

파이썬 데이터 분석

01. 환경 구축과 기본 문법

- 진행 기본 방식은 **1)개념 이해 2)코딩 따라하기 3)결과 확인** 입니다.
- **개념 이해** 단계에서는 스터디 진행자가 직접 개념에 대해 간단히 설명합니다.
- **코딩 따라하기** 단계에서는 PPT에 나와 있는 코드를 타이핑 하고 실행 결과를 확인합니다.
- **결과 확인** 단계에서는 따라 쳤던 코드의 결과를 확인하고 발생한 이슈/궁금점에 대해 논의합니다.
- 각 주차의 마지막 실습 예제 챕터는 **1)예제 설명 2)실습 진행** 순으로 진행됩니다.

스터디 진행 방법 - 개념 이해 단계

- 배우려는 개념을 이해하는 단계입니다.
- 해당 단계에서는 스터디 진행자가 직접 개념에 대해 설명합니다.

자료형과 변수(자료형) - 개념 이해

- 자료형(Data Type)**이란 프로그래밍 언어에서 정수/실수/문자 등 여러 종류의 데이터를 식별하는 분류
 - 파이썬에서는 숫자(int/float)와 문자열(str), 참 거짓을 따지기 위한 논리 자료형(bool) 여러 데이터가 모여 있는 묶음 형태 (list/tuple/dictionary/set)로 데이터를 분류
- | | |
|---|--|
| <ul style="list-style-type: none">int : 정수(양의 정수/0/음의 정수)를 의미. integer의 약자<ul style="list-style-type: none">1 / 0 / -1 / 1,000 / 921,119 등 다양한 정수들이 int에 포함float : 실수. <u>파이썬</u>에서는 소수로 표현<ul style="list-style-type: none">1.0 / 3.141592 / 110.00223 등 다양한 실수들이 float에 포함str : 문자 및 문자열. String의 약자<ul style="list-style-type: none">"a", "가", <u>"인사이저"</u>, "I like apple" 등 문자 및 단어/문자열 포함list : 순서가 있는 데이터 묶음<ul style="list-style-type: none">[1, 2, 3, 4, 5], ["안녕하세요", "반갑습니다"] 등 <u>대괄호</u>로 묶임 | <ul style="list-style-type: none">bool : 참 거짓을 따지기 위한 논리 자료형<ul style="list-style-type: none">True/False 두가지 형태tuple : 순서가 있고 수정 불가능한 데이터 묶음<ul style="list-style-type: none">(1, 2, 3), ("월", "화", "수") 등 <u>괄호</u>로 묶임set : 순서가 없고 중복 값이 존재하지 않는 데이터 묶음, 집합<ul style="list-style-type: none">{1, 2, 3}, {"a", <u>"b"</u>, "c", "d"} 등 <u>중괄호</u>로 묶임dictionary : <u>key:value</u>의 쌍으로 묶여 있는 데이터, key 중복X<ul style="list-style-type: none">{"name": "<u>sam</u>", "age": 19, "<u>score</u>": "<u>A</u>"}key와 value가 <u>콜론</u>으로 묶이고, 그 쌍이 <u>중괄호</u>로 묶임 |
|---|--|

- 코딩을 직접 따라 치고 실행시켜보는 단계입니다.
- 상단에 해당 코드를 학습하는 이유와 코드 내 표현 중 설명이 부족한 부분에 대한 추가적인 설명이 나와있습니다.

자료형과 변수(자료형) - 코딩 따라하기

- 직접 수치를 입력하고 어떤 타입의 데이터인지 확인
- `type()`은 해당 값의 타입을 알 수 있는 명령어

```
In [ ]: 1 type(1)

In [ ]: 1 type(3.141592)

In [ ]: 1 type("인사이저")

In [ ]: 1 type(True)

In [ ]: 1 type([1,2,3,4,5])

In [ ]: 1 type(("월", "화", "수"))

In [ ]: 1 type({10,11,12})

In [ ]: 1 type({"이름":"홍길동", "나이":300, "국적":"한국"})
```

INCIZOR

스터디 진행 방법 - 결과 확인 단계

- 코딩의 결과를 확인하는 단계입니다.
- 코딩 결과에 대해 간단한 리뷰를 듣고(진행자가 설명합니다), 발생한 이슈나 궁금한 사항에 대해 이야기 해보는 시간입니다.

자료형과 변수(자료형) - 결과 확인

```
In [1]: 1 type(1)
Out[1]: int

In [2]: 1 type(3.141592)
Out[2]: float

In [3]: 1 type("인사이드")
Out[3]: str

In [4]: 1 type(True)
Out[4]: bool

In [5]: 1 type([1,2,3,4,5])
Out[5]: list

In [6]: 1 type(("월", "화", "수"))
Out[6]: tuple

In [7]: 1 type({10,11,12})
Out[7]: set




In [8]: 1 type({"이름":"홍길동", "나이":300, "국적":"한국"})
Out[8]: dict
```

INCIZOR

1. 아나콘다, 주피터 노트북 설치
2. 파이썬 기본 문법 - 자료형과 변수
3. 조건문과 반복문
4. 파일 입출력
5. 함수
6. 실습 예제 - 성적 관리 테이블 만들기

1. 아나콘다, 주피터 노트북 설치

- 설치 링크에 접속에 개인의 컴퓨터 환경에 맞는 아나콘다를 설치
 - <https://www.anaconda.com/distribution/#download-section>
 - 파이썬 3.7 버전으로 다운
 - 운영체제 및 레지스터(x64/x32)에 맞게 다운로드

 Windows |  macOS |  Linux

Anaconda 2020.02 for Windows Installer

Python 3.7 version

Download

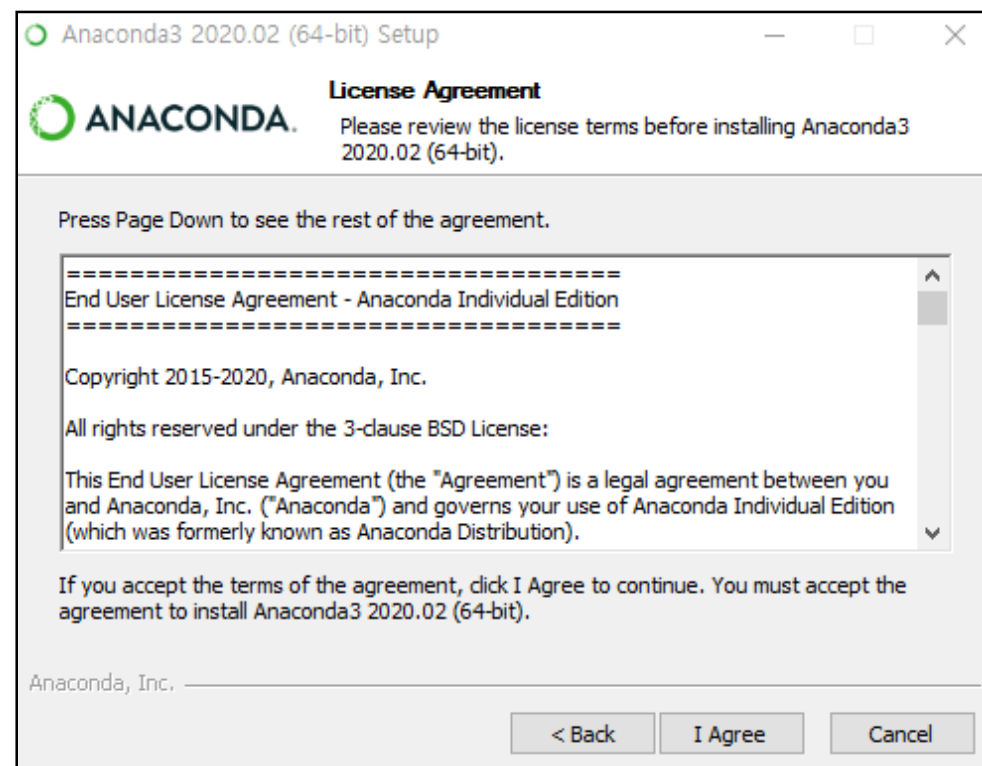
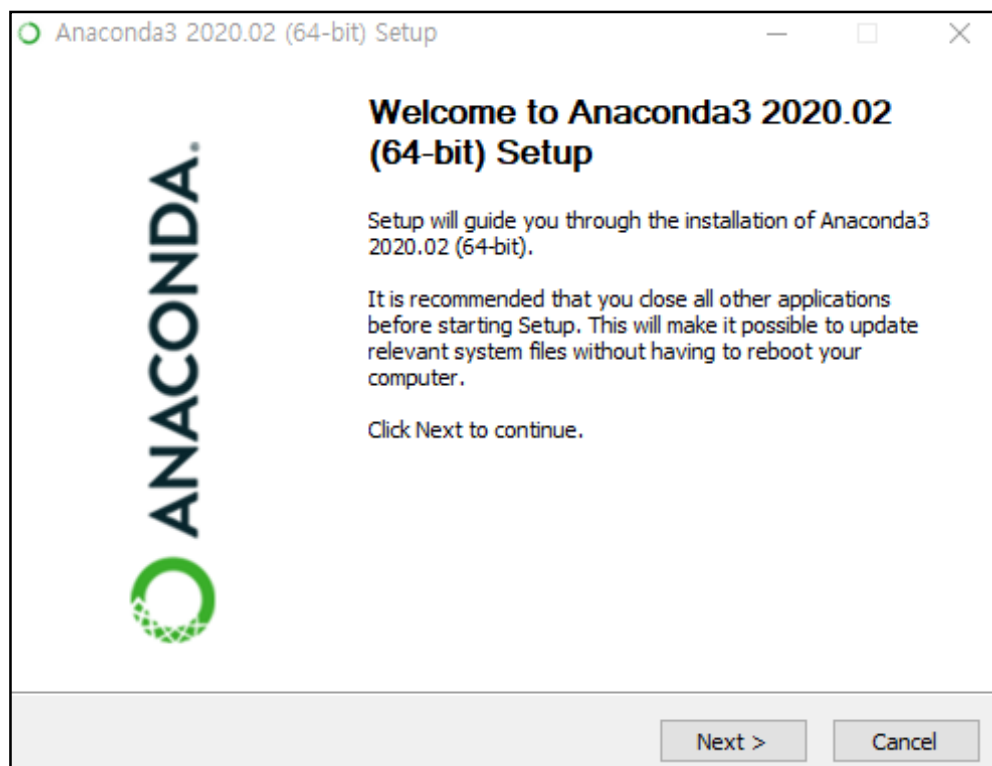
64-Bit Graphical Installer (466 MB)
32-Bit Graphical Installer (423 MB)

Python 2.7 version

Download

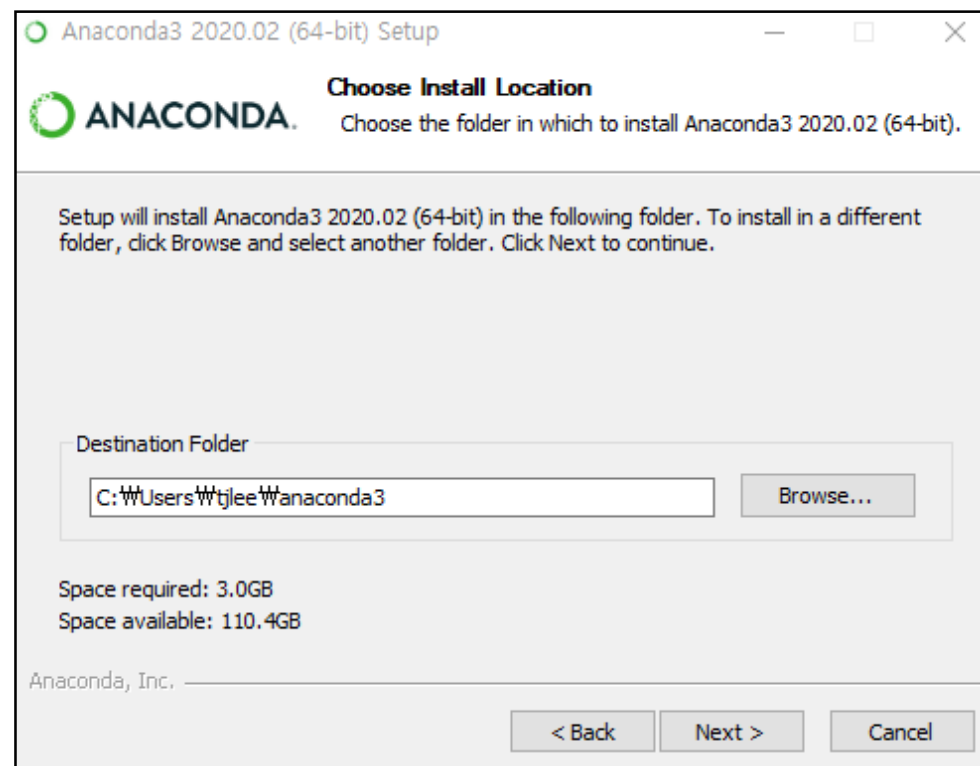
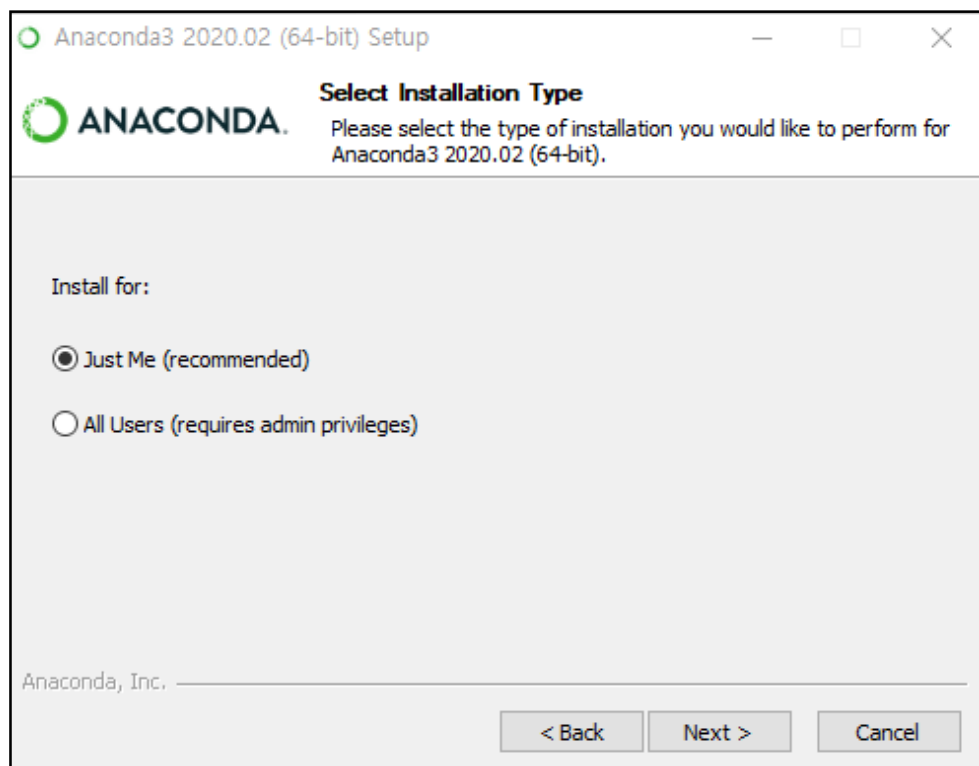
64-Bit Graphical Installer (413 MB)
32-Bit Graphical Installer (356 MB)

- setup 파일 실행 후 진행



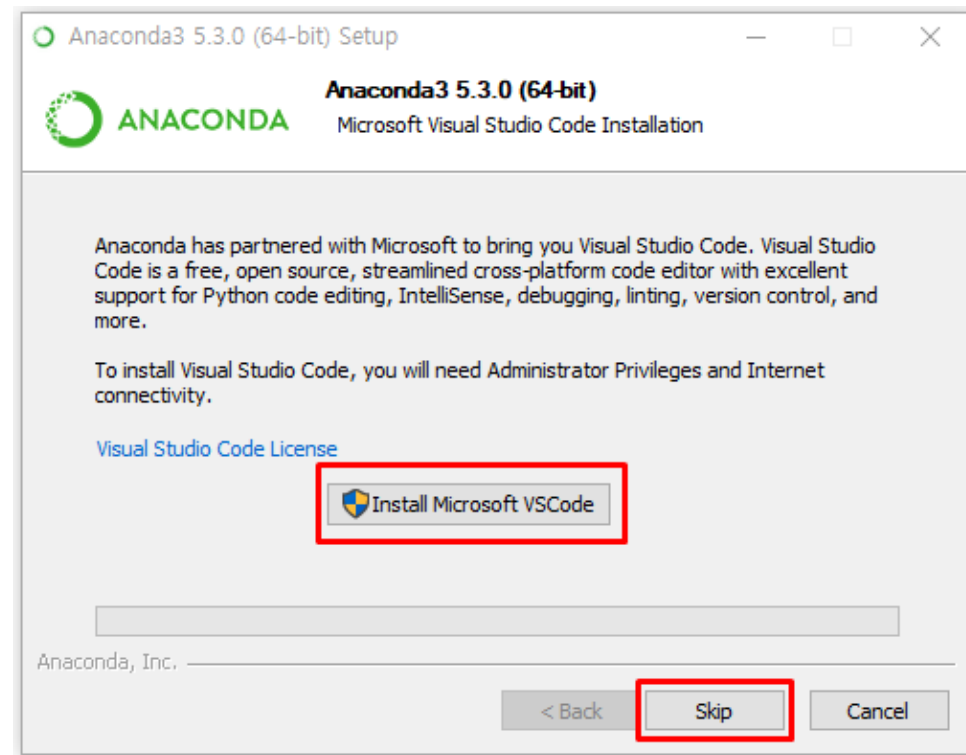
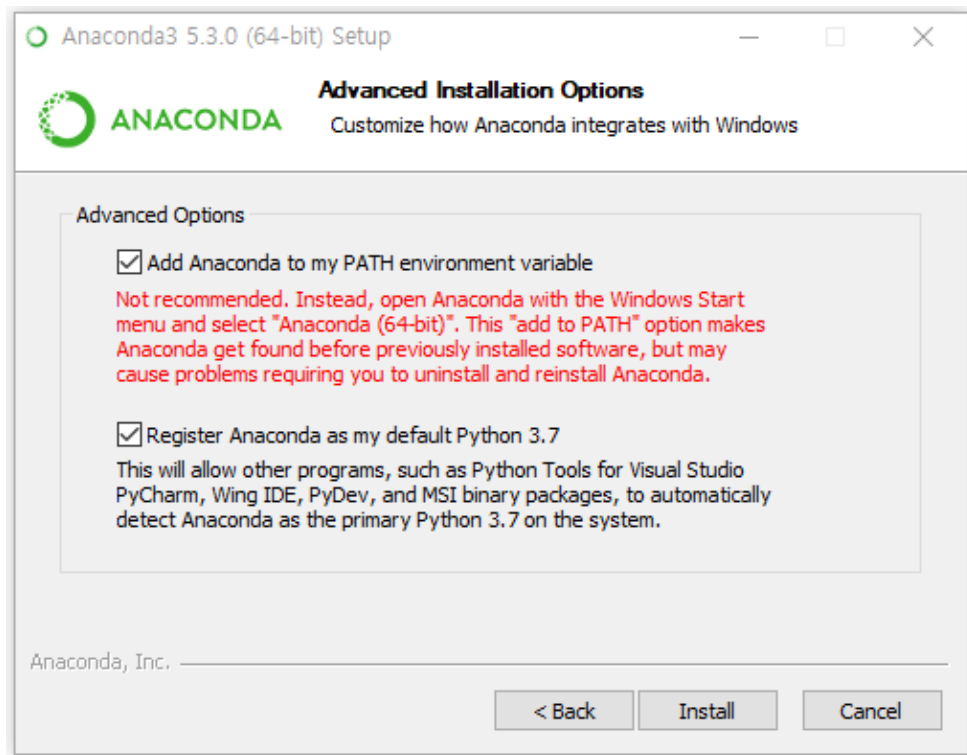
아나콘다, 주피터 노트북 설치 - setup 진행

- Just Me 항목을 선택
- 경로는 웬만하면 자동으로 설정된 경로로 지정



아나콘다, 주피터 노트북 설치 - setup 진행

- Advanced Options에서 두 개 다 선택
 - 이미 컴퓨터에 Anaconda가 설치되어 있을 경우엔 설치를 Cancel 하고 바로 학습 진행
 - Anaconda외에 파이썬이 설치되어 있는 경우에는 둘 다 체크하지 않고 진행
 - 설명 및 그림 참조 : <https://wonderbout.tistory.com/22>
- Install Microsoft VSCode는 Skip



- 마지막 체크박스는 설명에 관한 내용과 바로 실행하는 내용으로 굳이 선택할 필요는 없습니다
- 두개의 체크박스를 해제하고 Finish를 클릭

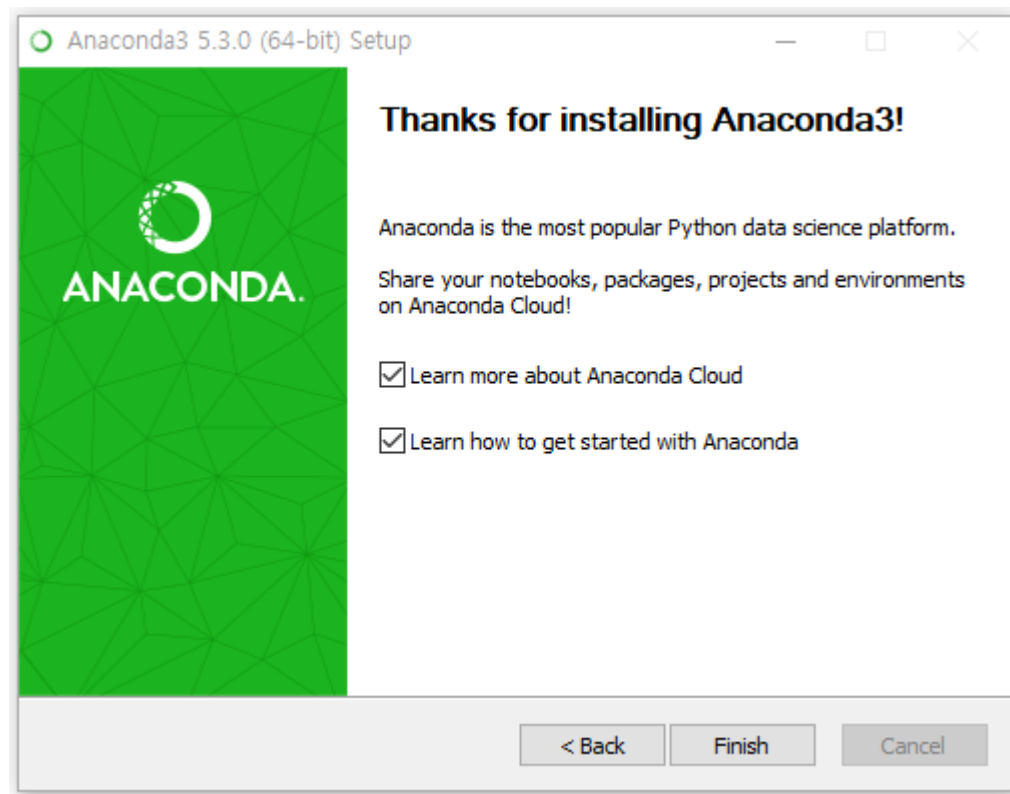
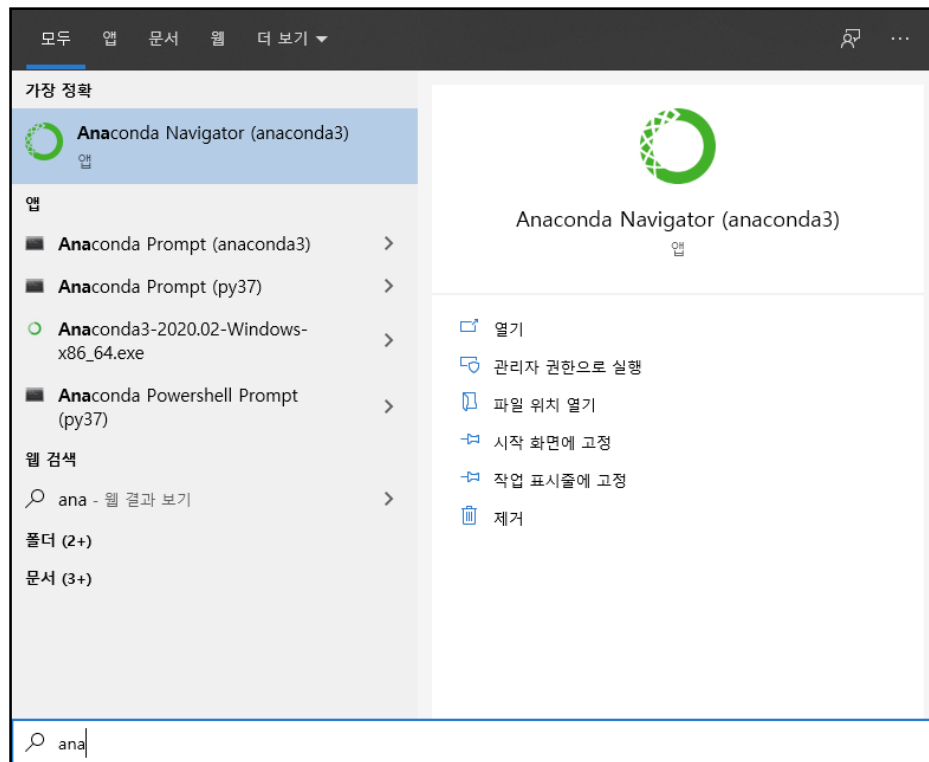


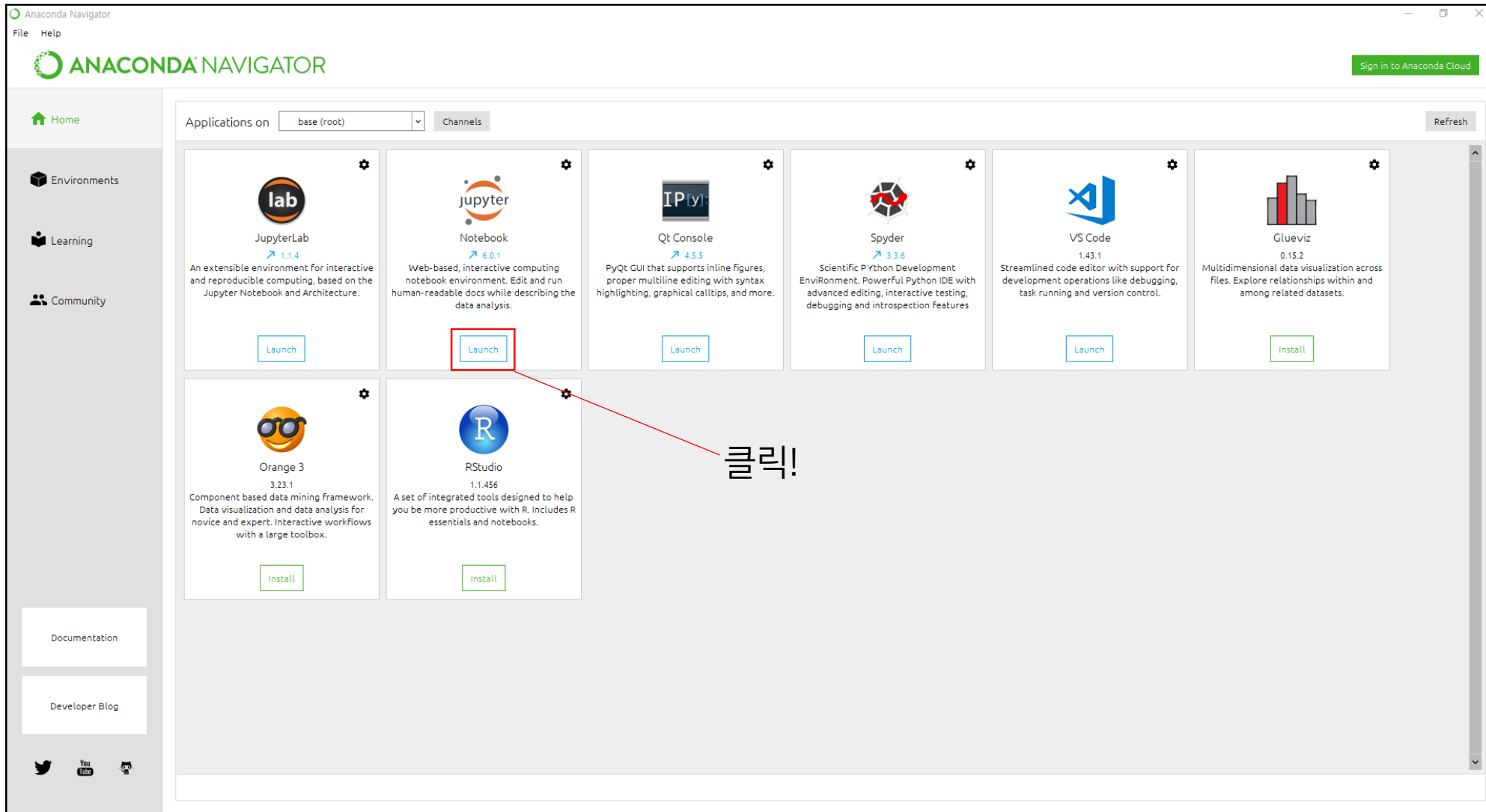
그림 참조 - <https://wonderbout.tistory.com/22>

아나콘다, 주피터 노트북 설치 - 실행

- 윈도우 검색창(윈도우키 + 'Q')을 띄우고 Anaconda Navigator를 검색
- 클릭하면 아나콘다 로고가 뜨면서 프로그램이 로딩

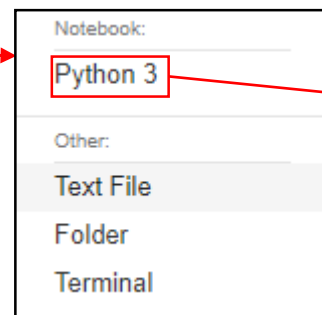
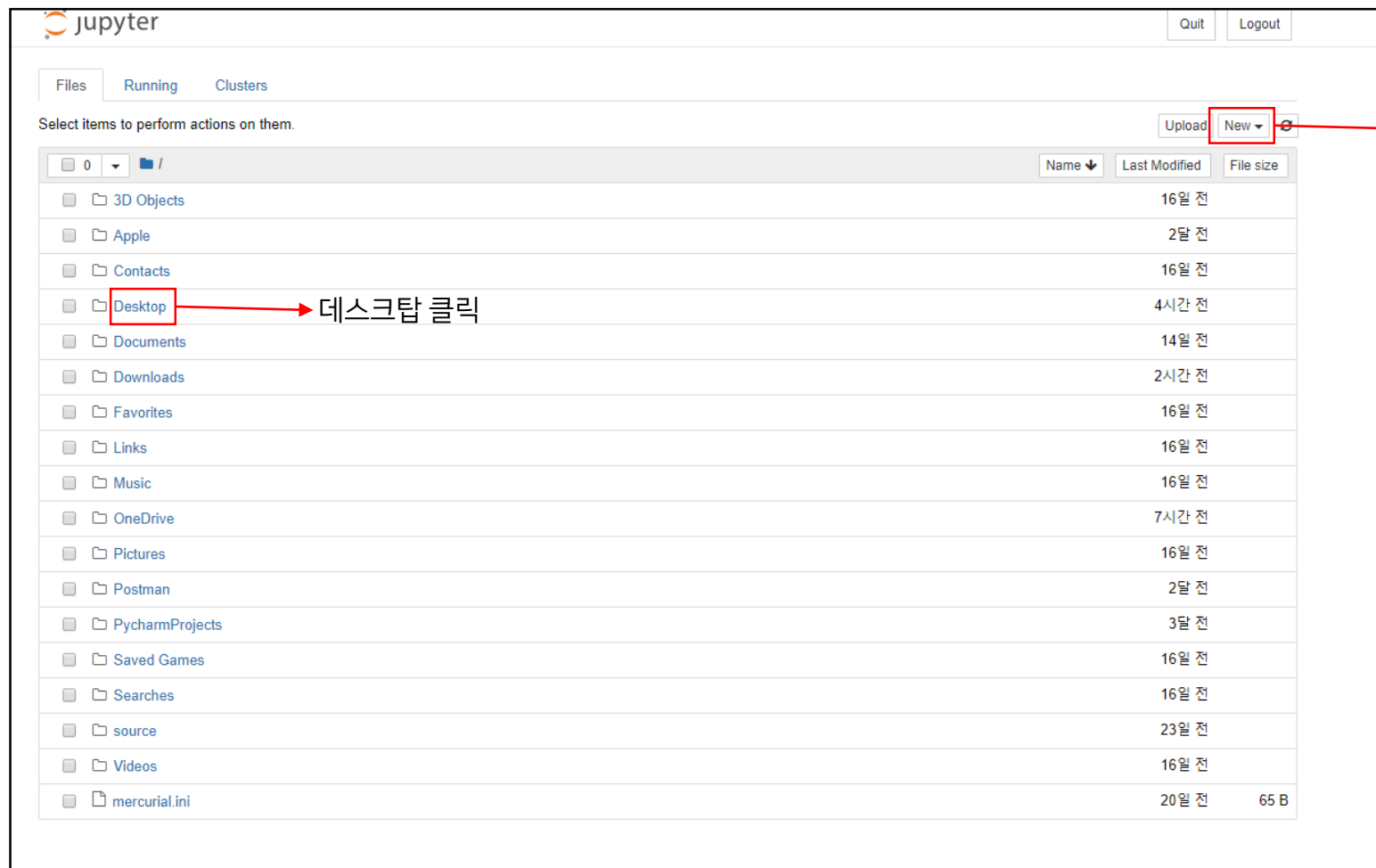


- Anaconda NAVIGATOR에서 Jupyter Notebook, Launch를 클릭



아나콘다, 주피터 노트북 설치 - 실행

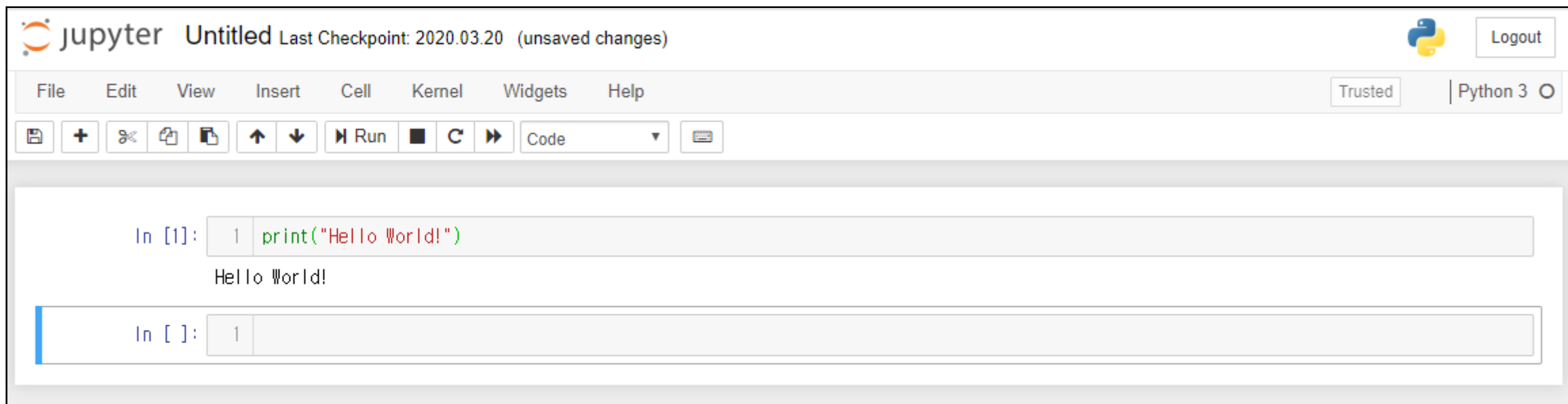
- Desktop 클릭(폴더명은 개인마다 다를 수 있음. Desktop은 바탕화면을 의미)
 - 코드를 다른 곳에 저장하고 싶으면 다른 폴더를 선택
- New 버튼 클릭 후 Python 3 클릭



Python 3 클릭 시 새로운 창 실행

아나콘다, 주피터 노트북 설치 - 실행

- 셀에 `print("Hello World!")` 입력, Shift + Enter 입력
- 설치 완료



- 키보드 단축키

- shift + enter : 코드 실행
- A : 선택한 셀 위쪽에 새로운 셀 생성
- B : 선택한 셀 아래쪽에 새로운 셀 생성
- D : 선택한 셀 삭제(두 번 입력)
- M : 선택한 셀 마크다운 모드로 전환
- Y : 선택한 셀 코드 입력 모드로 전환

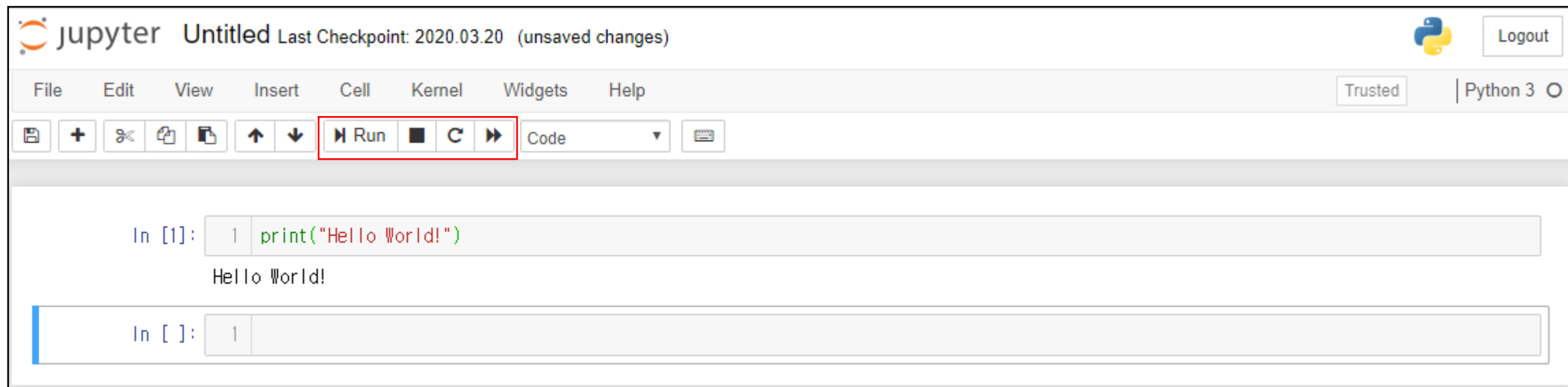
- 버튼

▶ Run : 선택한 셀 코드 실행

■ : 돌아가고 있는 코드를 강제 정지

↺ : 커널(서버)을 재시작, 돌아가고 있는 코드 강제 종료

▶▶ : 커널 재시작, 셀에 있는 모든 코드 다시 구동



2. 자료형과 변수

자료형과 변수(자료형) - 개념 이해

- **자료형(Data Type)**이란 프로그래밍 언어에서 정수/실수/문자 등 여러 종류의 데이터를 식별하는 분류
- 파이썬에서는 **숫자(int/float)**와 **문자열(str)**, 참 거짓을 따지기 위한 논리 자료형(bool), 여러 데이터가 모여 있는 **묶음 형태**(list/tuple/dictionary/set)로 데이터를 분류

- **int** : 정수(양의 정수/0/음의 정수)를 의미. integer의 약자
 - 1 / 0 / -1 / 1,000 / 921,119 등 다양한 정수들이 int에 포함
- **float** : 실수. 파이썬에서는 소수로 표현
 - 1.0 / 3.141592 / 110.00223 등 다양한 실수들이 float에 포함
- **str** : 문자 및 문자열. String의 약자. **따옴표**로 묶임
 - "a", "가", "인사이저", "I like apple" 등 문자 및 단어/문자열 포함
- **list** : 순서가 있는 데이터 묶음
 - [1, 2, 3, 4, 5], ["안녕하세요", "반갑습니다"] 등 **대괄호**로 묶임

- **bool** : 참 거짓을 따지기 위한 논리 자료형
 - True/False 두가지 형태
- **tuple** : 순서가 있고 수정 불가능한 데이터 묶음
 - (1, 2, 3), ("월", "화", "수") 등 **괄호**로 묶임
- **set** : 순서가 없고 중복 값이 존재하지 않는 데이터 묶음, 집합
 - {1, 2, 3}, {"a", "b", "c", "d"} 등 **중괄호**로 묶임
- **dictionary** : key:value의 쌍으로 묶여 있는 데이터, key 중복X
 - {"name":"sam", "age":19, "score":"A"}
 - key와 value가 **콜론**으로 묶이고, 그 쌍이 **중괄호**로 묶임

자료형과 변수(자료형) - 코딩 따라하기

- 직접 수치를 입력하고 어떤 타입의 데이터인지 확인
- `type()`은 해당 값의 타입을 알 수 있는 명령어

```
In [ ]: 1 type(1)
```

```
In [ ]: 1 type(3.141592)
```

```
In [ ]: 1 type("인사이저")
```

```
In [ ]: 1 type(True)
```

```
In [ ]: 1 type([1,2,3,4,5])
```

```
In [ ]: 1 type(("월", "화", "수"))
```

```
In [ ]: 1 type({10,11,12})
```

```
In [ ]: 1 type({"이름": "홍길동", "나이": 300, "국적": "한국"})
```

```
In [1]: 1 type(1)
```

```
Out[1]: int
```

```
In [2]: 1 type(3.141592)
```

```
Out[2]: float
```

```
In [3]: 1 type("인사이드")
```

```
Out[3]: str
```

```
In [4]: 1 type(True)
```

```
Out[4]: bool
```

```
In [5]: 1 type([1,2,3,4,5])
```

```
Out[5]: list
```

```
In [6]: 1 type(("월", "화", "수"))
```

```
Out[6]: tuple
```

```
In [7]: 1 type({10,11,12})
```

```
Out[7]: set
```

```
In [8]: 1 type({"이름": "홍길동", "나이": 300, "국적": "한국"})
```

```
Out[8]: dict
```

자료형과 변수(자료형_데이터 묶음) - 개념 이해

- 데이터 모음 형태는 4가지, list / tuple / set / dictionary로 나뉜다
- 모든 형태는 데이터 타입에 구애 받지 않고 값을 모아둘 수 있다
 - [1, "이", 3.0, [4,5]]
- list와 tuple의 경우 위치를 통해 값을 가져오는 인덱싱/특정 범위만 추출하는 슬라이싱이 가능하다

- list
 - 대괄호와 쉼표로 표현 - [1,2,3]
 - 위치를 통해 값을 가져올 수 있음(인덱싱)
- tuple
 - 괄호와 쉼표로 표현 - (1,2,3)
 - 내부의 값 수정, 변경이 불가능함, 인덱싱은 가능
- set
 - 중괄호와 쉼표로 표현 - {1,2,3}
 - 중복 값이 없고, 순서에 대한 개념이 없음
 - 연산자(&, |, -)를 통해 집합처럼 구현 가능
- dictionary
 - key와 value가 콜론으로 묶이고, 그 쌍이 중괄호로 묶임
 - {"age":30, "region":"seoul"}
 - key값을 통해 value를 읽는다

```
1 a = [1,2,3]
2 a[2]
```

list

3

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

tuple

```
TypeError                                Traceback (most recent call last)
<ipython-input-10-5536b44bdce3> in <module>
      1 a = (1,2,3)
----> 2 a[2] = 5
```

TypeError: 'tuple' object does not support item assignment

```
1 a = {1,2,3}
2 b = {2,3,6}
3 a & b
```

set

{2, 3}

```
1 a = {
2     "age" : 30,
3     "region" : "seoul",
4 }
5 a["age"]
```

dictionary

30

자료형과 변수(자료형_데이터 묶음) - 코딩 따라하기

- 직접 데이터 묶음을 컨트롤하여 감각 익히기
- 문자열도 list의 인덱싱/슬라이싱이 가능한 것 확인
- #<문자> 는 주석처리로, 코드를 실행할 때 영향을 끼치지 않는 코드를 의미
 - 코드의 설명 및 안 쓰는 코드를 해당 위치에 보관할 때 사용

```
1 a = [1,2,3,4,5]
2 print(a)
3 a[2] = 100
4 print(a)
5 b = a[2:4]
6 print(b)
```

```
1 a = {1,2,3,1,2,3,1,2,3}
2 b = {2,3,10}
3 print(a)
4 print(a & b)
5 print(a | b)
6 print(a - b)
```

```
1 a = {
2     "이름" : "인사이저",
3     "나이" : 2,
4     "키" : 350,
5 }
6 print(a["이름"])
```

```
1 #문자열 인덱싱 슬라이싱
2 a = "안녕하세요!"
3 print(a[5])
4 print(a[:2])
5 print(a[-2]) #음수 값을 넣으면 역수로 값을 가져옵니다
```

자료형과 변수(자료형_데이터 묶음) - 결과 확인

```
1 a = [1,2,3,4,5]
2 print(a)
3 a[2] = 100
4 print(a)
5 b = a[2:4]
6 print(b)
```

[1, 2, 3, 4, 5]
[1, 2, 100, 4, 5]
[100, 4]

```
1 a = {1,2,3,1,2,3,1,2,3}
2 b = {2,3,10}
3 print(a)
4 print(a & b)
5 print(a | b)
6 print(a - b)
```

{1, 2, 3}
{2, 3}
{1, 2, 3, 10}
{1}

```
1 a = {
2     "이름" : "인사이저",
3     "나이" : 2,
4     "키" : 350,
5 }
6 print(a["이름"])
```

인사이저

```
1 #문자열 왼쪽 슬라이싱
2 a = "안녕하세요!"
3 print(a[5])
4 print(a[:2])
5 print(a[-2]) #음수 값을 넣으면 역수로 값을 가져옵니다
```

!
안녕
요

자료형과 변수(변수) - 개념 이해

- 데이터를 담아두는 하나의 **바구니** 개념
- 바구니의 값은 언제든지 바꿀 수 있으며, 여러 개를 만들 수도 있다.

- 변수 명명 규칙1
 - 영어(대소문자), 숫자, 언더바(_) 조합으로 만든다
- 변수 명명 규칙2
 - 첫 자리에 숫자를 입력할 수 없다
- 변수 명명 규칙3
 - 언더바를 제외한 특수문자 사용 불가
- 변수 명명 규칙4
 - 파이썬 키워드를 변수명으로 사용할 수 없음
 - 주피터 노트북에 변수 명 입력 시 색이 바뀌면 변수로 쓰면 안 된다 라고만 기억

```
1 abc = 10
2 a1_b2_c3 = 'test'
3 helloworld2020 = 'hello world!'
```

규칙 1

```
1 abc00 = 10
2 abc&bdc = 'abcabc'
3 123a = 123
```

규칙 2, 3

File "<ipython-input-5-99d62c624e3b>", line 1

```
abc00 = 10
  ^
```

SyntaxError: invalid syntax

규칙 4

```
1 #파이썬 키워드 목록
2 False, None, True, and, as, assert, break, class, continue, def, del,
3 elif, else, except, finally, for, from, global, if, import, in, is,
4 lambda, nonlocal, not, or, pass, raise, return, try, while, with,
5 yield
```

자료형과 변수(변수) - 코딩 따라하기

- 변수로 사용할 수 있는 표현/사용할 수 없는 표현을 써서 규칙 이해하기
- 파이썬 키워드에 숫자, 문자를 조합할 경우엔 변수로 사용 가능

```
1 abc = 10
```

```
1 student_number = '888899'
```

```
1 testfile111 = "./test.txt"
```

```
1 01_document = "hello"
```

```
1 alpha&beta = {'a','b'}
```

```
1 True = 'true'
```

```
1 False = 0
```

```
1 #파이썬 키워드에 언더바와 숫자, 문자를 조합할 경우엔 사용할 수 있습니다.  
2 True_False = True  
3 finally_end = 999
```

```
1 abc = 10
```

```
1 student_number = '888899'
```

```
1 testfile111 = "./test.txt"
```

```
1 0l_document = "hello"
```

```
File "<ipython-input-4-6e0e322e4802>", line 1  
  0l_document = "hello"  
    ^
```

SyntaxError: invalid token

```
1 alpha&beta = {'a','b'}
```

```
File "<ipython-input-5-6ea1ae087b6>", line 1  
  alpha&beta = {'a','b'}  
    ^
```

SyntaxError: can't assign to operator

```
1 True = 'true'
```

```
File "<ipython-input-6-e651726200c3>", line 1  
  True = 'true'  
    ^
```

SyntaxError: can't assign to keyword

```
1 False = 0
```

```
File "<ipython-input-7-223dbc74e028>", line 1  
  False = 0  
    ^
```

SyntaxError: can't assign to keyword

```
1 #파이썬 키워드에 언더바와 숫자, 문자를 조합할 경우엔 사용할 수 있습니다.  
2 True_False = True  
3 finally_end = 999
```

- 프로그래밍에서 연산이란 데이터를 처리하여 결과를 산출하는 것
- 연산자란 연산을 수행하기 위한 기호
- 파이썬에서는 산술 연산자 / 비교 연산자 / 논리 연산자가 존재

산술 연산자

- 사칙연산 $+$, $-$, $*$, $/$
 - 사칙연산을 하는데 사용
 - 정수와 실수형 데이터를 함께 연산하면 실수형으로 형 변환(혹은 나눗셈 시)
 - list, 문자열 등에서도 일부 사칙연산 사용
- 몫과 나머지 $//$, $%$
 - 나눗셈을 할 때 몫과 나머지를 산출
 - $//$ 는 몫, $%$ 는 나머지
- 제곱 $**$
 - 제곱 연산 시 사용 ($<밑> ** <지수>$)
 - 지수에 분수, 실수 입력 시 제곱근 구현
 - 지수에 0 이하의 값 입력 시 역수를 구현

비교 연산자

- $==$, $!=$, $>$, $<$, $>=$, $<=$
 - 데이터의 같음/다름 및 차이를 비교할 때 사용
 - 결과는 bool type의 데이터로 반환(True, False)

논리 연산자

- and, or, not
 - bool type의 논리 연산 시 사용
 - and : 두 값이 모두 True 일 때 True
 - or : 두 값 중 한 값이 True 일 때 True
 - not : True일 땐 False, False일 땐 True
 - 여러 비교 연산자를 동시에 사용할 때 활용

자료형과 변수(연산자) - 코딩 따라하기

- 다양한 연산자들을 사용하고 이해하기
- 동일 변수에 연산을 시행할 때 += 등으로 축약할 수 있음($n = n + 1 \rightarrow n += 1$)
- 연산자 우선순위(빠름->느림) : 제곱 -> 부호(+, -) -> 곱셈, 나눗셈, 몫, 나머지 -> 덧셈, 뺄셈 -> 비교 연산(>, < 등) -> 논리 연산(and, or, not)
 - 외우기 힘들 경우 우선적으로 계산하고 싶은 식을 괄호로 씌워주자

```
1 10 + 5 - 2 * 9 / 3
```

```
1 n = 0
2 n = n + 1
3 print(n)
4 n += 1
5 print(n)
```

```
1 5 % 2
```

```
1 10 // 3
```

```
1 5 ** 2
```

```
1 4 ** 0.5
```

```
1 10 == 10
```

```
1 100 != 10
```

```
1 11 >= 1
```

```
1 print(True and True)
2 print(True or False)
3 print(not True)
4 print(10>=3 or 3>10)
```

```
1 #연산자 우선순위가 헷갈릴 때, 먼저 계산하고 싶은 연산에 괄호를 씌워주자
2 not((( -2)+(4**3)*5)<=((100%3)+(5**2)))
```

자료형과 변수(연산자) - 결과 확인

1	<code>10 + 5 - 2 * 9 / 3</code>
9.0	
1	<code>n = 0</code>
2	<code>n = n + 1</code>
3	<code>print(n)</code>
4	<code>n += 1</code>
5	<code>print(n)</code>
1	
2	
1	<code>5 % 2</code>
1	
1	<code>10 // 3</code>
3	
1	<code>5 ** 2</code>
25	
1	<code>4 ** 0.5</code>
2.0	
1	<code>10 == 10</code>
True	

1	<code>100 != 10</code>
True	
1	<code>11 >= 1</code>
True	
1	<code>print(True and True)</code>
2	<code>print(True or False)</code>
3	<code>print(not True)</code>
4	<code>print(10 >= 3 or 3 > 10)</code>
True	
True	
False	
True	
1	<i>#연산자 우선순위가 헷갈릴 땐, 먼저 계산하고 싶은 연산에 괄호를 씌워주자</i>
2	<code>not (((-2) + (4 ** 3) * 5) <= ((100 % 3) + (5 ** 2)))</code>
True	

3. 조건문/반복문

조건문/반복문(조건문) - 개념 이해

- 조건문이란 특정 조건에 부합할 때 특정 코드가 실행되도록 코드를 설계하는 문법
- if 문과 if ~ else, elif가 존재

if

- 조건에 부합할 시 아래 코드가 실행
- 조건식의 결과는 True, False로 나와야 함
- 실행 되는 코드는 탭(띄어쓰기 4칸)으로 들여쓰기
- 들여쓰기를 안 할 경우 if 문 적용 X
- 여러 개의 if문을 설정할 수 있음

if ~ else

- if의 조건식이 False일 경우 코드 실행
- 분기를 설정해주는 표현(Yes or No)

elif

- 여러 분기를 설정할 때 사용
- 맨 마지막에 else 사용 가능(모든 조건이 부합 X 시)
- if 여러 개를 쓰는 것보다 컴퓨터 자원 소모 효율적

if

```
1 if <조건식> :  
2     <코드>  
3     .  
4     .  
5     .
```

if ~ else

```
1 if <조건식> :  
2     <코드>  
3 else:  
4     <코드>
```

elif

```
1 if <조건식> :  
2     <코드>  
3 elif <조건식>:  
4     <코드>  
5 elif <조건식>:  
6     <코드>  
7 else:  
8     <코드>
```


조건문/반복문(조건문) - 코딩 따라하기

- 여러 케이스의 조건문을 다뤄 보고 이해하기
- if문 안에 if문을 사용할 수 있다 -> 다양한 조건문 조합으로 여러 결과를 도출할 수 있다

```
1 score = 80
2 if score >= 80:
3     print("B")
```

```
1 #들여쓰기를 안 하면 에러 발생
2 score = 80
3 if score >= 80:
4     print("B")
```

```
1 score = 70
2 if score >= 80:
3     print("B")
4 if score >= 70:
5     print("C")
```

```
1 #if 안에 if를 사용할 수도 있다
2 q1 = "to be"
3 q2 = "not to be"
4 if q1 == "to be":
5     if q2 == "not to be":
6         print("that is the question.")
```

```
1 power = 0
2 if power == 1:
3     print("작동 중")
4 else:
5     print("종료")
```

```
1 score = 68
2 if score >= 90 :
3     print("A")
4 elif score >= 80 :
5     print("B")
6 elif score >= 70 :
7     print("C")
8 else:
9     print("F")
```

조건문/반복문(조건문) - 결과 확인

```
1 score = 80
2 if score >= 80:
3     print("B")
```

B

```
1 #들여쓰기를 안 하면 에러 발생
2 score = 80
3 if score >= 80:
4     print("B")
```

File "<ipython-input-2-73811ddbc37a>", line 4
 print("B")
 ^

IndentationError: expected an indented block

```
1 score = 70
2 if score >= 80:
3     print("B")
4 if score >= 70:
5     print("C")
```

C

```
1 #if 안에 if를 사용할 수도 있다
2 q1 = "to be"
3 q2 = "not to be"
4 if q1 == "to be":
5     if q2 == "not to be":
6         print("that is the question.")
```

that is the question.

```
1 power = 0
2 if power == 1:
3     print("작동 중")
4 else:
5     print("종료")
```

종료

```
1 score = 68
2 if score >= 90 :
3     print("A")
4 elif score >= 80 :
5     print("B")
6 elif score >= 70 :
7     print("C")
8 else:
9     print("F")
```

F

조건문/반복문(반복문) - 개념 이해

- 반복문이란 특정 조건에 부합하는 만큼, 혹은 특정 범위 만큼 특정 코드를 반복시키고자 할 때 사용하는 문법
- for문과 while문

for

- 특정 범위 내에서 작업을 반복하고자 할 때 사용
- 반복 범위의 경우 **데이터 모음 형태**가 쓰임
 - list, dictionary, set, tuple
 - 묶음에서 데이터를 하나 씩 가져옴
- 반복 범위의 값을 모두 가져오면 반복이 끝난다
- 특정 횟수를 지정하여 반복하고 싶을 때 range()함수 사용
 - range(start, stop, step)
- for문 안에 for문 사용 가능

while

- 특정 조건에 부합할 때 작업을 반복
- 조건식이 False가 되었을 때 반복 종료
- 조건식이 False가 되지 않을 경우 **무한 루프**에 빠진다

for

```
1 for <반복 변수> in <반복 범위>:  
2     <코드>
```

while

```
1 while <조건식> :  
2     <코드>
```

조건문/반복문(반복문) - 코딩 따라하기 (1)

- 여러 케이스의 반복문을 다뤄 보고 이해하기
- dictionary 타입의 데이터를 for문에 입력할 경우 key 값만 가져온다

```
1 a = [1,2,3,4]
2 for i in a:
3     print(i)
```

```
1 a = {
2     "이름" : "막걸리",
3     "성격" : "전통술",
4     "재질" : "쌀",
5 }
6 for i in a:
7     print(i)
8 print("="*50)
9 for i in a:
10    print(a[i])
```

```
1 for i in range(10):
2     print(i, end=" ")
3 print()
4 for i in range(1, 10):
5     print(i, end=" ")
6 print()
7 for i in range(10, 0, -1):
8     print(i, end=" ")
```

```
1 #range에는 정수항만 들어갈 수 있습니다
2 range(1.5, 10.5, 0.5)
```

```
1 for i in range(0,6):
2     for j in range(i+1):
3         print("*", end=" ")
4     print()
```

```
1 n = 0
2 while n<10:
3     print(n, end=" ")
4     n = n + 1
```

```
1 #코드 실행 시 무한 루프에 빠집니다. 서둘러 강제종료 하세요.
2 n = 0
3 while n > -1:
4     n = n + 1
```

조건문/반복문(반복문) - 결과 확인 (1)

```
1 a = [1,2,3,4]
2 for i in a:
3     print(i)
```

```
1
2
3
4
```

```
1 a = {
2     "이름" : "막걸리",
3     "성격" : "전통술",
4     "재질" : "쌀",
5 }
6 for i in a:
7     print(i)
8 print("="*50)
9 for i in a:
10    print(a[i])
```

이름
성격
재질

=====

막걸리
전통술
쌀

```
1 for i in range(10):
2     print(i, end=" ")
3 print()
4 for i in range(1, 10):
5     print(i, end=" ")
6 print()
7 for i in range(10, 0, -1):
8     print(i, end=" ")
```

0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
10 9 8 7 6 5 4 3 2 1

```
1 #range에는 정수형만 들어갈 수 있습니다
2 range(1.5, 10.5, 0.5)
```

Traceback (most recent call last):
<ipython-input-9-639ffb6c68cd> in <module>
1 #range에는 정수형만 들어갈 수 있습니다
----> 2 range(1.5, 10.5, 0.5)

TypeError: 'float' object cannot be interpreted as an integer

```
1 for i in range(0,6):
2     for j in range(i+1):
3         print("+", end=" ")
4     print()
```

+
+ +
+ + +
+ + + +
+ + + + +
+ + + + + +

```
1 n = 0
2 while n<10:
3     print(n, end=" ")
4     n = n + 1
```

0 1 2 3 4 5 6 7 8 9

```
1 #코드 실행 시 무한 루프에 빠집니다. 서둘러 강제종료 하세요.
2 n = 0
3 while n > -1:
4     n = n + 1
```

조건문/반복문(반복문) - 코딩 따라하기 (2)

- 반복문을 제어하는 명령어 두개 (break, continue) 사용해보기
 - **break** : 반복문을 종료 시킨다
 - **continue** : continue 밑의 코드들을 무시하고 다음 반복 단계로 넘긴다
- import time을 하면 현실 시간을 측정할 수 있는 함수를 사용할 수 있다

```
1 # 홀수만 구하기
2 for i in range(0, 50):
3     if i % 2 == 0:
4         continue
5     print(i, end=" ")
```

```
1 n = 0
2 while n < 100:
3     print(n)
4     if n >= 3:
5         break
6     n += 1
```

```
1 #무한 루프 정지 시키기
2 import time
3
4 start = time.time()
5 while True:
6     if time.time() - start >= 3:
7         break
8     print("탈출")
```

조건문/반복문(반복문) - 결과 확인 (2)

```
1 # 홀수만 구하기
2 for i in range(0, 50):
3     if i % 2 == 0:
4         continue
5     print(i, end=" ")
```

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49

```
1 n = 0
2 while n < 100:
3     print(n)
4     if n >= 3:
5         break
6     n += 1
```

0
1
2
3

```
1 #무한 루프 방지 시키기
2 import time
3
4 start = time.time()
5 while True:
6     if time.time() - start >= 3:
7         break
8 print("탈출")
```

탈출

4. 파일 입출력

파일 입출력(화면상 입출력) - 개념 이해

- 주피터 노트북 상에서 바로 데이터를 입 출력할 수 있도록 하는 문법들 소개
- print()와 input()

print()

- 괄호 안에 출력하고 싶은 데이터, 혹은 변수 입력
- 사용 시 강제 개행(다음 행 넘김)
 - print(<데이터>, end="<연결표현>")
 - 다음 출력 시 연결 표현에 맞게 이어져서 출력
- format을 통해 변수와 문자를 섞어서 출력 가능

input()

- 코드 실행 시 입력 박스 생성
- input(<값>)입력 시 입력 박스와 함께 설명 추가
- 입력 값은 기본적으로 문자열 형태, 다른 타입으로 이용 시 형 변환 필요

print()

1	print("Hello")	print, 개행
2	print("world!")	
Hello world!		
1	print("Hello", end=", ")	print, 연결 표현
2	print("world!")	
Hello, world!		
1	a = "화란"	print, format 사용
2	b = "순정"	
3	print("{0}아, 나도 {1}이 있다.".format(a,b))	
4	print(f"{a}아, 나도 {b}이 있다.")	
화란아, 나도 순정이 있다. 화란아, 나도 순정이 있다.		

input()

1	<code>input()</code>	일반 input
<input type="text"/>		
1	<code>input("이름을 입력하세요! :")</code>	
이름을 입력하세요! : <input type="text"/>		
1	<code>a = input()</code>	
2	<code>print(a)</code>	
3	<code>print(type(a))</code>	
100	타입은 문자열	
100		
<class 'str'>		

- 화면상 입출력 기능 및, 기존에 학습한 내용을 종합하여 프로그램을 만들어 보기
- `list.append(<데이터>)` : 특정 list에 새로운 데이터를 추가한다

```
1 db_list = []
2 chk = "n"
3 while True:
4     save_db = {}
5     if chk == "y":
6         break
7     save_db["name"] = input("이름을 입력하세요. :")
8     save_db["area"] = input("지역을 입력하세요. :")
9     save_db["score"] = input("점수를 입력하세요. :")
10
11     print("=" * 100)
12     print("입력된 데이터 입니다.")
13     print(save_db)
14     db_list.append(save_db) #리스트에 값을 추가하는 함수입니다.
15
16     chk = input("종료 하시겠습니까? [y/n]")
17
18 print("=" * 100)
19 print("입력된 명단")
20 print("=" * 100)
21 for db in db_list:
22     print(db)
```

파일 입출력(화면상 입출력) - 결과 확인

```
1 db_list = []
2 chk = "n"
3 while True:
4     save_db = {}
5     if chk == "y":
6         break
7     save_db["name"] = input("이름을 입력하세요. :")
8     save_db["area"] = input("지역을 입력하세요. :")
9     save_db["score"] = input("점수를 입력하세요. :")
10
11     print("=" * 100)
12     print("입력된 데이터 입니다.")
13     print(save_db)
14     db_list.append(save_db) #리스트에 값을 추가하는 함수입니다.
15
16     chk = input("종료 하시겠습니까? [y/n]")
17
18 print("=" * 100)
19 print("입력된 명단")
20 print("=" * 100)
21 for db in db_list:
22     print(db)
```

이름을 입력하세요. :인사이저

지역을 입력하세요. :서울

점수를 입력하세요. :999

=====

입력된 데이터 입니다.

{'name': '인사이저', 'area': '서울', 'score': '999'}

종료 하시겠습니까? [y/n]n

이름을 입력하세요. :이동재

지역을 입력하세요. :서울

점수를 입력하세요. :100

=====

입력된 데이터 입니다.

{'name': '이동재', 'area': '서울', 'score': '100'}

종료 하시겠습니까? [y/n]n

이름을 입력하세요. :홍길동

지역을 입력하세요. :조선

점수를 입력하세요. :80

=====

입력된 데이터 입니다.

{'name': '홍길동', 'area': '조선', 'score': '80'}

종료 하시겠습니까? [y/n]y

=====

입력된 명단

=====

{'name': '인사이저', 'area': '서울', 'score': '999'}

{'name': '이동재', 'area': '서울', 'score': '100'}

{'name': '홍길동', 'area': '조선', 'score': '80'}

파일 입출력(파일 입출력) - 개념 이해

- 파이썬에서는 다양한 파일들을 읽을 수 있다
- open()을 통해 파일을 읽어오거나 생성할 수 있다
 - f=open("<경로+파일명>", "<모드>")

open()

- 파일 경로와 파일명은 문자열로 작성
 - 경로 지정은 안 할 시 현재 코드 파일이 있는 위치 지정
- 모드는 주로 "r(읽기)", "w(쓰기)", "wb", "rb" 가 쓰임
 - 'b'가 붙은 모드는 '바이너리'의 의미
 - 컴퓨터 내에서 사용하는 표현
- open()으로 파일을 다룬 후 종료 코드 입력 필수
 - f.close()
 - 안 할 시 파일이 계속 켜져 있는 상태로 지속

파일 쓰기('w')

- 코드 상에서 다룬 데이터를 파일에 덮어 씌울 수 있음
- 경로 상에 파일이 없을 경우 새로 생성
- f.write(<데이터>)형태로 입력
 - 데이터 입력 시 개행 필요, 안 할 시 데이터 덮어쓰기

파일 쓰기('r')

- 파일의 데이터를 읽어올 수 있음
- 경로상 파일이 존재하지 않을 경우 에러 발생
- 데이터를 읽을 때 전체/한줄 씩/줄 단위 한꺼번에 읽을 수 있음

with open("<경로+파일명>", "<모드>") as f:

- 자동으로 파일 close
- with 후 코드 들여쓰기. 들여 쓴 코드만 실행 후 파일 종료

파일 입출력(파일 입출력) - 코딩 따라하기

- txt 파일을 직접 만들고 다양한 형태로 읽어 들이면서 파일 입출력 이해하기
- `"""<문자열>"""`은 개행된 문자(문장, 긴 글)를 입력할 수 있는 방법. 혹은 따로 변수에 입력하지 않고 주석처럼 표기할 수도 있다.

```
1 txt = """텍스트 파일을 새로 생성합니다.  
2 이 파일을 통해 파일 입출력이  
3 어떻게 이루어지는 지 알 수 있습니다.  
4 """  
5 f = open("test.txt", "w") #코드 파일이 있는 곳(desktop)에 저장  
6 f.write(txt)  
7 f.close()
```

```
1 #모든 텍스트 데이터를 가져옵니다  
2 f = open("test.txt", "r")  
3 txt = f.read()  
4 f.close()  
5 print(txt)
```

```
1 #텍스트 데이터를 한 줄 씩 가져옵니다  
2 f = open("test.txt", "r")  
3 txt1 = f.readline()  
4 txt2 = f.readline()  
5 f.close()  
6 print(txt2)  
7 print(txt1)
```

```
1 #모든 텍스트 데이터를 리스트 형태로 가져옵니다.  
2 f = open("test.txt", "r")  
3 txt = f.readlines()  
4 f.close()  
5 print(txt)
```

```
1 """  
2 반복문으로 한 줄 씩 가져올 수 있고,  
3 with open 사용시 close를 안 사용해도 됩니다.  
4 """  
5 with open("test.txt", "r") as f:  
6     line = f.readline()  
7     while line:  
8         print(line)  
9         line = f.readline()
```

파일 입출력(파일 입출력) - 결과 확인

```
1 txt = """텍스트 파일을 새로 생성합니다.  
2 이 파일을 통해 파일 입출력이  
3 어떻게 이루어지는 지 알 수 있습니다.  
4 """  
5 f = open("test.txt", "w") #코드 파일이 있는 곳(desktop)에 저장  
6 f.write(txt)  
7 f.close()
```

```
1 #모든 텍스트 데이터를 가져옵니다  
2 f = open("test.txt", "r")  
3 txt = f.read()  
4 f.close()  
5 print(txt)
```

텍스트 파일을 새로 생성합니다.
이 파일을 통해 파일 입출력이
어떻게 이루어지는 지 알 수 있습니다.

```
1 #텍스트 데이터를 한 줄 씩 가져옵니다  
2 f = open("test.txt", "r")  
3 txt1 = f.readline()  
4 txt2 = f.readline()  
5 f.close()  
6 print(txt2)  
7 print(txt1)
```

이 파일을 통해 파일 입출력이
텍스트 파일을 새로 생성합니다.

test.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

텍스트 파일을 새로 생성합니다.
이 파일을 통해 파일 입출력이
어떻게 이루어지는 지 알 수 있습니다.

Wn : 개행 문자

- 행 넘김 문자. 출력 시에는 표기되지 않는다

Wt : 탭 문자

- 문자 사이에 탭을 추가. 출력 시 표기 X

```
1 #모든 텍스트 데이터를 리스트 형태로 가져옵니다  
2 f = open("test.txt", "r")  
3 txt = f.readlines()  
4 f.close()  
5 print(txt)
```

['텍스트 파일을 새로 생성합니다.\n', '이 파일을 통해 파일 입출력이\n', '어떻게 이루어지는 지 알 수 있습니다.\n']

```
1 """  
2 반복문으로 한 줄 씩 가져올 수 있고,  
3 with open 사용시 close를 안 사용해도 됩니다.  
4 """  
5 with open("test.txt", "r") as f:  
6     line = f.readline()  
7     while line:  
8         print(line)  
9         line = f.readline()
```

텍스트 파일을 새로 생성합니다.

이 파일을 통해 파일 입출력이
어떻게 이루어지는 지 알 수 있습니다.

5. 함수

함수(함수의 기본구조) - 개념 이해

- 함수란 특정 기능을 수행하는 코드의 묶음
- 코딩 시 특정 기능을 반복해야 하는 상황일 때 주로 사용된다
- 수학에서의 $y=f(x)$ 와 비슷함. x 를 넣어서 특정 연산($f()$)을 거친 후 y 라는 결과값을 도출하는 형태

함수의 기본 구조

- 기본 구조는 옆의 그림과 같다
- 함수는 <함수명>(<인자>)형태로 호출한다
- 인자 값이나 반환 값이 없는 함수도 만들 수 있다
- 인자 값이 여러 개 일 수도 있고, 반환 값이 여러 개 일 수도 있다
 - 여러 개의 값이 들어오고 나갈 때는 그 수에 맞게 변수를 셋팅 해야 한다
 - 반환 값이 여러 개의 경우 변수 하나만 쓰면 tuple 형태로 값을 받는다

함수의 기본 구조

```
1 def <함수명> (<인자>):  
2     <코드1>  
3     <코드2>  
4     return <반환 값>
```

함수의 호출 방식

```
1 def hello():  
2     print("hello!")
```

```
1 hello()
```

hello!

함수에 보내는 인자를 맞추지 않았을 때

```
1 def addition(a,b):  
2     return a+b
```

```
1 addition(10)
```

```
TypeError                                 Traceback (most recent call last)  
<ipython-input-6-dca566d4b68d> in <module>  
----> 1 addition(10)
```

TypeError: addition() missing 1 required positional argument: 'b'

함수의 반환 값을 변수 하나로 받았을 때

```
1 def lotto():  
2     return 2, 6, 8, 26, 43, 45
```

```
1 a=lotto()  
2 print(a)
```

(2, 6, 8, 26, 43, 45)

함수(함수의 기본구조) - 코딩 따라하기

- 몇 가지 간단한 함수를 선언해보고, 다양한 방법으로 호출해보면서 함수 개념 익히기
- 함수 선언 시 인자 값에 default 값을 선언해줄 수 있다. 이 경우 인자를 보내지 않아도 함수 사용이 가능

```
1 def four_op(a, b):
2     return a+b, a-b, a*b, a/b
3
4 def mul_power(a, b):
5     return a ** b
6
7 def pythagorean(a, b, c):
8     if (a**2) + (b**2) == (c**2):
9         print("이 삼각형은 직각 삼각형입니다.")
10    else:
11        print("이 삼각형은 직각 삼각형이 아닙니다.")
```

```
1 four_op(10, 5)
```

```
1 #반환 값의 갯수가 다를경우 에러가 발생합니다.
2 a, b, c = four_op(10, 5)
```

```
1 mul_power(4, 0.5)
```

```
1 pythagorean(3,4,5)
```

```
1 #인자를 보내줄 때도 설정한 갯수와 맞지 않을 경우 에러가 발생합니다.
2 pythagorean(5,10)
```

```
1 #인자에 디폴트를 설정하면 함수 호출 시 인자를 안 보내줘도 정상 작동 합니다.
2 def hello(name="그런데 누구시죠?"):
3     print(f"hello {name}!")
```

```
1 hello("인사여!")
```

```
1 hello()
```

함수(함수의 기본구조) - 결과 확인

```
1 def four_op(a, b):
2     return a+b, a-b, a*b, a/b
3
4 def mul_power(a, b):
5     return a ** b
6
7 def pythagorean(a, b, c):
8     if (a**2) + (b**2) == (c**2):
9         print("이 삼각형은 직각 삼각형입니다.")
10    else:
11        print("이 삼각형은 직각 삼각형이 아닙니다.")
```

```
1 four_op(10, 5)
```

(15, 5, 50, 2.0)

```
1 #반환 값의 갯수가 다를경우 에러가 발생합니다.
2 a, b, c = four_op(10, 5)
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-3-bb14aef8769b> in <module>
      1 #반환 값의 갯수가 다를경우 에러가 발생합니다.
----> 2 a, b, c = four_op(10, 5)
```

ValueError: too many values to unpack (expected 3)

```
1 mul_power(4, 0.5)
```

2.0

```
1 pythagorean(3,4,5)
```

이 삼각형은 직각 삼각형입니다.

```
1 #인자를 보내줄 때도 설정한 갯수와 맞지 않을 경우 에러가 발생합니다.
2 pythagorean(5,10)
```

```
TypeError                                Traceback (most recent call last)
<ipython-input-6-ef29194ad6f8> in <module>
      1 #인자를 보내줄 때도 설정한 갯수와 맞지 않을 경우 에러가 발생합니다.
----> 2 pythagorean(5,10)
```

TypeError: pythagorean() missing 1 required positional argument: 'c'

```
1 #인자에 디폴트를 설정하면 함수 호출 시 인자를 안 보내줘도 정상 작동 합니다.
2 def hello(name="그런데 누구시죠?"):
3     print(f"hello {name}!")
```

```
1 hello("인사이저")
```

hello 인사이저!

```
1 hello()
```

hello 그런데 누구시죠?!

함수(변수의 유효 범위) - 개념 이해

- 변수는 어디서 선언되고 쓰이는냐에 따라 사용할 수 있는 범위가 달라진다
- 공간의 종류에는 지역 영역(Local) / 전역 영역(Global) / 내장 영역(Built-in)이 있다

지역 변수

- 주로 함수 내부에서 선언되어 사용되는 변수
- 함수가 종료되면 변수도 제거된다

전역 변수

- 코드 파일 내에서 선언되어 사용되는 변수
- 코드 파일을 끄기 전까지 계속 유지된다
- 함수 내부에서도 호출 가능

내장 영역

- 내장 함수, 변수가 있는 곳
- 선언이나 import를 하지 않아도 사용 가능

스코핑 룰(LGB 룰)

- 파이썬이 변수를 읽는 순서, 규정
- 지역 영역 -> 전역 영역 -> 내장영역 순으로 읽는다

지역 변수와 전역 변수

```
1 a = 100
2
3 def func():
4     a = 1
5     print("함수 안에서 선언 및 출력! :",a)
6
7 func()
8 print("함수 밖에서 출력! :",a)
```

함수 안에서 선언 및 출력! : 1
함수 밖에서 출력! : 100

전역 변수를 함수 안에서 사용

```
1 a = 100
2
3 def func():
4     print("함수 안에서 출력! :",a)
5
6 func()
```

함수 안에서 출력! : 100

함수(변수의 유효 범위) - 코딩 따라하기

- 전역 변수와 지역 변수를 다뤄보면서 변수 선언 위치, 변수명 선정에 대한 중요성 이해하기
- global 명령어로 전역변수를 함수 내부에서 호출/수정 할 수 있다

```
1 # 지역 변수를 함수 밖에서 사용, 에러 발생
2 def func():
3     var = 100
4     print(var)
5 func()
6 print(var)
```

```
1 #지역 변수와 전역 변수 명이 같을 지라도 다른 변수로 취급된다
2 var = [1,2,3,4,5]
3 def func():
4     var = [1,2,3]
5     print(var) #지역변수 출력
6 func()
7 print(var) #전역변수 출력
```

```
1 #함수 내에 선언된 변수 명이 없을 때는 전역변수를 탐색
2 var = [1,2,3,4,5]
3 def func():
4     # var = [1,2,3]
5     print(var)
6 func()
```

```
1 # global 명령어를 통해 전역변수를 수정할 수 있다
2 var = [1,2,3,4,5]
3 def func():
4     global var
5     print(var)
6     var[2] = 100 #전역 변수 값 수정
7     print(var)
8 func()
9 print(var)
```

함수(변수의 유효 범위) - 결과 확인

```
1 # 지역 변수를 함수 밖에서 사용, 에러 발생
2 def func():
3     var = 100
4     print(var)
5 func()
6 print(var)
```

100

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-1-64432f19a885> in <module>
      4     print(var)
      5 func()
----> 6 print(var)
```

NameError: name 'var' is not defined

```
1 #지역 변수와 전역 변수 명이 같을 지라도 다른 변수로 취급된다
2 var = [1,2,3,4,5]
3 def func():
4     var = [1,2,3]
5     print(var) #지역변수 출력
6 func()
7 print(var) #전역변수 출력
```

[1, 2, 3]

[1, 2, 3, 4, 5]

```
1 #함수 내에 선언된 변수 명이 없을 때는 전역변수를 탐색
2 var = [1,2,3,4,5]
3 def func():
4     # var = [1,2,3]
5     print(var)
6 func()
```

[1, 2, 3, 4, 5]

```
1 # global 명령어를 통해 전역변수를 수정할 수 있다
2 var = [1,2,3,4,5]
3 def func():
4     global var
5     print(var)
6     var[2] = 100 #전역 변수 값 수정
7     print(var)
8 func()
9 print(var)
```

[1, 2, 3, 4, 5]

[1, 2, 100, 4, 5]

[1, 2, 100, 4, 5]

함수(람다 함수와 내장 함수) - 개념 이해

- 람다(lambda) 함수는 함수를 따로 선언하지 않고 함수를 표현하는 기법
- 내장 함수는 파이썬에 이미 내장되어 있어 따로 선언하지 않고도 사용할 수 있는 함수

람다(lambda) 함수

- 기본 구조는 아래의 그림과 같다
- 단순한 계산의 경우, 람다 함수를 통해 간결하게 만들 수 있다
- 여러 개의 인자를 보낼 수도 있다

람다 함수

```
1 lambda <인자> : <인자를 포함한 식>
```

일반 함수와 람다 함수

```
1 #일반 함수
2 def circle(x):
3     area = 3.14 * (x**2)
4     return area
5
6 #람다 함수
7 circle = lambda x : 3.14 * (x**2)
```

유용한 내장 함수 목록

- 형 변환 함수
 - <타입명>(<값>)
 - 데이터 모음 형태 끼리, 혹은 숫자형 끼리 변환 가능
 - 형태가 맞지 않으면 에러 발생
- 최솟값, 최댓값 함수
 - max(<값>), min(<값>)
 - 데이터 모음 형태가 들어간다(list, set, tuple 등)
- 절대값, 전체 합 함수
 - abs(<값>), sum(<값>)
 - 절대값의 경우 음/양 부호를 뺀 값의 크기를 리턴(반올림 X)
 - 전체 합은 주로 숫자형 데이터가 모인 list, set, tuple에 쓰임
- 항목의 개수를 구하는 함수
 - len(<값>)
 - 문자열 및 데이터 모음 내 데이터의 개수를 리턴

함수(람다 함수와 내장 함수) - 코딩 따라하기

- 람다 함수 및 내장 함수를 다양하게 사용하여 개념 이해하기

```
1 def calc(x):  
2     result = x**2 + 10  
3     return result  
4  
5 calc(10)
```

```
1 calc = lambda x: x**2 + 10  
2 calc(10)
```

```
1 #input은 문자열 형태로 값을 받으므로, 숫자로 사용하고 싶을 경우 형 변환을 거쳐야 합니다.  
2 num = input("숫자 입력 : ")  
3 #에러 발생  
4 print(100 + num)
```

```
1 num = int(input("숫자 입력 : "))  
2 print(100 + num)
```

```
1 #숫자 형태의 문자열이 아닌 경우, int, float으로 형 변환이 안됩니다.  
2 print(int("100000"))  
3 #에러 발생  
4 print(int("백만원"))
```

```
1 #list, tuple에서 set으로 형 변환 할 경우 중복 값이 제거됩니다.  
2 a = [1,2,3,1,2,3]  
3 print(a)  
4 print(set(a))
```

```
1 a = [1, 100, 0.1, 1000, 50, 99]  
2 print(max(a))  
3 print(min(a))  
4 print(sum(a))  
5 print(len(a))  
6  
7 #len()은 문자열의 갯수도 셀 수 있습니다.  
8 print(len("특수 문자 및 공백의 갯수까지 전부 포함됩니다."))
```

```
1 a = -26  
2 b = -26.5  
3 print(abs(a), abs(b))
```

함수(람다 함수와 내장 함수) - 결과 확인

```
1 def calc(x):
2     result = x**2 + 10
3     return result
4
5 calc(10)
```

110

```
1 calc = lambda x: x**2 + 10
2 calc(10)
```

110

```
1 #input은 문자열 형태로 값을 받으므로, 숫자로 사용하고 싶을 경우 형 변환을 거쳐야 합니다.
2 num = input("숫자 입력 : ")
3 #에러 발생
4 print(100 + num)
```

숫자 입력 : 100

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-3-6229e4c2b7a4> in <module>
      2 num = input("숫자 입력 : ")
      3 #에러 발생
----> 4 print(100 + num)
```

TypeError: unsupported operand type(s) for +: 'int' and 'str'

```
1 num = int(input("숫자 입력 : "))
2 print(100 + num)
```

숫자 입력 : 100
200

```
1 #숫자 형태의 문자열이 아닌 경우, int, float으로 형 변환이 안됩니다.
2 print(int("100000"))
3 #에러 발생
4 print(int("백만원"))
```

100000

```
-----
ValueError                                 Traceback (most recent call last)
<ipython-input-5-4ed3db1d0a7d> in <module>
      2 print(int("100000"))
      3 #에러 발생
----> 4 print(int("백만원"))
```

ValueError: invalid literal for int() with base 10: '백만원'

```
1 #list, tuple에서 set으로 형 변환 할 경우 중복 값이 제거됩니다.
2 a = [1,2,3,1,2,3]
3 print(a)
4 print(set(a))
```

[1, 2, 3, 1, 2, 3]
{1, 2, 3}

```
1 a = [1, 100, 0.1, 1000, 50, 99]
2 print(max(a))
3 print(min(a))
4 print(sum(a))
5 print(len(a))
6
7 #len()은 문자열의 갯수도 셀 수 있습니다.
8 print(len("특수 문자 및 공백의 갯수까지 전부 포함됩니다."))
```

1000
0.1
1250.1
6
26

```
1 a = -26
2 b = -26.5
3 print(abs(a), abs(b))
```

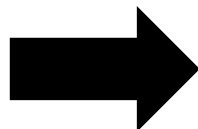
26 26.5

6. 실습

실습 (성적관리테이블)_예제 설명

- 엑셀로 정리된 성적 데이터 읽기, 위에서 배운 파이썬 문법을 토대로 데이터 다뤄 보기
- 이후 판다스 데이터프레임을 다루기 위한 초석 다지기
- https://drive.google.com/file/d/13LNdFXDYDXtI_Jgipm4Bao1-9VQHmd7C1/view?usp=sharing

	A	B	C	D	E	F
1	이름	성별	반	국어	영어	수학
2	주원	남	1	79	73	70
3	지후	남	1	97	77	96
4	도현	남	1	72	61	96
5	선우	남	1	98	62	97
6	은우	남	1	50	70	76
7	준영	남	1	58	67	64
8	지원	남	1	97	79	97
9	재민	남	1	65	95	63
10	은찬	남	1	97	85	59
11	예성	남	1	76	75	83
12	규민	남	1	52	86	79
13	지민	남	1	56	92	69
14	서연	여	1	51	57	55
15	시은	여	1	62	55	52
16	가은	여	1	75	80	64
17	시연	여	1	94	54	78
18	아윤	여	1	64	98	92
19	나현	여	1	66	77	90
20	도연	여	1	87	88	63
21	지훈	남	2	75	85	78
22	송민	남	2	83	53	89
23	민성	남	2	90	72	71
24	재윤	남	2	97	82	79
25	민규	남	2	67	62	77
26	민우	남	2	70	56	70
27	성현	남	2	99	90	71
28	민석	남	2	67	64	73
29	예찬	남	2	96	71	69
30	정현	남	2	88	54	54
31	채은	여	2	72	59	94
32	다인	여	2	50	93	88
33	현서	여	2	79	88	81
34	서율	여	2	77	77	53



	A	B	C	D	E	F
1	반	학생 수	평균 성적	초과/미달	분산	표준편차
2	1	19	74.84211	초과	94.44875	9.718475
3	2	16	74.85417	초과	43.99957	6.633217
4	3	23	75.37681	초과	57.85801	7.606445
5	4	12	71.86111	미달	39.8233	6.310571
6	5	21	75.80952	초과	39.73091	6.303246
7	6	24	75.27778	초과	90.40432	9.508119
8	7	12	77.38889	초과	83.27469	9.125497
9	8	21	74.8254	초과	65.28697	8.080036
10	9	20	73.91667	미달	46.70972	6.834451
11	10	32	72.25	미달	78.0625	8.835299

- 마지막 심화 과정 코드를 다음 주차에 해오시는 분께 소정의 상품을 드립니다!

3. 새로운 리스트 만들기(심화)

추출한 데이터를 새로운 리스트로 정제하고, txt파일로 만들어 보시다.

반 기준 데이터 리스트 만들기

```
1 from IncizerStudy01 import *
2 """
3 위에서 시행한 방법들을 총 동원해서 새로운 데이터를 구축해봅시다!
4 만들어야 할 리스트는 다음과 같습니다.
5 [
6 [1, 1반 학생수, 1반 평균 성적, 전체 평균 대비 초과/미달 라벨, 분산, 표준편차]
7 [2, 2반 학생수, 2반 평균 성적, 전체 평균 대비 초과/미달 라벨, 분산, 표준편차]
8 .
9 .
10 .
11 [10, 10반 학생수, 10반 평균 성적, 전체 평균 대비 초과/미달 라벨, 분산, 표준편차]
12 ]
13 """
14 #데이터 읽기
15 students = read_data()
16 all_avg = 74.52333333333334
17
18 #1반 ~ 10반 각각 데이터 추출
19 all_class = []
20 for idx in range(1,11):
21     tmp_list = [idx]
22
23     #반별 데이터 추출
24
25
26     #반 학생 수 구하기
27
28
29     #반 평균 성적 산출
30
31
32     #전체 평균 대비 초과/미달 라벨
33
34     #반 분산 구하기
35     variance = 0
36
37
38     #반 표준편차 구하기
39     std_dev = 0
40
41     #새로운 리스트에 반 데이터 넣기
42     all_class.append(tmp_list)
43
44 for data in all_class:
45     print(data)
```

실습 (성적관리테이블)_실습 진행(출력 답안)

1-1 전체 학생 수 구하기

200

1-2 학생 별 평균 성적 구하기

['민준', '남', 8, 85, 72, 72, 76.33333333333333, '초과']
['서준', '남', 3, 70, 58, 97, 75.0, '초과']
['예준', '남', 10, 97, 85, 60, 80.66666666666667, '초과']
['도윤', '남', 5, 73, 68, 93, 78.0, '초과']
['시우', '남', 8, 86, 68, 50, 68.0, '미달']
['주원', '남', 1, 79, 73, 70, 74.0, '미달']
['하준', '남', 3, 66, 90, 63, 73.0, '미달']
['지호', '남', 8, 51, 78, 74, 67.66666666666667, '미달']
['지후', '남', 1, 97, 77, 96, 90.0, '초과']
['준서', '남', 6, 55, 57, 57, 56.33333333333333, '미달']
['준우', '남', 8, 81, 71, 66, 72.66666666666667, '미달']
['현우', '남', 3, 75, 57, 57, 63.0, '미달']
['지훈', '남', 2, 75, 85, 78, 79.33333333333333, '초과']
['도현', '남', 1, 72, 61, 96, 76.33333333333333, '초과']
['건우', '남', 4, 64, 56, 92, 70.66666666666667, '미달']
['우진', '남', 10, 87, 79, 65, 77.0, '초과']
['민재', '남', 7, 87, 96, 73, 85.33333333333333, '초과']
['현준', '남', 10, 61, 62, 57, 60.0, '미달']
['선우', '남', 1, 98, 62, 97, 85.66666666666667, '초과']

1-4 전체 평균 성적에 초과/미달 인 학생 라벨링

['민준', '남', 8, 85, 72, 72, 76.33333333333333, '초과']
['서준', '남', 3, 70, 58, 97, 75.0, '초과']
['예준', '남', 10, 97, 85, 60, 80.66666666666667, '초과']
['도윤', '남', 5, 73, 68, 93, 78.0, '초과']
['시우', '남', 8, 86, 68, 50, 68.0, '미달']
['주원', '남', 1, 79, 73, 70, 74.0, '미달']
['하준', '남', 3, 66, 90, 63, 73.0, '미달']
['지호', '남', 8, 51, 78, 74, 67.66666666666667, '미달']
['지후', '남', 1, 97, 77, 96, 90.0, '초과']
['준서', '남', 6, 55, 57, 57, 56.33333333333333, '미달']
['준우', '남', 8, 81, 71, 66, 72.66666666666667, '미달']
['현우', '남', 3, 75, 57, 57, 63.0, '미달']
['지훈', '남', 2, 75, 85, 78, 79.33333333333333, '초과']
['도현', '남', 1, 72, 61, 96, 76.33333333333333, '초과']
['건우', '남', 4, 64, 56, 92, 70.66666666666667, '미달']
['우진', '남', 10, 87, 79, 65, 77.0, '초과']
['민재', '남', 7, 87, 96, 73, 85.33333333333333, '초과']
['현준', '남', 10, 61, 62, 57, 60.0, '미달']
['선우', '남', 1, 98, 62, 97, 85.66666666666667, '초과']

2-1 10반 학생 수 구하기

32

2-2 10반이 전체 평균 성적에 초과인지 미달인지 확인

미달

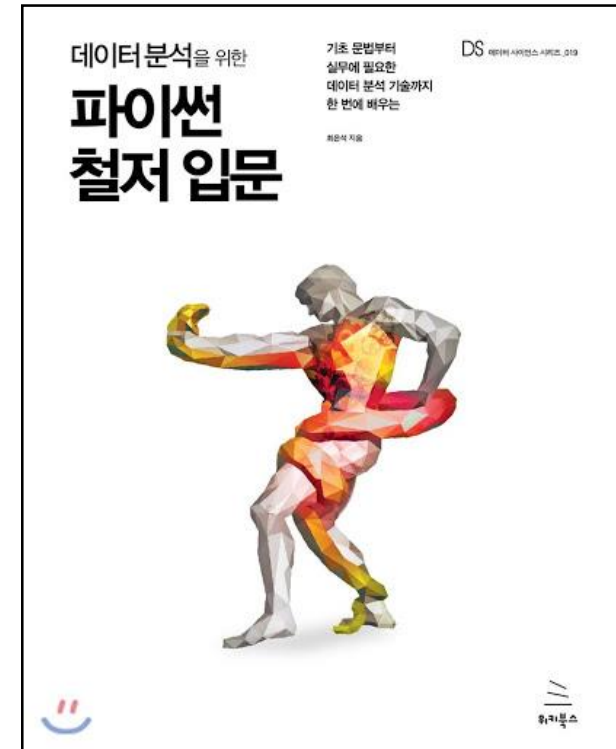
2-2 10반 평균 성적의 분산, 표준편차 구하기

78.06250000000003
8.835298523536148

1-3 전체 학생의 평균 성적 구하기

74.52333333333334

- 점프 투 파이썬
 - <https://wikidocs.net/book/1>
- 왕초보를 위한 파이썬
 - <https://wikidocs.net/43>
- 데이터 분석을 위한 파이썬 철저 입문(교재)
 - <https://wikibook.co.kr/python-for-data-analysis/>



감사합니다