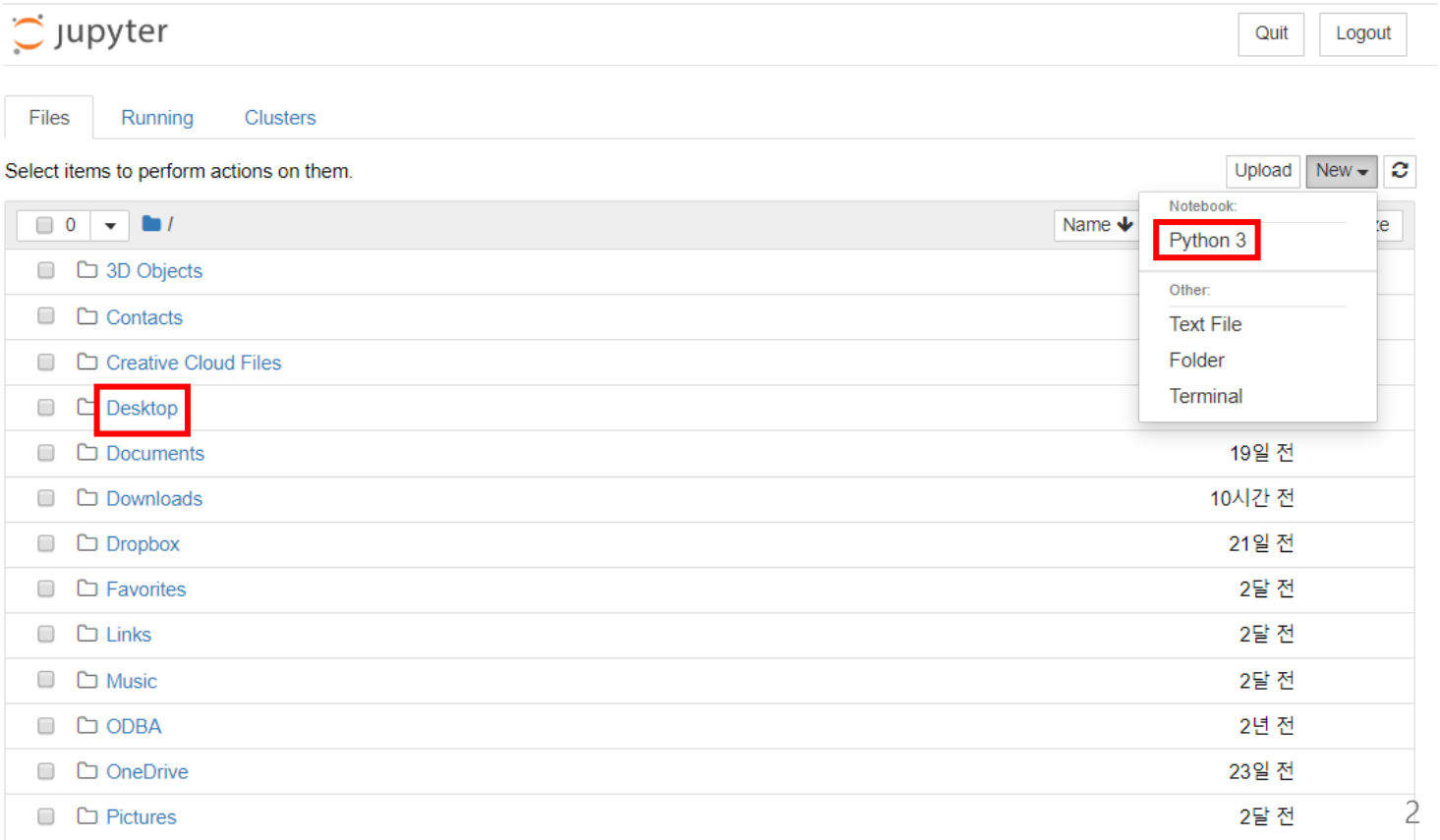
# 파이썬 기초문법 튜토리얼

# 파이썬 시작하기 : Jupyter notebook

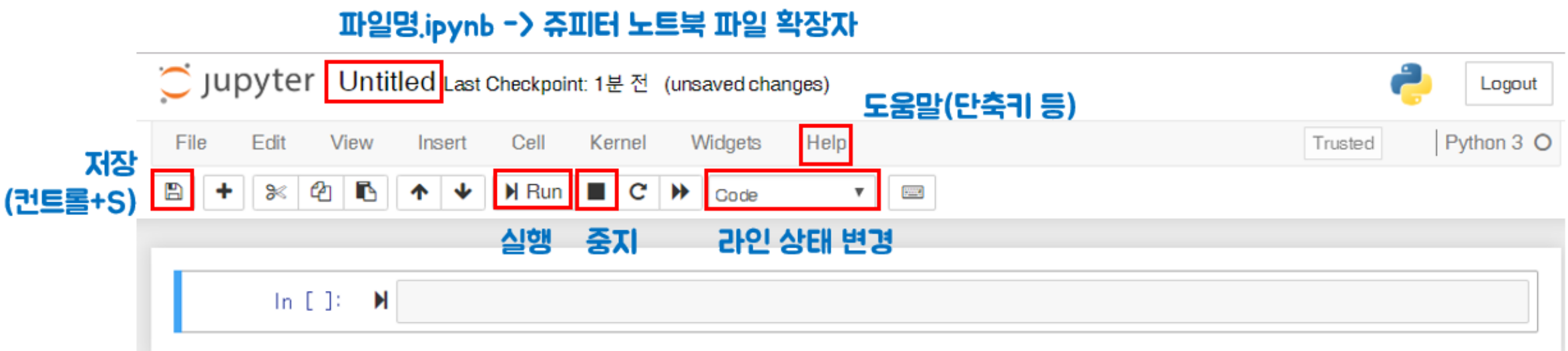
1. Jupyter notebook 아이콘 클릭

2. "Desktop" -> "Python_basic" 폴더 클릭

3. 파이썬 파일 생성



# 쥬피터 노트북 사용법 – 외워야할 단축키




<많이 쓰는 단축키> (Help -> Keyboard Shortcuts)

코드 실행 : 컨트롤+엔터
쉬프트+엔터
위에 라인 추가 : a
아래에 라인 추가 : b
라인 제거 : dd

라인 나누기 : 컨트롤+쉬프트+'-'
라인 합치기 : 쉬프트+M
저장 : 컨트롤+S
tab : 들여쓰기
shift + tab : 내어쓰기
ctrl+z : 되돌리기
ctrl+y : 되돌리기 취소

# 쥬피터 노트북 사용법

커맨드 모드 전용 (에디트 모드에서 Esc 키 입력 or 셀의 바깥 클릭)
Command Mode (press Esc to enable)

F : 찾기
Ctrl-Shift-F : 커맨드 파렛트 열기
Ctrl-Shift-P : 커맨드 파렛트 열기
Enter : 에디트 모드로 변경
P : 커맨드 파렛트 열기
Shift-Enter : 현재 셀 실행 + 아래 셀로 이동
Ctrl-Enter : 현재 셀 실행 (셀 이동 없음)
Alt-Enter : 현재 셀 실행 + 아래 셀 삽입
Y : 현재 셀을 코딩 옵션으로 변경
M : 현재 셀을 마크다운 옵션으로 변경
R : 현재 셀을 로우 옵션으로 변경
1 : 현재 셀을 헤딩1 옵션으로 변경
2 : 현재 셀을 헤딩2 옵션으로 변경
3 : 현재 셀을 헤딩3 옵션으로 변경
4 : 현재 셀을 헤딩4 옵션으로 변경
5 : 현재 셀을 헤딩5 옵션으로 변경
6 : 현재 셀을 헤딩6 옵션으로 변경
K : 현재 셀을 헤딩6 옵션으로 변경
Up : 바로 윗 셀 선택
Down : 바로 아래 셀 선택
J : 바로 아래 셀 선택
Shift-K : 현재 셀 포함하여 윗 셀 선택
Shift-Up : 현재 셀 포함하여 윗 셀 선택

단축키 수정
Edit Shortcuts

Shift-Down : 현재 셀 포함하여 아래 셀 선택
Shift-J : 현재 셀 포함하여 아래 셀 선택
A : 현재 셀 위에 셀 삽입
B : 현재 셀 아래에 셀 삽입
X : 현재 셀 잘라내기
C : 현재 셀 복사하기
Shift-V : 현재 셀 위로 붙여넣기
V : 현재 셀 아래로 붙여넣기
Z : 셀 삭제 되돌리기
D , D : 현재 셀 삭제 (d 두 번 연타)
Shift-M : 여러 셀 합치기
셀 하나 선택 후 이 명령을 내리면
바로 아래 셀과 합치기
Ctrl-S : 전체 저장
S : 전체 저장
L : 줄번호 옵션 켜기/끄기
O : 현재 셀 수행 결과 표시
켜기/끄기
Shift-O : 현재 셀 수행 결과 표시
스크롤 켜기/끄기
H : 키보드 단축키 모음 보기
I , I : 커널 중지 (i 두 번 연타)
0 , 0 : 커널 재시작 (숫자 0 두 번 연타)
※ 숫자패드 0 안됨
Esc : 팝업 닫기
Q : 팝업 닫기
Shift-L : 모든 셀 줄번호 옵션
켜기/끄기
Shift-Space : 화면 스크롤 올리기
Space : 화면 스크롤 내리기

https://m.blog.naver.com/jsk6824/221789162355

# 첫 번째 파이썬 명령어! #

파이썬을 실행시키면 프롬프트가 `>>>` 로 바뀝니다. 이제부터는 파이썬 언어 명령어만 사용할 수 있다는 뜻이예요. `>>>` 를 입력할 필요 없어요. 파이썬이 대신해 줄 테니까요.

파이썬 콘솔에서 나가려면 `exit()` 을 입력하거나, 윈도우에서는 `Ctrl + Z` 를, 맥이나 리눅스에서는 `Ctrl + D` 를 입력하면 됩니다. 그러면 `>>>` 는 사라질 거에요.

물론 지금 파이썬 콘솔을 종료할 필요가 없답니다. 아직 파이썬 콘솔에 대해 좀 더 배워야 하니까요. 아주 간단한 사칙연산부터 해봅시다. `2 + 3` 을 입력하고 `엔터` 를 쳐보세요.

command-line

```
>>> 2 + 3
5
```

잘했어요! 값이 출력되었나요? 파이썬은 수학을 할 줄 알아요. 다른 사칙연산도 입력해보세요.

- `4 * 5`
- `5 - 1`
- `40 / 2`

# 문자열(String) #

이름을 써봅시다. 따옴표(")를 양쪽에 감싸서 입력해보세요.

command-line

```
>>> "Ola"
'Ola'
```

처음으로 문자열을 만들었군요! 문자열은 컴퓨터가 처리할 수 있는 연속된 문자를 말해요. 문자열은 반드시 시작과 끝이 문자여야 하고, 양쪽을 작은따옴표( ' ) 나 큰따옴표( " )로 감싸야 해요. (차이점은 없어요) 이 따옴표는 파이썬에게 안에 문자열이 들어있다고 알려줍니다.

문자열은 줄줄이 사탕처럼 연결될 수 있어요. 이렇게요.

command-line

```
>>> "Hi there " + "Ola"
'Hi there Ola'
```

문자열과 숫자를 함께 곱할 수도 있어요.

command-line

```
>>> "Ola" * 3
'OlaOlaOla'
```

## 문자열에 따옴표 넣기

문자열 안에 작은따옴표를 넣고 싶다면, 두 가지 방법이 있어요.

큰따옴표("")를 사용하는 방법과

command-line

```
>>> "Runnin' down the hill"
"Runnin' down the hill"
```

백슬래시( \ )를 이용하는 방법이예요.

command-line

```
>>> 'Runnin\' down the hill'
"Runnin' down the hill"
```

command-line

```
>>> "Ola".upper()
'OLA'
```

# 메소드, 함수

방금 문자열에 `upper` 이라는 **메소드(method)**를 사용했어요! 메소드는( `upper()` 같은 것)는 파이썬이 객체( `"Ola"` 같은)를 대상으로 수행할 수 있는 일련의 명령을 말해요.

총 글자 수를 알고 싶을 때, `len()` **함수(function)**을 사용해요.

command-line

```
>>> len("Ola")
3
```

왜 어떤 함수는 문자열 뒤에 `.` 를 붙이고, ( `"Ola".upper()` 처럼), 또 어떤 함수는 뒤에 문자열이 들어간 괄호( `len("Ola")처럼` )를 붙이는지 궁금하셨죠? `upper()` 같은 함수는 문자열만 쓸 수 있는 함수로, **메서드(method)**라고 합니다. 반면에 `len()` 과 같은 함수는 문자열이나 숫자 등 여러 객체를 사용할 수 있어요. 그래서 `len()` 함수에 `"Ola"` 를 매개변수(parameter)로 준 거예요.

# 요약하기

자, 문자열은 충분히 다루었어요. 지금까지 우리가 공부한 내용을 정리해봅시다.

- **프롬프트(the prompt) -** 명령어(코드)를 파이썬 프롬프트에 입력하면 파이썬이 응답합니다.
- **숫자와 문자열(numbers and strings) -** 사칙연산을 할 때는 숫자를 사용하고 글을 다룰 때는 문자열을 사용합니다.
- **연산자(operators) -** `+` 나 `*` 같은 연산자를 사용합니다.
- **함수(functions) -** 객체에 `upper()` 나 `len()` 처럼 명령을 수행합니다.

지금까지 가장 기본적인 프로그래밍 문법을 살펴보았어요. 더 어려운 부분으로 넘어가도 괜찮겠죠?

# 오류(error)

새로운 것을 배워봅시다. 총 글자 수도 알아냈듯이 숫자 수도 알 수 있을까요? `len(304023)`을 입력하고 `엔터`를 입력하세요.

command-line

```
>>> len(304023)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

첫 번째 에러가 나타났어요! 여기서 ❗ 아이콘은 `이 코드는 여러분이 예상하는 대로 실행되지 않을 것입니다.`를 의미합니다. (의도적으로 만들어낸 것이라도) 실수도 배움의 중요한 과정 중 하나랍니다!

"int"(정수)객체는 길이가 없기 때문이지요. 이제 어떻게 할 수 있을까요? 숫자(number)를 문자열(String)로 바꿔 쓸 수 있지 않을까요? 문자열은 길이를 갖고 있으니까요, 그렇죠?

command-line

```
>>> len(str(304023))
6
```

잘 되네요! 앞에서 우리는 `len()` 함수 내부에 `str()` 함수를 사용했어요. `str()` 함수는 모든 대상을 문자열로 변환한답니다.

# 변수

변수는 프로그래밍에서 중요한 개념입니다. 변수란 여러 번 사용될 수 있게 이름을 붙인 거랍니다. 작성한 코드를 다 기억할 필요없이 코드를 읽기 쉽게 하기 위해 사용하지요. 프로그래머들은 변수를 사용해 데이터를 저장합니다.

`name` 이라는 새로운 변수를 만들어 보세요.

command-line

```
>>> name = "Ola"
```

봤죠? 간단해요. 매우 쉬어요! "이름은 올라와 같다"라고 쓰는 거에요.

여러분도 알겠지만, 프로그램은 이전에 수행한 내용을 보여주지 않아요. 그렇다면 변수값을 알려면 어떻게 해야할까요? `name` 를 입력하고 `엔터` 를 입력하세요.

command-line

```
>>> name
'Ola'
```

야호! `name` 은 여러분이 만든 첫 변수랍니다! :) 언제든지 변수값을 바꿀 수 있어요.

# 변수

command-line

```
>>> name = "Sonja"
>>> name
'Sonja'
```

또 함수 안에 변수명을 사용할 수 있습니다.

command-line

```
>>> len(name)
5
```

멋지죠? 물론 변수는 무엇이든지 담을 수 있으니, 당연히 숫자도 가능하죠! 해봅시다. :

command-line

```
>>> a = 4
>>> b = 6
>>> a * b
24
```

# print() 함수

아래와 코드를 따라 입력해보세요. :

command-line

```
>>> name = 'Maria'
>>> name
'Maria'
>>> print(name)
Maria
```

`name` 을 쳤을 때, 파이썬 해석기는 'name' 변수의 문자열 *표현(representation)*으로 응답하는데, 그 모양은 작은따옴표(')로 둘러싸인 M-a-r-i-a 문자입니다. `print(name)` 라고 치면, 파이썬은 화면에 따옴표없이 더 깔끔한 화면을 보여( `print` )줍니다.

나중에 차차 알게 될 거지만, `print()` 는 함수 안에 있는 내용을 출력할 때, 여러 내용들을 확인하고 싶을 때 유용하게 사용됩니다.

# 데이터 보관소 : 리스트, 딕셔너리

# 리스트(list)

문자열과 정수 외에도 객체를 정렬할 수 있는 여러 방법들이 있습니다. **리스트**라고 불리는 아이에 대해 알아봅시다. 리스트란 서로 다른 객체들을 일렬로 나열한 것이라고 생각하시면 됩니다.

아래와 같이 리스트를 만듭시다. :

command-line

```
>>> []
[]
```

리스트가 비어있네요. 별로 유용해 보이지 않죠? 이제 로또 번호 목록을 만들어 봅시다. 매번 직접 우리가 반복하기는 귀찮으니, 리스트에 변수를 넣어봅시다.

command-line

```
>>> lottery = [3, 42, 12, 19, 30, 59]
```

좋아요, 리스트를 만들었어요! 이제 무엇을 할 수 있을까요? 리스트 안에 들어간 로또 번호는 몇 개인지 알려면 어떻게 해야할까요? `len()` 함수를 사용할 수 있겠죠? 이미 앞에서 해봤어요!

command-line

```
>>> len(lottery)
6
```

# 리스트 정렬

네! `len()` 함수는 리스트 안에 있는 객체의 수를 알려줍니다. 유용하죠? 이제 정렬을 해봅시다.

command-line

```
>>> lottery.sort()
```

숫자를 오름차순으로 정렬했지만 출력값이 보이지 않네요. 어떻게 변했는지 리스트를 출력해 확인해 봅시다. :

command-line

```
>>> print(lottery)
[3, 12, 19, 30, 42, 59]
```

리스트 안에 있는 숫자들은 오름차순으로 정렬되었어요. 잘했습니다!

이번에는 내림차순으로 정렬해볼까요?

command-line

```
>>> lottery.reverse()
>>> print(lottery)
[59, 42, 30, 19, 12, 3]
```

# 리스트에 요소 추가

참 쉽죠? 여러분이 리스트에 새로운 값을 추가하고 싶다면 이렇게 해보세요. :

command-line

```
>>> lottery.append(199)
>>> print(lottery)
[59, 42, 30, 19, 12, 3, 199]
```

# 리스트 인덱싱 : 요소 찾기

첫 번째 숫자 하나만 보여주고 싶다면, **인덱스(indexes)**를 사용하세요. 인덱스는 리스트 내 아이템 위치를 나타내는 숫자입니다. 프로그래머는 0부터 세는 것을 선호합니다. 따라서 리스트에 있는 첫 번째 객체는 인덱스 0부터 시작하고, 그 다음은 1, 그 다음은 2.. 순번대로 번호를 매길 수 있습니다. `.[index숫자]` 를 입력해볼까요.

command-line

```
>>> print(lottery[0])
59
>>> print(lottery[1])
42
```

이렇게 리스트 이름과 대괄호`([])`안 객체의 인덱스를 사용해서, 리스트 안에 있는 다른 객체로도 접근이 가능합니다.

리스트 내 아이템을 지우려면 인덱스와 함께 `pop()` 메소드를 사용하면 됩니다. 아래 예제 코드를 따라하면서 배운 내용을 다시 기억해봅시다. 리스트에 맨 처음 있는 객체를 삭제해봅시다.

command-line

```
>>> print(lottery)
[59, 42, 30, 19, 12, 3, 199]
>>> print(lottery[0])
59
>>> lottery.pop(0)
59
>>> print(lottery)
[42, 30, 19, 12, 3, 199]
```

# 딕셔너리(dictionary)

딕셔너리는 리스트와 유사하지만, 인덱스가 아닌 키(key)로 값을 찾습니다. 키는 문자열이든, 숫자든지 상관없습니다. 빈 딕셔너리를 만드는 문법은 아래와 같습니다.

command-line

```
>>> {}
{}
```

방금 비어있는 딕셔너리를 하나 만들었답니다. 만세!

자, 이제 아래 명령어를 따라 작성해 봅시다. (딕셔너리 안에 있는 값은 마음대로 수정해도 됩니다)

command-line

```
>>> participant = {'name': 'Ola', 'country': 'Poland', 'favorite_numbers': [7, 42, 92]}
```

위 명령어로 `paticipant` 라는 새 변수를 만들었습니다. 이 변수 안에는 3개의 키-값 쌍이 들어있어요.

- 키(key) 는 `name` 이고, 값(value) 는 `'Ola'` 를 가리킵니다. ( `string` 입니다)
- 키 `country` 의 값은 `'Poland'` 입니다. (또 다른 `string` 입니다)
- 그리고 키 `favorite_numbers` 는 `[7, 42, 92]` 를 가리킵니다. ( `list` 에 숫자 세 개가 있습니다)

# 딕셔너리(dictionary)

아래와 같은 문법으로 개별 키의 값에 접근할 수 있습니다.

command-line

```
>>> print(participant['name'])
Ola
```

리스트와 조금 비슷해보이지만 인덱스가 아닌, 이름을 사용해 값을 찾는다는 것을 기억하세요.

만약 파이썬에 키(key)에 대응하는 값이 없다면 어떻게 될까요? 그럼 테스트 해봅시다!

command-line

```
>>> participant['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```

보세요, **KeyError**에러가 났습니다. 파이썬은 친절하게 `'age'` 키가 딕셔너리에 존재하지 않는다고 알려주네요.

# 딕셔너리 vs 리스트

그렇다면 언제 딕셔너리를 쓰고, 언제 리스트를 사용해야할까요? 이를 위한 판단 기준은 다음과 같습니다.

- 리스트 : 아이템 정렬이 필요할 때
- 딕셔너리 : 키(key)과 값(value)이 서로 연관되어 있거나, 효과적으로 (키를 사용해서) 어떤 값을 찾을 때

딕셔너리는 리스트와 유사하지만, *변경(mutable)*할 수 있습니다. 딕셔너리가 만들어진 후에도, 그 값을 마음대로 변경할 수 있다는 뜻이지요. 키/값을 나중에 추가할 수 있답니다.

command-line

```
>>> participant['favorite_language'] = 'Python'
```

리스트처럼 딕셔너리도 `len()` 메서드를 사용하여 키-값 쌍의 수를 리턴합니다. 한 번 커맨드라인에서 확인합시다.

command-line

```
>>> len(participant)
4
```

# 요약

훌륭하네요! 이제 모두가 프로그래밍에 대해서 많은 것을 알게 되었어요. 지금까지 우리가 배운 내용을 정리해볼까요.

- **에러** - 파이썬이 작성된 명령어를 이해하지 못할 때 에러가 발생합니다.
- **변수** - 객체에 적절한 이름을 붙여 코드를 가독성 좋게 작성할 수 있습니다.
- **리스트** - 특정한 순서로 정렬된 객체들이 저장된 목록입니다.
- **딕셔너리** - 키(key)-값(value) 쌍으로 이루어진 객체들이 저장됩니다.

다음 내용이 기대되나요? :)

# 값 비교하기

비교는 프로그래밍에서 매우 중요한 부분이에요. 비교를 할 때, 제일 쉬운 방법은 무엇일까요? 바로 숫자입니다. 어떻게 하는지 알아볼게요. :

command-line

```
>>> 5 > 2
True
>>> 3 < 1
False
>>> 5 > 2 * 2
True
>>> 1 == 1
True
>>> 5 != 2
True
```

파이썬이 두 숫자를 비교했네요. 파이썬은 메서드 결과도 비교 가능합니다. 멋지죠?

두 숫자를 비교할 때 왜 등호 부호 `==` 를 사용하는지 궁금했죠? 변수에 값을 넣을 때 등호 기호 `=` 하나만 사용했기 때문에, 두 대상이 같은지 서로 비교하기 위해서는 **항상** 두 개의 등호 기호 `==` 를 사용해야 합니다. 마찬가지로 서로 다른 대상일 경우에는 `!=` 기호를 사용합니다.
파이썬은 얼마든지 원하는 만큼 숫자를 비교할 수 있고 답을 알려줄 거랍니다! 똑똑한 녀석이죠?

- **and -** `and` 연산자는, 두 값 모두 참( `True` )일 경우에만 결과가 참( `True` )이 됩니다.
- **or -** `or` 연산자는, 둘 중 하나만 참( `True` )일 경우에 결과가 참( `True` )이 됩니다.

# True or False (불리언 : Boolean)

방금 전 파이썬의 새로운 객체 종류를 배웠는데요. 바로 **Boolean**입니다. 이번 장에서 가장 쉬운 내용이에요.

불리언의 객체 두가지 입니다.

- True(참)
- False(거짓)

파이썬은 항상 `True` 라고 써야 이해합니다. 첫 글자가 대문자고 나머지는 소문자여야만, Boolean으로 이해합니다. **true, TRUE, tRUE는 모두 틀린 표현이에요. -- `True` 만 올바른 표현입니다.** ( `False` 도 마찬가지입니다)

불리언을 변수로 사용할 수도 있습니다! 아래 코드를 실행해보세요.

command-line

```
>>> a = True
>>> a
True
```

# 조건문 :
# 논리적으로 판단

# If...elif... else문

대부분의 코드는 조건문을 만났을 때 실행됩니다. 파이썬은 **if**문을 사용합니다.

**python_intro.py** 파일을 다음과 같이 수정하세요.

python_intro.py

```
if 3 > 2:
```

저장하고 실행하면 아래와 같은 에러가 보일 거에요.

command-line

```
$ python3 python_intro.py
File "python_intro.py", line 2
          ^
SyntaxError: unexpected EOF while parsing
```

파이썬은 조건문 `3 > 2` 가 참인 경우(또는 값이 `True` 인 경우)에 어떻게 할 것인지 물어보네요. 이 경우에 파이썬이 "It works!"를 출력하게 해봅시다. **python_intro.py** 파일을 아래와 같이 수정하세요.

python_intro.py

```
if 3 > 2:
    print('It works!')
```

## 조건이 참(True) 이 아니라면 어떻게 되나요?

앞 예제에서 조건이 참( `True` )인 경우에만 실행되게 만들었어요. 하지만 파이썬은 `elif` 문과 `else` 문도 사용할 수 있습니다.

python_intro.py

```
if 5 > 2:
    print('5 is indeed greater than 2')
else:
    print('5 is not greater than 2')
```

실행하면 다음과 같이 출력됩니다.

command-line

```
$ python3 python_intro.py
5 is indeed greater than 2
```

## 조건에 따라 출력이 달라진다

만약 2가 5보다 크다면 두번째 명령이 실행됩니다. 참 쉽죠? 이제 `elif` 가 어떻게 작동하는지 봅시다.

python_intro.py

```
name = 'Sonja'
if name == 'Ola':
    print('Hey Ola!')
elif name == 'Sonja':
    print('Hey Sonja!')
else:
    print('Hey anonymous!')
```

그리고 실행해보세요.

command-line

```
$ python3 python_intro.py
Hey Sonja!
```

# 조건에 따라 출력이 달라진다

`if` 문을 쓴 다음에 원하는 만큼 `elif` 문을 계속 추가할 수 있어요. 이렇게요.

python_intro.py

```
volume = 57
if volume < 20:
    print("It's kinda quiet.")
elif 20 <= volume < 40:
    print("It's nice for background music")
elif 40 <= volume < 60:
    print("Perfect, I can hear all the details")
elif 60 <= volume < 80:
    print("Nice for parties")
elif 80 <= volume < 100:
    print("A bit loud!")
else:
    print("My ears are hurting! :(")
```

파이썬이 각 테스트를 순서대로 실행하고 출력합니다:

command-line

```
$ python3 python_intro.py
Perfect, I can hear all the details
```

# 주석 :
# 코드설명

# # 설명할 글

주석은 `#` 으로 시작하는 줄입니다. 파이썬은 `#` 의 내용을 무시합니다. 주석은 코드를 읽는 다른 사람들이 보다 쉽게 이해할 수 있도록 작성합니다.

아래 예제를 보세요. :

python_intro.py

```
# volume 값을 바꿔보세요
if volume < 20 or volume > 80:
    volume = 50
    print("That's better!")
```

모든 라인마다 주석을 작성할 필요는 없지만, 코드의 역할과 복잡한 수행 내용을 정리 요약할 때 사용됩니다.

사용 방법 :
1. # 설명할 글
2. 텍스트 선택 후 컨트롤+/

# 함수 :
# 필요한 기능

# 나만의 함수 만들기

앞에서 했던 `len()` 와 같은 함수를 사용했었죠? 지금부터는 직접 함수를 만드는 법을 배울 거에요!

함수는 파이썬이 명령어의 나열이에요. 파이썬의 각 함수는 `def` 로 시작하고, 이름을 붙일 수 있고, 여러 매개변수를 가질 수 있어요. 쉬운 것부터 시작할게요. **python_intro.py** 파일을 열어 코드를 다음과 같이 고치세요.

python_intro.py

```
def hi():
    print('Hi there!')
    print('How are you?')

hi()
```

우리가 만든 첫 번째 함수네요!

파일 맨 밑에 왜 함수의 이름을 적었는지 궁금할 거에요.왜냐하면 파이썬은 파일을 위에서 아래로 읽어 실행하기 때문이에요. 함수를 사용하기 위해서는 하단에 다시 적어야 합니다.

## 함수 속의 매개변수

다음으로 매개 변수와 함께 첫 번째 함수를 만들어 볼게요. 전에 사용했던 예제를 다시 사용할 거에요. 함수를 실행하면, 이름을 부르고 'hi'를 말하게요.

python_intro.py

```
def hi(name):
```

함수에 매개변수인 `name`를 넣었어요. :

python_intro.py

```
def hi(name):
    if name == 'Ola':
        print('Hi Ola!')
    elif name == 'Sonja':
        print('Hi Sonja!')
    else:
        print('Hi anonymous!')

hi()
```

이런, 에러가 나왔네요. 다행히도, 파이썬 에러는 유용한 오류 메세지를 보여준답니다. 에러 메시지를 보면 함수 `hi()` 가 한 개의 인자값( `name` ) 을 필요로 하는데, 그 함수를 호출할 때 빼먹고 실행했다는 것을 알려주고 있어요. 이제 아래와 같이 코드를 수정해봅시다.

python_intro.py

```
hi("Ola")
```

그리고 다시 실행해보세요.

command-line

```
$ python3 python_intro.py
Hi Ola!
```

이름을 바꿔볼까요?

python_intro.py

```
hi("Sonja")
```

그리고 실행하세요.

command-line

```
$ python3 python_intro.py
Hi Sonja!
```

이제 다른 이름(Ola와 Sonja가 아닌)을 넣는다면 어떻게 될까요? 아래처럼 나오게 만들어보세요. :

command-line

```
Hi anonymous!
```

어때요, 정말 굉장하지 않나요? 이로써 인사 함수가 호출될 때마다, 매번 사람 이름을 반복할 필요가 없어요. 그래서 함수가 필요답니다. 매번 똑같은 코드를 반복해 작성하지 않아도 돼요!

좀 더 똑똑하게 만들어 볼게요. - 2명 이상 이름이 있는 경우, 모든 이름마다 조건문을 추가하는 건 꽤 귀찮은 일이 될거에요. 그렇죠?

python_intro.py

```
def hi(name):
    print('Hi ' + name + '!')

hi("Rachel")
```

이제 코드를 실행해 봅시다. :

command-line

```
$ python3 python_intro.py
Hi Rachel!
```

축하합니다! 우리는 이제 함수를 만들 수 있게 되었어요! :)

# 반복문 :
# 코딩의 힘, 자동 반복

# 반복문

이제 마지막 부분입니다. 금방 했죠? :)

프로그래머는 반복되는 일을 하는 것을 좋아하지 않아요. 프로그래밍은 모든 것을 자동화하는 것입니다. 모든 사람의 이름을 일일이 입력해서 hi 라는 메세지를 출력하게 만들고 싶지 않겠죠. 반복문을 사용하면 이를 해결할 수 있습니다.

리스트를 기억하고 있죠? 아래 girls 리스트를 보세요.

python_intro.py

```
girls = ['Rachel', 'Monica', 'Phoebe', 'Ola', 'You']
```

우리는 여자 아이들 이름을 불러 인사를 하고 싶어요. 우리는 이미 `hi` 함수를 만들었으니 반복문을 사용해봅시다. :

python_intro.py

```
for name in girls:
```

The `for` 문은 `if` 문과 다음 줄은 4칸 들여쓰기를 해야합니다.

아래 완성된 코드입니다.

# 반복문

python_intro.py

```
def hi(name):
    print('Hi ' + name + '!')

girls = ['Rachel', 'Monica', 'Phoebe', 'Ola', 'You']
for name in girls:
    hi(name)
    print('Next girl')
```

실행하면 다음과 같습니다.

command-line

```
$ python3 python_intro.py
Hi Rachel!
Next girl
Hi Monica!
Next girl
Hi Phoebe!
Next girl
Hi Ola!
Next girl
Hi You!
Next girl
```

# 반복문

`girls` 리스트의 요소마다 여러분이 `for` 문에 넣은, 들여쓰기를 한 모든 코드들이 반복됩니다.

`for` 문에 `range()` 함수로 숫자범위를 지정해 사용할 수 있어요.

python_intro.py

```
for i in range(1, 6):
    print(i)
```

이렇게 출력됩니다.

command-line

```
1
2
3
4
5
```

`range()` 는 매개변수로 넘겨진 숫자부터 시작하는 숫자 리스트를 만들어주는 함수입니다.

두 숫자 중 두 번째(마지막)숫자는 리스트에 포함되지 않는다는 것을 기억하세요. (즉 `range(1, 6)` 는 1부터 5까지 카운트 하며 숫자 6은 리스트에 포함되지 않습니다) "range()"는 반만 열려 있기 때문에, 첫번째 숫자는 포함되지만 마지막 숫자는 포함되지 않아요.

# Python 기초문법 요약 시트

## Beginner's Python Cheat Sheet

### Variables and Strings

*Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.*

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

### Lists

*A list stores a series of items in a particular order. You access items using an index, or within a loop.*

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

### Lists (cont.)

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

### Tuples

*Tuples are similar to lists, but the items in a tuple can't be modified.*

Making a tuple

```
dimensions = (1920, 1080)
```

### If statements

*If statements are used to test for particular conditions and respond appropriately.*

Conditional tests

```
equals            x == 42
not equal         x != 42
greater than      x > 42
  or equal to     x >= 42
less than         x < 42
  or equal to     x <= 42
```

Conditional test with lists

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

### Dictionaries

*Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.*

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

### User input

*Your programs can prompt the user for input. All input is stored as a string.*

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```

# pandas 기초문법 요약시트

## Python For Data Science *Cheat Sheet*

### Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

### Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

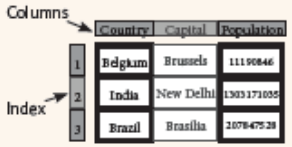
### Pandas Data Structures

#### Series

A one-dimensional labeled array capable of holding any data type


```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

#### DataFrame




A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

### Asking For Help

```
>>> help(pd.Series.loc)
```

### Selection

Also see NumPy Arrays

#### Getting

```
>>> s['b']
 7

>>> df[1:]
   Country    Capital  Population
1   India  New Delhi  1303171035
2  Brazil   Brasilia   207847528
```

Get one element

Get subset of a DataFrame

#### Selecting, Boolean Indexing & Setting

**By Position**

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat[[0],[0]]
'Belgium'
```

Select single value by row & column

**By Label**

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
```

Select single value by row & column labels

**By Label/Position**

```
>>> df.ix[2]
Country      Brazil
Capital    Brasilia
Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasilia
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```

Select rows and columns

**Boolean Indexing**

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame

**Setting**

```
>>> s['a'] = 6
```

Set index a of Series s to 6

### Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

### Sort & Rank

```
>>> df.sort_index(by='Country')
>>> s.order()
>>> df.rank()
```

Sort by row or column index
Sort a series by its values
Assign ranks to entries

### Retrieving Series/DataFrame Information

#### Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

#### Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idmin()/df.idmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

### Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

### Data Alignment

#### Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a    10.0
  b     NaN
  c     5.0
  d     7.0
```

#### Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c     5.0
  d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

### I/O

#### Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
```

#### Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

#### Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()

```
>>> pd.to_sql('myDf', engine)
```

DataCamp
Learn Python for Data Science Interactively

# 10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the Cookbook.

Customarily, we import as follows:

```
In [1]: import numpy as np

In [2]: import pandas as pd
```

# Object creation

See the Data Structure Intro section.

Creating a `Series` by passing a list of values, letting pandas create a default integer index:

```
In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])

In [4]: s
Out[4]:
0    1.0
1    3.0
2    5.0
3    NaN
```

On this page



https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html