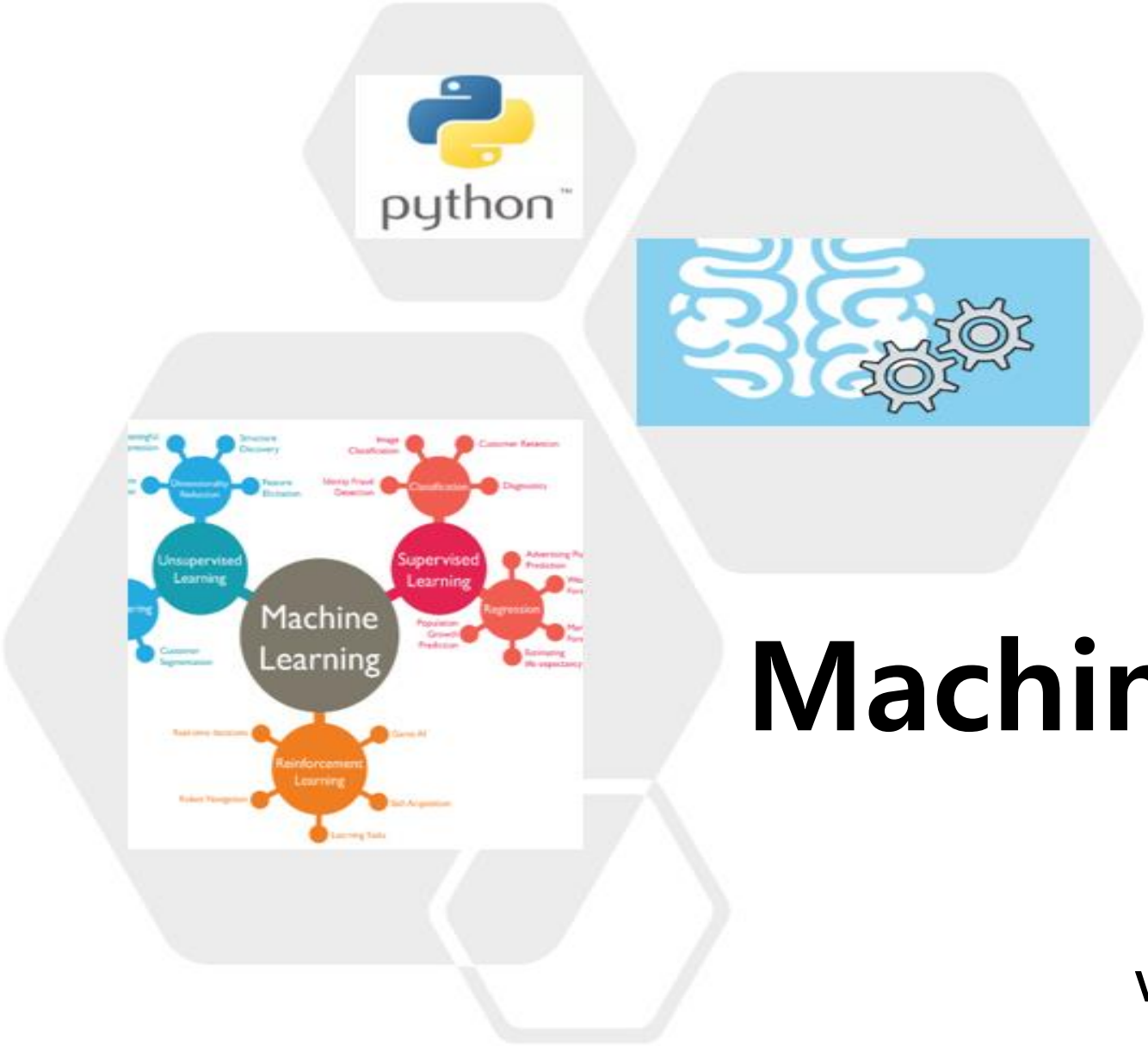


Machine Learning

김진숙
Maso Campus
www.masocampus.co.kr

Contents

I 파이썬 개요

1. 파이썬 개발 환경 설정
2. 파이썬 기초
3. 파이썬 자료구조 - 리스트, 튜플, 딕셔너리
4. 함수와 모듈

II 데이터 분석 및 시각화

1. 고성능 수치 계산 - Numpy
2. 데이터 핸들링 - Pandas
3. 데이터 시각화 - Matplotlib
4. 데이터 시각화 - Seaboarn

■ Contents

III 사이킷런으로 시작하는 머신러닝

1. 머신 러닝의 개념
2. 사이킷런 소개와 특징
3. 사이킷런 기반 프레임워크
4. Model Selection 모듈

IV 데이터 전처리

1. 데이터 인코딩
2. 피처 스케일링과 정규화
3. StandardScaler
4. MinMaxScaler
5. 실습 예제 - 공공데이타 건강진단 데이터 전처리

V 지도학습 : 회귀

1. 회귀(Regression) 소개
2. 단순 선형 회귀
3. 다중 선형 회귀
4. 회귀 모델 평가

캐글 보스턴 주택가격 예측 데이터 세트

Contents

파이썬 개요

파이썬 개발 환경 설정

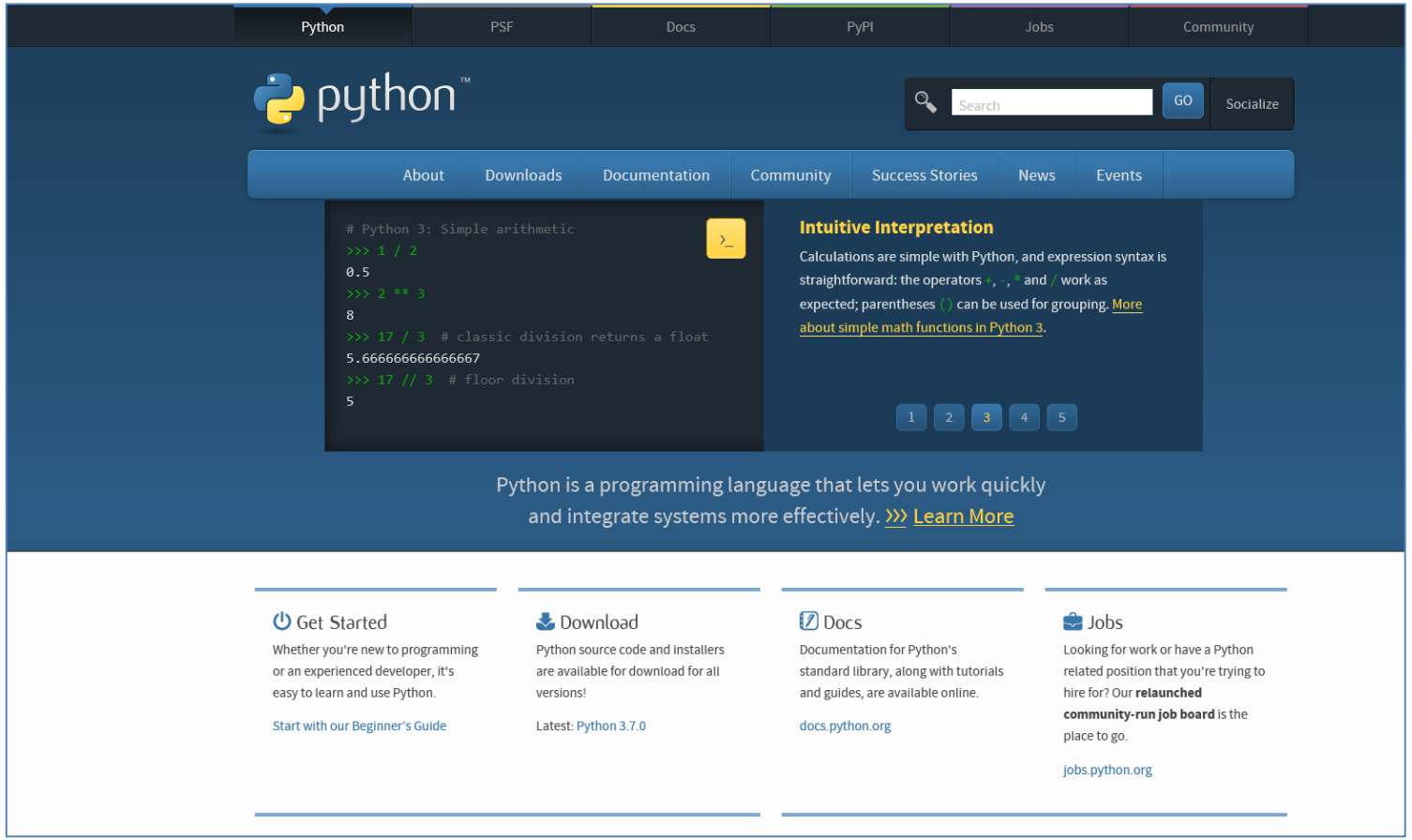
- **개발 환경이란?**
  - 프로그램을 작성하고 실행시키는 환경, '코딩 환경 ' 이라고 함

- **개발 환경 설정 시 고려할 사항 3가지**
  1) 운영체제(Operating System, OS) : Windows, Linux, Mac OS
  2) 파이썬 인터프리터(Python Interpreter)
     - Python 2.7, **Python 3.x** : 일반적인 파이썬으로 기본적인 모듈을 포함
     - Anaconda, Canopy : 다양한 과학 계산용 모듈들을 묶어서 패키지 형태로 제공
  3) 코드 편집기
     - 메모장, vi editor, sublime text, Atom, **Jupyter notebook**, Pycharm 등

- **파이썬 개발 환경**

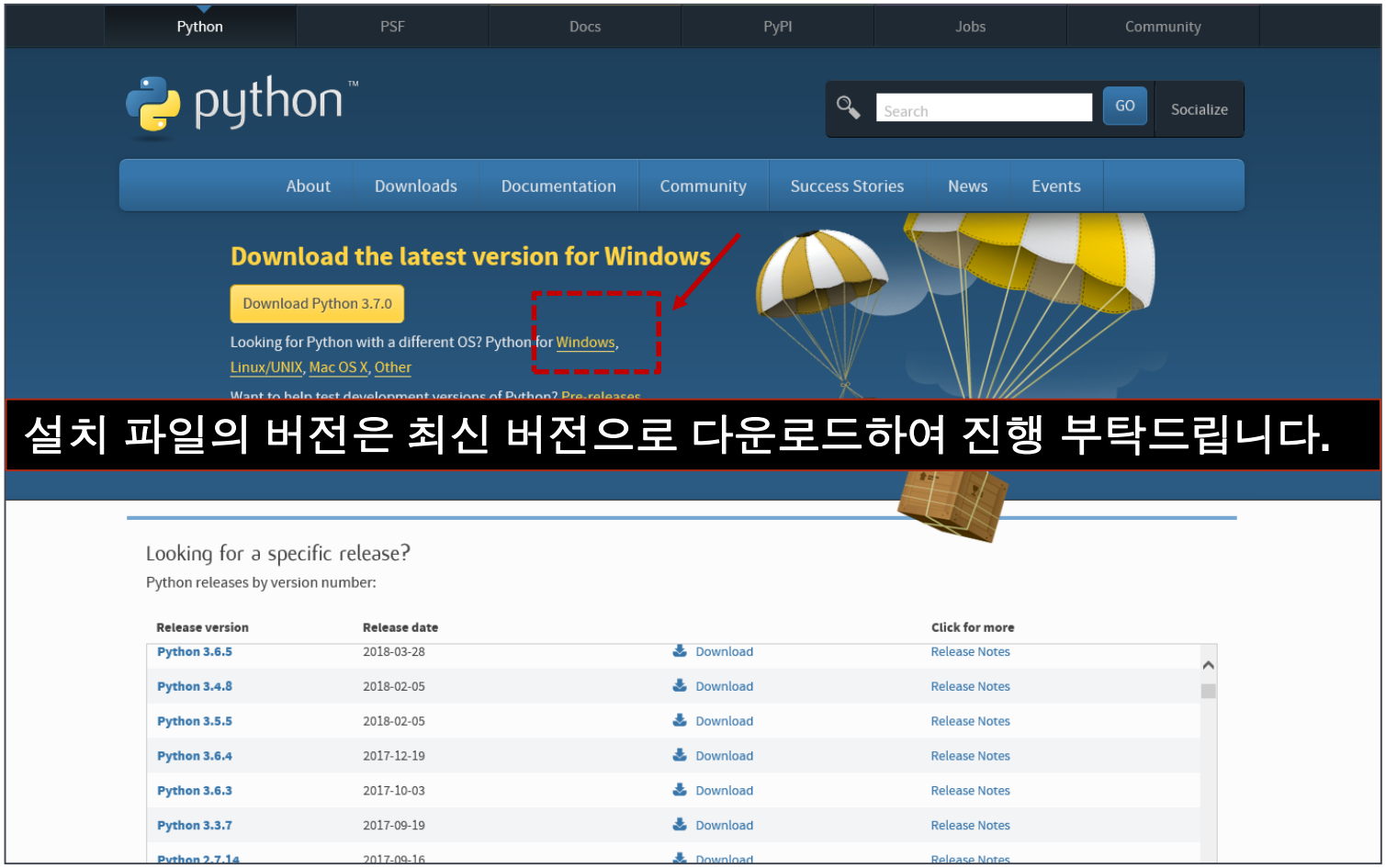
파이썬 프로그램 다운로드

- 파이썬 다운로드는 https://www.python.org/ 으로 들어가면 설치파일을 다운로드



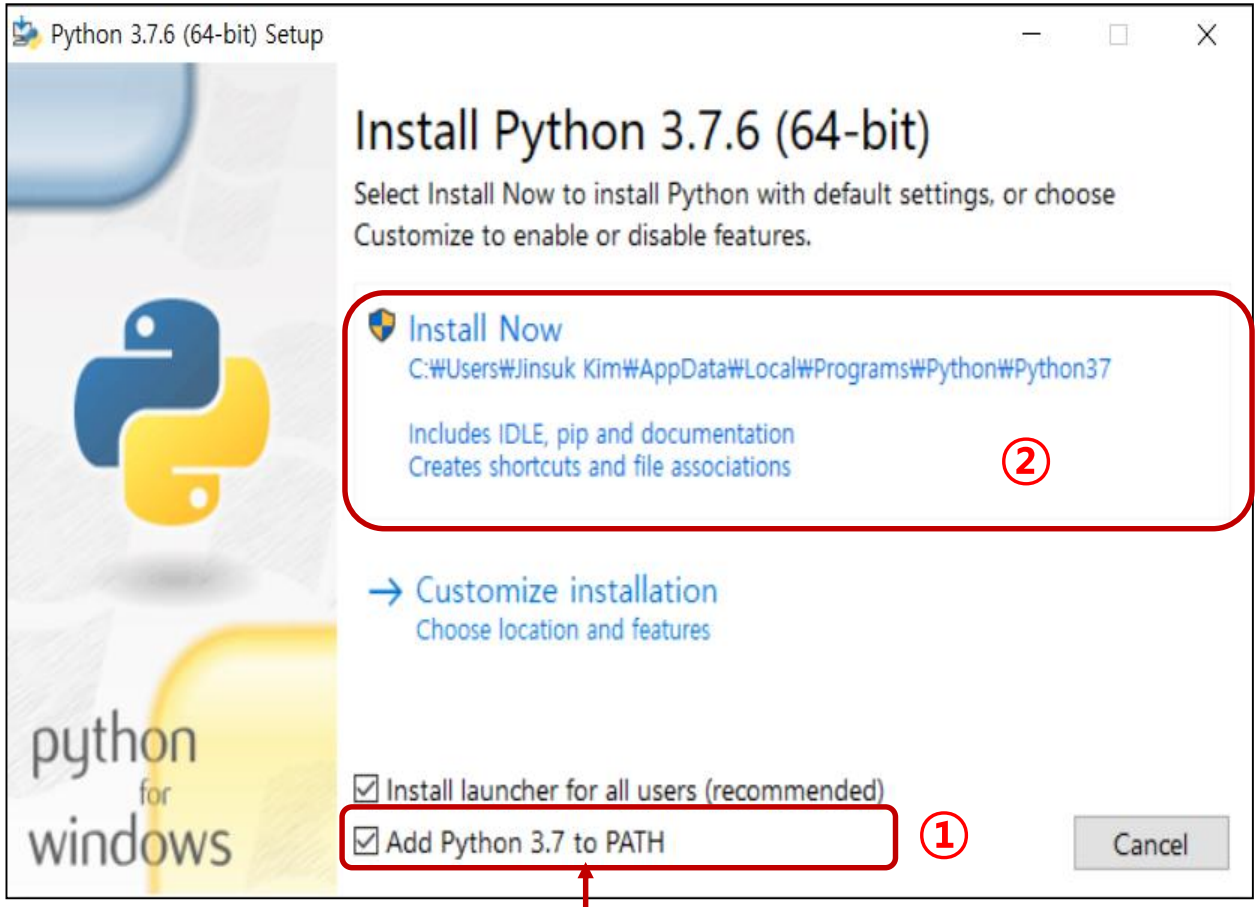
파이썬 프로그램 다운로드

- 상단의 메뉴 중 다운로드(Downloads)를 클릭한다.
- 컴퓨터 환경에 맞는 운영체제를 클릭한다.(Windows)

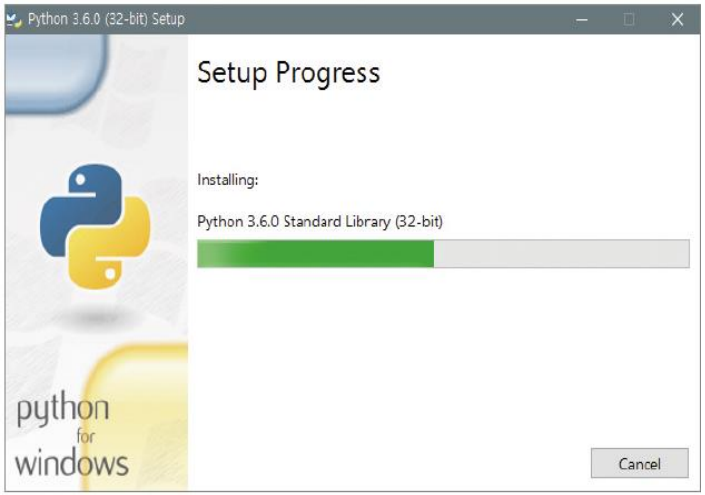


파이썬 프로그램 설치

- 파이썬 설치하기(관리자 권한으로 실행 권장)
- Add Python 3.7 to PATH 체크



환경변수 설정 추가하는 것으로 반드시 체크



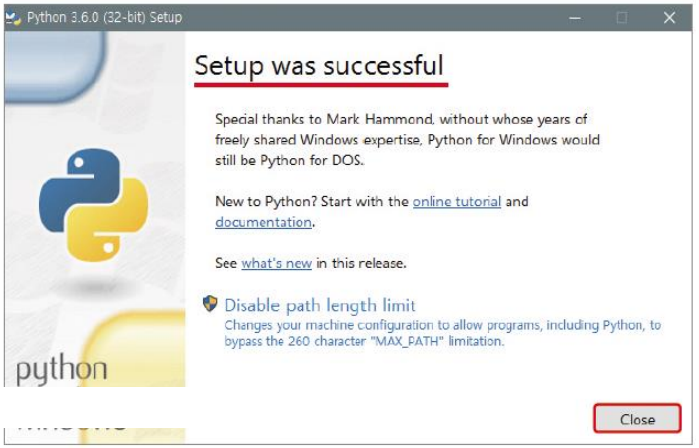


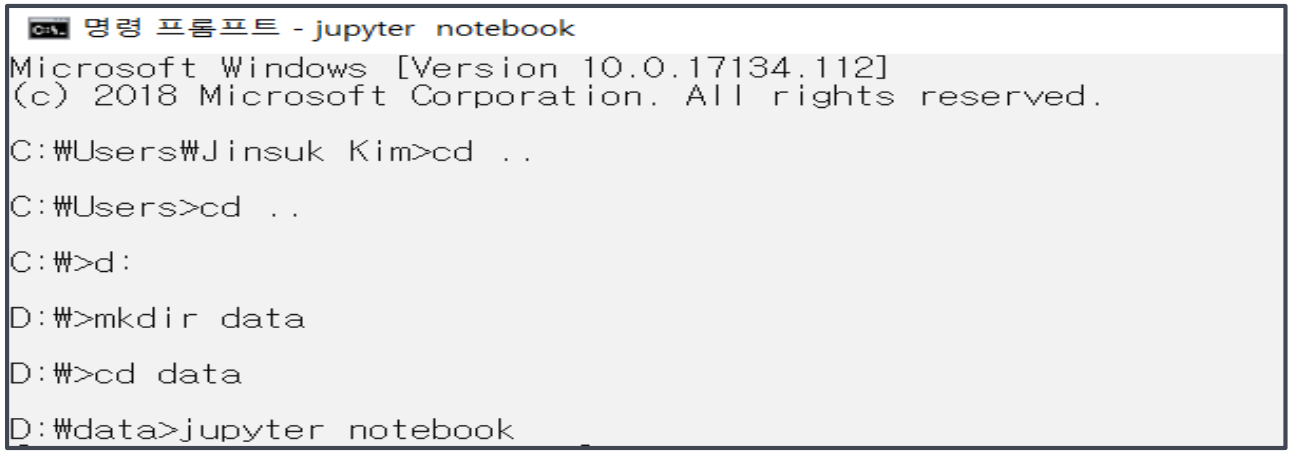
그림 1-8 파이썬 설치 화면

jupyter notebook 설치 및 실행

- **jupyter notebook 설치**
  - cmd 창(윈도우 버튼 + R) 실행 → pip 으로 jupyter notebook 설치
    > pip install jupyter notebook

- **jupyter notebook 실행(2가지 방법)**

(1) cmd 창(윈도우 버튼 + R)에서 jupyter notebook 입력
> jupyter notebook

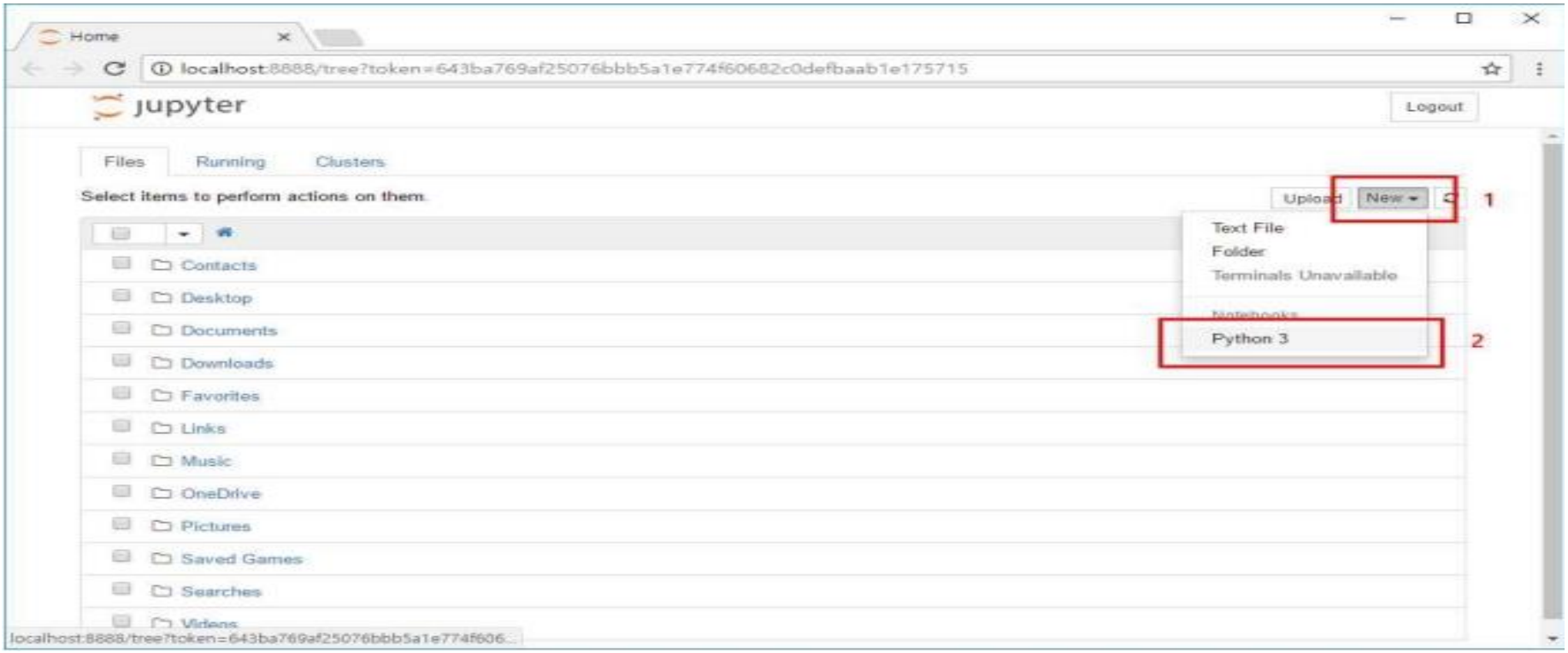


(2) cmd 창
> jupyter notebook --notebook-dir c:/python

디렉토리명

jupyter notebook 사용법

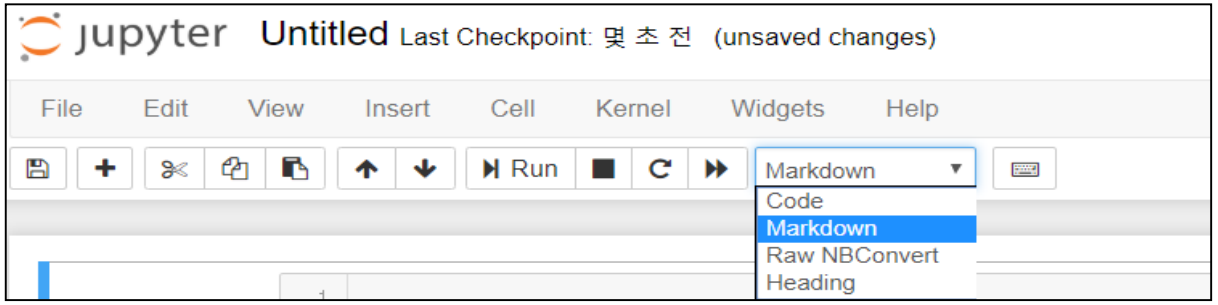
- 주피터 노트북 시작화면



- 파이썬 파일 생성하기
  - New > Python 3

jupyter notebook 사용법

- Markdown 사용하기
  - 메뉴 > Markdown 선택



1. 제목 만들기

제목 1

제목 2

제목 3

제목 4

제목 5

제목 6

2. 목록 만들기

- 리스트1
- 리스트2
- 리스트3

Run
Ctrl+Enter

```
## 1. 제목 만들기

# 제목 1
## 제목 2
### 제목 3
#### 제목 4
##### 제목 5
###### 제목 6
```

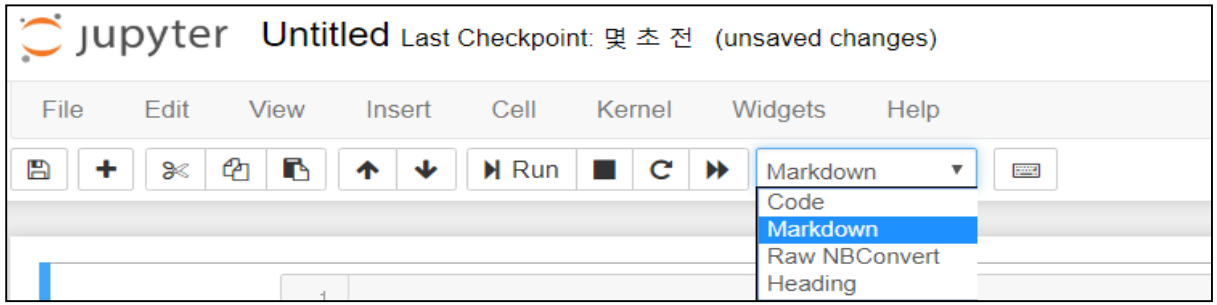
```
## 2. 목록 만들기

- 리스트1
- 리스트2
- 리스트3
```

파이썬 개발 환경 설정

jupyter notebook 사용법

- Markdown 사용하기
  - 메뉴 > Markdown 선택



3. 이미지 가져오기

4. 표 만들기

구분	의미	비고
주피터	주피터 사용법	파이참
Numpy	Python의 수학 계산기	
Pandas	Python의 엑셀	
Matplotlib	Python 챠트	

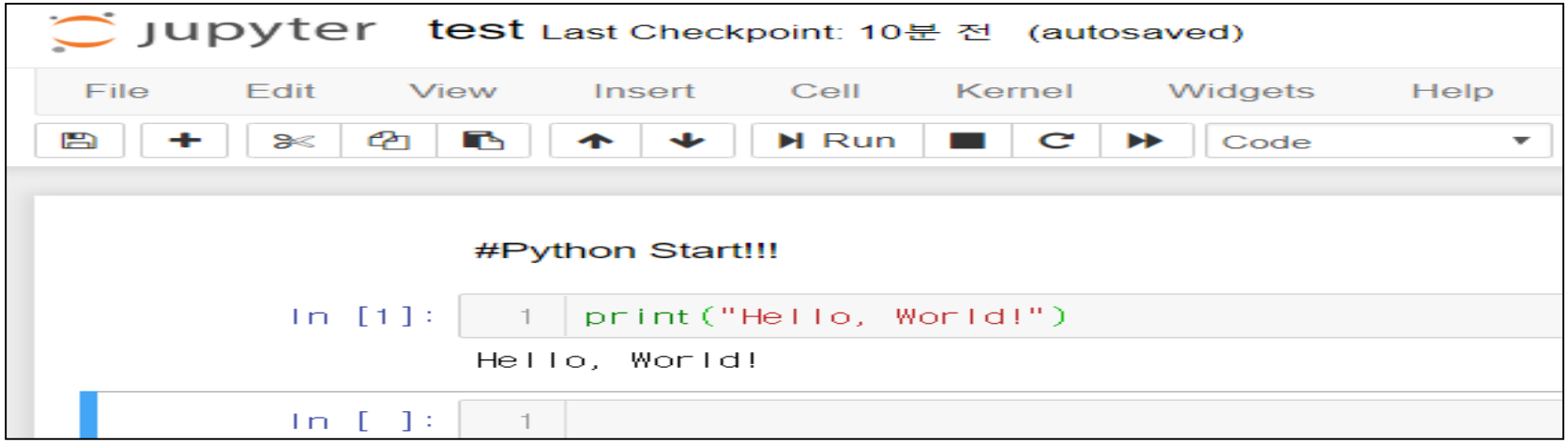
Run
Ctrl+Enter

```
## 3. 이미지 가져오기
<img src="http://story.wisedog.net/wp-content/uploads/2015/02/python_language_logo_300x300.png">
```

```
## 4. 표 만들기
| 구분 | 의미 | 비고 |
|---|---|---|
| `주피터` | 주피터 사용법 | `파이참` |
| `Numpy` | Python의 수학 계산기 | |
| `Pandas` | Python의 엑셀 | |
| `Matplotlib` | Python 챠트 | |
```

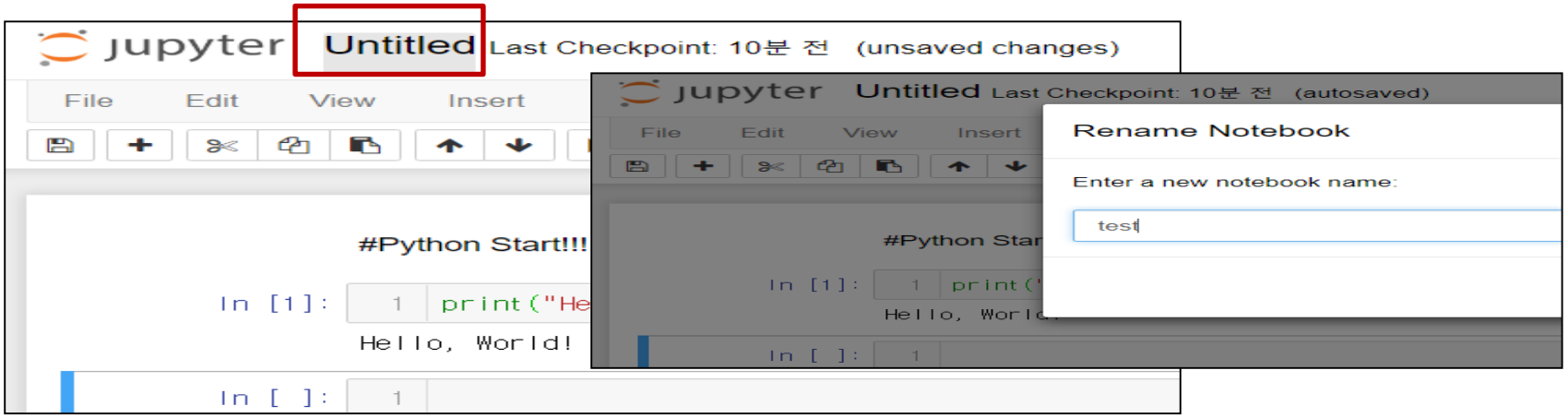
jupyter notebook 사용법

- 코드 입력하고 실행하기
  ① print(“Hello, World!”) 입력
  ② Run 버튼 클릭
  ③ 실행결과 확인



jupyter notebook 사용법

- 파일명 변경하기
  - 화면 상단의 Untitled 클릭 > 파일명 입력



- 파일 저장하기
  - File 메뉴 > Save as > 파일명 입력 (*.ipynb)
  - File 메뉴 > Download as > Notebook (*.ipynb)> 파일명 입력
  - File 메뉴 > Download as > HTML (*.html) > 파일명 입력
  - File 메뉴 > Download as > Python (*.py) > 파일명 입력

jupyter notebook 사용법

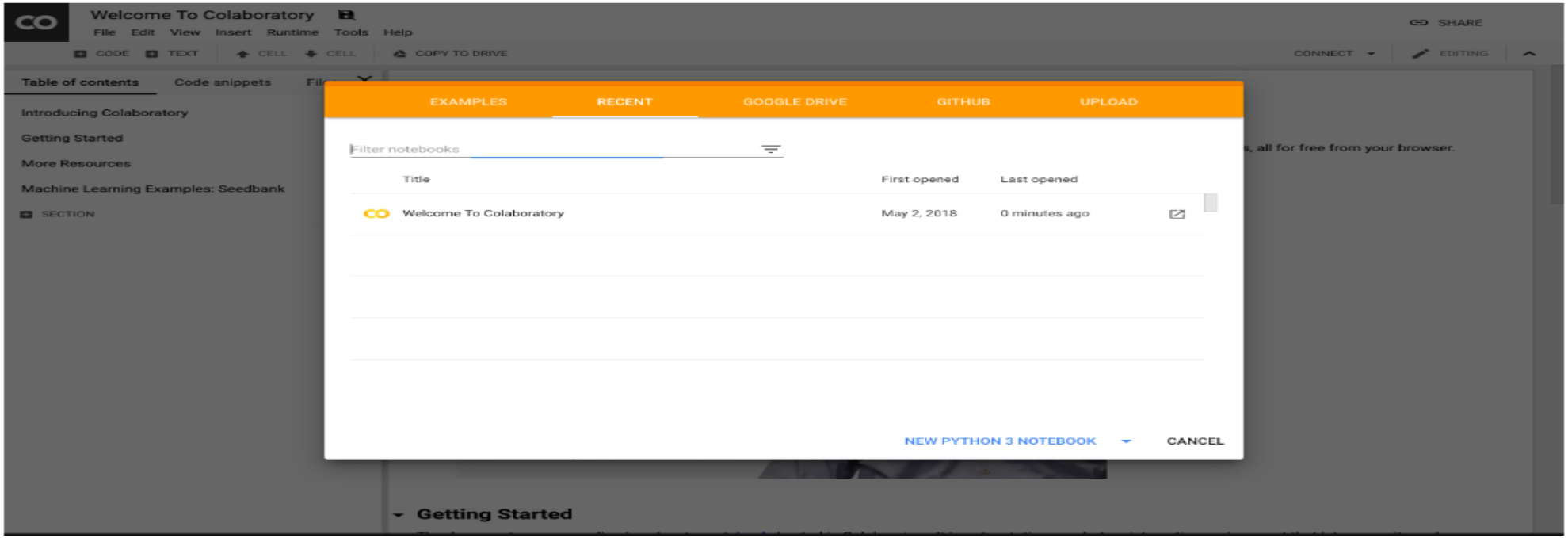
- **jupyter notebook cell 안의 코드 실행 방법(3가지)**
  1) 상단의 run cell 버튼 클릭
  2) 명령어 입력 후 Ctrl+Enter
  3) 명령어 입력 후 Shift+Enter 또는 Alter+Enter(명령 실행 후 아래쪽에 새로운 cell 생성)

- **주피터 단축키**

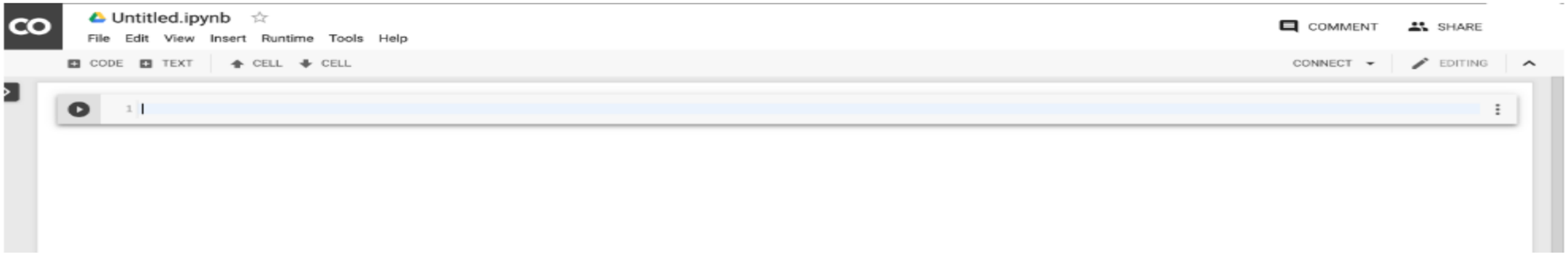
Command Mode	**Enter**:에디트모드(EditMode)로진입	**Y**:코드(Code)셀로전환
	R:RawNB셀로전환	**M**:마크다운(Markdown)셀로전환
	1~6:마크다운H1~H6	**a**:위에셀삽입
	b:아래셀삽입	

Edit Mode	**Tab**:코드자동완성,들여쓰기	**Ctrl+]**:들여쓰기(intent)
	Ctrl+[:내어쓰기(detent)	**Ctrl+a**:전체선택
	Ctrl+z:실행취소(undo)	**Esc**:커맨드모드(Commandmode)로진입
	Shift+Enter:셀 실행하고 다음셀선택	**Ctrl+Enter**:셀 실행
	Alt+Enter:셀 실행하고 아래 셀 삽입	

구글 Colaboratory notebook 시작 : http://colab.research.google.com



- 메뉴 - '파일' > '새 Python 3 노트' 명령을 선택하거나
- 다이얼로그 아래의 "새 PYTHON 3 노트" 버튼을 누르면 새 노트북 시작



파이썬 기초

- **파이썬의 역사**
  - 배우기도 쉽고 결과도 바로 확인할 수 있어 초보자에게 적합한 프로그래밍 언어
  - 귀도 반 로섬(1956년~)이라는 프로그래머가 C 언어로 제작해 1991년에 공식으로 발표
  - 사전적인 의미는 비단뱀으로 로고도 파란색과 노란색 비단뱀 두 마리가 서로 얽혀 있는 형태

파이썬 로고(출처 : https://www.python.org)

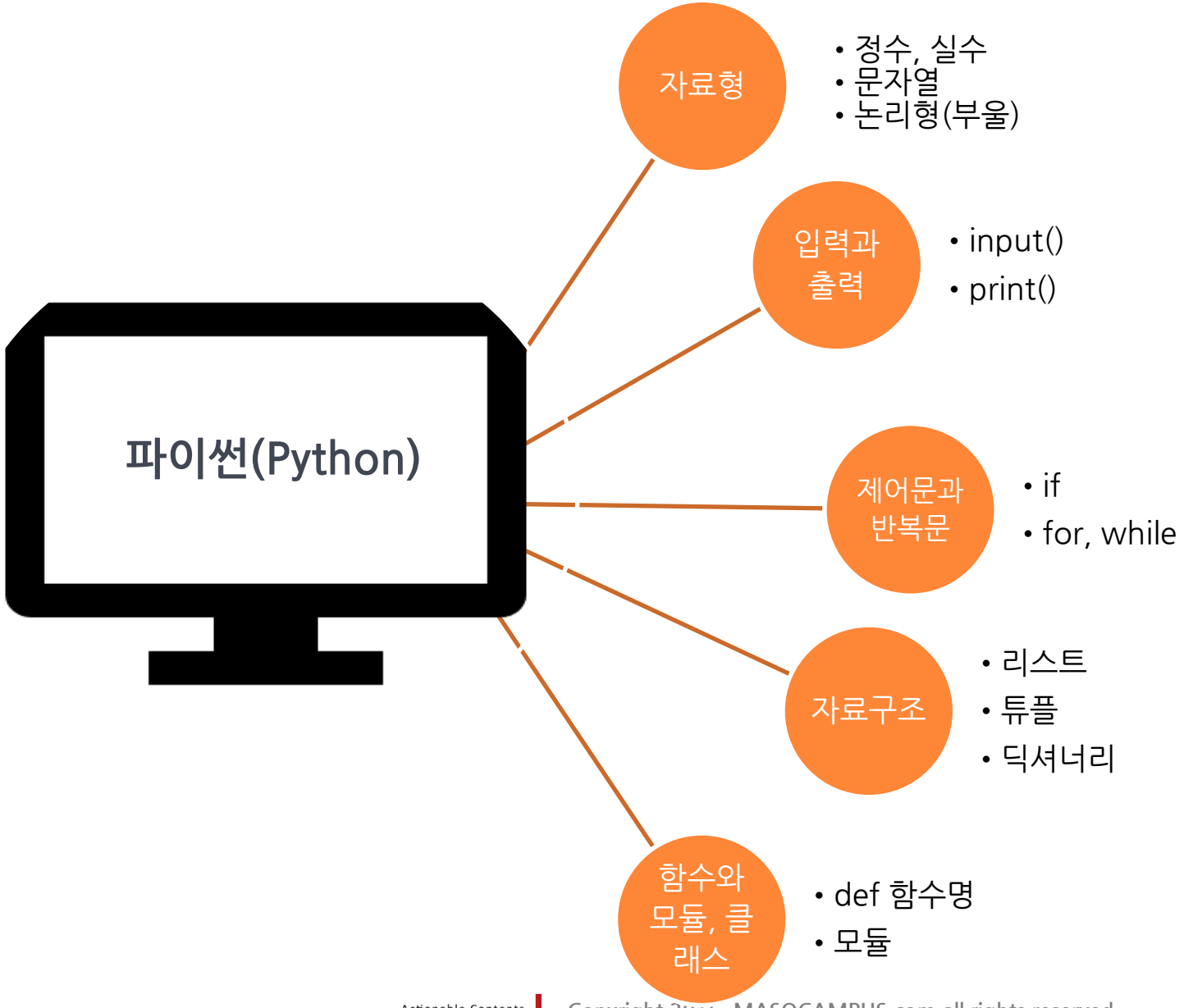
파이썬의 창시자 귀도 반 로섬(출처 : 위키피디아)

- 파이썬의 장점과 단점
  - 스크립트 언어 - 한줄 한줄 작성할 때마다 실행이 됨
  - 장점
    - 강력한 기능을 무료로 사용할 수 있다.
    - 읽기 쉽고 사용하기 쉽다.
    - 다양하고 강력한 외부 라이브러리들이 풍부하다.
    - 컴파일이 필요 없다.
    - 상대적으로 쉽다.
  - 단점
    - 속도가 느리다.(느낄 수 있는 정도는 아님)

- 컴파일러 vs 인터프리터

비교	컴파일러	인터프리터
작동방식	소스코드를 기계어로 먼저 번역 해당 플랫폼에 최적화시켜 프로그램을 실행함	별도의 번역 과정 없이 소스코드를 실행시점에 해석하여 컴퓨터가 처리할 수 있도록 함
장점	실행속도가 빠름	간단히 작성, 메모리가 적게 필요
단점	한번에 많은 기억장소가 필요	실행속도가 느림
주요언어	C, 자바, C++, C#	파이썬, 자바스크립트, 스칼라

모든 프로그램 언어에는 이것이 있다!!



필요한 모듈 import 및 버전 확인

```
1 import sys                    # 파이썬 시스템 모듈을 import
```

```
1 sys.version_info              # 파이썬 버전 확인
```

```
sys.version_info(major=3, minor=7, micro=4, releaselevel='final', serial=0)
```

```
1 import numpy as np            # 선형대수를 처리하는 넘파이 모듈 import, 별칭으로 np를 사용
```

```
1 np.__version__                # 모듈의 버전을 확인한다.
```

```
'1.16.5'
```

```
1 import pandas as pd           #  데이터프레임 처리를 판다스 모듈을 import, 별칭으로 pd를 사용
```

```
1 pd.__version__                # 모듈의 버전을 확인한다.
```

```
'0.25.1'
```

필요한 모듈 import 및 버전 확인

```
1 import matplotlib as mpl        #  기본 시각화 모듈을 import, 별칭으로  mpl를 사용
```

```
1 mpl.__version__                 # 모듈의 버전을 확인한다.
```

```
'3.1.1'
```

```
1 import seaborn as sns           #  데이터 프레임을 시각화할 수 있는 모듈을 import, 별칭으로  sns 사용
```

```
1 sns.__version__                       ## 모듈의 버전을 확인한다.
```

```
'0.9.0'
```

```
1 import sklearn as sk            # 머신러닝을 지원하는 모듈을 import, 별칭으로 sk 사용
```

```
1 sk.__version__                   # 모듈의 버전을 확인한다.
```

```
'0.22.1'
```

변수와 데이터 타입(Data Type)

<변수명의 명명규칙>
•영문 문자와 숫자를 사용할 수 있다.
•대소문자를 구분한다.
•문자부터 시작해야 하며 숫자부터 시작하면 안 된다.
•_(밑줄 문자)로 시작할 수 있습니다.
•특수 문자(+, -, *, /, $, @, &, % 등)는 사용할 수 없다..
•파이썬의 키워드(if, for, while, and, or 등)는 사용할 수 없다.

- 변수의 선언
  - 변수는 어떠한 값을 저장하는 메모리 공간(그릇)
  - 변수 선언은 그릇을 준비하는 것
  - 파이썬은 C/C++, 자바 등과는 달리 변수를
    선언하지 않아도 되지만 긴 코드를 작성할 때는 사용될 변수를 미리 계획적으로 준비하는 것이 더 효율적

```
boolVar = True
intVar = 0
floatVar = 0.0
strVar = ""
```

<변수 여러 개를 한꺼번에 만들기>
1) x, y, z = 10, 20, 30 #x=10; y=20; z=30;
2) x = y = z = 10 #x=10; y=10; z=10;

TIP • 이 구문은 다음과 같이 표현해도 된다.

```
boolVar, intVar, floatVar, strVar = True, 0, 0.0, ""
```

  - 가장 많이 사용하는 변수는 정수형, 실수형, 문자열 , 불형(Boolean, True 또는 False 저장)

변수와 데이터 타입(Data Type)

- **파이썬 식별자**
  - 파이썬의 식별자는 파이썬에 있는 모든 이름을 정의하는 것
  - 키워드로 사용되는 것을 식별자로 사용하지 않도록 조심해야 함

```
import keyword
import pprint
pprint.pprint(dir(keyword))
```

```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'iskeyword',
 'kwlist',
 'main']
```

```
count = 0
for i in keyword.kwlist:
    count += 1
    if count % 5 == 0:
        print()
    print(i, end=' ')
```

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

변수와 데이터 타입(Data Type)

- 변수의 삭제

```
1 a = 1
2 b = 1
3 print(id(a), a)
4 print(id(b), b)
```

```
140734056604048 1
140734056604048 1
```

```
1 del a
2 try :
3     print(a)
4 except Exception as e :
5     print(e)
```

```
name 'a' is not defined
```

```
1 print(id(b), b)
```

```
140734056604048 1
```

데이터 타입(Data Type)

참고) 파이썬은 변수 생성시 데이터형이 결정되지 않고
변수에 값을 넣는 순간에 변수의 데이터형이 결정된다.
→ '동적 변수 할당'(동적 특징)

숫자 (numbers)	문자열 (string)	논리형 (boolean)
정수형 (int) 100 실수형 (float) 12.345 복소수형 (complex)	'Hello world!' A sequence set of characters Enclose by ' ' or " "	True or False

변수와 데이터 타입(Data Type)

- **데이터 타입 확인 - type( ) 함수**
  - bool(불형), int(정수), float(실수), str(문자열) 타입 확인

```
type(boolVar), type(intVar), type(floatVar), type(strVar)
```

출력 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

```
1  a=100
2  type(a)
```

```
int
```

```
1  b=0.5
2  type(b)
```

```
float
```

```
1  c=True
2  type(c)
```

```
bool
```

```
1  d="hohoho"
2  type(d)
```

```
str
```

- **데이터 타입 변환 함수 : 캐스트 함수(cast function)**
  - int() 함수 : 문자열을 int(정수) 데이터 타입으로 변환
  - float() 함수 : 문자열을 float(실수) 데이터 타입으로 변환
  - str() 함수 : 다른 데이터 타입의 데이터를 str(문자열) 데이터 타입으로 변환
  - bool() 함수 : 다른 데이터 타입의 데이터를 불형(bool) 데이터 타입으로 변환

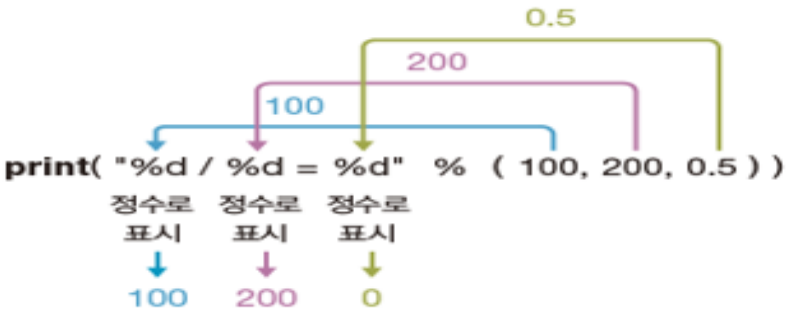
print 출력문

- **값을 여러 개 출력하기**
  - print에는 변수나 값 여러 개를 ,(콤마)로 구분
  - **print(값1, 값2, 값3)**
  - **print(변수1, 변수2, 변수3)**

```
1 #값을 여러개 출력하기
2 print(1, 2, 3)
3 print('Hello', 'Python')
```

```
1 2 3
Hello Python
```

- **서식이 있는 print( ) 함수를 사용한 다양한 출력**

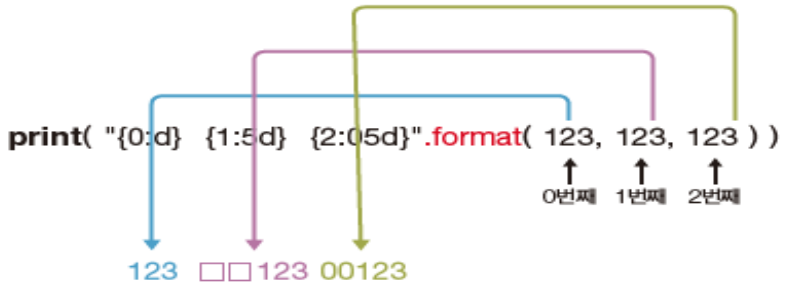



```
print("%d / %d = %5.1f" % (100, 200, 0.5))
```

print() 함수에서 사용할 수 있는 서식

서식	값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수, 16진수, 8진수)
%f	0.5, 1.0, 3.14	실수(소수점이 붙은 수)
%c	"b", "한"	한 글자
%s	"안녕", "abcdefg", "a"	두 글자 이상인 문자열

- **format( ) 함수와 { }를 함께 사용해 서식 지정**



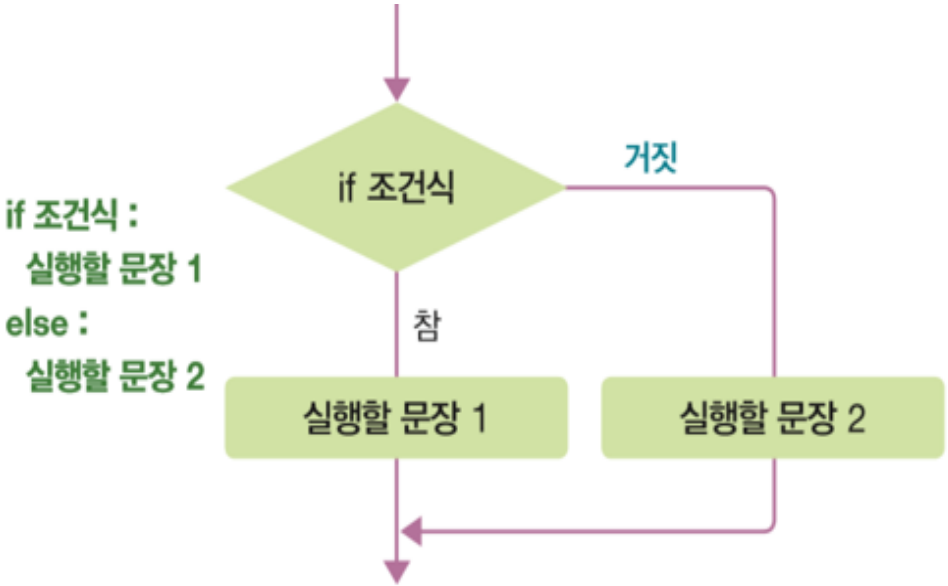

```
print("{0:d} {1:5d} {2:05d}".format(123, 123, 123))
```

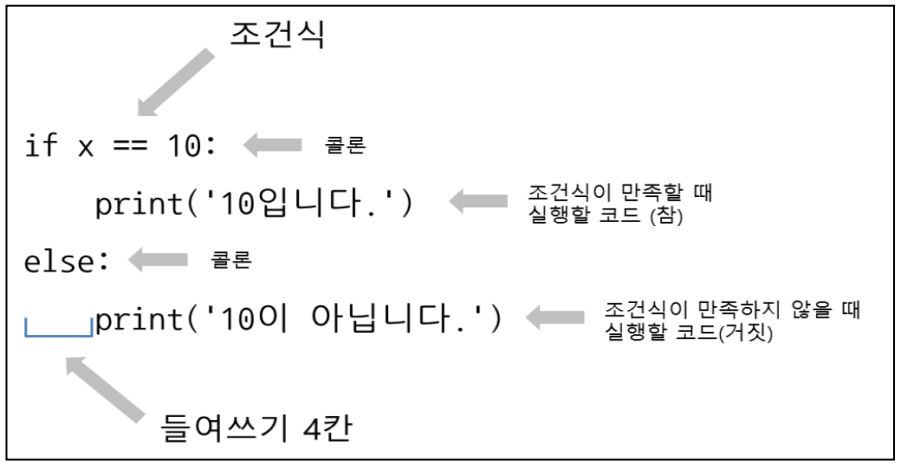
파이썬 프로그램 흐름 제어(flow control)

조건문 (branch statement)	반복문 (loop statement)
if..else 한가지 조건 분기문	for 순서열의 처음부터 끝까지 반복 for(시작값, 끝값+1, 증가값)
if..elif..else 2가지 이상 조건 분기문	while 조건이 참인 동안 반복
nested if statement if..if..else if..else 분기문 안에 if..else문	continue, break 반복문 중단 : break 해당 반복만 건너뜀 : continue

if~else 조건문

- if~else 문
  - 조건이 참일 때와 거짓일 때 실행할 문장이 다름





- if..else..의 축약(conditional expression; 조건부 표현식)
  - 변수 = 값 if 조건문 else 값

```
1 #if..else..축약(조건부 표현식 )
2 x = 5
3 y = x if x == 10 else 0
4 y
```

```
0
```

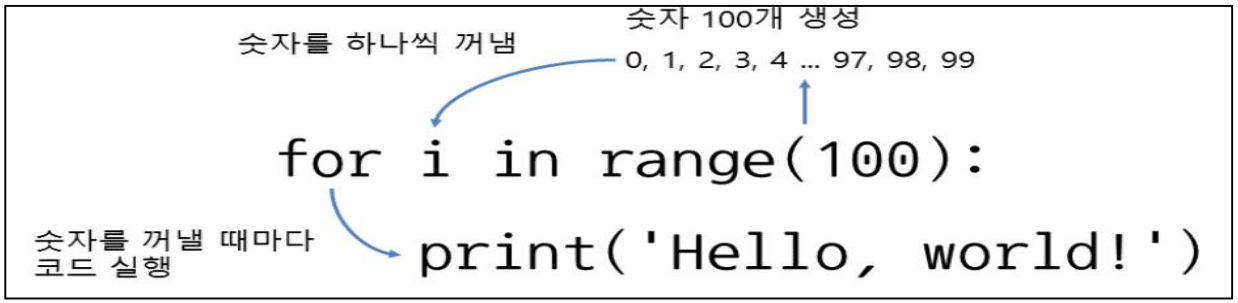
for 반복문

- **for 문 기본 형식**

```
for 변수 in range(시작값, 끝값+1, 증가값) :
    이 부분을 반복
```

for 변수 in range(끝값+1)
for 변수 in range(시작, 끝값+1)
for 변수 in range(시작, 끝값+1, 증가값)

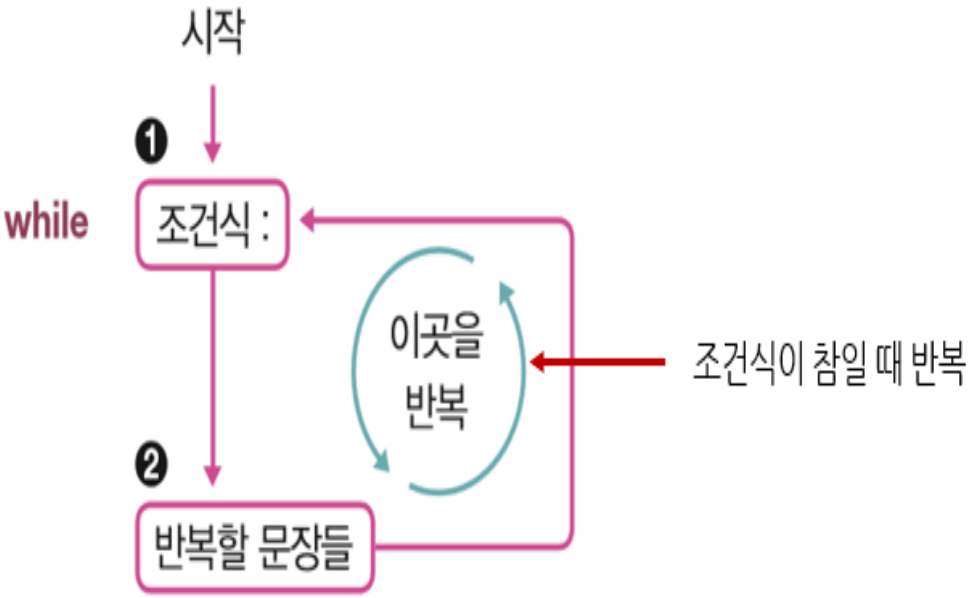
- **for 반복문 : for와 range 사용**
  - for 반복문은 'for' 다음에 '반복 변수'를 지정해주고
  - 'in' 다음에 순서열(sequence), 그리고 마지막에 '콜론(:)'을 써주고 , 그 다음줄에 반복을 시킬 코드 블록을 쓴다. 이때 들여쓰기 (indentation)을 반드시 해주어야 함
  - range(시작값, 끝값+1, 증가값)
    - 시작값을 생략하면 순서열은 '0'부터 디폴트로 시작하고,
    - 증가값은 순서열의 간격을 설정할 수 있으며,
    - 증가값을 생략하면 순서열의 간격은 '1'이 디폴트이고,
    - 시작값, 끝값, 중간값은 정수만 인식하고 부동소수형(float)은 TypeError 가 발생

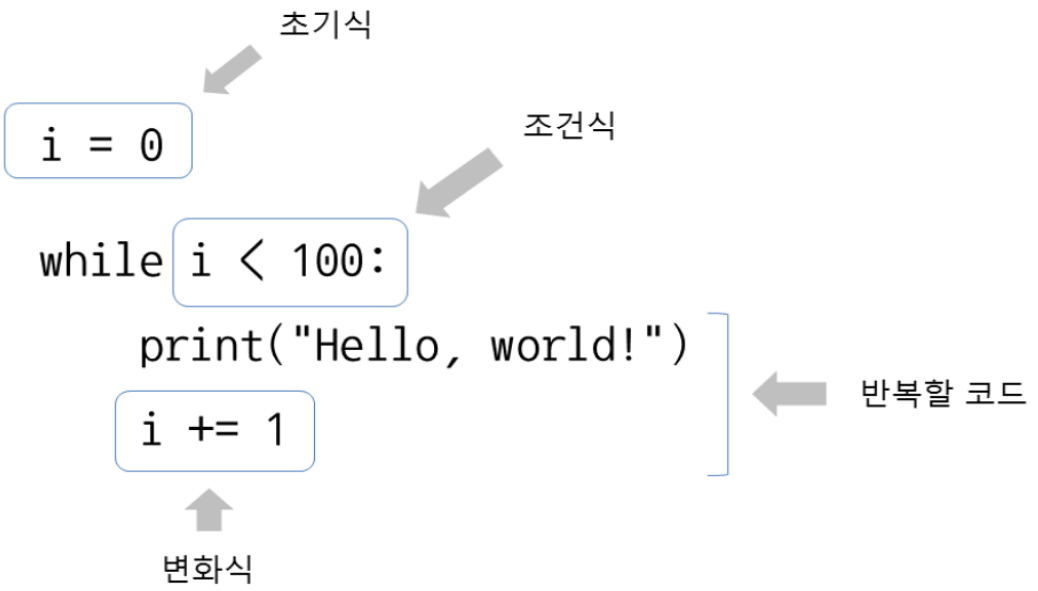


while 반복문

- while 문
  - while 문은 반복 횟수를 결정하기보다는 조건식이 참일 때 반복하는 방식

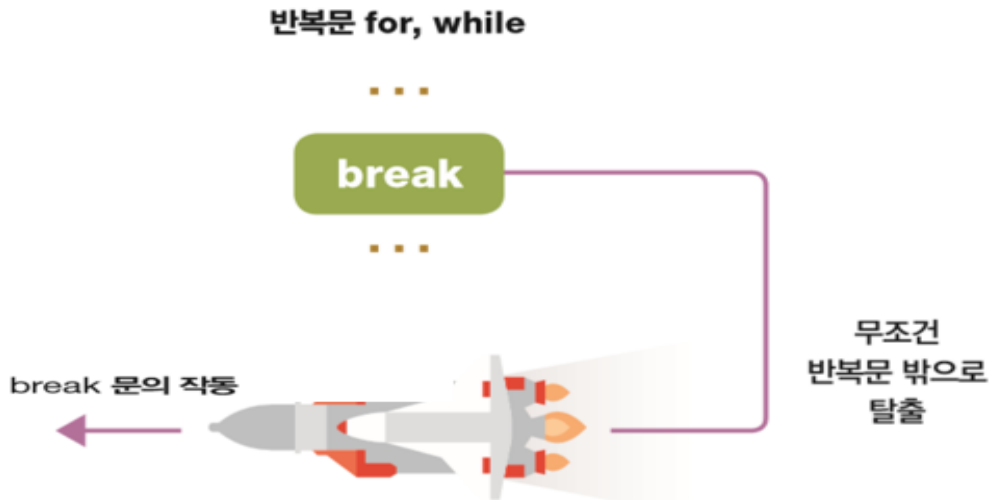
```
변수 = 시작값
while 변수 < 끝값 :
    이 부분을 반복
    변수 = 변수 + 증가값
```





break 문과 continue 문

- **반복문을 탈출시키는 break 문**
  - 계속되는 반복을 논리 적으로 빠져나가는 방법



- **반복문으로 다시 돌아가게 하는 continue 문**

- **break문 예제**

```
for i in range(1, 100) :
    print("for 문을 %d번 실행했습니다." % i)
    break
```

출력 결과

```
for 문을 1번 실행했습니다.
```

- **continue문 예제**
  1~100의 합계를 구하되,
  3의 배수를 제외하고 더하기

```
1  hap, i = 0, 0
2
3  for i in range(1, 101) :
4      if i % 3 == 0 :
5          continue
6
7      hap += i
8
9  print("1~100의 합계(3의 배수 제외) : %d" % hap)
```

출력 결과

```
1~100의 합계(3의 배수 제외) : 3367
```

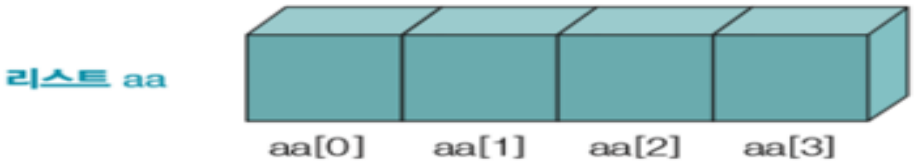
파이썬 자료구조

파이썬 자료구조

리스트 (list)	튜플 (tuple)	딕셔너리 (dictionary)
['abc', 123]	('abc', 123)	{'학번': 1000, '이름': '홍길동', '학과':'컴퓨터 ' }
1 dimensional sequence of different data type objects	1 dimensional sequence of different data type objects	Hash table type Associative array Key-value pairs
Can be updated	Can Not be updated	
Enclosed by [ ]	Enclosed by ( )	Enclosed by { }

리스트

- **리스트의 개념**
  - 하나씩 사용하던 변수를 한 줄로 붙여 놓은 형태



- **리스트를 생성하는 방법**
  - 리스트 안에 문자열, 정수, 부동소수형, 리스트, 튜플 등 다양한 형태의 자료들이 들어갈 수 있으며, 콤마로 구분해주고, 대괄호로 감싸준다.
  - 리스트의 첨자는 0부터 시작한다.

```
리스트명 = [값1, 값2, 값3, …]
```

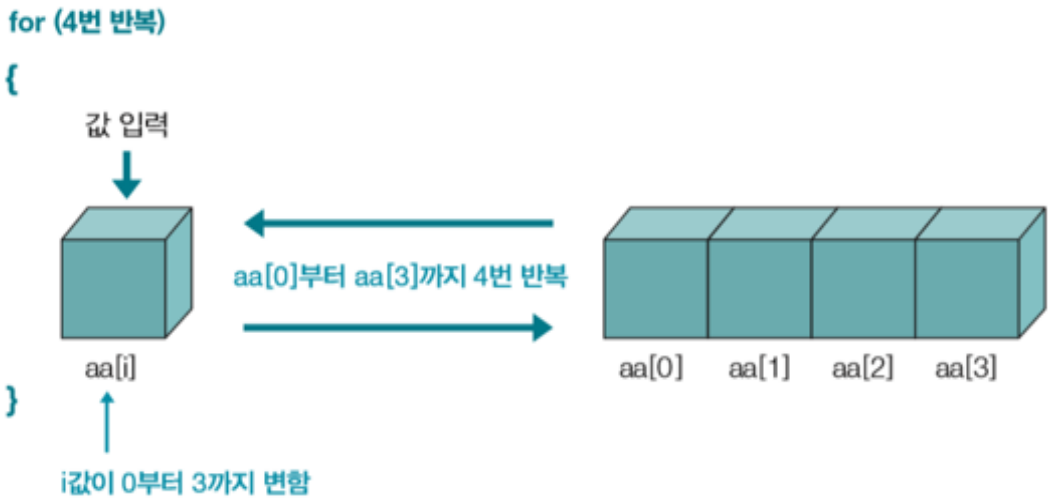
(예) aa = [10, 20, 30, 40] list1 = ['abc', 123, 3.14, 'cde']

- **for 반복문으로 요소 출력하기**

[형식] for 변수 in 리스트 :
반복할 코드

```
1  #for로 모든 리스트 출력
2  a = [10, 20, 30, 40]
3  for i in a:
4      print(i)
```

```
10
20
30
40
```

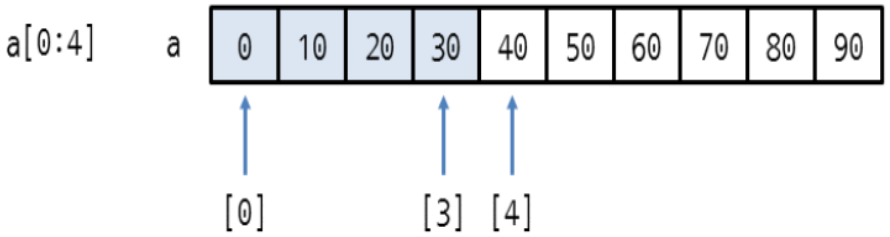


리스트의 인덱싱과 슬라이싱(slicing)

- 1) 리스트[시작인덱스:끝인덱스]

```
1  # 인덱스로 범위를 지정하여 리스트의 일부만 가져오기
2  a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
3  a[0:4]
```

```
[0, 10, 20, 30]
```



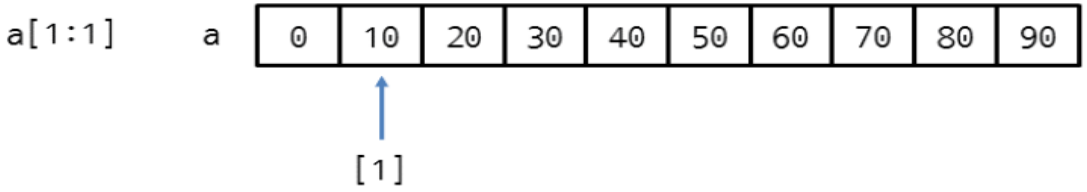

```
1  a[0:10]
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

- 2) 시작 인덱스와 끝 인덱스가 같을 때 - 값을 가져오지 못함

```
1  # 시작 인덱스와 끝 인덱스가 같을 때
2  a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
3  a[1:1]
```

```
[]
```



리스트의 인덱싱과 슬라이싱(slicing)

- 3) 리스트의 중간 부분을 가져 가져오기

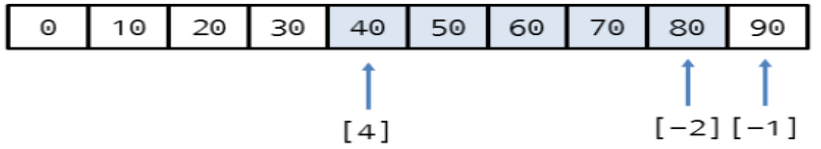
```
1  #리스트 중간 부분 가져오기
2  a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
3  a[4:7]
```

```
[40, 50, 60]
```

```
1  a[4:-1]
```

```
[40, 50, 60, 70, 80]
```

a[4:-1] a



- 4) 시작 인덱스와 끝 인덱스 생략하기 → 리스트의 마지막 일부분만 출력할 때 사용

```
1  # 시작 인덱스와 끝 인덱스 생략하기 -> 리스트의 마지막 일부분만 출력할 때 사용
2  a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
3  a[:7]
```

```
[0, 10, 20, 30, 40, 50, 60]
```

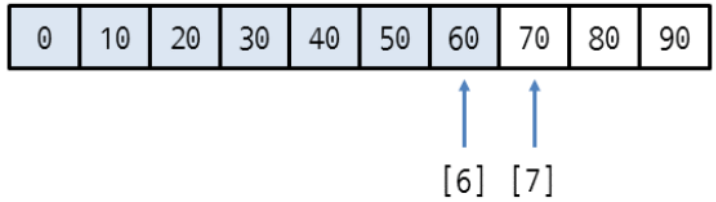
```
1  a[7:]
```

```
[70, 80, 90]
```

```
1  a[:]
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

a[:7] a

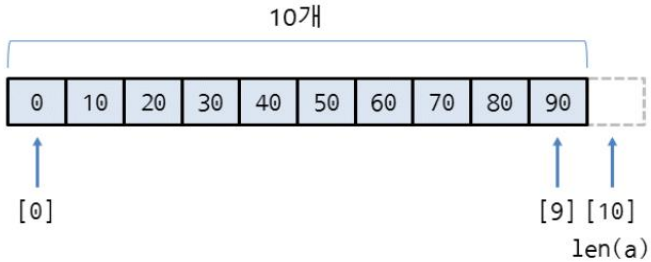


- 5) len으로 전체 리스트 가져오기

```
1  # len으로 전체 리스트 가져오기
2  a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
3  a[0:len(a)]
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

a[0:len(a)] a

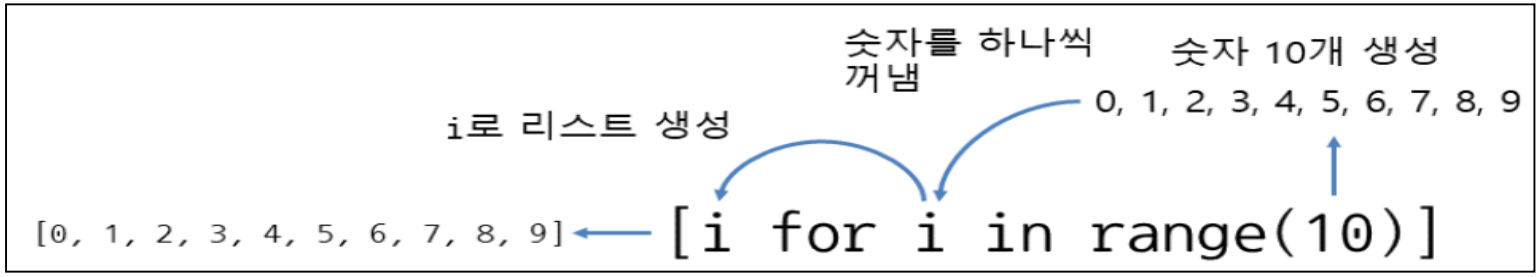


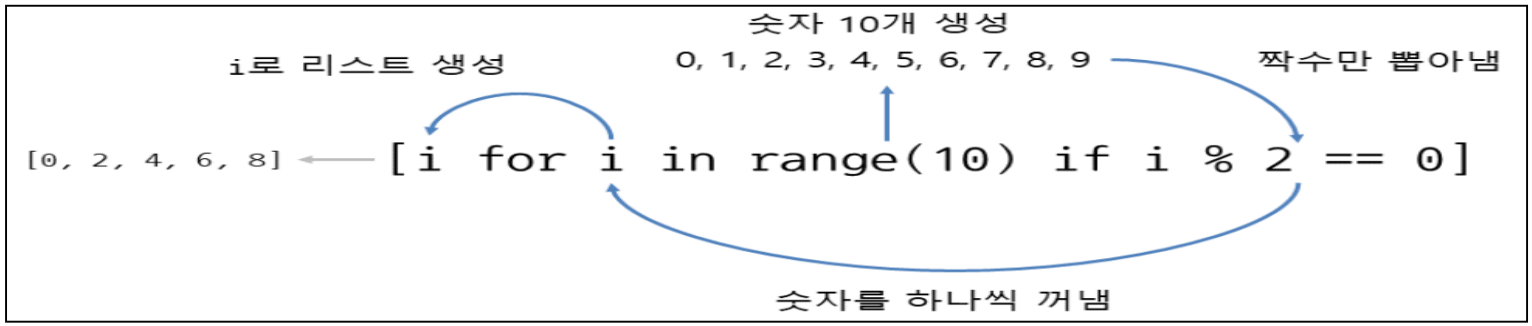
리스트 컨프리헨션((list comprehension)

- 리스트 내포, 리스트 내장, 리스트 축약, 리스트 해석, 리스트 표현식

[형식] [식 for 변수 in 리스트]

list(식 for 변수 in 리스트)





튜플(tuple)

- **튜플의 생성**
  - 튜플은 소괄호 ( )로 생성(소괄호 생략 가능), 리스트는 대괄호 [ ]로 생성
  - 튜플은 값을 수정할 수 없으며, 읽기만 가능해 읽기 전용 자료를 저장할 때 사용

- **빈 튜플 생성하기**
  - **a = () 또는 a = tuple()**

- **튜플의 오류**
  - 읽기 전용이므로 튜플 항목을 추가, 수정, 삭제 불가능 **→ 내용 변경 불가(불변, immutable)**
  - 단, 튜플 자체는 삭제 가능 (예) del t1

- **튜플의 인덱싱과 슬라이싱**
  - 튜플 항목에 접근 : 튜플명[위치]
  - 튜플 범위에 접근 : 리스트와 동일한 방법으로 접근, 튜플명[시작값:끝값+1]

- **튜플의 연산 : 덧셈(튜플들끼리 연결) 및 곱셈 연산(숫자만큼 반복)**

딕셔너리(Dictionary)

- 영어사전이나 국어 사전과 비슷
- **{키(Key) : 값(Value)}** 이 콜론(:)으로 구분이 되어서 쌍(pair)을 이루고 있고, 중괄호({ })로 싸여져 있는 자료형
  딕셔너리변수 = {키:값, 키:값, 키:값, …} **(예) {'apple': '사과', 'banana':'바나나', 'cherry':'체리 ' }**

- 다른 프로그래밍 언어에서는 해시(Hash), 연관 배열(Associative Array)이라 함
- 사전(Dictionary) 자료형의 키(Key)는 튜플 처럼 변경 불가능(immutable)하고 유일한 값(unique)을 가지며, 값(Value)은 리스트처럼 변경이 가능(mutable)

- **딕셔너리 키의 접근**
  - 1) 딕셔너리 키를 이용하여 값에 접근하는 코드 : [] 사용 (예) student['학번']
  - 2) 딕셔너리명.get(키) 함수를 사용해 키로 값에 접근 (예) student.get('학번')

- **딕셔너리 함수**
  - 딕셔너리명.keys() : 딕셔너리의 모든 키 반환
  - 딕셔너리명.values() : 딕셔너리의 모든 값을 리스트로 만들어 반환
  - 딕셔너리명.items() : 튜플 형태로 키와 값을 모두 구할 수 있음

- **in 연산자 사용**
  - 딕셔너리 안에 해당 키가 있는지 없는지는 in을 사용해 확인
  - 딕셔너리에 키가 있다면 True를 반환하고, 없다면 False를 반환 (예) '주소' in student

파이썬 자료구조

문자열(String)

- 문자열(String)
  - 문자열을 담는 자료형
  - ‘ ‘, “ “, ‘‘‘ ’’’, “”” “””
- 문자열의 연산
  - + : 문자열 붙이기
  - * : 문자열 반복하기

- 여러줄 짜리 문자열을 담기 위해 ‘’’ ’’’ 사용

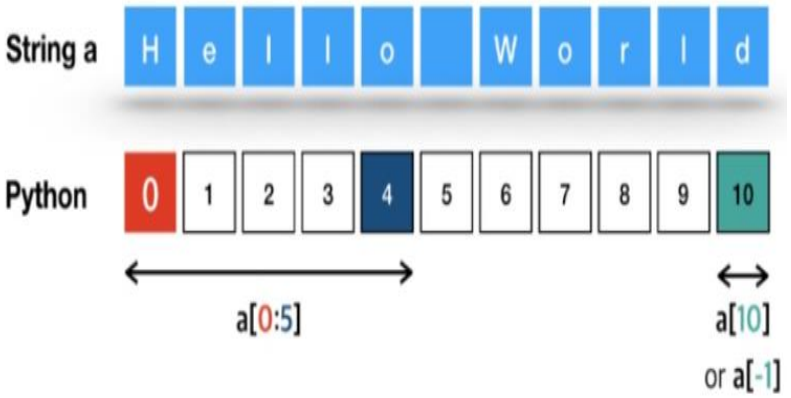
```
1  a = '''python
2  is
3  awesome'''
4  print(a)
```

```
python
is
awesome
```

```
1  a = "Python"
2  b = "awesome"
3  print(a + " is " + b)
4  print("="*30)
```

```
Python is awesome
==============================
```

- 문자열 - 인덱싱과 슬라이싱
  - 인덱싱 : 문자열에서 특정 문자를 가리키는 것을 의미
  - 슬라이싱 : 문자열에서 특정 문자열을 가리키는 것을 의미, 처음과 끝은 생략 가능




```
1  # 문자열의 인덱싱과 슬라이싱
2  a = "Hello World"
3  print(a[0])
4  print(a[10])
5  print(a[-1])
6  print(a[0:5])
7  print(a[6:])
8  print(a[:5])
```

```
H
d
d
Hello
World
Hello
```

Tip) 문자열과 리스트와 비교 : 리스트는 대괄호 [ ]로 묶고, 문자열은 따옴표로 묶어 출력

함수와 모듈

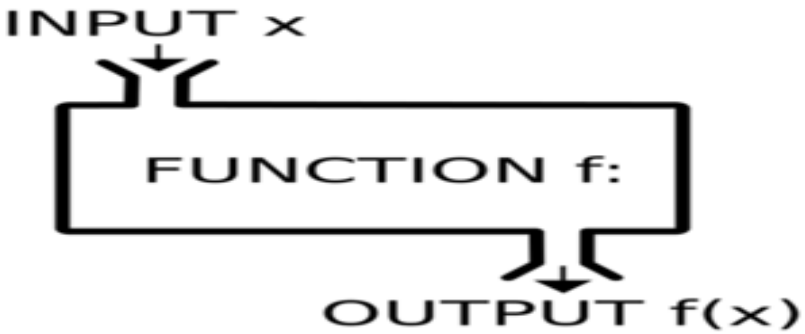
함수의 기본

- **함수의 개념과 필요성**
  - 함수(Function) : '무엇'을 넣으면, '어떤 것'을 돌려주는 요술 상자, '어떤 일 ' 이 일어나기도 함
  - 함수와 메서드(Method)와 차이점 : 함수는 외부에 별도로 존재, 메서드는 클래스 안에 존재
  - 함수의 입력 값이 없을 수도, 결과값이 없을 수도 있음

- **함수의 형식**

```
함수명()
```

- **함수 사용시 주의 사항**
  - **함수 호출이 되기 전까지 함수 안에 있는 문장은 수행이 안됨!!!**
  - 함수는 호출 되기 전까지 먼저 만들어져야 함
  - 입력 값은 함수안에서 변수로 사용됨
  - 매개변수는 파라메타(parameter)라고 부르기도 함



람다 함수

- 람다 함수(lambda)
  - 함수의 이름 없이, 함수처럼 사용할 수 있는 익명의 함수
  - 함수를 한 줄로 간단하게 만들어 줌

```
1  # 일반함수
2  def f(x, y) :
3      return x + y
4
5  print(f(1, 4))
```

```
5
```

```
1  # 람다 함수 이용하여 일반함수 표현
2  f2 = lambda x, y : x + y
3  print(f2(1, 4))
```

```
5
```

- 람다 함수에서 매개변수의 기본값을 설정
  - 매개변수를 지정하지 않으면 기본값으로 설정, 매개변수를 넘겨주면 기본값 무시

```
1  f3 = lambda x=10, y=20 : x + y
2  print(f3())
3  print(f3(100, 200))
```

```
30
300
```

모듈(module)

- **모듈(Module) : 함수의 집합**
  - 만들어진 함수를 다른 프로그램에서 사용하고 싶을 때 사용하는 방식
  - 파이썬 파일로 별도로 분리, import 문을 통해 모듈 사용

```
1 import math
2 print(dir(math), end="")  #모듈이 제공하는 함수 확인하기
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'c
opysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'p
i', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

- **모듈을 import 하는 방법**
  - **import 모듈이름** (예) import os
  - **import 모듈이름 as 별칭** (예) import pandas as pd
  - **from 모듈이름 import 모듈함수** (예) from os import listdir … 모듈이름을 앞에 붙이지 않고도 사용
  - **from 모듈이름 import *** (예) from mod1 import *

파이썬 모듈

모듈(module)

- **모듈의 종류**
  (1) 사용자 정의 모듈 : 직접 만들어서 사용하는 모듈
  (2) 표준 모듈(코어 모듈) : 파이썬에서 제공하는 모듈
  (3) 서드 파티(3rd Party) 모듈 : 파이썬이 아닌 외부 회사나 단체에서 제공하는 모듈
    - 서드파티 모듈 설치 관리자 : pypi, 파이썬 모듈 중앙 저장소, https://pypi.python.org/pypi

- **서드 파티 모듈의 설치**
  - pip 패키지 매니저를 이용하여 파이썬 모듈을 설치한다.
  - pip install 모듈명 예) pip install requests

- **모듈과 패키지(package)**
  - 모듈 : 함수와 클래스를 정리해서 파일로 분리시키는 방법, 하나의 *.py 파일 안에 함수가 여러 개 들어 있는 것
  - 패키지 : 여러 모듈을 정리하는 방법을 제공, 모듈을 주제별로 분리할 때 주로 사용

- **패키지(package)**
  - 파이썬 모듈을 계층적(디렉터리) 구조로 관리할 수 있게 모아 놓은 디렉토리(폴더)
  - __init__.py 파일이 디렉토리에 위치하면 파이썬은 패키지로 인식, 내용은 없는 파일
  - import는 from .. import 문법 사용