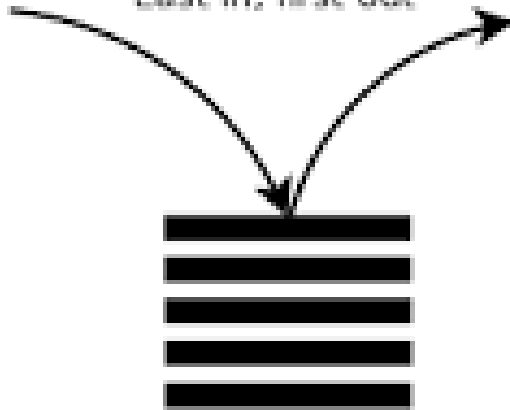
```
a = [1, 2, 3]
print(a)
a.append(5)
print(a)
a.insert(3, 4) # 인덱스 3에 요소 4를 추가
print(a)
print(a.count(2)) # 리스트 내 요소 2의 개수를 반환
a.reverse()
print(a)
a.sort()
print(a)
a.remove(3) # 내부에 있는 요소 3을 제거
print(a)
a.extend([6, 7, 8])
print(a)
```

리스트

: 리스트를 Stack과 Queue로 사용하기

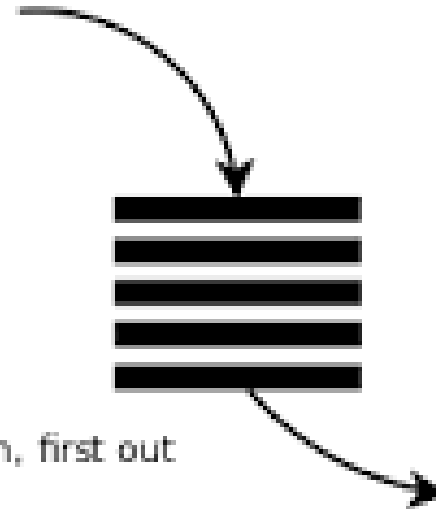
Stack:

Last in, first out



Queue:

First in, first out



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
<code>add(x)</code>	세트에 <code>x</code> 를 추가
<code>remove(x)</code>	세트에서 <code>x</code> 를 제거. <code>x</code> 가 세트에 없으면 오류 발생
<code>discard(x)</code>	세트에서 <code>x</code> 를 제거. <code>x</code> 가 세트에 없으면 무시
<code>update({set})</code>	세트에 여러 개의 값을 추가
<code>clear()</code>	세트를 비움

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

```
s.add(4)
```

```
s.add(1)
```

```
s.discard(2)
```

```
s.remove(3)
```

```
s.update({2, 3})
```

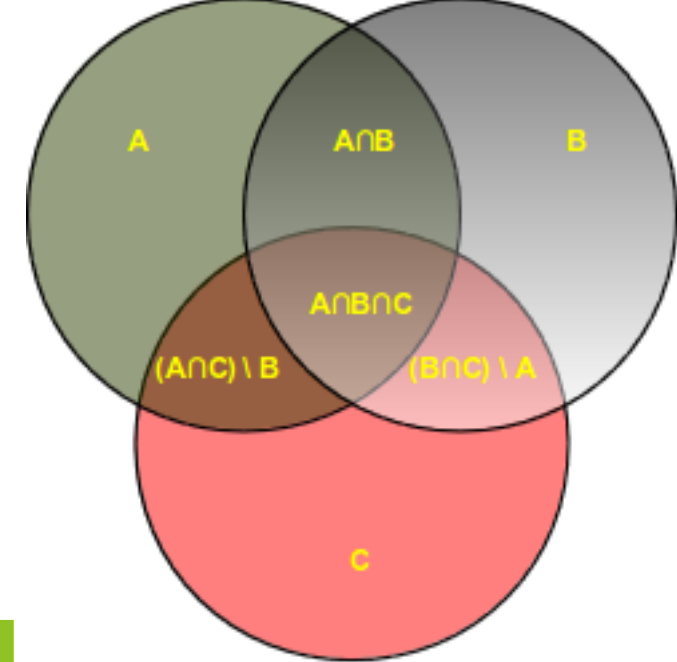
```
s.clear()
```

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 매핑형 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
<code>keys()</code>	사전내 키 목록을 dict_keys 객체로 반환
<code>values()</code>	사전내 값 목록을 dict_values 객체로 반환
<code>items()</code>	사전내 키-값 쌍을 튜플로 묶은 dict_items 객체로 반환
<code>get(key [, default])</code>	사전내 key 에 대응하는 값을 반환 default 를 지정하면 key 에 대응하는 값이 없을 때 default 를 반환
<code>del dic[key]</code>	dic 사전 내 key 에 대응하는 객체를 삭제
<code>clear()</code>	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 초과 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))
```