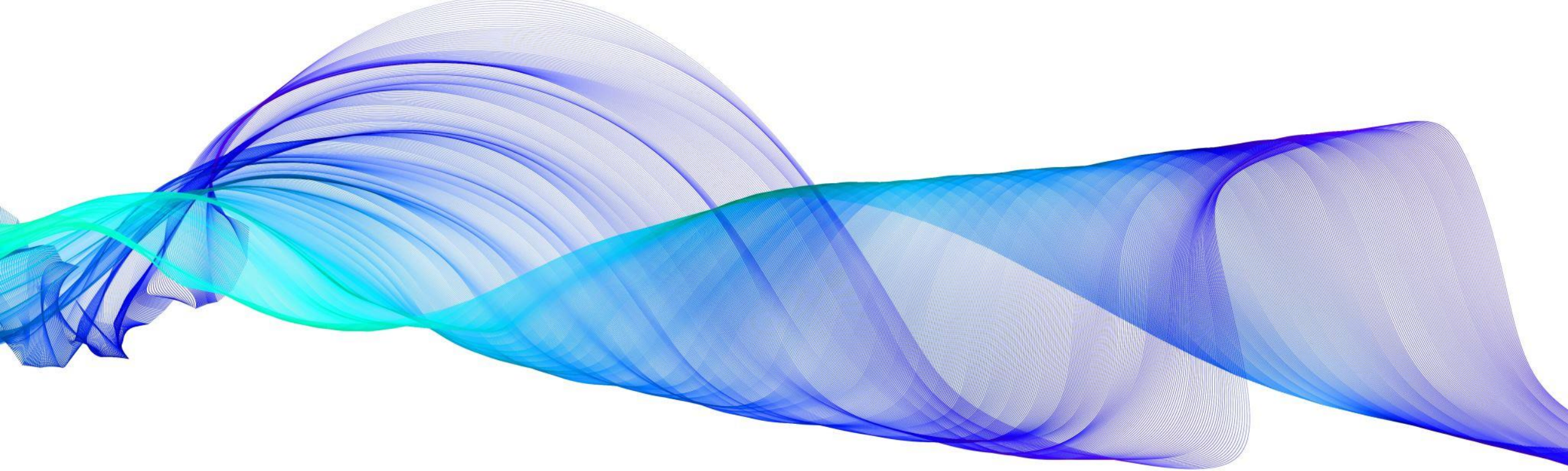
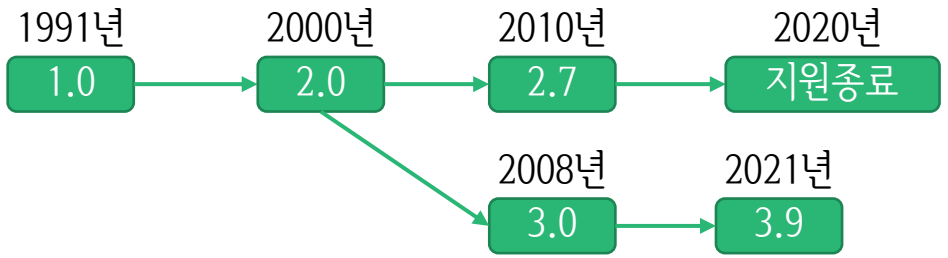


Python

Soft Engineer Society

▣ Prologue

- **Python 특징**
  - 1991년 2월
  - 네덜란드의 귀도 반 로섬(Guido van Rossum) 발표
  - 현재 파이썬 버전 : 3.9
  - 파이썬 2.x와 3.x는 서로 호환되지 않음



- **언어로써의 특징**
  - 인터프리터 언어
  - 객체지향 언어
  - 개발기간이 짧다
  - 플랫폼 독립적 ⇨ 웹에 유리
  - 활용분야가 다양
  - 라이브러리가 많다.
  - C언어와 융합이 좋다

▣ Prologue

- **에피소드**
  - 1989년 크리스마스에 연구실이 닫혀 있어 할 일이 없어 파이썬 개발
  - “Monty Python’s Flying Circus” -> 파이썬 이름의 유래
  - 구글, DropBox 근무

Hi Guido,

I came across your resume in a Google web search.
You seem to have an awesome expertise on Python .
I would be glad if you can reply my email and let me know your interest and availability.
……..
Our client immediately needs a PYTHON Developers at its location in * , NJ. Below are the job details. If interested and available, kindly fwd me your updated resume along with the expected rate and the availability.

I’m not interested and
not Available

Source: https://goo.gl/GysBnU

▣ 개발 환경 구축 및 실행

- **Python 설치 및 버전 확인**
  - 2020. 10. : v3.9
  - https://www.python.org/downloads/

    c:₩> python --version

- **실행 예**

  c:₩> python
  Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)]
  on win32
  Type "help", "copyright", "credits" or "license" for more information.
  >>>
  >>> Ctrl + Z ➔ 종료

▣ 개발 환경 구축 및 실행

- **Interactive Mode**
  - 하나의 명령문을 즉시 번역해서 실행

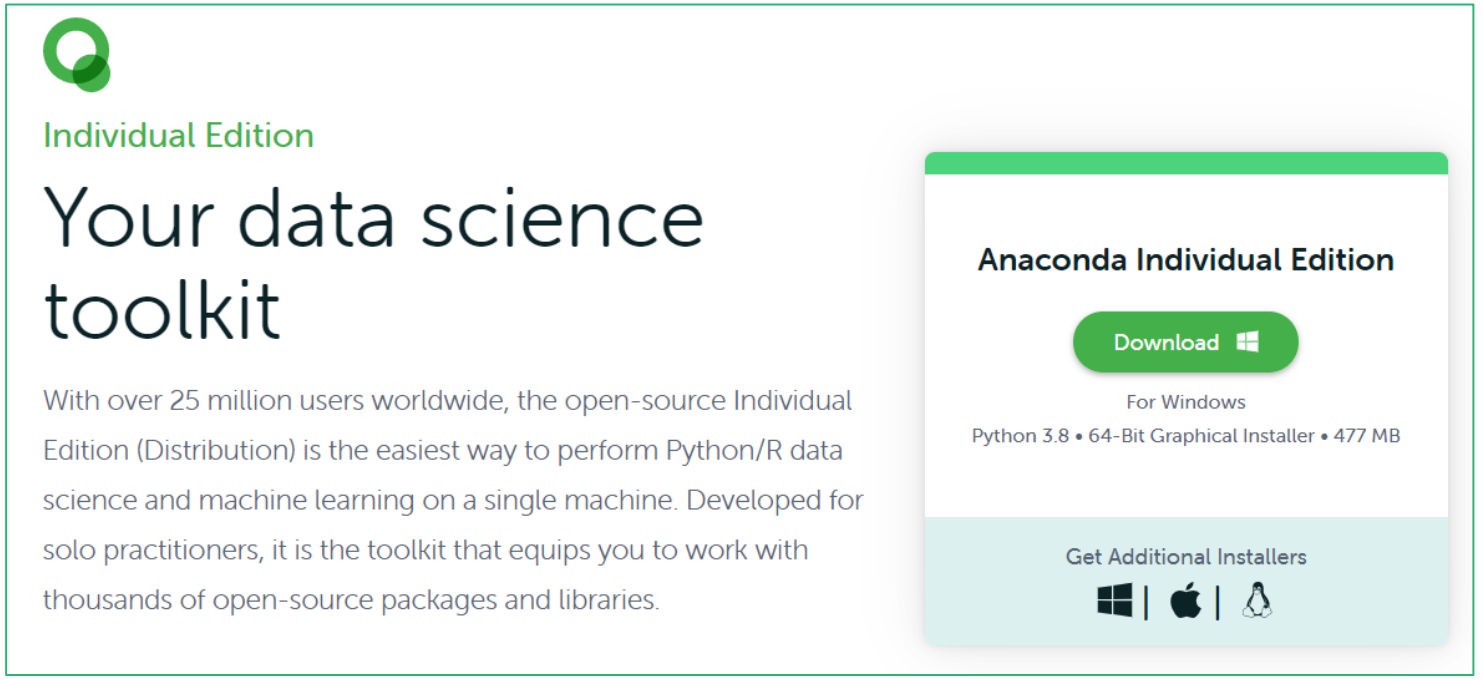
  >>> Prompt에서 실행

- **Script Mode**
  - 일련의 명령을 모아서 한꺼번에 실행
  - 파일로 저장한 후 실행 (*.py로 저장)
  - Python이 제공하는 IDLE을 이용하거나 Editor 사용

▣ Anaconda 개발 환경 구축

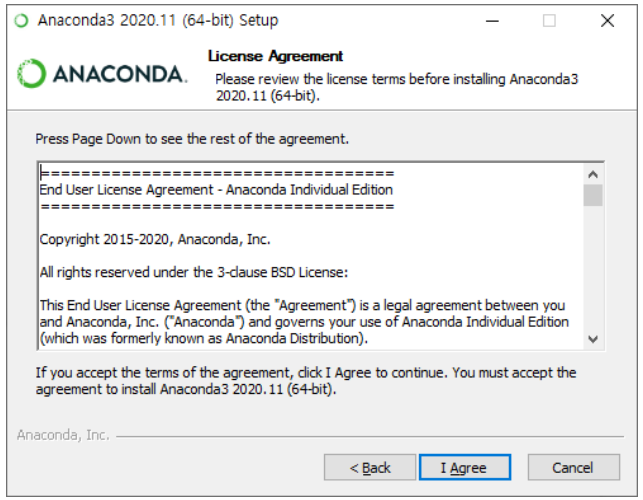
- **Anaconda 다운로드**
  - 다운로드 URL : https://www.anaconda.com/products/individual
  - 아나콘다는 개발에 필요한 패키지(Numpy, Pandas...)들을 모아놓은 개발환경

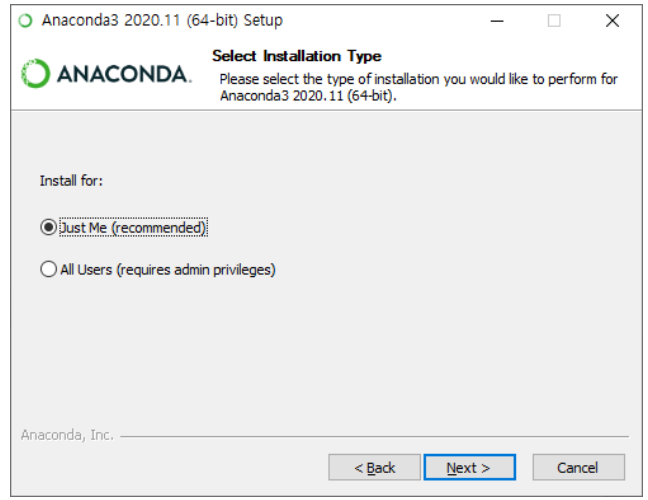
- **다운로드 화면**

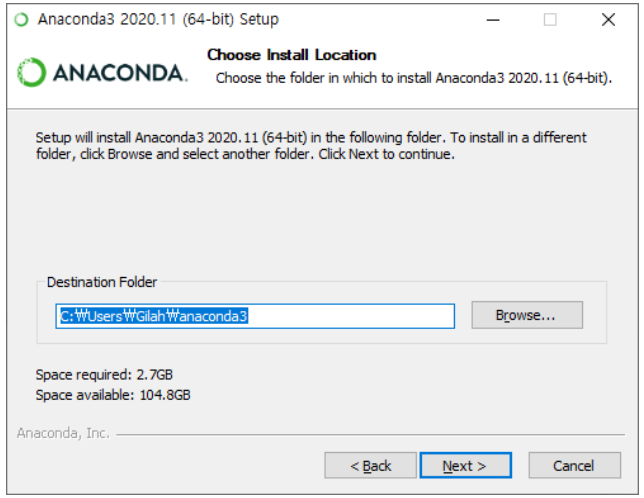


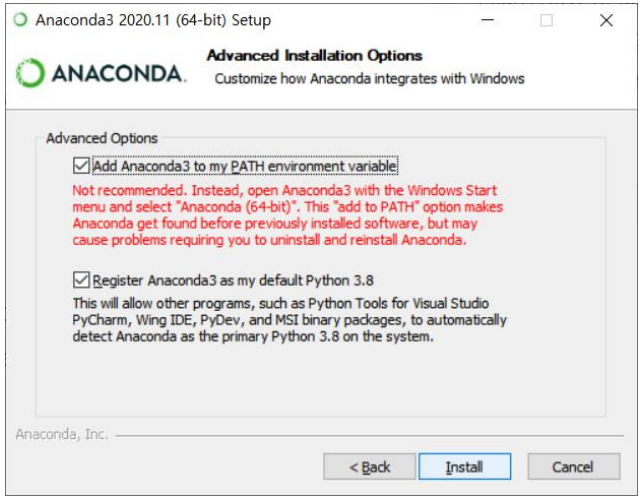
▣ 개발 환경 구축

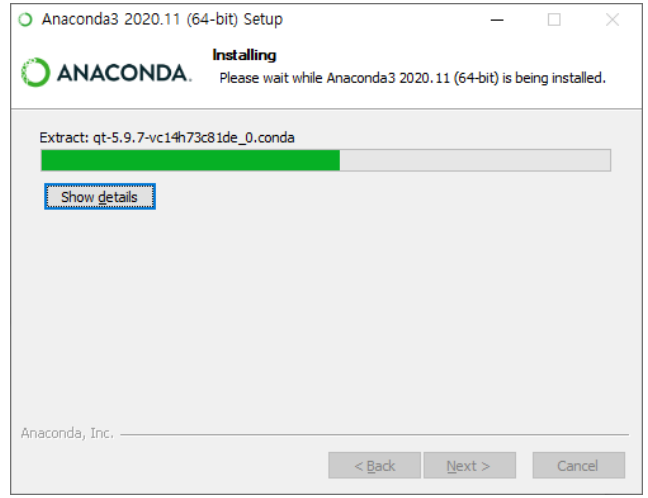

Anaconda3 2020.11 (64-bit) Setup
ANACONDA.
Welcome to Anaconda3 2020.11 (64-bit) Setup
Setup will guide you through the installation of Anaconda3 2020.11 (64-bit).
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel


Anaconda3 2020.11 (64-bit) Setup
ANACONDA.
License Agreement
Please review the license terms before installing Anaconda3 2020.11 (64-bit).
Press Page Down to see the rest of the agreement.
End User License Agreement - Anaconda Individual Edition
Copyright 2015-2020, Anaconda, Inc.
All rights reserved under the 3-clause BSD License:
This End User License Agreement (the "Agreement") is a legal agreement between you and Anaconda, Inc. ("Anaconda") and governs your use of Anaconda Individual Edition (which was formerly known as Anaconda Distribution).
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2020.11 (64-bit).
Anaconda, Inc.
< Back
I Agree
Cancel


Anaconda3 2020.11 (64-bit) Setup
ANACONDA.
Select Installation Type
Please select the type of installation you would like to perform for Anaconda3 2020.11 (64-bit).
Install for:
Just Me (recommended)
All Users (requires admin privileges)
Anaconda, Inc.
< Back
Next >
Cancel

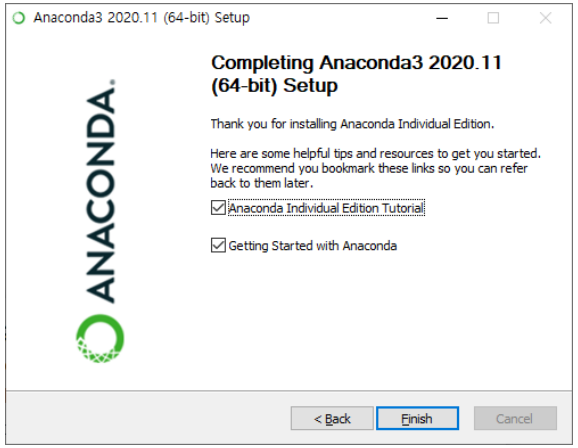

Anaconda3 2020.11 (64-bit) Setup
ANACONDA.
Choose Install Location
Choose the folder in which to install Anaconda3 2020.11 (64-bit).
Setup will install Anaconda3 2020.11 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:₩Users₩Gilah₩anaconda3
Browse...
Space required: 2.7GB
Space available: 104.8GB
Anaconda, Inc.
< Back
Next >
Cancel

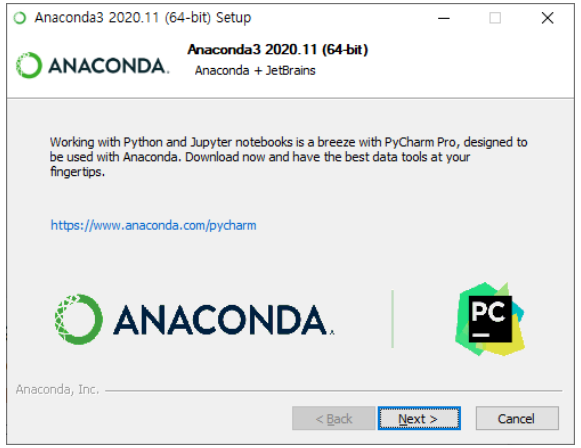

Anaconda3 2020.11 (64-bit) Setup
ANACONDA.
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda3 to my PATH environment variable
Not recommended. Instead, open Anaconda3 with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda3 as my default Python 3.8
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 3.8 on the system.
Anaconda, Inc.
< Back
Install
Cancel

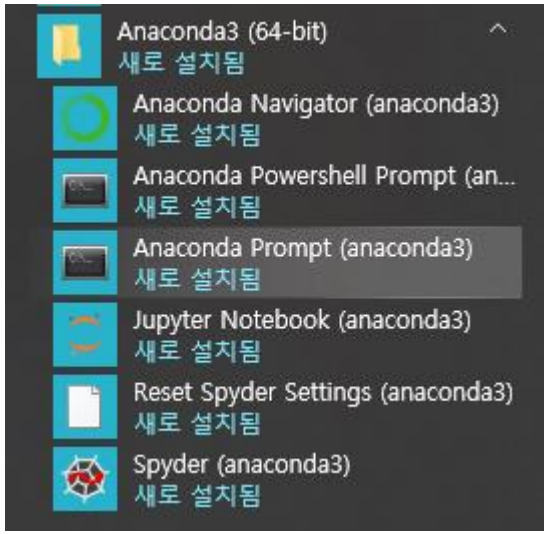

Anaconda3 2020.11 (64-bit) Setup
ANACONDA.
Installing
Please wait while Anaconda3 2020.11 (64-bit) is being installed.
Extract: qt-5.9.7-vc14h73c81de_0.conda
Show details
Anaconda, Inc.
< Back
Next >
Cancel

▣ 개발 환경 구축

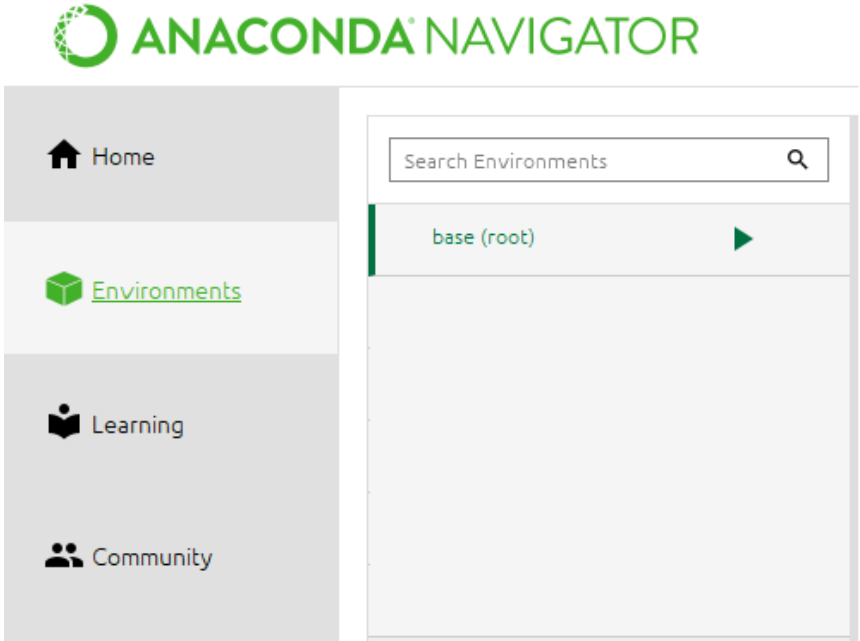
- 설치 완료





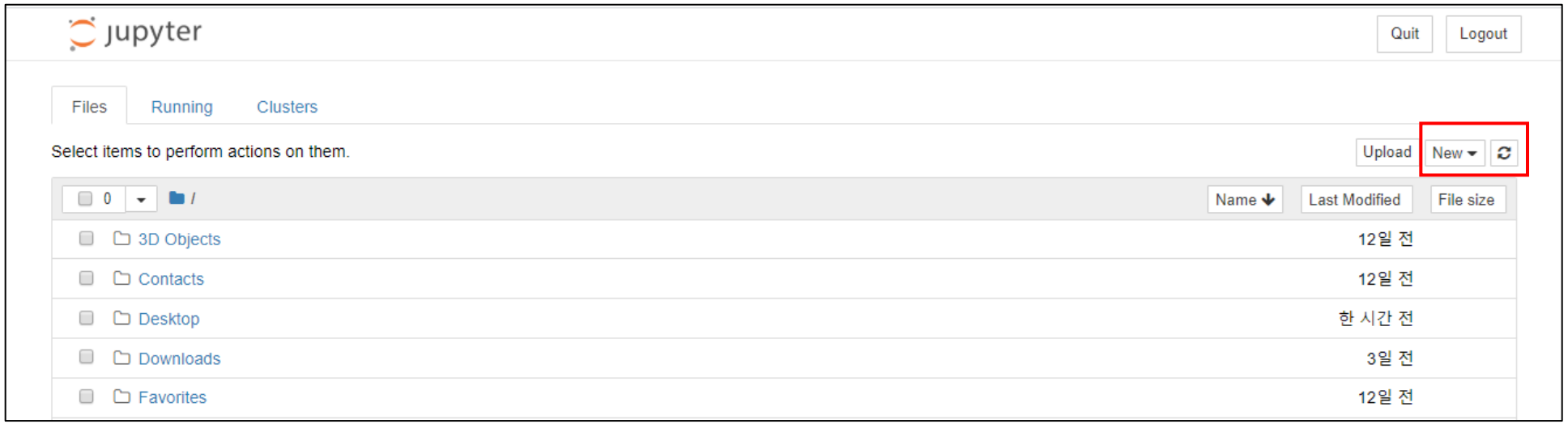


- Jupyter Notebook 환경 확인
  - [시작]-[Anaconda3]-[Anaconda Navigator] 클릭
    → Environments
    → base(root) ▶ 클릭
    → “Open with Jupyter Notebook” 또는
  - [시작]-[Anaconda3]-[Jupyter Notebook] 클릭

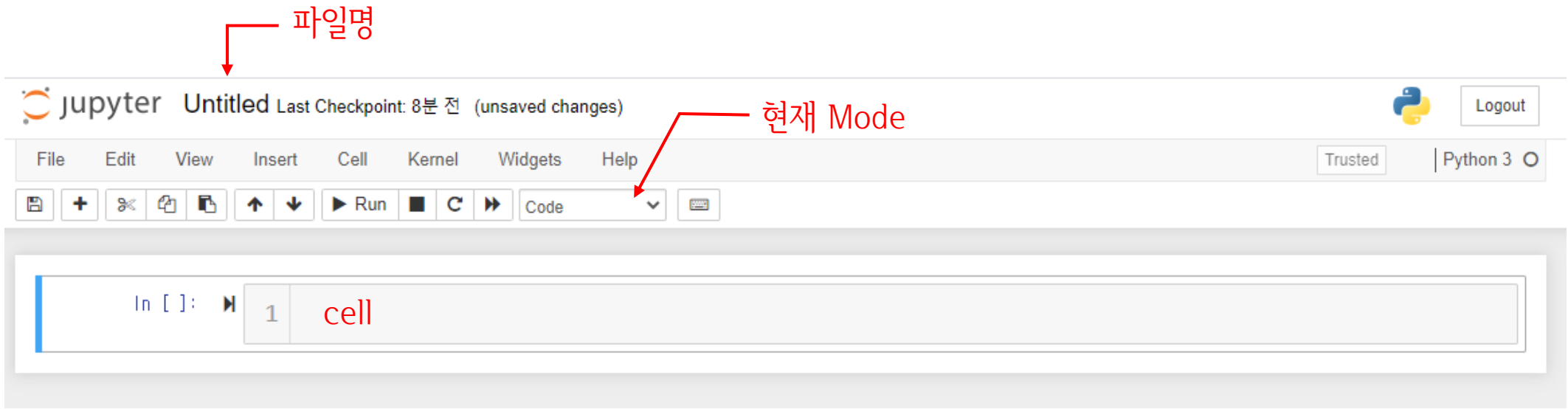


▣ 파일 생성

- **파일의 생성**
  - Jupyter Notebook을 실행 시킨 후 아래의 화면에서 우 상단의 “New 버튼 ⇨ Python 3”을 클릭하면 새 Notebook 파일이 생성된다.
  - Jupyter Notebook의 Root 디렉토리는 “C:₩Users₩사용자명”
  - Jupyter Notebook에서 생성된 파일의 확장명은 *.ipynb이다.



▣ Jupyter notebook 사용



Command	기능	Command	기능
a	현재 셀 위에 셀 추가	shift + tab	현재 입력 중인 함수의 docstring
b	현재 셀 아래에 셀 추가	명령? (ctrl + enter)	입력한 명령의 docstring
dd	현재 셀 삭제	ctrl + /	주석 on / off
m / y	mark down 모드 / code 모드		
shift + enter	현재 셀 실행 후 아래 셀로 이동		
ctrl + enter	현재 셀 실행		

▣ 파이썬 기본 문법 (변수)

- **파이썬에서 변수의 작성**
  - 변수 선언 시 자료형을 사용할 필요 없다.
  - 값에 의해 변수의 타입이 결정
  - 레퍼런스 변수만 존재

- **변수명**
  - 소문자, 대문자, 숫자, 언더스코어(_) 가능
  - 숫자로 시작할 수 없다.
  - 예약어를 변수로 사용할 수 없다.
  - 유니코드를 지원한다.

▣ 예약어 (Reserved Word)

- **예약어란?**
  - 개발을 할 때 파이썬 인터프리터 내에서 이미 특정 용도로 사용하기 위해 지정된 단어를 예약어라고 한다.
  - 예약어는 변수나 함수명 등 identifier로 사용할 수 없다.

- **예약어의 종류**

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	

▣ 표현식과 명령문

- **표현식(Expression)**
  - 값, 변수, 연산자의 조합. 값은 그 자체로 표현식이다.
  - 예) 42, n, n+25 등

- **문장(Statement)**
  - 프로그램에 영향을 주는 코드의 단위 변수를 생성하거나 값을 표시.
  - 문장을 입력하면 인터프리터는 문장을 실행한다.

    예) n = 17
        print(n) 등

▣ 주석 (Comment)

- **주석이란?**
  - 개발을 할 때 개발 코드에 설명문을 달아 놓은 것을 주석이라고 하는데, 주석을 적절하게 사용하는 것은 코드를 파악하는데 도움이 된다.

- **파이썬 주석**
  - 한 줄 주석 : #
  - 파이썬에는 여러 줄 주석이 없으며, # 이후에 입력된 문장을 주석으로 처리한다.
  - 여러 줄 주석이 필요할 때 ''' 로 하는 경우는 있으나 주석은 아님 (여러 줄 문장)

- **Mark Down을 이용한 설명문 작성**
  - Jupyter Notebook으로 프로그래밍을 할 때 보다 Visual 한 설명문을 넣고자 할 때 Mark Down을 사용할 수 있다.
  - Mark Down은 Github 등과 같은 버전관리 툴에서도 많이 사용된다.

▣ 자료형과 literal

- **자료형 기본**
  - 모든 데이터는 객체이다.
  - 객체의 타입을 바꿀 수 없다. (강형 타입)
  - 라인유지 : 문자열에서 ₩ 를 이용하면 연산식이나 표현식을 여러 줄에 나눠서 쓸 수 있다.

```
type(데이터)  # 특정 데이터의 타입 확인하는 함수
```

- **자료형의 종류**

① Boolean

- 참(True)과 거짓(False)을 나타내는 자료형
- Boolean 자료형 상수 : True / False

② 정수

- 소수점이 없는 수 : 42, 1000000, 123, -123, 05(숫자 앞에 0 올 수 없다.)
- 10진수 : 일반적인 수 (113)
- 2진수 : 0b or 0B 가 붙은 수(0b10)
- 8진수 : 0o or 0O가 붙은 수 (0o10)
- 16진수 : 0x or 0X가 붙은 수 (0x10)

▣ 자료형 con't

③ 실수

- 소수점이 있는 수 (3.14158 , 1.0e8)

④ 복소수

- 제곱해서 음수가 되는 수 (실수부 + 허수부j)
- 실수부 : 복소수.real
- 허수부 : 복소수.imag
- 켤레 복소수 : 복소수.conjugate()

⑤ 문자열 : 4가지 방법으로 작성

- 큰 따옴표 : "문자열"
- 작은 따옴표 : '문자열'
- 큰 따옴표 세 개 연속 """문자열"""
- 작은 따옴표 세 개 연속 : '''문자열'''

▣ 자료형 con’t

- **Boolean 역할을 하는 데이터의 종류**

값	True / False
"python"	True
1	True
[1, 2, 3]	True
"", ''	False
[]	False
()	False
{}	False
set()	False
0	False
0.0	False
null	False

▣ 연산자

연산자	설명	예	결과
+	더하기 문자열 결합 (Python은 문자열과 숫자를 연결할 수 없다)	5 + 8 'a'+ 'b' 'a' + 1	13 'ab' 오류
-	빼기	90 - 10	80
*	곱하기 문자열 연결 반복	4 * 7 'la' * 3	28 'lalala'
/	부동소수점 나누기	7 / 2 13/3	3.5 4.333333333333333
//	정수나누기(소수점 이하 버림) 나눗셈의 결과값보다 작은 정수 중, 가장 큰 정수를 리턴	7 // 2	3
%	나머지	7 % 3 -25.5 % 2.25	1 1.5
**	지수(거듭제곱)	3 ** 4	81
<< >>	비트 이동 연산	2 << 2 11 >> 1	8 5

▣ 연산자 con't

연산자	설명	예	결과
&	비트 and 연산	5 & 3	1
|	비트 or 연산	5 | 3	7
^	비트 XOR 연산자	5 ^ 3	6
not	True라면, False, False라면 True 반환	x = True not x	False
~	비트 반전 연산자 숫자 x의 비트 반전 연산 값 -(x+1)을 반환	~5	-6
and	논리 AND (Short Circuit : 좌측 연산이 False이면 우측 연산 안함) 3 >= 3 > 2 ⇒ 3 >= 3 and 3 > 2 위와 같은 비교연산은 and 연산을 축약해서 사용하는 것이다.	x = False y = True x and y	False
or	논리 OR (Short Circuit : 좌측 연산이 True이면 우측 연산 안함)	x = False y = True x or y	True
=	대입연산자	name = '홍길동'	

▣ 연산자 con't

연산자	설명	예	결과
*= += /=	연산자	a = 3 a *= 2	a = 6
== !=	같다. 같지 않다.	a = 10 b = '10' a == b	False
< >	작다 크다	3 >= 3	True
<= >=	작거나 같다, 크거나 같다	3 <= 3 3 >= 3 > 2	True True
in ...	특정 데이터가 집합 내에 있는지 여부 판단 값과 타입이 맞아야 True 리턴 boolean이나 문자열에서 사용하며, 숫자에는 사용하지 않는다.	'hello' in 'Hello everyone'	False
not in ...	특정 데이터가 집합 내에 없는지 여부 판단	'hello' not in 'Hello world'	True
is	변수의 레퍼런스 값 비교. 레퍼런스의 값이 같으면 True	a = 10; b = 10 a is b	True
is not	변수의 레퍼런스 값 비교 레퍼런스의 값이 같으면 False	a = 10; b = 10 a is not b	False

▣ 자료형 변환

- 데이터의 자료를 변환하려면 파이썬에서 제공하는 내장함수를 사용해야 한다.

함수명	설명	예	결과
float(x)	숫자 또는 문자열 *x* 로 부터 실수를 만들어 리턴	a = float('42.15') b = float('62')	a = 42.15 b = 62.0
int(x)	숫자 나 문자열 *x* 로 부터 만들어진 정수 객체를 리턴. 인자가 주어지지 않으면 0 리턴	a = int('62') b = int('42.15')	a = 62 ValueError
hex(x)	정수를 "0x" 접두사가 붙은 소문자 16진수 문자열로 변환	hex(27)	'0x1b'
oct(x)	정수를 "0o"로 시작하는 8진수 문자열로 변환	oct(27)	'0o33'
str(x)	숫자 형태의 자료를 일반 문자열로 변환	str(27)	'27'
bool(x)	x값을 판정하여 결과에 따라 True, 아니면 False 리턴	bool([]) bool("Korea") bool(0) bool(42.195)	False True False True

▣ 표준 입출력 (Input / Output)

- **표준 입력**
  - 입력 받은 모든 데이터는 문자열.
  - 형식

```
input()
input(prompt='문자열')   # 입력 받을 때 안내되는 메시지 문자열
```

- **표준 출력**
  - 형식

```
print(value, ...,  sep=' ', end='₩n', file=sys.stdout, flush=False)
```

  - sep : 값을 화면에 출력할 때 출력할 데이터들 사이를 표현하는 문자열. Default는 공백
  - end : 마지막 출력 데이터(value)를 화면에 출력한 후 마지막에 추가되는 문자열, Default는 new line
  - file : value를 출력할 객체. 기본값은 현재 sys.stdout (모니터)
  - flush : 스트림을 강제로 플러시 할지 여부 Default 는 False

▣ 제어문 (if)

- **if문**
  - 조건문, 다른 언어와 유사
  - 수행할 문장은 반드시 들여쓰기
  - 들여쓰기는 언제나 같은 깊이로 해야 한다.
  - 탭, 스페이스 2가지를 혼용해서 쓰면 안됨. 최근에는 공백 4개
  - if 조건문, while, for문 끝에 콜론(:)을 사용

  - 조건문에 비교 연산식 대신 변수를 바로 쓸 수도 있으며, 변수 자체가 논리식이 되는데 이때 각 변수의 논리값은 다음과 같이 결정된다.

타입	참	거짓
숫자	0이 아닌 숫자	0
문자열	비어 있지 않은 상태	“”
리스트, 튜플, 딕셔너리	비어 있지 않은 상태	빈 상태

▣ 제어문 (if) con't

- **if문의 형식**

```
if 조건문 :
    수행할 문장1
    수행할 문장2
    ...
else :
    수행할 문장A
    수행할 문장B
    ...
```

```
if 조건문1 :
    수행할 문장1
    수행할 문장2
    ...
elif 조건문2 :
    수행할 문장A
    수행할 문장B
    ...
else :
    수행할 문장A
    수행할 문장B
    ...
```

▣ 제어문 (if)

- **들여쓰기 오류의 예**

오류 예 1 : 들여쓰기 오류

```
if 조건문 :
    수행할 문장1
수행할 문장2
    수행할 문장3
```

오류 예 2 : 들여쓰기 오류

```
if 조건문 :
    수행할 문장1
    수행할 문장2
        수행할 문장3
```

▣ 제어문과 반복문

- **while / break / continue**
  - Java 언어의 문법과 동일하다.
  - while else 문이 있어, 조건식에 거짓인 경우 하나의 문장으로 처리가능

- **while else문**

```
n = 10; sum = 0; i = 1

while i <= n:
    sum += i
    i += 1    # update counter
print("The sum is", sum)

# 코드의 결과는 위와 동일
n = 10; sum = 0; i = 1

while i <= n:
    sum += i
    i += 1    # update counter
else : print("The sum is", sum)
```

▣ 반복문 (for)

- **for / break / continue**
  - range() 함수와 함께 사용하여 지정된 횟수 만큼 반복하거나 문자열이나 리스트와 같은 자료형을 순회할 때 사용.

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2 ...

for 변수 in range(start, end, step) :
    수행할 문장1
    수행할 문장2 ...
```

```
test_list = ['one', 'two', 'three']

for i in test_list:
    print(i)
```

```
a = [(1,2), (3,4), (5,6)]

for first, last in a :
    print(first + last)
```

▣ 반복문 (for) con't

- **for문과 range() 함수의 사용**
  - range는 지정한 start, end 사이의 숫자를 step의 간격으로 발생시킨다.
  - range의 결과는 start부터 end 바로 앞 숫자까지 컬렉션을 만든다.
  - start와 step은 생략 가능하며, start를 생략하면 0부터, step을 생략하면 1씩 증가된다.

```
range(start, end, step)                # start~end까지 step 간격으로 데이터 발생
```

```
sum = 0
for i in range(1, 11):
        sum = sum + i
        print(sum)
```

- **for와 list (파이썬스럽지 못한 코드)**

```
a = [1,2,3,4]
result = []
for num in a:
    result.append(num*3)

print(result)
```

▣ 반복문 (for) con't

- **for ~ else문**
  - while else문과 동일하며 조건에서 벗어났을 때 실행할 문장을 하나로 처리할 수 있다.

```
sum = 0
for i in range (1, 11) :
    sum += i

print("The sum is", sum)

# 코드의 결과는 위와 동일
sum = 0

for i in range (1, 11) :
    sum += i
else : print("The sum is", sum)
```

▣ 반복문 (for) con't

- **for와 list (파이썬다운 코드) - List Comprehension**

```
a = [1,2,3,4]
result = [num * 3 for num in a]
print(result)

a = [1,2,3,4]
result = [num * 3 for num in a if num % 2 == 0]
print(result)

result = [x*y for x in range(2,10) for y in range(1,10)]
print(result)
```

▣ 문자열

- **문자열의 특징**
  - 파이썬 3은 Unicode 지원한다.
  - 문자열은 immutable한 데이터이다.
  - '문자열', "문자열", '''문자열''', """문자열"""
  - Single quotation(''') , double quotation(""")을 3개씩 문장의 앞 뒤로 감싸면 여러 줄로 구성된 문자열을 만들 수 있다.

- **Multiline 처리**

  ① '''문자열''', """문자열"""

```
mystr = """Hello, it's me
I was wondering if after all these years you'd like to meet
To go over everything
They say that time's supposed to heal ya
But I ain't done much healing"""
```

- **문자열 관련 연산자 : + ***
  - 문자열 연결 : + (문자열과 숫자 연결 불가)
  - 문자열 반복 : *

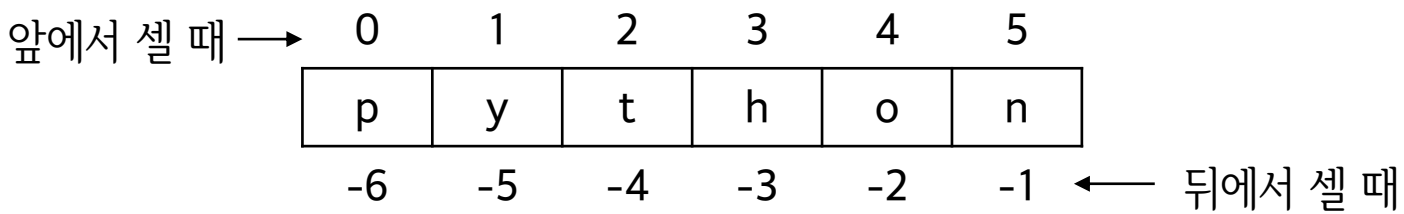
▣ Escape Sequence

- **Escape Sequence**
  - 문자 앞에 ₩를 붙여 언어에서 정의되어 있는 원래의 의미를 벗어나는 문자들을 말한다.

이스케이프 시퀀스	의미
₩₩	역 슬래시 (₩)
₩'	작은따옴표 (')
₩"	큰따옴표 (")
₩a	ASCII 벨 (BEL)
₩b	ASCII 백스페이스 (BS)
₩f	ASCII 폼 피드 (FF)
₩n	ASCII 라인 피드 (LF)
₩r	ASCII 캐리지 리턴 (CR)
₩t	ASCII 가로 탭 (TAB)

▣ 문자열 연산

- **문자열 분리**
  - 문자열을 구성하는 개별 문자를 읽을 때는 [] 괄호와 문자의 위치인 첨자를 적는다.
  - 문자열의 길이는 len() 함수를 통해 얻는다.



▣ 문자열 연산

- **추출(슬라이싱)**
  - 파이썬 문자열은 자바의 배열처럼 동작
  - 음수 사용 가능

표현	의미
[:]	처음부터 끝까지
[start:]	start offset부터 끝까지
[:end]	처음부터 end-1까지
[start:end]	start offset부터 end-1까지
[start:end:step]	step만큼 문자를 건너뛰면서, start 오프셋부터 (end-1) 오프셋까지 시퀀스를 추출

▣ 문자열 관련 함수

함수명	설명
capitalize()	첫 번째 단어를 대문자로
lower()	모든 글자를 소문자로
title()	모든 단어의 첫 글자를 대문자로 변환
upper()	모든 글자를 대문자로
swapcase()	대문자는 소문자로, 소문자는 대문자로 바꿈
count(문자열)	해당 문자열이 몇 개 있는지 출력
len(문자열)	문자열의 길이를 센다
endswidth()	지정된 문자열로 끝나는지 여부 출력(True, False)
startswith(문자열)	지정된 문자열로 시작하는지 여부 출력(True, False)
find(문자열)	지정된 문자열의 오프셋 값 리턴
format()	출력 시 포맷을 지정할 수 있다.

▣ 문자열 관련 함수

함수명	설명
구분문자.join(list)	문자열 리스트를 하나의 문자열로 결합
replace(원본문자열, 바꿀문자열 [, 바꿀횟수])	문자열을 지정한 문자열로 바꿈
rfind(문자열)	지정 문자열을 오른쪽에서 찾아서 출력
center(숫자)	문자열을 지정한 공간에서 중앙에 배치
ljust(숫자)	문자열을 지정한 공간에서 왼쪽에 배치
rjust(숫자)	문자열을 지정한 공간에서 오른쪽에 배치
split(구분자)	구분자를 기준으로 하나의 문자열을 리스트로 나눔 구분자 없이 사용하면 기본 구분자 (공백, 줄나눔, 탭)를 사용
str(데이터)	괄호 안의 모든 데이터를 문자열로 바꾼다.
strip(문자열)	문자열의 맨 앞과 맨 뒤에 지정 문자열을 삭제
rstrip(문자열)	문자열의 맨 뒤에 White Space 삭제할 때 주로 사용
lstrip(문자열)	문자열의 맨 앞에 White Space 삭제할 때 주로 사용

▣ 문자열 대입

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

▣ 문자열 대입

- **문자열 자리배치**

```
%[-]폭[.유효자리수]서식
```

  - 별도의 폭 지정이 없으면 변수의 자리수만큼 차지하지만 폭을 지정하면 최소 지정된 만큼 폭을 확보한다.

```
value = 123
print("###%d###" % value)
print("###%5d###" % value)
print("###%10d###" % value)
print("###%1d###" % value)
```

```
###123###
###   123###
###         123###
###123###
```

  - 서식이 지정하는 폭은 강제 폭이 아니라 최소 폭이어서 자리수보다 작은 폭을 주어도 최소한 숫자의 자리수만큼은 차지한다.
  - 폭에 -를 지정하면 왼쪽으로 정렬한다.

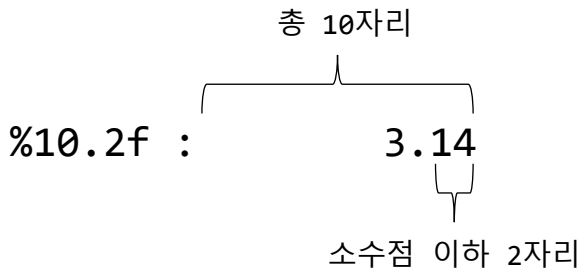
▣ 문자열 대입

- **문자열 자리배치**

%[-]폭[.유효자리수]서식

- 별도의 폭 지정이 없으면 변수의 자리수만큼 차지하지만 폭을 지정하면 최소 지정된 만큼 폭을 확보한다.

```
pie = 3.14159265
print("%10f" % pie)
print("%10.8f" % pie)
print("%10.5f" % pie)
print("%10.2f" % pie)
```




```
  3.141593
3.14159265
   3.14159
      3.14
```

- 실수는 폭 지정 외에 . 기호와 함께 실수 부분의 유효 자리수를 지정할 수 있다.
- 유효자리수 지정이 없으면 소수점 이하 6자리까지 반올림하여 표시한다.

▣ 문자열 대입

- **포맷팅**
  - 문자열 내에 {} 괄호를 입력하고 format 함수의 인수로 삽입할 변수 또는 값을 입력하면 {} 괄호 자리에 차례대로 인수 값이 전달된다.

```
name = '한결'
age = 16
height = 162.5
print('이름:{}, 나이:{}, 키:{}'.format(name, age, height))
```

```
이름:한결, 나이:16, 키:162.5
```

  - {} 괄호 안에 0부터 시작하는 순서 값을 지정할 수도 있다.

```
'이름:{2}, 나이:{0}, 키:{1}'.format(age, height, name)
```

▣ List

- **List 특징**
  - 배열과 같은 자료(시퀀스 구조, 즉 순서가 있다.), 수정 가능하다. 가변적이다.
  - 리스트의 요소는 일반적으로 같은 타입이지만 다른 타입을 섞어서 저장하는 것도 가능하다 (일반적이지는 않음)
  - 첨자로 음수 가능
  - + 연산자를 이용하면 두 개의 서로 다른 리스트를 붙일 수 있다.
  - [] 나 list()를 이용해 생성

- **List slicing**
  - 문자열과 동일하게 슬라이싱 가능

- **Deep Copy**
  - [:] 또는 copy() Module 이용하여 Deep copy 가능

- **List 연산**

  \+ : 리스트 연결

  * : 리스트 반복

▣ List 관련 함수

- **List 관련 함수**

함수명	설명
append()	리스트의 뒤에 데이터 추가하기
sort()	리스트를 정렬, 역순으로 정렬하기 위해서는 reverse=True 옵션
reverse()	리스트 요소순서를 뒤집는다.
index()	특정 요소의 위치 값 반환
insert(위치, 데이터)	지정된 위치에 데이터를 삽입한다.
pop()	마지막 요소 리턴 후 삭제
count(요소값)	특정 요소의 개수 세기
extend(리스트)	원래의 리스트를 매개변수로 전달한 리스트와 합쳐서 확장
len(리스트)	리스트 요소의 개수를 구함

▣ Dictionary

- **개요**
  - 파이썬의 가장 강력한 데이터 컬렉션
  - 파이썬에서 빠르게 데이터베이스 같은 연산을 가능하게 함
  - 다른 언어에서는 다른 이름으로 불림
    - Associative Arrays (연관 배열) - Perl / PHP
    - Properties or Map or HashMap(속성, 맵, 해쉬맵) - Java
    - Property Bag (속성 가방) - C# / .Net

▣ Dictionary

- **특징**
  - {} 기호 혹은 dict() 를 이용해 생성
  - key : value의 쌍으로 이루어짐. 변환되는 값은 Key로 사용 불가(예 : list)
  - Key를 통해 Value를 얻는다
  - {Key1:Value1, Key2:Value2, Key3:Value3 ...}

```
dic = {'name':'Gildong', 'phone':'0119993323', 'birth': '1118’}
```

  - 키 값이 반드시 문자열일 필요 없으며 변경 불가능한 데이터만 키가 될 수 있다.
    (immutable 데이터인 문자열, 튜플, 숫자 값은 키가 될 수 있으나, **리스트는 키가 될 수 없다.)**

  - 사전변수명[키](ex: dic[‘address’]) 형식으로 찾을 경우 찾는 키가 없으면 예외를 발생한다.
  - get 함수는 키가 없을 때 예외를 발생하는 대신 None을 리턴한다.

▣ Dictionary

- **Dictionary 관련 함수**

함수명	설명
min()	키 값 중에서 최솟값 (사전적 순서)
max()	키 값 중에서 최대값 (사전적 순서)
get(키 [, 값])	특정 키를 통해 값을 얻음. 해당 키가 없으면 디폴값으로 대체
keys()	키 값을 리스트로 얻어옴
values()	밸류값을 리스트로 얻어옴
items()	키와 밸류의 쌍을 튜플로 얻어옴
sorted(iterable, *, key=None, reverse=False)	내장함수, 딕셔너리의 키를 오름차순 정렬

▣ Tuple

- **tuple의 특징**
  - tuple과 list는 기본적으로 동일
  - list는 [과]으로 둘러싸지만 tuple은 (과)으로 둘러싼 값이며 immutable 데이터이다.
  - list는 그 값의 생성, 삭제, 수정이 가능하지만 tuple은 그 값을 바꿀 수 없다.
    즉, 리스트의 항목 값은 변화가 가능하고 tuple의 항목값은 **변화가 불가능하다.**
  - tuple을 사용하는 이유는 내부 구조가 단순하고 읽는 속도가 빠르다. 즉, 가볍고, 빠르고, 안전한 이유로 tuple을 사용한다.

```
# 튜플의 생성
t1 = ()
t2 = (1,)               # 요소가 1개일 경우 끝에 ,
t3 = (1, 2, 3)
t4 = 1, 2, 3            # 값만 나열해도 튜플 생성 가능

t5 = ('a', 'b', ('ab', 'cd'))

t1 = (1, 2, 'a', 'b')
#del t1[0]              # 삭제 불가, 삭제 시 오류 발생
```

▣ Set(집합)

1) 개요

- {}, set, set()로 생성
- 집합(set)은 파이썬 2.3부터 지원되기 시작한 자료형
- set()을 이용해 만듦
- 중복을 허용하지 않는다.
- 순서가 없다(Unordered)
- 인덱싱으로 값을 얻을 수 없다(순서가 없으므로) 리스트나 튜플로 변경 후 값 추출함
- 자료형의 중복을 제거하기 위해 사용
- 교집합(&), 차집합(-), 합집합(|) 연산 가능

```
coffee = set(['아메리카노', '카푸치노', '캬라멜 마키아또', '바닐라 라떼'])
print(coffee)

fruit = {'apple', 'pear', 'banana', 'orange', 'pineapple'}
print(fruit)
print(type(fruit))

coffee.add('아이스라떼')
coffee.add('아메리카노')  # 동일한 데이터가 있으므로 삽입되지 않음

print(coffee)
```

▣ Set(집합)

1) Set 관련 함수

함수명	설명
add()	요소 추가
update()	요소 수정
remove()	요소 삭제
intersection()	교집합
difference()	차집합
union()	합집합
symmetric_difference	배타적 차집합 (^)

▣ Comprehension

- **Comprehension이란?**
  - 하나 이상의 iterator로부터 python의 자료구조를 만드는 방법

1) List comprehension
   - append() 이용
   - range()를 이용

```
[ 표현식 for 항목 in 순회 가능한 객체 ]

[ 표현식 for 항목 in 순회 가능한 객체 if 조건]
```

2) dictionary comprehension
   - List comprehension과 같이 if, for … 다중 절을 가질 수 있다.

```
{키표현식 : 값표현식 for 표현식 in 순회가능한 객체}
```

3) set comprehension

```
{ 표현식 for 표현식 in 순회가능한 객체 }
```

▣ iterator 함수

- **zip()을 이용한 순회**
  - 여러 시퀀스를 병렬로 순회하는 함수
  - 여러 시퀀스 중 가장 짧은 시퀀스가 완료되면 멈춘다.
  - 두 개 이상의 list, 튜플의 값을 **병렬적**으로 추출할 때 사용.
    추출된 데이터는 문자열, 리스트, 튜플 등으로 추출 가능

- **enumerate()을 이용한 순회**
  - 순서 값과 요소 값 둘을 한꺼번에 구해주는 내장함수
  - 리스트의 순서 값과 요소 값을 튜플로 묶은 컬렉션을 리턴한다.

▣ 함수

- **함수란?**
  - 이름이 있는 코드 조각
  - 매개변수와 반환 값이 있다.
  - 정의하고 난 후 호출을 한다.
  - 내장함수 (print() 등...) 와 외장함수가 있다.

1) 함수 정의

```
def 함수명() :                  # 매개변수가 없는 함수
        [ return 코드 ]

def 함수명(a, b) :              # 매개변수가 있는 함수
        [ return 코드 ]
```

2) 함수 호출

```
함수명()
결과값을 받을 변수 = 함수명(입력 인수 1, 입력 인수 2, ...)
```

```python
# 함수정의(매개변수가 없는 함수)
def something() :
  print("hahaha")

x = something()
Print(x)

# 매개변수가 있는 함수
def mymax(a, b) :
  if a > b :
    print(a,  ' 가 크다 ' )
  elif a < b :
    print(b,  ' 가 크다 ' )
  else:
    print( ' 두 값이 같다. ' )

mymax(1, 2)
mymax(4, 3)
mymax(5, 5)
```

▣ 함수(매개변수와 인수)

- **매개변수와 인수**
  - 매개변수(parameter) : 함수에 입력으로 전달된 값을 받는 변수를 의미
  - 인수(arguments) : 함수를 호출할 때 전달하는 입력 값을 의미한다.

- **인자를 지정한 함수의 호출**
  - Python은 매개변수의 수만큼 전달해줘야 하는 것이 기본이지만, 가변인자 사용도 가능
  - 매개변수에 초기값을 설정할 수 있다.

- **위치 인자를 이용한 호출(Keyword argument)**
  - 호출할 때 매개 변수명과 값을 지정하여 호출하는 방식
  - 위치 인수와 키워드 인수를 섞어서 호출할 수 있으며, 이때 위치 인수를 먼저 적고 뒤쪽에 키워드 인수를 적는다.
  - ex)

```
def calcstep(begin, end, step):
    ...


calcstep(end=3, 5, 1) # error
```

▣ 함수 초기값 설정

- **초기값(default parameter)의 설정**
  - 함수 정의부의 매개변수에 초기값을 설정하면, 해당 매개 변수의 값이 전달되지 않았을 경우 지정된 초기 값으로 세팅된다.
  - (C++의 default parameter와 유사)
  - default parameter 값은 오른쪽에서부터 채워야 한다 (즉, 기본값을 가지는 인수 뒤에 일반 인수가 올 수 없다.)

    ex) def calcstep(begin, step=1, end)

- **None**
  - 아무것도 없다는 것을 뜻하는 상수 값.
  - False와는 다르다

▣ 변수의 scope

- **함수는 정의된 위치에 따라 지역과 전역으로 나눈다.**

  ① 지역변수 (local): 함수 내부에 선언되는 변수

  ② 전역변수 (global) : 함수 밖에 선언되는 변수

  - 함수 내부에서 초기화하는 변수는 전역변수와 이름이 같더라도 지역변수로 간주된다.

```
price = 1000              # 전역변수

print(x)

def sales() :
  price = 500             # sales 함수 안에서만 쓰이는 지역변수
```

▣ *arg와 **kwargs 매개변수

- ***arg와 **kwargs 매개변수의 사용**
  - *args는 전달된 데이터를 튜플로 묶는다.
  - **kwargs : 가변 인자를 dict() 타입으로 받음 (위치 인수를 넘기면 에러 발생)
  - 위치 인수와 키워드 인수를 동시에 가변으로 취할 수도 있는데, 이때는 위치 인수가 먼저 오고 키워드 인수가 뒤에 온다.

▣ Filter, Map, Reduce

- **Filter, Map, Reduce**
  - filter()　　: 리스트의 요소 중 조건에 맞는 것만 골라 list로 리턴
  - map()　　: 리스트를 함수에 의해 처리 후 그 결과를 list로 리턴
  - reduce()　　: 리스트를 함수로 처리 후 그 결과를 단일의 값으로 리턴
  - Sequence 자료형 각 element에 동일한 function을 적용함. 빅데이터에서 매우 중요

```
① map(function_name, list_data)

② reduce(function_name, list_data)
```

▣ Lambda

- **Lambda 함수?**
  - 단일문으로 표현되는 익명함수
  - Lambda, map, reduce는 간단한 코드로 다양한 기능을 제공
  - 그러나 코드의 직관성이 떨어져서 lambda나 reduce는 pyton3에서 사용을 권장하지 않음
  - 파이썬 3부터는 권장하지는 않으나 Legacy library나 다양한 머신러닝 코드에서 여전히 많이 쓰임
  - 작은 함수를 정의하고 이들을 호출해서 얻은 모든 결과값을 저장해야 하는 경우에 유용
  - 콜백 함수를 정의하는 GUI에서 람다를 사용할 수 있다.

    lamda 매개변수 : 리턴할 수식

▣ 값으로서의 함수

- **Python에서는 함수를 값으로 처리한다.**
  - 함수를 다른 변수에 대입할 수 있다

```
def add(a, b):
    print(a+b)
plus = add
plus(1, 2)
```

```
def calc(op, a, b):
    op(a, b)

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

def multi(a, b):
    print(a*b)

calc(add, 1, 2)
calc(multi, 3, 4)
```

▣ 내부 함수와 Closure

- **내부함수**
  - 함수 안에 정의된 다른 함수
  - 코드 중복을 피하기 위해 주로 사용

```
def outer(a, b) :
    def inner(c, d) :
        return c + d
    return inner(a, b)

print(outer(4, 7))    # 출력결과?
```

▣ Closure

- **closure란?**
  - 함수 안에 정의된 다른 함수
  - 함수 내부의 반복되는 코드를 통합하기 위해 주로 사용
  - 내부함수는 외부에서 호출할 수 없다

```
def student(saying) :
    def inner(quote) :
        return "우리는 '%s' 학생들이다. " %quote
    return inner(saying)

k = student('Master')
print(k)   # 출력 결과는?

def student2(saying) :
    def inner2() :
        return "우리는 '%s' 학생들이다. " % saying  # 바깥 쪽 변수에 직접 접근
    return inner2                    # 내부 변수가 리턴

a = student2('SCIT 41기')            # a의 결과는?
b = student2('SWDO 7기')             # b의 결과는?
print(a())
print(b())
```

▣ docstring

- **docstring이란?**
  - 함수에 대한 설명문
  - help()와 __doc__ 를 이용해 출력할 수 있다.
  - help()는 자세히 출력, __doc__ 은 docstring만 출력

```
def echo(anything) :
    'echo returns its input argument'
    return anything

print(echo('I am'))
help(echo)
 # Help on function echo in module __main__:
# echo(anything)
#     echo returns its input argument

print(echo.__doc__)
# echo returns its input argument
```

docstring

```
def echo(anything) :
    'echo returns its input argument'
    return anything

print(echo('I am'))
help(echo)

# Help on function echo in module __main__:
#
# echo(anything)
#     echo returns its input argument

print(echo.__doc__)
# echo returns its input argument
```

▣ docstring

```
def print_if_true(thing, check) :
    '''
    Prints the first argument if a second argument is true,
    The operation is :
        1. Check whether the *second* argument is true.
        2. If it is, print the *first* argument.
    '''
    if check :
        print(thing)

help(print_if_true('abc', True))
print(print_if_true.__doc__)
```

▣ 표준 모듈

- **import**
  - 모듈을 파이썬 코드를 작성해 놓은 스크립트 파일이다.
  - 스크립트 파일 안에는 함수, 변수, 클래스 등이 정의되어 있다.
  - 외부의 모듈을 가져와 사용할 때는 import 명령을 사용한다.
  - 모듈에 포함된 함수를 호출할 때는 함수명 앞에 모듈명을 붙인다.

```
import math
print(math.sqrt(2))
```

  - 특정 함수만 임포트하고자 할 때는 다음 구문을 사용한다.

```
from 모듈 import 함수명
```

  - 모듈의 모든 함수를 불러올 때는 함수명 자리에 * 문자를 사용
  - 모듈의 이름이 길고 복잡할 때는 as 다음에 별칭을 지정할 수 있다.

```
import  math  as m
```

▣ 표준 모듈

- **math 모듈**
  - 수학 연산에 필요한 상수와 연산 함수를 제공

함수명	설명
sqrt(x)	x의 제곱근을 구한다.
pow(x, y)	x의 y승을 계산한다.
ceil(x)	올림 값을 구한다.
floor(x)	내림 값을 구한다.
fabs(x)	x의 절대값을 구한다.
trunc(x)	x의 소수점 이하를 버린다.

▣ 표준 모듈

- **time 모듈**
  - 날짜와 시간 관련 기능을 제공

```
import time

t = time.time()
print(t)
print(time.ctime(t))
```

- **calendar 모듈**
  - calendar 함수는 인수로 받는 년도의 달력 객체를 반환
  - month 함수는 년도와 달을 인수로 받은 해당 월의 달력 객체를 반환

```
import calendar

print(calendar.calendar(2021))
print(calendar.month(2021,8))
```

▣ 표준 모듈

- **random 모듈**
  - 난수를 생성하는 기능을 제공

함수명	설명
random()	0에서 1 미만의 실수 하나를 생성
randint(begin, end)	begin~end 사이의 정수 난수 하나를 생성(end포함)
randrange(begin, end)	begin~end 사이의 정수 난수 하나를 생성(end제외)
uniform(begin, end)	begein~end 사이의 실수 난수 하나를 생성
choice(list 객체)	리스트에서 임의의 요소를 하나 골라 리턴
shuffle(list 객체)	리스트의 요소를 무작위로 섞는다
sample(list 객체, n)	리스트 항목 중 n개를 무작위로 뽑아 새로운 리스트를 만든다.

▣ 표준 모듈

- **sys 모듈**
  - 파이썬 해석기가 실행되는 환경과 해석기의 여러 가지 기능을 조회하고 관리하는 모듈
  - sys.exit(0) : 프로그램 강제 종료

- **명령행 인수**

```
#file name: sysarg.py
import sys

print(sys.argv)
```

```
> sysarg.py korea option
> ['c:\\python test\\sayarg.py', 'korea, 'option']
```

  - argv[0]에 실행파일 전체 경로가 들어가고 이 후 인수는 argv[1], argv[2]로 전달된다.

▣ I/O (Input / Output)

- **표준 입력**
  - 형식

```
input()
Input(문자열)
```

  - 입력 받은 모든 데이터는 문자열이다.

- **표준 출력**

```
print(출력데이터)
```

▣ File IO

- **File Open**

```
파일 객체 = open(파일 이름, 파일 열기 모드)
```

- **File Open Mode**

파일열기모드	설명
r (읽기모드)	파일을 읽기전용으로 open 할 때 사용. 모드를 생략할 경우 r 모드로 설정
w (쓰기모드)	파일에 내용을 쓸 때 사용 파일을 쓰기 모드로 열게 되면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고, 해당 파일이 존재하지 않으면 새로운 파일이 생성
a (추가모드)	파일의 마지막에 새로운 내용을 추가시킬 때 사용
t (문자열)	파일에 저장되는 데이터가 문자열임을 나타냄. (default)
b (binary)	binary data를 쓸 때 사용

▣ File IO

- **데이터 읽기**

①	readline()	: 한 줄 읽어서 리턴
②	readlines()	: 파일의 모든 라인을 읽어서 각각의 줄을 요소로 갖는 리스트로 리턴
③	read()	: 파일의 내용 전체를 문자열로 리턴

```
f = open("test.txt", mode="r", encoding="utf-8")
line = f.readline()
print(line)
f.close()                    # 사용이 완료되면 close()
########
f = open("test.txt", 'r', encoding='UTF-8')
while True:
    line = f.readline()
    if not line: break
    print(line , end='')
f.close()
```

▣ File IO

- **데이터 읽기**

①	readline()	: 한 줄 읽어서 리턴
②	readlines()	: 파일의 모든 라인을 읽어서 각각의 줄을 요소로 갖는 리스트로 리턴
③	read()	: 파일의 내용 전체를 문자열로 리턴

```
f = open("test.txt", mode="r", encoding="utf-8")
line = f.readline()
print(line)
f.close()                    # 사용이 완료되면 close()
########
f = open("test.txt", 'r', encoding='UTF-8')
while True:
    line = f.readline()
    if not line: break
    print(line , end='')
f.close()
```

▣ File IO

- **데이터 저장 / 추가**
  - 파일을 오픈할 때 mode를 ‘w’로 하면 저장, ‘a’로 하면 기존 내용에 추가

① write(저장할 데이터)

```
poem = '''
Programming is fun
When the work is done
if you wanna make your work also fun:
use Python!
'''
f = open('poem.txt', 'w')          # ‘a’로 오픈하면 추가모드로 파일 오픈
f.write(poem)                      # 문자열 변수 poem의 데이터를 파일로 출력
f.close()
```

▣ File IO

- **입출력 위치**
  - 현재 입출력 위치를 조사할 때는 tell 함수를 사용
  - 위치를 변경할 때는 seek 함수 사용
  - seek(위치, 기준) : 기준이 0이면 파일 처음부터, 2이면 끝에서부터, 1이면 현재 위치를 기준으로 한다

▣ 예외처리

- **예외란?**
  - 프로그램의 실행 도중에 만날 수 있는 오류들을 Exception이라고 한다.
  - 파이썬에서는 안정적인 프로그래밍 작성을 위해 try, except를 이용해서 오류처리를 할 수 있다.

```
# 방법 1
try:
    오류발생 코드
except:
    오류처리 코드

# 방법 2
try:
    오류발생 코드
except 발생 오류 :
    오류처리 코드

# 방법 3
try:
    오류발생 코드
except 발생 오류 as 오류 메시지 변수:
    오류처리 코드
```

```
# 방법 4
try:
    오류발생 코드
except 발생 오류 as 오류 메시지 변수:
    오류처리 코드
else :
    오류가 발생하지 않았을 때 코드

# 방법 5
try:
    오류발생 코드
except 발생 오류 as 오류 메시지 변수:
    오류처리 코드
finally :
    반드시 처리해야하는 코드
```

▣ Python 클래스

- **Python Class의 특징**
  - 멤버가 있으나 자바와는 다르게 멤버변수는 전부 public이다.

- **클래스의 선언과 생성**

```
class 클래스명:
    pass                                # 빈 클래스임을 나타냄

someone = 클래스명()
```

▣ Python 클래스

- **Python의 생성자**
  - 파이썬은 기본생성자도 작성해야 한다.
  - 두 개의 생성자를 가질 수 없다.
  - 생성자 작성 방법
    - __init__()은 클래스당 한 개만 작성가능
    - self : 클래스 내부에서 __init__() 함수의 첫 매개변수여야 한다.
      (키워드가 아니어서 다른 이름을 쓸 수 있지만 self 명칭을 쓰는 것이 관행이다.)

```
def __int__(self) :                 # 기본 생성자

def __int__(self, a, b) :           # 전달인자 2개 받는 생성자
```

```
# 객체 선언방법 1
class Person :
    def __init__(self):             # __init()__ 필수 x (기본생성자)
        pass
```

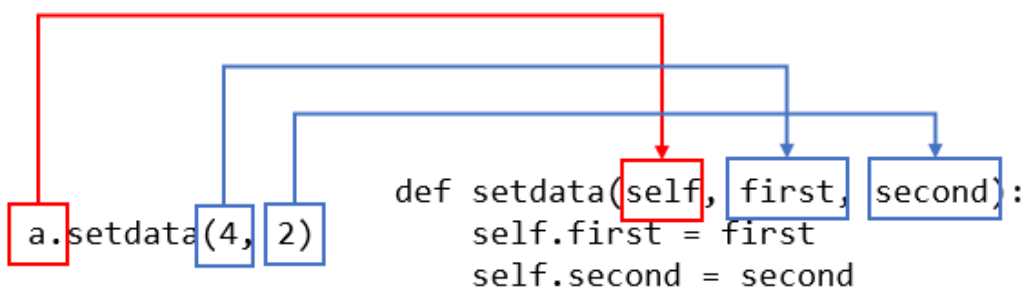
▣ Python 클래스

```
class FourCal :
    def setdata(self, first, second) :          # self변수는 객체명을 받는 용도이며, 생략 불가
        self.first  = first
        self.second = second

a = FourCal()                                   # 객체 생성
a.setdata(3, 4);                                # 첫 번째 호출 방법

b = FourCal()
FourCal.setdata(b, 5, 6)                        # 두 번째 호출 방법, b 생략 불가

print(a.first, a.second)
print(b.first, b.second)
```



▣ Python 클래스

- **상속**
  - 문법

```
class 자식클래스명(부모클래스명) :
```

```
class Car() :
    def exclaim(self) :
        print("I'm a car")

class Yogo(Car) :
    pass

car = Car()
print(car.exclaim())        # I'm a car 출력

yogo = Yogo()
print(yogo.exclaim())       # I'm a car 출력, 메서드 상속됨
```

▣ Python 클래스

```
class Person() :
    def __init__(self, name):
        self.name = name

class MDPerson(Person) :
    def __init__(self, name) :
        self.name = "Doctor " + name

class JDPerson(Person) :
    def __init__(self, name) :
        self.name = name +", Esquire"

person = Person('Fudd')
doctor = MDPerson('Fudd')
lawer  = JDPerson('Fudd')

print(person.name)      # Fudd 출력
print(doctor.name)      # Doctor Fudd 출력
print(lawer.name)       # Fudd, Esquire 출력
```

▣ Python 클래스

- **super()**
  - 부모클래스의 생성자 호출

```
class Person() :
    def __init__(self, name):
        self.name = name

class EmailPerson(Person) :
    def __init__(self, name, email) :
        super().__init__(name)  # 부모클래스의 __init__() 호출
        self.email = email

bob = EmailPerson( ' 박길동', 'bob@frapples.com')
print(bob.name)
print(bob.email)
```

▣ Python 클래스

- **self 키워드**
  - 객체 그 자체를 나타내는 키워드

- **접근지정자**
  - 파이썬은 멤버의 접근지정자 개념이 없다. (기본적으로 public)
  - __(언더스코어 2개) 는 private, _(언더스코어 1개) protected
    아무것도 없을 때에는 public이라고 약속했으나 크게 의미는 없음

- **setter / getter**
  - 파이썬에는 private 멤버의 개념이 없으나 멤버 이름을 감춘 것 같은 효과를 줄 수는 있다.
  - 프로퍼티를 이용하면 특정 값을 읽거나 할당할 때 항상 특정 메서드가 실행될 수 있도록 강제할 수 있다.

    ① property() 함수를 이용한 방법

    ② @ 이용한 방법
    getter 메서드 앞 : @property
    setter 메서드 앞 : @name.setter

▣ Python 클래스

- **instance 메서드 / class 메서드 / static 메서드**
  - 정적메소드 : 클래스에서 직접 접근할 수 있는 메서드
  - static method와 class method의 차이점은 상속
  - 파이썬에서는 다른 언어와는 다르게 정적 메소드임에도 불구하고 인스턴스에서도 접근이 가능.

- **특징**

① instance method : 첫 번째 인자로 self

② class method : 첫 번째 인자로 친

③ static method : 특별히 추가되는 인자가 없다.

▣ Python 클래스

- **추상 클래스**
  - 추상클래스란 구현되지 않은 추상메소드를 한 개 이상 가지며, 자식클래스에서 해당 추상 메소드를 반드시 구현하도록 강제
  - 상속받은 클래스는 추상메소드를 구현하지 않아도, import할 때까지 에러는 발생하지 않으나 객체를 생성할 시 에러가 발생한다.
  - 추상클래스를 만들기 위한 형식은 아래와 같다.
    ① 반드시 abc 모듈을 import 해야 한다.
    ② 추상메소드는 생략하면 기본적인 클래스 기능은 동작하지만, 추상메소드를 추가한 후에는 객체를 생성하면 에러가 발생한다.

- **추상클래스 작성 방법**

```
from abc import *
class 추상클래스명(metaclass=ABCMeta):

    @abstractmethod
    def 추상메소드(self):
        pass
```

The End!