

DAY 1

Python

Day1

- 파이썬 소개
- 파이썬의 기본
- 파이썬의 자료형
- 조건문
- 반복문
- 파일쓰기
- 파일읽기
- 함수

파이썬 소개

파이썬?

- 파이썬 소개
- 다른 언어와 비교
- 개발환경 구축하기

파이썬?



1991년 구도 반 로섬(Guido van Rossum)이 발표

why 파이썬?

1. General-purpose

- 시스템 유ти리티
- GUI 프로그래밍
- C/C++ 과의 결합
- 웹 프로그래밍
- 데이터 베이스
- 데이터 분석
- 사물인터넷

why 파이썬?

2. 간결하고 쉬운 문법

(1) 간결함

C	JAVA	Python
int array[2]; array[0] = 1; array[1] = 2;	int[] array = new int[2]; array[0] = 1; array = 2;	array = [1,2]

(2) 들여쓰기(Indentation)

C	JAVA	Python
for (int a=0; a<3; a++) { print(a) }	for (int a=0; a<3; a++){ System.out.println(a) }	for a in range(3) : print(a)

why 파이썬?

3. 현업에서 사용 중



Gmail



Dropbox

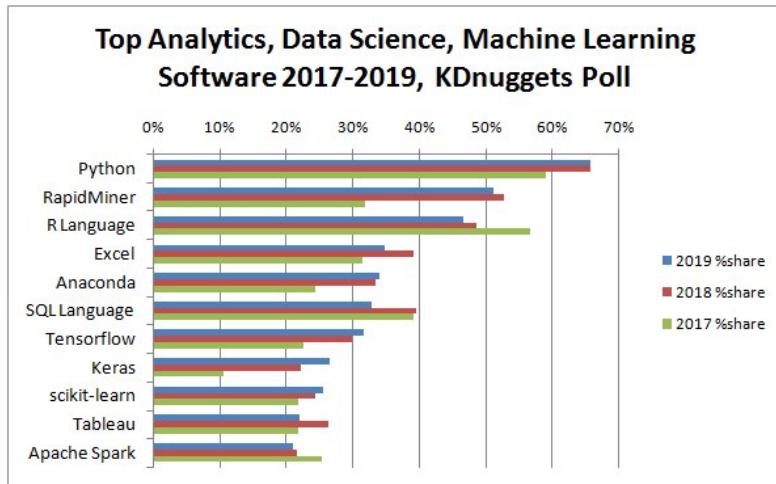


Instagram

- 구글, 드롭박스, 인스타그램 등 현업에서 사용 중
- 미국 Top 10 컴퓨터공학과 중 80%는 컴퓨터언어 입문강좌로 C가 아닌 Python을 채택

why 파이썬?

3. 현업에서 사용 중



Software	2019 % share	2018 % share	2017 % share
Python	65.8%	65.6%	59.0%
RapidMiner	51.2%	52.7%	31.9%
R Language	46.6%	48.5%	56.6%
Excel	34.8%	39.1%	31.5%
Anaconda	33.9%	33.4%	24.3%
SQL Language	32.8%	39.6%	39.2%
Tensorflow	31.7%	29.9%	22.7%
Keras	26.6%	22.2%	10.7%
scikit-learn	25.5%	24.4%	21.9%
Tableau	22.1%	26.4%	21.8%
Apache Spark	21.0%	21.5%	25.5%

파이썬 개발환경



ANACONDA[®]

- 손쉬운 패키지 설치
- 가상 환경을 통한 패키지 버전관리가 용이

파이썬 개발환경



- web을 통한 접근 가능
- Coding 결과를 실시간으로 확인 가능
- 자동완성 기능
- 다양한 언어 지원
- markdown을 통해 문서화 가능

ANACONDA 설치

<https://www.anaconda.com/products/individual>

Anaconda Installers



Python 3.8

[64-Bit Graphical Installer \(466 MB\)](#)

[32-Bit Graphical Installer \(397 MB\)](#)



Python 3.8

[64-Bit Graphical Installer \(462 MB\)](#)

[64-Bit Command Line Installer \(454 MB\)](#)

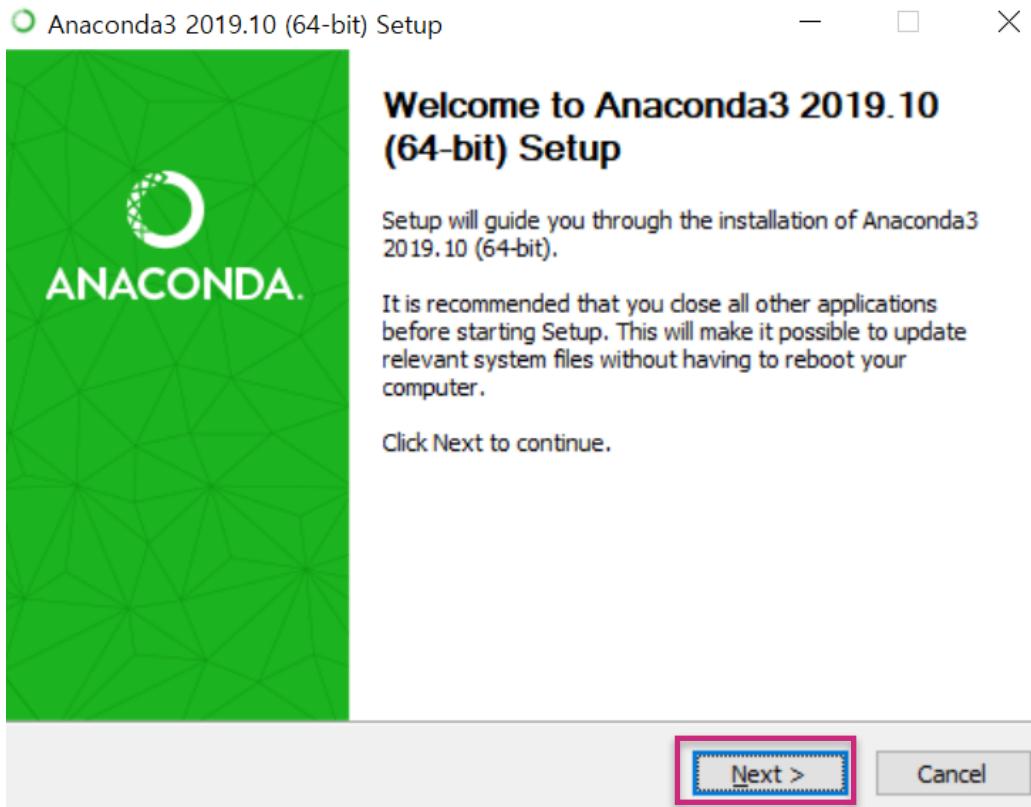


Python 3.8

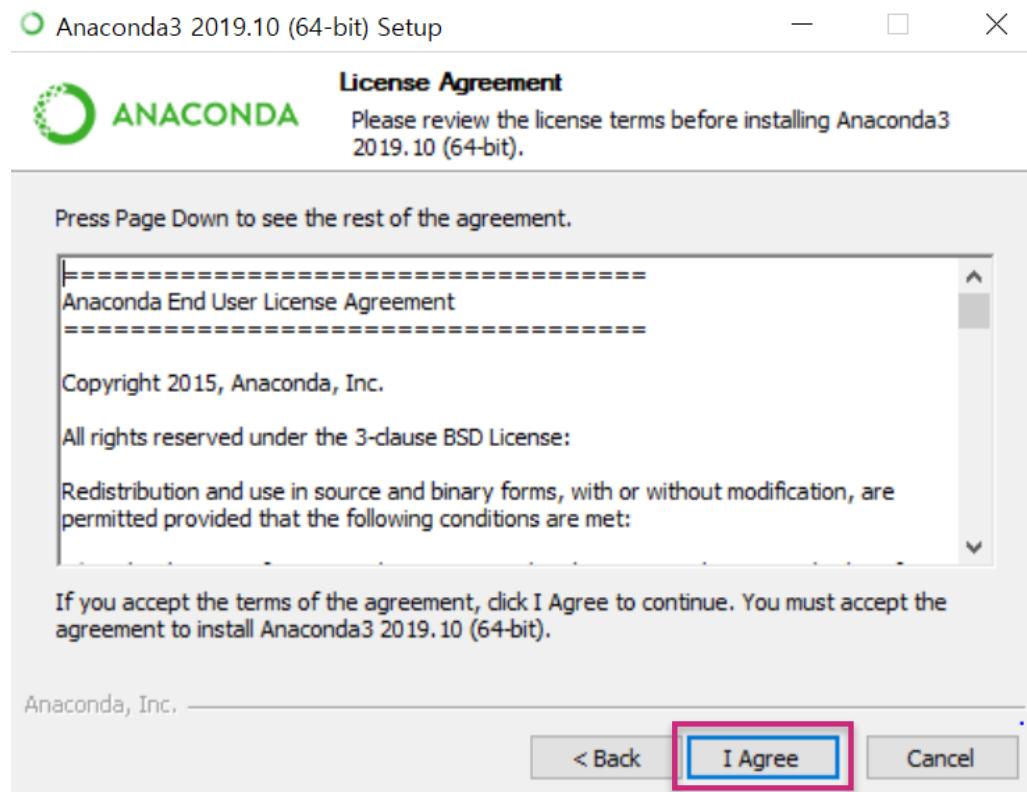
[64-Bit \(x86\) Installer \(550 MB\)](#)

[64-Bit \(Power8 and Power9\) Installer \(290 MB\)](#)

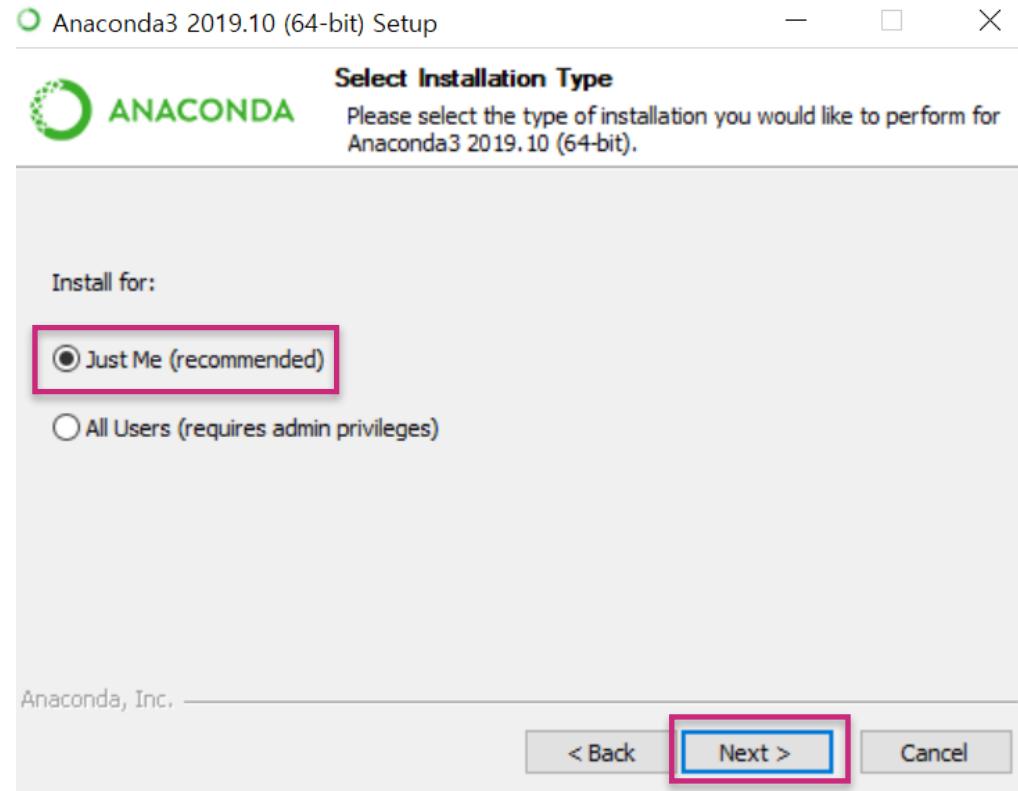
ANACONDA 설치



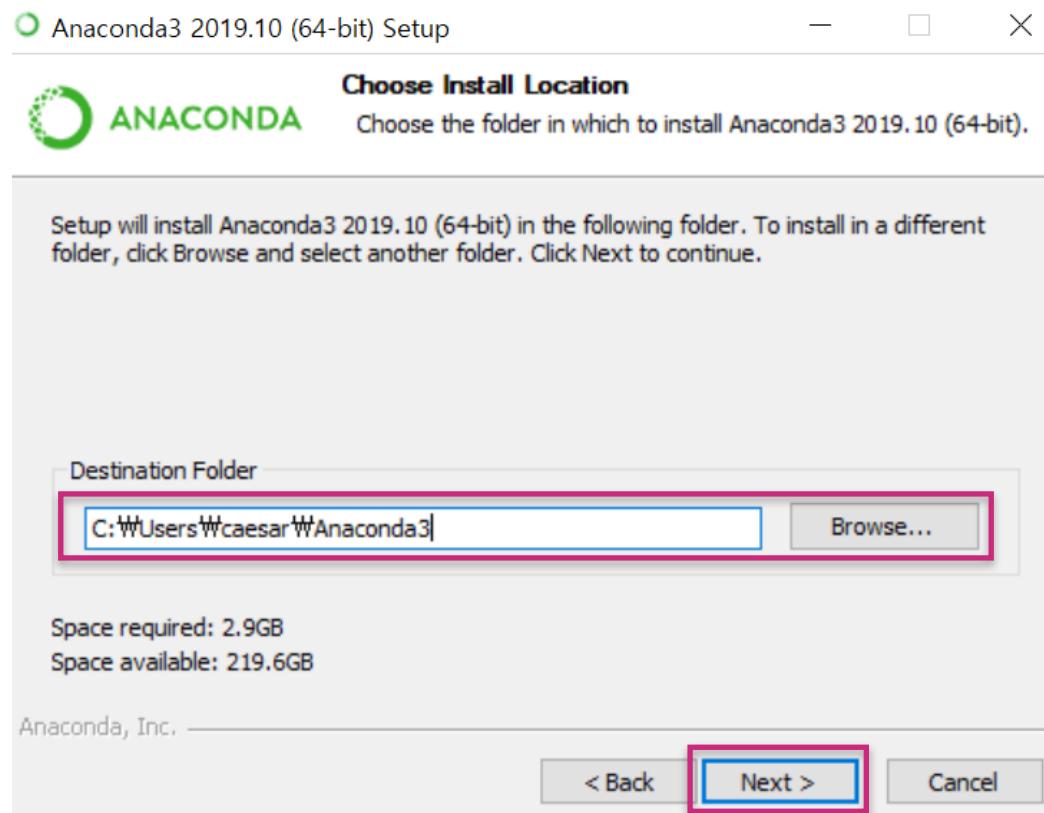
ANACONDA 설치



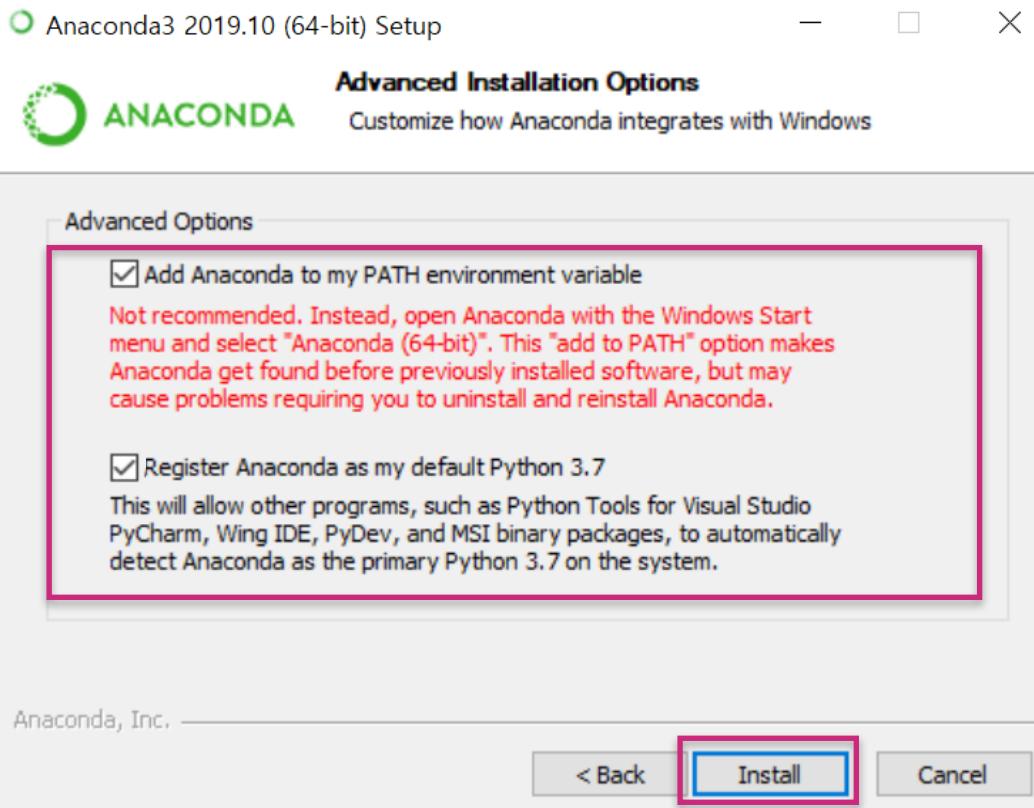
ANACONDA 설치



ANACONDA 설치

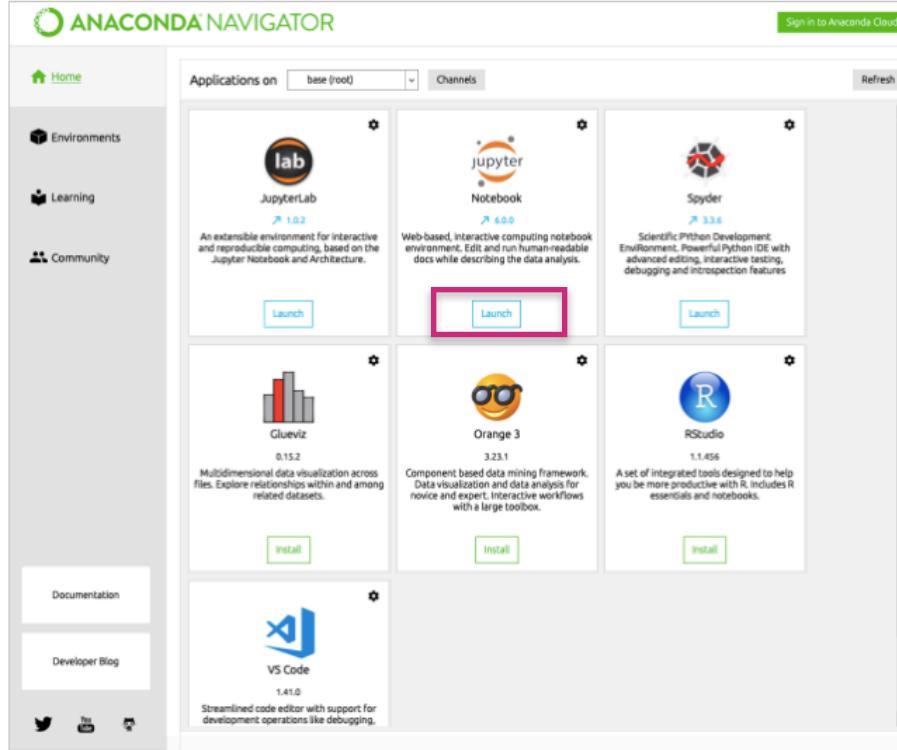


ANACONDA 설치



Jupyter Notebook 사용하기

- 아나콘다 네비게이터 실행



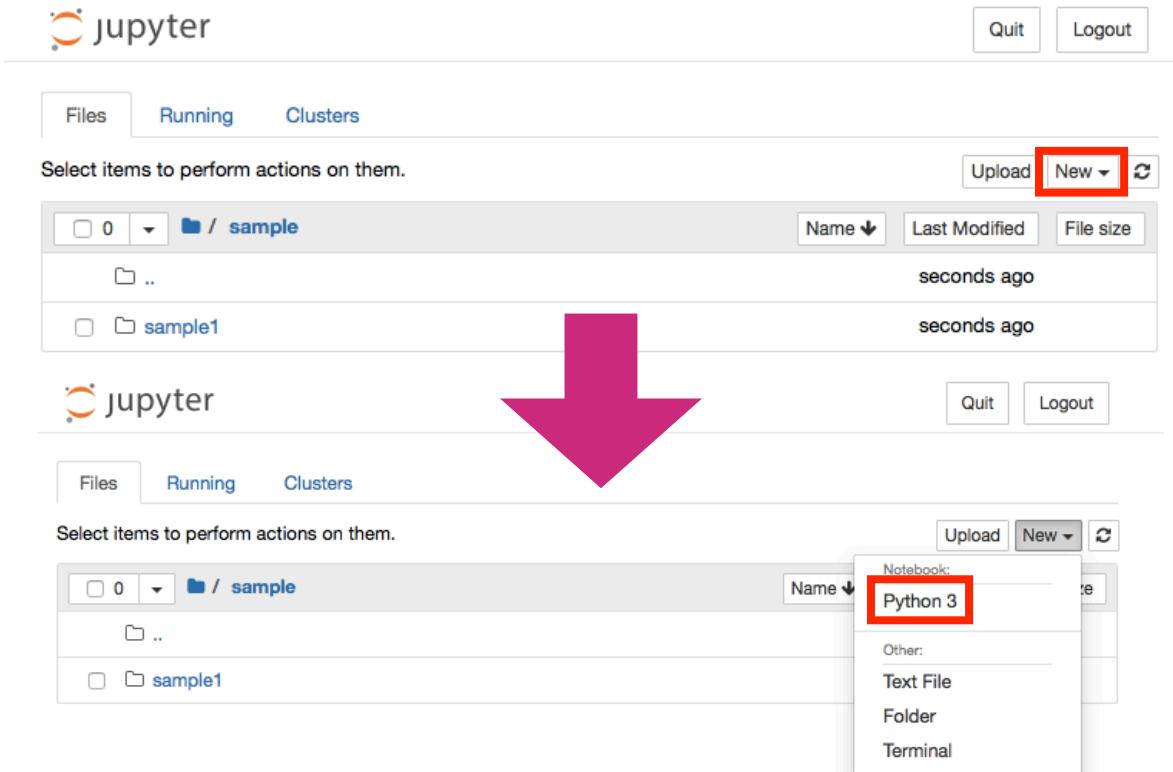
Jupyter Notebook 사용하기

- Anaconda Prompt 실행 후 아래 명령어 실행
→ jupyter notebook

```
Last login: Wed Aug  5 15:41:48 on ttys000
→ ~ jupyter notebook
```

Jupyter Notebook 사용하기

- 새로운 노트북 생성하기



Jupyter Notebook 사용하기

- 유용한 단축키

```
In [ ]: # Edit mode : Enter  
# Command mode : ESC
```

```
In [ ]: # 셀 추가 (Command mode에서)  
# 현재 셀 위로 : a  
# 현재 셀 아래로 : b
```

```
In [ ]: # 셀 삭제 (Command mode에서) : dd
```

```
In [ ]: # 선택한 셀을  
# 복사하기 : c  
# 잘라내기 : x
```

```
In [ ]: # 셀 붙여넣기  
# 선택한 셀 위로 : Shift + v  
# 선택한 셀 아래로 : v
```

```
In [ ]: # 셀에 라인 추가하기 (Command mode에서) : l
```

Jupyter Notebook 사용하기

- 유용한 단축키

```
In [ ]: # 셀 변경하기 (Command mode에서)  
# code : y  
# markdown : m
```

```
In [ ]: # 해당 셀을 실행하기 : Ctrl + Enter
```

```
In [ ]: # 해당 셀을 실행하고 아래에 셀 선택 : Shift + Enter
```

```
In [ ]: # 코드 자동 완성 : Tab  
str.|
```

capitaliz

```
In [1]: str| casefold
```

center

count

encode

endswith

expandtabs

find

format

format_map

Jupyter Notebook 사용하기

- 유용한 단축키
 - 함수나 변수에 대한 설명1 : Shift + Tab

In []: str

In []: Init signature: str(self, /, *args, **kwargs)
Docstring:
In []: str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

A screenshot of a Jupyter Notebook interface. In the top cell, 'str' is typed. A tooltip-like pop-up window appears, showing the function signature 'Init signature: str(self, /, *args, **kwargs)' and the docstring 'Docstring'. Below this, another cell shows the implementation of the str() function for different types.

- 함수나 변수에 대한 설명2 : ? 셀을 실행 해야함

In [1]: str?

Init signature: str(self, /, *args, **kwargs)
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

A screenshot of a Jupyter Notebook interface. In the top cell, 'str?' is typed. The bottom cell displays the same help information as the previous screenshot, including the function signature, docstring, and implementation details for str() and str().

Jupyter Notebook 사용하기

Jupyter 맛보기

markdown

문서처럼 작성할 수도 있습니다.

```
In [2]: print("당연히 코딩도 할 수 있습니다.")
```

당연히 코딩도 할 수 있습니다.

```
In [7]: from IPython.display import Image  
Image(url="data:image/jpeg;base64,/9j/4AAQSkZJRqABAQAAAQABAAAD/2wCEAAkGBxMTEhUTExiWFhUWGBkYC
```

Out[7]:



파이썬 기본

파이썬의 기본

- 산술 연산자

1. 더하기 : +

[] 1 + 2



3

[] 250.7 + 300.2

[] 550.9



550.9

▼ 2. 빼기 : -

[] 5 - 3



2

[] 250.5 - 100.3



150.2

▼ 3. 곱하기 : *

[] 5 * 3



15

[] 20.5 * 2



41.0

▼ 4. 나누기 : /

[] 10 / 2



5.0

[] 7 / 3



2.3333333333333335

파이썬의 기본

- 산술 연산자

- 4. 나누기 : /

```
[ ] 10 / 2
```

5.0

```
[ ] 7 / 3
```

2.3333333333333335

- 5. //

```
[ ] 10 // 2
```

5

```
[ ] 7 // 3
```

2

/의 연산 결과 : 소수점 표시

→ 실수형

//의 연산 결과 : 소수점 없음(나누기의 몫) → 정수형

파이썬의 기본

- 산술 연산자

- 6. 나머지 : %

```
[ ] 10 % 2
```



0

```
[ ] 7 % 3
```



1

- 7. 그 밖에 연산자

제곱 : **

```
[ ] 2 ** 3
```



8

```
[ ] 5 ** 3
```



125

파이썬의 기본

- 연산자의 우선순위

사칙연산의 연산자 우선순위와 같음
괄호()가 있다면 가장 우선 처리

[] 5 + 3 * 4

17

[] (8 - 3) * 2

10

연습문제 1

파이썬의 기본

- 할당 연산자(Assignment) : =

변수 = 연산/조건/함수

할당 연산자 우측을 연산을 한 후, 그 결과를 좌측에 있는 변수에 할당(저장)

파이썬의 기본

- 할당 연산자(Assignment) : =

```
[ ] result = 1 + 2
```

```
[ ] print(result)
```

3

```
[ ] result
```

3

할당 연산자 우측을 연산을 한 후, 그 결과를 좌측에 있는 변수에 할당(저장)

파이썬의 기본

- 할당 연산자(Assignment) : =

```
[ ] result += 1  
      result
```



4

```
[ ] result = result + 1
```

산술 연산자와 할당 연산자를 함께 쓸 수 있음
코드를 더욱 간결하게 쓸 수 있음

파이썬의 기본

- 할당 연산자(Assignment) : $=$

```
[ ] result += 1  
result
```



4

```
[ ] result -= 5  
result
```



-1

```
[ ] result *= -1  
result
```



1

```
[ ] result /= 5  
result
```



0.2

파이썬의 기본

- 할당 연산자(Assignment) : **=**

단순 산술 연산이 아닌 변수간의 연산도 가능

```
[ ] a = 5  
    b = 7  
    c = a + b  
    print(c)
```



12

주의 파이썬은 대소문자를 구분함

```
[ ] c = A + B
```



```
NameError                                 Traceback (most recent call last)  
<ipython-input-26-b698ae2737d5> in <module>  
----> 1 c = A + B
```

```
NameError: name 'A' is not defined
```

파이썬의 기본 연산자

- 산술 연산자
- 할당 연산자

연습문제 2

파이썬의 자료형

파이썬의 자료형

- 숫자(int, float)
- 문자(string)
- 리스트(list)
- 튜플(tuple)
- 딕셔너리(dict)
- 셋(set)
- 불(bool)

숫자(int, float)

- int와 float

```
[5] a = 3  
    type(a)
```

⇨ int

```
[6] b = 5.0  
    type(b)
```

⇨ float

int는 정수, float는 실수

type(변수) : 변수의 자료형을 알려주는 함수

숫자(int, float)

- int와 float

```
[7] c = a + b  
    c
```

↳ 8.0

```
[8] type(c)      int + float 는 float
```

↳ float

숫자(int, float)

- int와 float

```
[9] d = int(c)
    print(d)
    type(d)
```

↳ 8
int

```
[10] e = float(5)
      print(e)
      type(e)
```

↳ 5.0
float

숫자는 형 변환을 할 수 있음

문자(string)

- string

“(작은 따옴표) 혹은 “”(큰 따옴표)을 이용하여 문자형(string)임을 표시

```
[11] f = '문자형 예시입니다.'  
      print(f)  
      type(f)
```

⇒ 문자형 예시입니다.
str

문자(string)

[12] 따옴표 = '문장안에 따옴표를 쓰고 싶다면? """ 번갈아 쓰면 됩니다.'
print(따옴표)

☞ 문장안에 따옴표를 쓰고 싶다면? "" 번갈아 쓰면 됩니다.

[13] escape = "escape 문자인 \" \\ \"(역슬래시 혹은 원화표시)도 사용가능"
print(escape)

☞ escape 문자인 " \" (역슬래시 혹은 원화표시)도 사용가능

문자(string)

```
[21] g = 3  
     h = '5'  따옴표가 있으므로 string
```

```
[22] type(g)
```

↳ int

```
[23] type(h)
```

↳ str

문자(string)

```
[21] g = 3  
     h = '5'
```

```
[24] i = str(g)  
      print(i)  
      type(i)
```

```
↪ 3  
str
```

```
[ ] j = int(h)  
print(j)  
type(j)
```

```
↪ 5  
int
```

str와 int도 형 변환이 가능

문자(string)

- string의 연산

```
[26] str_1 = '문자'  
     str_2 = '의 연산'  
     result = str_1 + str_2  
     result
```

⇨ '문자의 연산'

```
[27] str_3 = '더하기'  
     result += str_3  
     result
```

⇨ '문자의 연산더하기'

더하기를 하면 두 문자열을 붙여줌

문자(string)

- string의 연산

```
[28] str_4 = '곱하기'  
      print(str_4 * 5)
```

⇨ 곱하기곱하기곱하기곱하기곱하기

```
[29] str_5 = '반복'  
      print((str_4 + str_5)*2)
```

⇨ 곱하기반복곱하기반복

곱하기는 그 수만큼 문자열을 반복함

문자(string)

- string의 연산

단, 자료형이 다르다면 연산이 불가

```
[31] int_1 = 8
     str_6 = '문자'
     error = str_6 + int_1
```

```
↳ -----
TypeError                                         Traceback (most recent call last)
<ipython-input-31-1a7e46269a93> in <module>()
      1 int_1 = 8
      2 str_6 = '문자'
----> 3 error = str_6 + int_1

TypeError: must be str, not int
```

SEARCH STACK OVERFLOW

문자(string)

- string의 indexing(인덱싱)

인덱싱? 순서가 있는 변수의 특정 위치에 접근 할 수 있게 해줌

변수[위치]

주의 파이썬의 인덱싱은 0부터 시작

문자(string)

- string의 indexing(인덱싱)

인덱싱? 순서가 있는 변수의 특정 위치에 접근 할 수 있게 해줌

변수[위치]

```
[47] str_5 = '인덱싱을 하기 위한 string입니다.'  
      len(str_5)
```

↳ 21

len(변수)는 변수의 길이를 재는 함수

```
[48] str_5[2]
```

↳ '싱'

문자(string)

- string의 slicing(슬라이싱)

슬라이싱? 변수의 일정 부분에 접근하여 잘라옴

변수[시작 : 끝 : 간격]

주의

- 슬라이싱은 시작부터 끝-1까지 가져옴
- 간격이 1이라면 생략해도 됨

문자(string)

- string의 slicing(슬라이싱)

슬라이싱? 변수의 일정 부분에 접근하여 잘라옴

변수[시작 : 끝 : 간격]

```
[49] str_5 = '인덱싱을 하기 위한 string입니다.'
      str_5[2:7]
```

⇒ '싱을 하기'

문자(string)

- string의 slicing(슬라이싱)

슬라이싱? 변수의 일정 부분에 접근하여 잘라옴

변수[시작 : 끝 : 간격]

```
[50] str_6 = '안녕하세요. 홍길동 고객님!'
```

```
[52] str_6[7:10]
```

↳ '홍길동'

문자(string)

- string의 slicing(슬라이싱)

```
[54] str_6 = '안녕하세요. 홍길동 고객님!'
      print(str_6[:2])
```

↳ 안녕

→ 시작점이 없다면 처음부터

```
[57] print(str_6[11:])
```

↳ 고객님!

→ 끝이 없다면 끝까지

```
[58] print(str_6[:])
```

↳ 안녕하세요. 홍길동 고객님!

→ 시작과 끝이 없다면 전체

문자(string)

- string의 slicing(슬라이싱)

파이썬은 마이너스 인덱싱을 지원함

```
[60] str_6 = '안녕하세요. 홍길동 고객님!'
      print(str_6[-1])
```

→ -1은 끝에서부터 거꾸로

⇒ !

```
[63] print(str_6[-4:])
```

⇒ 고객님!

```
[64] print(str_6[:-2])
```

⇒ 안녕하세요. 홍길동 고객

문자(string)

- string의 slicing(슬라이싱)

파이썬은 마이너스 인덱싱을 지원함

```
[65] str_6 = '안녕하세요. 홍길동 고객님!'
      print(str_6[::-1])
```

→[::-1]은 str전체를 뒤집음

⇨ !님객고 동길홍 .요세하녕안

문자(string)

- string의 method

method? 파이썬의 객체가 가지고 있는 고유의 함수
.을 이용하여 사용

1. 대소문자 변환 : lower, upper

```
[66] str_7 = 'Alphabet'  
      print(str_7.lower())
```

↳ alphabet

```
[67] print(str_7.upper())
```

↳ ALPHABET

문자(string)

- string의 method
2. 해당 문자의 갯수 세기 : count

```
[68] str_7 = 'Alphabet'  
     str_7.count('a')
```

→ count는 대소문자를 구분함

⇨ 1

Quiz! str_7에서 대소문자 구분없이 문자의 갯수를 세려면?

```
[71] str_7.lower().count('a')
```

⇨ 2

문자(string)

- string의 method

3. 문자열의 위치 찾기

→ 해당하는 문자열의 위치(인덱스)를 반환

1) find

```
[72] str_7 = 'Alphabet'  
      str_7.find('t')
```

↳ 7

2) index

```
[73] str_7.index('t')
```

↳ 7

문자(string)

- string의 method

3. 문자열의 위치 찾기

1) find

```
[74] str_7 = 'Alphabet'  
      str_7.find('z')
```

↳ -1

→문자열에 없으면 -1 반환

2) index

```
[75] str_7.index('z')
```

↳ -----

ValueError: substring not found

→문자열에 없으면 error

문자(string)

- string의 method

4. 문자열 바꾸기 : replace

```
[82] str_8 = 'Life is C between B and D'  
      str_8.replace('C', 'Chicken')
```

↳ 'Life is Chicken between B and D'

5. 문자열 나누기 : split

```
[83] str_8.split(' ')
```

→따옴표 ‘ ’사이에 기준이 되는 문자를 넣기
생략 시 기준은 띄어쓰기

↳ ['Life', 'is', 'C', 'between', 'B', 'and', 'D']

문자(string)

- string의 method

6. 문자열 삽입 : join

```
[84] str_9 = 'abcd'  
     ','.join(str_9)
```

↳ 'a,b,c,d'

→ 따옴표 ''사이에 삽입할 문자를 넣기

7. join과 split

```
[85] str_10 = str_8.split()
```

→ split한 결과에 join을 사용하면
다시 문자열로 만들 수 있음

```
[86] ' '.join(str_10)
```

↳ 'Life is C between B and D'

문자(string)

- string의 method

더 많은 메소드는 [python 공식문서](#)나 아래와 같은 방법으로 확인 가능

```
[21] print(dir(str_8))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__ge__attribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

리스트(list)

- list

[]을 이용하여 리스트(list)임을 선언

```
[1] list_1 = [3,2,5]
    list_2 = list()
```

```
[2] type(list_1)
```

⇨ list

리스트(list)

- list의 특징

리스트는 모든 자료형을 담을 수 있음

리스트는 삽입, 수정, 삭제가 자유로움

리스트는 순서가 있는 자료형 → 인덱싱과 슬라이싱이 가능

```
[4] list_3 = [1, 2, '문자', ['이중 리스트', '가능'], ('리스트 속', '튜플')]
```

```
[5] list_3[3]
```

↳ ['이중 리스트', '가능']

```
[6] list_3[4][1]
```

↳ '튜플'

주의 python의 인덱스는 0부터 시작

리스트(list)

- list의 인덱싱

```
[7] list_4 = [[1,2,3,4],  
             [2,4,6,8],  
             [3,6,9,12],  
             [4,8,12,16],  
             [5,10,15,20]]
```

가장 바깥쪽부터 괄호부터

```
[9] list_4[1][3]
```

앞에서부터 순서대로

리스트(list)

- list의 인덱싱

```
[ ] list_5 = [[[1,2,3],[4],[5,[6,7,8,9]]],  
             [10,11,[12,13,[14]]]]]
```

```
[ ] list_5[1][2][2]
```

```
↳ [14]
```

리스트(list)

- list의 슬라이싱

```
[7] list_4 = [[1,2,3,4],  
             [2,4,6,8],  
             [3,6,9,12],  
             [4,8,12,16],  
             [5,10,15,20]]
```

```
[ ] list_4[1][:3]
```

```
↳ [2, 4, 6]
```

```
[ ] list_4[:2][1]
```

```
↳ [2, 4, 6, 8]
```

리스트(list)

- list의 슬라이싱

```
[9] list_5 = [[[[1, 2, 3], [4], [5, [6, 7, 8, 9]]],  
              [10, 11, [12, 13, [14]]]]]
```

```
[13] list_5[:2][0]
```

```
⇒ [[1, 2, 3], [4], [5, [6, 7, 8, 9]]]]
```

```
[14] list_5[0][:2]
```

```
⇒ [[1, 2, 3], [4]]
```

리스트(list)

- list의 연산

```
[15] list_6 = ['a', 'b', 'c']
```

```
[16] list_7 = ['가', '나', '다']
```

```
[17] list_6 + list_7
```

⇒ ['a', 'b', 'c', '가', '나', '다']

```
[18] list_6 * 3
```

⇒ ['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']

리스트(list)

- list의 메소드

리스트는 삽입, 수정, 삭제가 자유로움

1. 삽입 : append, insert

```
[19] list_8 = ['파이썬', 'C', 'java']
```

→ insert는 원하는 위치에

```
[20] list_8.append('R')  
list_8
```

```
↳ ['파이썬', 'C', 'java', 'R']
```

→ append는 리스트 제일 뒤에

```
[21] list_8.insert(2, 'C++')  
list_8
```

```
↳ ['파이썬', 'C', 'C++', 'java', 'R']
```

```
[22] list_8.insert(5, 'C')  
list_8
```

```
↳ ['파이썬', 'C', 'C++', 'java', 'R', 'C']
```

리스트(list)

- list의 메소드
2. 수정 : 덮어쓰기 → 인덱스나 슬라이싱을 통해 수정

```
[23] list_8[0] = 'Python'  
list_8  
⇒ ['Python', 'C', 'C++', 'java', 'R', 'C']
```

리스트(list)

- list의 메소드

- 3. 삭제 : remove

```
[24] list_8
```

```
↳ ['Python', 'C', 'C++', 'java', 'R', 'C']
```

```
[25] list_8.remove('C')
      list_8
```

```
↳ ['Python', 'C++', 'java', 'R', 'C']
```

→ remove는 지우고자 하는 요소를 명시해야함

→ 가장 앞에 있는 요소가 삭제됨

리스트(list)

- list의 메소드

- 4. 꺼내오기 : pop

```
[26] list_8.pop()
```

```
↳ 'C'
```

```
[28] list_8
```

```
↳ ['Python', 'C++', 'java', 'R']
```

- pop은 괄호()안 인덱스에 해당하는 요소를 반환
- 인덱스를 생략하면 가장 마지막 요소를 꺼내옴

리스트(list)

- list의 메소드
 - 5. 정렬 : sort

```
[29] list_9 = [1,5,7,2,6]
```

```
[30] list_9.sort()  
list_9
```

```
⇒ [1, 2, 5, 6, 7]
```

→ 리스트 자체를 정렬

list.sort(key=None, reverse=False)

- key : 정렬해주고 싶은 기준, 기본적으로 < 비교에 의해 정렬(오름차순)
- reverse : True로 설정하게 되면 > 비교에 의해 정렬(내림차순)

리스트(list)

- list의 메소드

- 5. 정렬 : sort

```
[32] list_10 = [1,47,12,6]
      list_10.sort(reverse=True)
      list_10
```

↳ [47, 12, 6, 1]

리스트(list)

- list의 메소드

- 6. 뒤집기 : reverse

```
[37] list_9
```

```
↳ [1, 2, 5, 6, 7]
```

```
[38] list_9.reverse()  
list_9
```

```
↳ [7, 6, 5, 2, 1]
```

→ 현재 리스트 요소 순서를 뒤집음

→ 리스트 자체를 뒤집음

리스트(list)

- list의 메소드

6. 뒤집기 : reverse

```
[32] list_10 = [1,47,12,6]
      list_10.sort(reverse=True)
      list_10
```

```
↪ [47, 12, 6, 1]
```

```
[41] list_10.reverse()
      list_10
```

```
↪ [1, 6, 12, 47]
```

연습문제 3

튜플(tuple)

- tuple
()을 이용하여 튜플(tuple)임을 선언

```
[42] tuple_1 = ()  
     tuple_2 = tuple()
```

```
[43] tuple_3 = (1, 2)  
     tuple_4 = (3,)  
     tuple_5 = (4, 5, (6, 7))  
     tuple_6 = 8, 9, 10
```

한 개의 요소만 사용 할 때에는 콤마(,)를 반드시 사용해야함
괄호가 없어도 튜플로 선언할 수 있음

튜플(tuple)

- tuple

인덱싱과 슬라이싱이 가능

리스트와 달리 요소를 **삽입/삭제/수정** 할 수 없음

```
[44] tuple_3 = (1, 2)
     tuple_3[0] = 3
```

```
-----  
TypeError                                         Traceback (most recent call last)  
<ipython-input-44-c8372825bae8> in <module>()  
      1 tuple_3 = (1, 2)  
----> 2 tuple_3[0] = 3  
  
TypeError: 'tuple' object does not support item assignment
```

SEARCH STACK OVERFLOW

튜플(tuple)

- tuple의 연산

```
[45] tuple_3 = (1, 2)
     tuple_4 = (3, )
     tuple_3 + tuple_4
```

↳ (1, 2, 3)

```
[46] tuple_4 * 2
```

↳ (3, 3)

딕셔너리(dictionary)

- dictionary

{ }을 이용하여 딕셔너리(dictionary)임을 선언

```
[47] dict_1 = {'key' : 'value'}
     dict_2 = dict()
```

- 딕셔너리는 **key**와 **value** 값으로 이루어진 자료형
- 순서가 있는 자료형이 아니며, **key**를 통해 **value**값에 접근이 가능
- **key**는 고유한 값으로 중복될 수 없음
- **value**는 중복 가능

딕셔너리(dictionary)

- dictionary 추가/삭제/수정

1. 요소 추가하기 : 새로운 **key**에 **value**를 할당

```
[48] dict_3 = {'홍길동' : 100, '홍계월' : 200}
```

```
[49] dict_3['슈퍼맨'] = 300  
dict_3
```

⇒ {'슈퍼맨': 300, '홍계월': 200, '홍길동': 100}

주의 인덱스 대신 **key** 사용

2. 요소 수정하기 : 덮어쓰기

```
[51] dict_3['홍길동'] = 1  
dict_3
```

⇒ {'홍계월': 200, '홍길동': 1}

딕셔너리(dictionary)

- dictionary 추가/삭제/수정

3. 요소 삭제하기

```
[50] del dict_3['슈퍼맨']
      dict_3
```

↳ {'홍계월': 200, '홍길동': 100}

주의 인덱스 대신 **key** 사용

del : 파이썬 자체의 명령어로 메모리 자체에서 삭제

딕셔너리(dictionary)

- key와 value에 접근하기

1. key에 접근하기 : keys()

```
[1] dict_3 = {'홍길동' : 100, '홍계월' : 200, '슈퍼맨' : 150, '배트맨' : 250}
    print(dict_3.keys())
    print(type(dict_3.keys()))
```

```
⇒ dict_keys(['홍길동', '홍계월', '슈퍼맨', '배트맨'])
    <class 'dict_keys'>
```

2. value에 접근하기 : values()

```
[2] print(dict_3.values())
    print(type(dict_3.values()))
```

```
⇒ dict_values([100, 200, 150, 250])
    <class 'dict_values'>
```

딕셔너리(dictionary)

- key와 value에 접근하기
- 3. key와 value에 함께 접근하기

```
[3] print(dict_3.items())
    print(type(dict_3.items()))
```

```
↪ dict_items([('홍길동', 100), ('홍계월', 200), ('슈퍼맨', 150), ('배트맨', 250)])
<class 'dict_items'>
```

주의

리스트처럼 보이나 실제로는 **dict**의 개체이므로
리스트의 메소드를 사용할 수 없음

리스트처럼 사용하고 싶다면 리스트로 형 변환을 해야함

```
[4] list(dict_3.keys())[0]
```

```
↪ '홍길동'
```

딕셔너리(dictionary)

- key와 value에 접근하기
 - 4. key로 value에 접근하기
 - 1) 인덱싱처럼 키값을 이용하여 바로 접근
 - 2) 메소드 get을 사용

```
[5] dict_3
```

```
↳ {'배트맨': 250, '슈퍼맨': 150, '홍계월': 200, '홍길동': 100}
```

```
[6] print(dict_3['배트맨'])
print(dict_3.get('배트맨'))
```

```
↳ 250
250
```

딕셔너리(dictionary)

- key와 value에 접근하기
- 4. key로 value에 접근하기

```
[7] print(dict_3['펭수'])
```



```
-----  
KeyError                                         Traceback (most recent call last)  
<ipython-input-7-43368d90b0e9> in <module>()  
----> 1 print(dict_3['펭수'])
```

```
KeyError: '펭수'
```

→ 키값을 이용하여 접근 할 경우, 해당 키
가 없으면 오류 발생

SEARCH STACK OVERFLOW

```
[8] print(dict_3.get('펭수'))
```

→ get을 통해 접근할 경우, **None**을 반환



```
None
```

셋(집합)

- set

{ }을 이용하여 set을 선언

```
[9] set_1 = set()  
set_2 = set('Hello')  
set_3 = set([1, 2, 3])  
set_4 = {3,5, 'hi'}
```

중복을 허용하지 않음
순서가 없음



```
[10] set_2  
↳ {'H', 'e', 'l', 'o'}
```

```
[11] set_3  
↳ {1, 2, 3}
```

```
[12] set_4  
↳ {3, 5, 'hi'}
```

셋(집합)

- set의 연산

1. 교집합

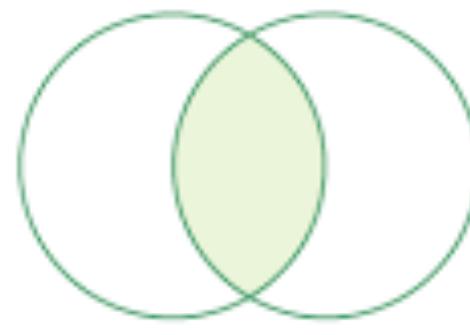
```
[13] set_5 = {1,2,3,4}  
      set_6 = {4,5,6,7}
```

```
[14] set_5 & set_6
```

```
⇒ {4}
```

```
[15] set_5.intersection(set_6)
```

```
⇒ {4}
```



집합 1 집합 2
(a) 교집합

셋(집합)

- set의 연산

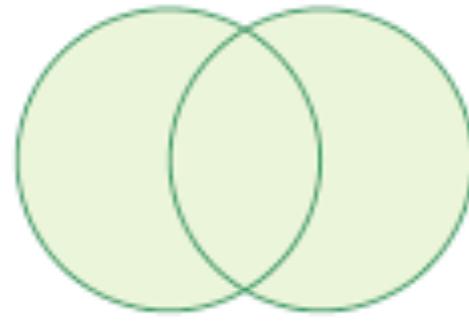
2. 합집합

```
[16] set_5 | set_6
```

↳ {1, 2, 3, 4, 5, 6, 7}

```
[17] set_5.union(set_6)
```

↳ {1, 2, 3, 4, 5, 6, 7}



(b) 합집합

셋(집합)

- set의 연산

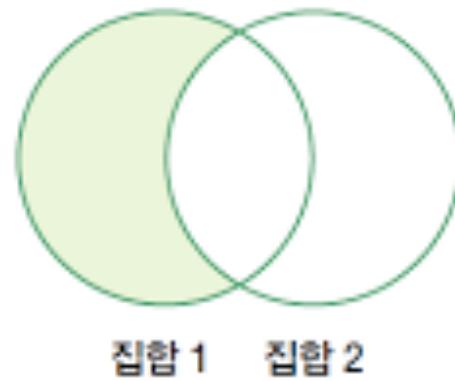
3. 차집합

```
[18] set_5 - set_6
```

☞ {1, 2, 3}

```
[19] set_5.difference(set_6)
```

☞ {1, 2, 3}



(c) 차집합

셋(집합)

- set의 메소드

1. 한 개 추가하기 : add

```
[22] set_6 = {4, 5, 6, 7}  
      set_6.add(8)  
      set_6
```

↳ {4, 5, 6, 7, 8}

2. 여러 개 추가하기 : update

```
[23] set_6.update([1,2,3])  
      set_6
```

↳ {1, 2, 3, 4, 5, 6, 7, 8}

셋(집합)

- set의 메소드
- 3. 한 개 지우기 : remove

```
[24] set_6.remove(3)  
      set_6
```

↳ {1, 2, 4, 5, 6, 7, 8}

불(bool)

- bool : 참과 거짓을 나타내는 자료형

참 : True

거짓 : False

[26] `True`

↳ `True`

[28] `1 == 1`

↳ `True`

[27] `type(False)`

↳ `bool`

[29] `1 > 2`

↳ `False`

→ 비교 연산자와 조건문의 반환값으로 불자료형이 사용 됨

비교 연산자	의미
<code>==</code>	같음
<code>!=</code>	같지 않음
<code>>, >=, <, <=</code>	크거나 같음/ 작거나 같음

布尔(bool)

- bool : 참과 거짓을 나타내는 자료형
자료형에도 참/거짓이 있음

자료	참 / 거짓
'python'	참
" "	거짓
[1,2,3]	참
[]	거짓
()	거짓
{ }	거짓
1	참
0	거짓
None	거짓

```
[30] bool(1)
```

↳ True

```
[31] bool('')
```

↳ True

```
[32] bool('')
```

↳ False

bool()함수를 통해 해당 자료형의 참/거짓을 확인할 수 있음

파이썬의 자료형

- 자료형의 특징
- 인덱싱
- 슬라이싱
- 메소드

- 숫자(int, float)
- 문자(string)
- 리스트(list)
- 튜플(tuple)
- 딕셔너리(dict)
- 셋(set)
- 불(bool)

연습문제 4

Quiz 1

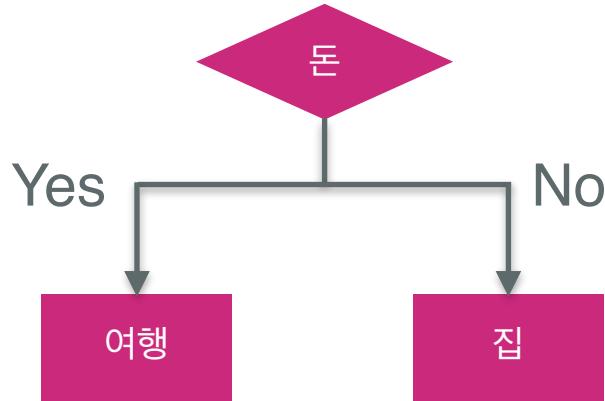
조건문

조건문

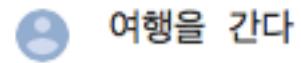
- if와 else
- elif
- 조건문 만들기
- 조건문 중첩하기
- pass

조건문

- if / else



```
[ ] money = True  
  
if money :  
    print("여행을 간다")  
else :  
    print("집에서 쉰다")
```



조건문

- if / else

```
[ ] if 조건 :  
    수행 할 문장1  
    수행 할 문장2  
else :  
    수행 할 문장3  
    수행 할 문장4
```

```
[ ] money = True  
  
if money :  
    print("여행을 간다")  
else :  
    print("집에서 쉰다")
```



여행을 간다

주의

콜론(:)과 들여쓰기에 주의할 것

조건문

- if / elif / else

```
[ ] if 조건1 :  
    수행 할 문장1  
elif 조건2 :  
    수행 할 문장2  
else :  
    수행 할 문장3
```

```
[ ] money = 20000  
  
if money < 5000 :  
    print('라면을 먹는다')  
elif 5000 < money < 25000 :  
    print('치킨을 먹는다')  
elif 25000 < money < 50000 :  
    print('삼겹살을 먹는다')  
else :  
    print('소고기를 먹는다')
```



치킨을 먹는다

연습문제 5

조건문

- 조건문?

조건문 : 참과 거짓을 판단하게 하는 문장

1. 비교연산자

```
[ ] a = 3  
      b = 5
```

a < b



True

```
[ ] c = 3  
      d = 3
```

c <= d



True

```
[ ] c != d
```



False

조건문

- 조건문?

조건문 : 참과 거짓을 판단하게 하는 문장

2. 논리연산자 : and / or / not

[] True and True



True

1) and

비교하는 대상이 모두 참이여야만
True반환

[] True and False



False

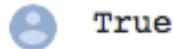
조건문

- 조건문?

조건문 : 참과 거짓을 판단하게 하는 문장

- 논리연산자 : and / or / not

[] True or True

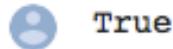


True

2) or

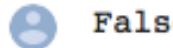
비교하는 대상 중 하나만 참이여도
True 반환

[] True or False



True

[] False or False



False

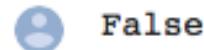
조건문

- 조건문?

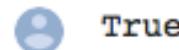
조건문 : 참과 거짓을 판단하게 하는 문장

2. 논리연산자 : and / or / not

[] not True



[] not False



3) not

참이면 False를, 거짓이면 True를 반환

조건문

- 조건문?

조건문 : 참과 거짓을 판단하게 하는 문장

3. 요소인지 파악하기 : in / not in

```
[ ] e = [1, 3, 5, 7]
```

```
[ ] 0 in e
```



False

```
[ ] 2 not in e
```



True

```
[ ] 1 in e
```



True

```
[ ] 3 not in e
```



False

1) x in s

x가 s의 요소인가

2) x not in s

x가 s의 요소가 아닌가

조건문

- if문 안에 if문

조건 안에 조건을 만들 수 있음

```
[ ] money = 20000  
card = True
```

```
[ ] if card :  
    if money < 30000 :  
        print("삼겹살을 먹는다")  
    else :  
        print("소고기를 먹는다")  
else :  
    if money <= 1000 :  
        pass  
    else :  
        print("라면을 먹는다")
```



삼겹살을 먹는다

조건문

- pass

조건을 만족해도 아무 일도 일어나지 않게 하려면?

```
[ ] money = 1000
    card = False
```

```
[ ] if card :
    if money < 30000 :
        print("삼겹살을 먹는다")
    else :
        print("소고기를 먹는다")
else :
    if money <= 1000 :
        pass
    else :
        print("라면을 먹는다")
```

조건문

- if와 else
- elif
- 조건문 만들기
- 조건문 중첩하기
- pass

반복문

반복문

- while 문
- for 문

반복문 - while

조건문이 참인 동안 반복해서 문장을 수행함

```
[ ] while 조건 :  
    수행할 문장1  
    수행할 문장2
```

```
[ ] # 100이하의 짝수 프린트하기  
i = 1  
→ 조건문에 쓰일 변수  
  
while i <= 10 :  
    if i % 2 == 0 :  
        print(i)  
    i += 1  
→ 변수를 증감
```

2
4
6
8
10

반복문 - while

- break

반복문에서 빠져나오고 싶다면?

```
[ ] # 100번째 방문자 찾기  
i = 90
```

```
while i :  
    i += 1  
    if i == 100 :  
        print("축하합니다. %d번째 방문자입니다." % i)  
        break  
print("감사합니다. 이벤트가 종료되었습니다.")
```

→ 변수는 0이 아니라면 항상 참 (무한 반복)

→ 반복문이 종료되었으므로 수행됨



축하합니다. 100번째 방문자입니다.
감사합니다. 이벤트가 종료되었습니다.

break를 사용하면 반복문이 더 이상 작동하지 않고 멈춤

반복문 - while

- continue

반복문의 첫부분으로 돌아가고 싶다면?

```
[ ] i = 0

while i < 11 :
    i += 1
    if i == 6 :
        continue
    if i % 2 == 0 :
        print(i)
```

→ 6일 때는 continue를 사용해 반복문의 가장 처음으로 가도록 함



2
4
8
10

→ 6일 때는 짹수임에도 불구하고 프린트가 안됨

반복문 - for

- for문

```
[ ] for 변수 in range(변수가 속한 자료형, 혹은 변수의 범위) :  
    수행해야할 문장1  
    수행해야할 문장2
```

주의

콜론(:)과 들여쓰기에 주의할 것

```
[ ] for x in range(0,5) :  
    print(x)
```



0

1

2

3

4

반복문 - for

- for문

```
[ ] for x in range(0,5) :  
    print(x)
```



0
1
2
3
4



변수의 범위 지정하기
1) 숫자로 범위주기
2) 자료형으로 범위주기

반복문 - for

- 변수의 범위 지정하기
 - 1) 숫자로 범위주기 : range()

range(시작, 끝, 간격)

- 시작부터 끝 - 1 까지
- 간격이 1이라면 생략가능
- 0부터 시작하여 1씩 증가한다면 시작과 간격 생략가능

반복문 - for

- 변수의 범위 지정하기

- 숫자로 범위주기 : range()

연습문제 7! 5에서부터 0까지 카운트 다운을 세어 볼까요?

```
[ ] # 카운트 다운  
for count in range(5, -1, -1):  
    print(count)
```



5
4
3
2
1
0

→ 하나씩 작아지므로 간격은 -1
→ 0까지 출력해야 하므로 -1이 끝 인덱스가
되어야 함

반복문 - for

- 변수의 범위 지정하기
- 2) 자료형으로 범위

```
[ ] word = 'Hello!'  
      for w in word:  
          print(w)
```



H
e
l
l
o
!

```
[ ] for a, b in [(2,1), (2,2), (2,3), (2,4)] :  
      print(a*b)
```



2
4
6
8

반복문 - for

- 이중으로 반복문을 사용하기

```
[ ] # 구구단 2단과 3단을 출력하기  
[5] for i in range(2,4) :  
    print('===' , i, '단 ===')  
    for j in range(1, 10) :  
        print(i * j)
```



===== 2 단 =====
2
4
6
8
10
12
14
16
18
===== 3 단 =====
3
6
9
12
15
18
21
24
27

반복문

- while 문

break

continue

- for 문

범위 지정하기

Quiz 2

파일

파일

- 파일 쓰기
- 파일 읽기

파일 쓰기

- open

open은 파일을 여는데 사용하는 함수

open(file_path, mode=‘ ’, encoding=‘ ’)

- **file_path** : 파일의 주소
- **mode** : 파일을 열 때 필요한 파일 모드(읽기/쓰기/수정)
- **encoding** : 파일을 열 때 필요한 인코딩 (생략 가능)

```
[ ] new_file = open('/content/drive/My Drive/Colab Notebooks/python/toy_data/new.txt', 'w')

[ ] print(new_file)

↳ <_io.TextIOWrapper name='./drive/My Drive/Colab Notebooks/day2/toy_data/new.txt' mode='w' encoding='UTF-8'>
```

* 주의 * open으로 연 파일은 파일객체를 반환함

파일 쓰기

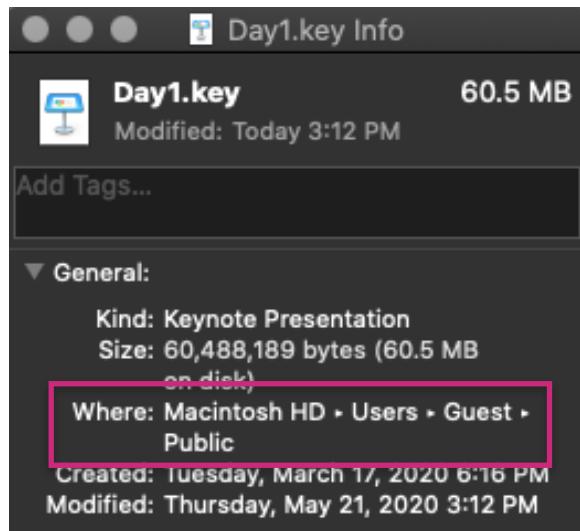
- open
 - 1. file path
 - file path(파일 경로)는 string
 - 파일은 두 가지 방식으로 표현할 수 있음
 - 1) 절대 경로 : 파일이 가지고 있는 고유한 경로
 - 2) 상대 경로 : 현재 내 위치에서부터 해당 파일까지의 경로

파일 쓰기

- open

1. file path

- 1) 절대 경로 : 파일이 가지고 있는 고유한 경로
현재 위치와 무관하게 항상 해당 파일에 접근 가능



→ 'Users/Guest/Public/Day1.key'

파일 쓰기

- open

1. file path

2) 상대 경로 : 지금 내가 작업하고 있는 위치에서부터 해당 파일까지의 경로
내 위치에 따라 달라질 수 있음

현재 파일의 위치	./
현재 파일의 상위 폴더	../

파일 쓰기

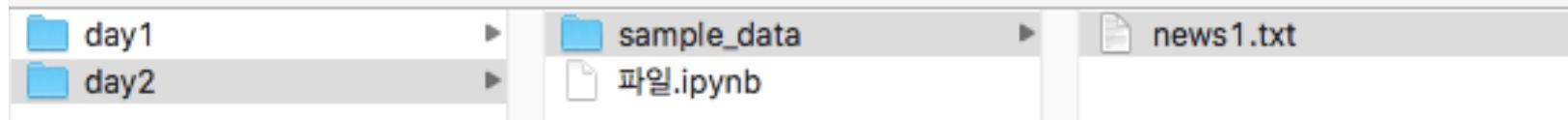
- open

1. file path

2) 상대 경로 : 지금 내가 작업하고 있는 위치에서부터 해당 파일까지의 경로
내 위치에 따라 달라질 수 있음

현재 파일의 위치	./
현재 파일의 상위 폴더	../

Quiz. 현재 ‘파일.ipynb’에서 작업 중이다. ‘news1.txt’파일을 열기 위해 필요한 상대 경로는?



→ './sample_data/news1.txt'

파일 쓰기

- open

1. file path

2) 상대 경로 : 지금 내가 작업하고 있는 위치에서부터 해당 파일까지의 경로
내 위치에 따라 달라질 수 있음

현재 파일의 위치	./
현재 파일의 상위 폴더	../

Quiz. 현재 ‘파일.ipynb’에서 작업 중이다. ‘day1’디렉토리(폴더)에 있는 ‘quiz.txt’ 파일을 열기 위해 필요한 상대 경로는?



→ './day1/quiz.txt'

파일 쓰기

- open

1. file path

- file path(파일 경로)는 **string**
- **윈도우** 환경에서 파일 열 때 주의사항

윈도우 환경에서는 '\U'가 파일 경로에 포함된 경우 안 열리는 경우(유니코드 에러)가 존재함.

→ 이럴 땐, r‘파일경로’를 사용하거나 \를 두번씩 써주면 됨.

경로 : ‘Users/Guest/Public/Day1.key’

→ r‘Users/Guest/Public/Day1.key’

→ ‘Users\\Guest\\Public\\Day1.key’

파일 쓰기

- open

- mode

파일 모드	설명	비고
r	읽기 모드	파일에 새로운 내용을 쓰거나 수정이 불가능
w	쓰기 모드	파일에 새로운 내용을 덮어씀(기존의 내용 모두 삭제)
a	수정 모드	파일에 새로운 내용이 추가됨(기존의 내용 + 새로운 내용)
rb	바이너리 읽기 모드	바이너리 파일을 읽을 때 사용
wb	바이너리 쓰기 모드	바이너리 파일을 쓸 때 사용

파일 쓰기

- open
- 3. encoding

지정하지 않으면 플랫폼에 따르게 됨

- UTF-8 : 대표적인 조합형 유니코드 인코딩 방식

파일 쓰기

- write

파일.write(내용)

- 파일 : open으로 열었던 파일 객체
- 내용 : 파일에 쓰고 싶은 내용

```
[6] data = '새로운 내용을 쓰고 싶다면, \n 이렇게 작성하면 됩니다.'
```

```
[7] new_file.write(data)
```

↳ 29

→ 파일 안의 문자의 갯수를 반환

* 주의 * 파일을 열고, 작업을 마쳤다면 반드시 닫을 것

```
[8] new_file.close()
```

파일 쓰기

- with

with open(file_path, mode=) as 변수 :

수행할 문장 1

수행할 문장 2

- with문을 사용하면 with블록을 벗어나는 순간 파일을 자동으로 close해줌
- **as 변수** : open으로 연 파일 객체를 **변수**로 받아줌

```
[ ] with open('/content/drive/My Drive/Colab Notebooks/python/toy_data/new.txt', 'w') as f :  
    data = '이렇게도 작성이 가능합니다. close를 따로 안해도 되요.'  
    f.write(data)
```

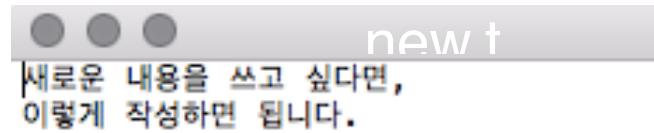
파일 읽기

- `read` / `readline` / `readlines`

함수	설명
<code>read</code>	파일 전체를 한번에 읽어서 <code>string</code> 으로 반환
<code>readline</code>	파일을 한 줄만 읽어서 <code>string</code> 으로 반환
<code>readlines</code>	파일을 줄단위로 모두 읽어서 <code>list</code> 로 반환

파일 읽기

- read / readline / readlines



1. read

```
[ ] with open('/content/drive/My Drive/Colab Notebooks/python/toy_data/new.txt', 'r') as f:  
    lines = f.read()  
    print(lines)  
    print(type(lines))
```

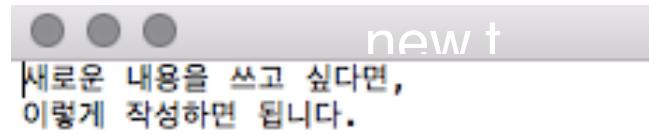
☞ 새로운 내용을 쓰고 싶다면,
이렇게 작성하면 됩니다.

```
<class 'str'>
```

→ 파일 전체를 **string**으로 불러옴

파일 읽기

- read / readline / readlines



2. readline

```
[ ] with open('/content/drive/My Drive/Colab Notebooks/python/toy_data/new.txt', 'r') as f:  
    line = f.readline()  
    print(line)  
    print(type(lines))
```

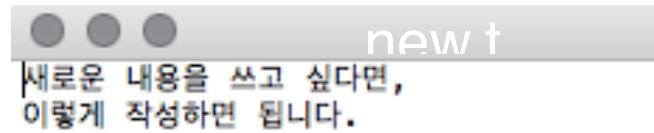
☞ 새로운 내용을 쓰고 싶다면,

```
<class 'str'>
```

→ 파일에서 한 줄만 **string**으로 불러옴

파일 읽기

- read / readline / readlines



2. readline

(참고) readline + while문으로 전체파일을 불러올 수 있음

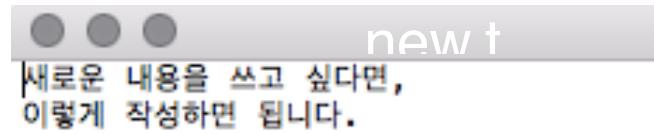
```
[ ] with open('/content/drive/My Drive/Colab Notebooks/python/toy_data/new.txt', 'r') as f:  
    while True :  
        line = f.readline()  
        if not line : break  
        print(line)
```

☞ 새로운 내용을 쓰고 싶다면,

이렇게 작성하면 됩니다.

파일 읽기

- read / readline / readlines



2. readlines

```
[ ] with open('/content/drive/My Drive/Colab Notebooks/python/toy_data/new.txt', 'r') as f:  
    lines = f.readlines()  
    print(lines)  
    print(type(lines))
```

```
↳ ['새로운 내용을 쓰고 싶다면,\n', '이렇게 작성하면 됩니다.'][  
<class 'list'>
```

- 파일에서 줄단위로 읽어서 list로 불러옴
- 개행문자(\n)가 남아 있음

파일

- colab에서 파일 사용하기
- 파일 쓰기

open

with문

- 파일 읽기

read

readline

readlines

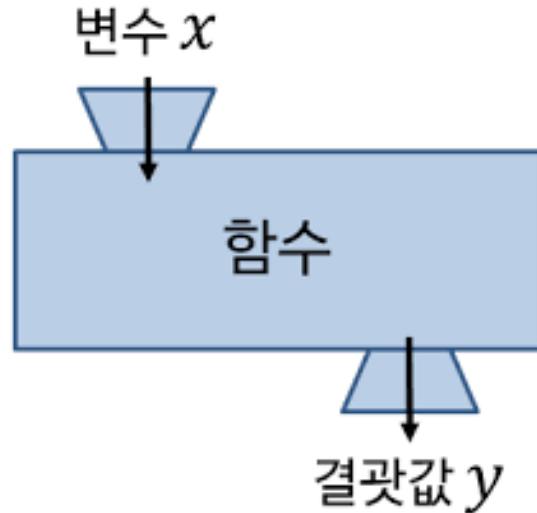
함수(Function)

함수

- 함수란?
- 함수 정의하기
- 함수와 변수

함수(Function)

- 함수란?



```
[ ] def 함수이름(함수인자) :  
    수행할 문장1  
    수행할 문장2  
    return 반환값
```

일정한 프로세스를 할 수 있도록 하는 것

함수(Function)

- 함수 정의하기

```
def 함수이름( 매개변수 ) :  
    함수의 내용  
    return 반환값
```

- **함수이름** : 사용자가 정의하는 함수 이름, 기존에 사용되는 함수나 예약어들을 제외하고 사용
- **매개변수** : 함수 안에서 사용 할 변수들 (생략 가능)
- **return** : 함수 안에서 모든 연산을 마친 후 반환할 값(생략 가능)

함수(Function)

- 함수 정의하기

pop quiz! 두 개의 input를 받으면 더하기를 하여 값을 돌려주는 함수를 만들어봅시다.

```
[ ] def add(a, b) :  
    result = a + b  
    return result
```



```
[ ] c = add(3, 5)  
c
```



8

→ 함수 정의

→ 함수 호출

함수(Function)

- 함수 정의하기
- 1. 매개변수와 return값이 모두 있는 함수

```
[ ] def add(a, b) :  
    result = a + b  
    return result
```

```
[ ] c = add(3, 5)  
c
```



8

함수(Function)

- 함수 정의하기
- 2. return값이 비어있는 함수

```
[ ] def sub(a, b) :  
    print('뺄셈의 결과는 %d입니다.' %(a-b))  
    return
```

```
[ ] d = sub(1, 2)
```



뺄셈의 결과는 -1입니다.

```
[ ] print(d)
```



None

함수(Function)

- 함수 정의하기

3. return이 없는 함수

```
[ ] def mul(a, b) :  
    print('%d 와 %d의 곱은 %d입니다.' %(a, b, a*b))
```

```
[ ] mul(3, 2)
```



3 와 2의 곱은 6입니다.

함수(Function)

- 함수 정의하기
4. 매개변수와 return값이 모두 없는 함수

```
[ ] def start():
    print("Hello World!")
```

```
[ ] start()
```



Hello World!

연습문제 6

함수(Function)

- 더 알아보기
- 1. 매개변수의 초기값 미리 설정하기

```
[ ] def function(a, n=2) :  
    print("%d의 제곱은 %d 입니다." %(a, a**n))
```

```
[ ] function(4)
```



4의 제곱은 16 입니다.

→ 초기값을 설정하는 경우, 매개변수의 순서에 주의!
초기값 없는 변수, 초기값 있는 변수 순서로 배치

함수(Function)

- 더 알아보기

2. 매개변수가 몇 개가 필요할지 모를 때

```
[3] def test(*args) :  
    print(args)
```

```
[4] test(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

⇒ (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

```
[5] test([1,2],3)
```

⇒ ([1, 2], 3)

→ 매개변수에 *를 붙이면 함수가 받은 input 모두를 tuple로 묶어줌

Quiz! 왜 tuple로 묶어줄까요?

연습문제 7

함수(Function)

- 함수와 변수

Quiz! 함수 안에 선언한 변수를 함수 밖에서도 사용할 수 있을까?

NO!

함수 안의 블록은 함수 고유의 영역임.

따라서, 함수 안에서 선언한 변수는 함수 밖에서 사용할 수 없음.

함수(Function)

- 함수와 변수

예시 1)

```
[ ] def myfunction() :  
    in_val = '함수 안의 변수'
```

```
[ ] myfunction()
```

```
[ ] in_val
```



```
-----  
NameError Traceback (most recent call last)  
<ipython-input-30-a23adeea90d3> in <module>  
----> 1 in_val  
  
NameError: name 'in_val' is not defined
```

함수(Function)

- 함수와 변수

예시 2)

```
[ ] a = 1
```

```
[ ] def myfunction2(a) :  
    a += 1  
    print(a)
```

```
[ ] myfunction2(a)
```

2

```
[ ] a
```

1

함수 실행 시 출력값은 무엇일까요?

a는 어떤 값일까요?

함수(Function)

- 함수와 변수

함수 안의 블록은 함수 고유의 영역임.

따라서, 함수 안에서 선언한 변수는 함수 밖에서 사용할 수 없음.

```
[ ] def 함수이름(함수인자) :  
    수행할 문장1  
    수행할 문장2  
    return 반환값
```

함수(Function)

- 함수와 변수

함수 안에 있는 변수를 밖에서도 쓰고 싶다면?

1. return 사용하기
2. global 사용하기 ← 권장하지 않음

함수(Function)

- 함수와 변수

함수 안에 있는 변수를 밖에서도 쓰고 싶다면?

1. return 사용하기

```
[ ] def myfunction3(a) :  
    a += 1  
    print(a)  
    return a
```

```
[ ] b = myfunction3(a)
```



2

```
[ ] b
```



2

함수(Function)

- 함수와 변수

함수 안에 있는 변수를 밖에서도 쓰고 싶다면?

2. global 사용하기

```
[ ] a = 1
```

```
[ ] def myfunction4():
    global a
    a += 1
    print(a)
```

```
[ ] myfunction4()
```



2

```
[ ] a
```



2

함수(Function)

- 함수와 변수

return 사용하기

```
[21] for _ in range(5) :  
    c = myfunction3(a)  
    print('함수의 결과 : %d' %c)  
    print('a의 값 : %d' %a)
```

```
↳ 3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2
```

global 사용하기

```
[25] for _ in range(5) :  
    myfunction4()  
    print('a의 값 : %d' %a)
```

```
↳ 2  
a의 값 : 2  
3  
a의 값 : 3  
4  
a의 값 : 4  
5  
a의 값 : 5  
6  
a의 값 : 6
```

← 함수 밖의 변수(global variable)이
함수의 작동에 따라 그 값이 변함

함수

- 함수란?
- 함수 정의하기
 - 매개변수
- 함수와 변수
 - return과 global

Quiz 3