

# Introduction to Python

SEUNGWOO SCHIN

Language AI Lab, NCSOFT

본 문서는 실습환경 설정과 파이썬 문법 소개, 그리고 기초적인 프로그래밍의 개념들의 학습을 위해서 작성되었습니다. 예시 코드는 examples 폴더에 들어 있습니다.

## 1 실습환경 설정

실습환경 설정은 크게 언어 설치와, 그 언어로 된 소스코드를 편집하기 위한 편집기나 IDE를 설치하는 것 두 가지입니다.

### 1.1 파이썬 설치

본 수업에서는 파이썬을 이용할 것이기 때문에, 파이썬을 먼저 설치해야 합니다. 본 환경설정법은 장고 결스 튜토리얼<sup>1</sup>의 해당 부분을 참고하여 작성되었습니다.

**Windows** 설치할 수 있는 방법이 크게 2개가 있습니다. 두 방법 모두 설치 전에 사용 중인 컴퓨터 윈도우 운영체제가 32비트인지 64비트인지 확인해야 합니다. 확인법은 마이크로소프트 링크에서 찾아보실 수 있습니다.

- Vanilla Python

Vanilla Python이란 가장 기본적인 파이썬 언어만 포함하는 버전을 말합니다. 가볍고 설치가 용이한 장점이 있습니다.

윈도우용 파이썬 설치파일을 파이썬 공식 다운로드 링크에서 다운로드 할 수 있습니다. 본 교재에서는 파이썬 3 버전을 사용할 것이므로, Latest Python 3 Release - Python 3.x.x 를 찾아서 다운로드받으면 됩니다. 64 비트 버전의 Windows인 경우 Windows x86-64 executable installer를 다운로드하시고, 이외에는 Windows x86 executable installer를 다운로드하면 됩니다. 설치 프로그램을 다운로드 한 후에 실행하고 지시 사항을 따르세요.

설치하는 동안 "Setup(설치하기)"이라고 표시된 창이 나타납니다. 다음과 같이 "Add Python 3.x to PATH(python3.x를 경로에 추가)" 체크 박스를 체크하고 "Install Now(지금 설치하기)"를 클릭하세요.

- 아나콘다 설치

아나콘다는 파이썬에 더해서, 수치계산 및 머신 러닝에 필수적으로 쓰이는 라이브러리들을 같이 설치하는 방법입니다.

기본적으로는 위 바닐라 파이썬의 설치와 같습니다. 아나콘다 설치 페이지에서 위에서 확인한 비트수에 맞게 아나콘다를 내려받아 설치하면 됩니다.

설치 후, 윈도우 + r 키를 눌러 실행한 후, Run(실행) 창이 뜨면 cmd를 입력하고 실행하여 윈도우 명령 프롬프트를 실행한 후, python 혹은 python3X<sup>2</sup> 을 실행해 보세요. 'python'은(는) 내부 또는 외부

---

<sup>1</sup>장고결스 튜토리얼 : 파이썬 설치

<sup>2</sup>파이썬 버전에 맞는 숫자를 입력하세요. python3X를 그대로 입력하면 안됩니다.



Figure 1: 파이썬 설치과정

명령, 실행할 수 있는 프로그램, 또는 배치 파일이 아닙니다.이라는 메세지가 보이면 설치가 되었지만 이를 컴퓨터에서 찾지 못하는 중인 것입니다. 이런 경우, 환경 변수를 설정하여 컴퓨터에게 파이썬의 위치를 알려주어야 합니다.

환경 변수는 다음 과정을 통해 설정할 수 있습니다.

- 윈도우 탐색기를 연 후(아무 폴더나 클릭해서 열면 됩니다.), 왼쪽에서 '내 pc'를 오른쪽 클릭하고 속성으로 들어감.
- 오른쪽에서 고급 시스템 설정 클릭
- 고급/환경 변수 클릭
- (사용자이름)에 대한 사용자 변수(U)에서 Path 선택, 편집 클릭
- 파이썬, 혹은 아나콘다가 설치된 폴더의 경로를 추가

**OS X** 파이썬 공식 사이트로 가서 파이썬 설치 파일을 다운 받으세요.

- Mac OS X 64-bit/32-bit installer 파일을 다운받습니다.
  - python-3.6.1-macosx10.6.pkg을 더블클릭해 설치합니다.
- OS 버전에 따라 다릅니다만, 이미 설치되어 있을 수도 있습니다.

**Linux** 이미 파이썬 3이 설치되어 있을 것입니다.

## 1.2 커맨드 라인 익숙해지기

모든 os에는 텍스트 기반으로 명령을 내릴 수 있는 방법이 있습니다. 이렇게 명령을 내리는 방법을 command line interface(cli)라고 합니다. 윈도우의 경우는 위 설치 확인 과정에서 턴 명령 프롬프트가 윈도우에서 주로 사용할 cli이며, 맥이나 리눅스의 경우는 터미널을 실행하면 됩니다. 대부분의 cli는 거의 비슷한 방법으로 구동합니다.

*command\_name argument1 argument2 ...*

여기서 `command_name`은 명령어이고, `argument`는 그 `argument`를 실행하기 위해서 필요한 입력입니다. 본 수업에서는 다음의 명령어만 알아도 충분히 활용할 수 있습니다.

- `cd` : 특정 디렉토리로 들어갑니다. `cd ..`은 상위 디렉토리로 이동합니다.
- `dir(윈도우) / ls(OS X, linux)` : 현 디렉토리에 있는 파일과 디렉토리의 목록을 출력합니다.

## 1.3 소스코드 편집기

프로그램은 소스코드로 이루어져 있습니다. 이러한 소스코드는 본질적으로 텍스트 파일이며, 이는 여러분이 쓰는 기본 메모장으로도 소스 코드를 작성할 수 있음을 의미합니다. 하지만 대부분의 경우 소스 코드를 문법에 맞추어 작성하기 편하게 하기 위한 편집기들을 사용합니다. 아래와 같은 편집기들을 쓸 수 있습니다.

- Notepad++ : 가장 가벼운, 메모장에 syntax highlighting 기능만 추가된 것입니다. 필요에 따라 더 많은 플러그인을 설치하여 사용할 수 있습니다.
- Atom : Github에서 만든 소스코드 편집기입니다.
- Sublime Text

위와 같은 프로그램을 통해서 파이썬 코드를 작성할 수 있습니다. 이렇게 작성된 프로그램을 `filename.py`에 저장하고, os에 맞는 프롬프트를 이용해서 `python filename.py`를 실행하면 파이썬 소스 코드를 실행할 수 있습니다. 하지만 매번 이런 방식으로 실행시키는 것은 번거로울 수 있습니다. 이러한 문제점을 해결하기 위해서 통합 개발 환경(Integrated Development Environment, IDE)이 사용됩니다. IDE는 코딩, 디버깅<sup>3</sup>, 배포 등 프로그램 개발에 필요한 작업들을 처리할 수 있는 환경을 제공하는 소프트웨어를 말합니다. 파이썬용 IDE는 개인 취향에 따라서 아래의 옵션 중 골라서 설치하시면 됩니다.

- PyCharm : 가장 범용적으로 쓰이는 파이썬 IDE입니다.
- Spyder
- Eclipse : PyDev 플러그인 사용
- Visual Studio Code : 파이썬 플러그인 사용

## 1.4 Git

Git은 버전 관리 도구로써, 본 수업에서 필수적으로 사용되는 것은 아니지만 사용하는 것에 익숙해지면 쓰일 곳이 많습니다. 특히, 개발자에게 자신이 작성한 코드를 관리하고 공유하기 위해서는 필수적으로 알아야 합니다. 여기서는 따로 Git의 사용법을 다루지는 않습니다. 본 단락 역시 장고결스 튜토리얼의 해당 부분<sup>4</sup>을 참고하였습니다.

**Windows, OS X** git-scm를 링크에서 다운로드하면 됩니다.

---

<sup>3</sup>작성한 프로그램의 오류를 찾는 작업을 말합니다.

<sup>4</sup>장고결스 튜토리얼 : 배포

Linux sudo apt-get install git 으로 git을 다운로드할 수 있습니다.

## 2 프로그램과 프로그래밍

### 2.1 프로그램이란?

프로그램이란 컴퓨터에게 특정 형태의 입력을 처리하여, 원하는 형태의 출력을 반환하게 명령하는 것을 말합니다. 이는 수학에서의 함수의 정의와 거의 일치합니다.

프로그램을 작성하기 위해서는 컴퓨터에게 명령을 내려야 하므로, 컴퓨터가 이해할 수 있는 방식으로 명령을 내릴 수 있어야 합니다. 이 때 쓰이는 언어를 프로그래밍 언어라고 하며, 명령을 작성하는 것을 프로그래밍이라고 합니다. 또한 이렇게 작성한 명령을 소스 코드라고 합니다.

프로그래밍 언어는 다양하고, 각각 장단점이 있기 때문에 원하는 작업에 적절한 프로그래밍 언어를 사용하는 것은 중요합니다. 본 수업에서는 빅데이터를 다루기 위해서 가장 기본적으로 파이썬 언어를 사용할 것입니다. 파이썬은 가장 배우기 쉬운 언어 중 하나로, 기계학습을 비롯한 인공지능의 개발이나 빅데이터 분석에 광범위하게 이용되는 언어입니다.

### 2.2 프로그램의 구조

위에서 프로그램이란 특정 형태의 입력을 처리하여, 원하는 형태의 출력을 반환하는 것이라고 하였습니다. 이 때, 특정 형태라는 것은 컴퓨터가 이해할 수 있는 형태로 전달해야 합니다. 컴퓨터는 본질적으로 0과 1만 이해할 수 있으므로, 우리가 전달하고자 하는 어떤 것을 그대로 전달하는 것은 불가능합니다. 따라서 어떤 식으로 전달할지를 미리 정해야 하는데, 이는 프로그래밍 언어가 미리 정해놓기도 하고, 입력이 많이 복잡하다면 프로그래밍 언어를 이용하여 어떤 식으로 전달할지를 명시적으로 작성하기도 합니다. 예를 들어서, 내가 숫자를 컴퓨터에 전달하고 싶으면 숫자에 대응되는 형태를 미리 정해놓는 것입니다. 이를 자료형(Type)라고 합니다.

대개의 프로그래밍 언어는 각 자료형에서 많이 이용되는 가공법 역시 같이 제공합니다. 예를 들어서, 대부분의 언어는 숫자형 데이터에 대해서 사칙연산을 제공해줍니다. 이러한 가공법 또한 언어마다 쓰는 법이 다르므로 이에 대해서도 숙지해야 합니다.

입력을 컴퓨터가 이해할 수 있는 형태로 바꾼 후에는 이를 원하는 형태로 가공하도록 명령을 내려야 합니다. 이 때 가공하는 과정은 하나의 큰 함수를 만드는 것과 같습니다.

함수를 만드는 것 역시 프로그래밍 언어에 따라 차이가 있습니다. 보통 이 과정에서 익혀야 하는 문법<sup>5</sup>은 아래의 4 가지입니다.

- 함수의 정의 및 함수의 이용
- 변수의 정의
- 반복문
- 흐름 제어문

즉, 프로그램을 만든다는 것은

- 입력/출력을 정의
- 정의된 입출력에 맞는 함수를 구현

---

<sup>5</sup>여러분이 영어나 한국어에서 말하는 문법과 정확하게 같은 의미의 문법입니다. 프로그래밍 언어도 말 그대로 언어이기 때문입니다.

하는 것이며, 이 두 가지를 특정 언어로 할 수 있을 때 프로그래밍 언어를 익혔다고 할 수 있습니다. 다시 말해서, 입력/출력을 정의하기 위해서 적절한 자료형을 쓰거나 만들 수 있으며, 함수와 변수를 정의하고, 필요한 반복문과 흐름 제어를 할 수 있으면 언어의 기본적인 문법을 익혔다고 볼 수 있습니다. 요약하면, 다음의 항목들을 이해했을 때 기본적인 문법을 이해했다고 볼 수 있습니다.

- 언어의 자료형
  - 언어에서 제공하는 자료형과 그 가공법 이해
  - 언어로 새로운 자료형 생성
- 언어의 문법
  - 변수의 지정
  - 함수의 정의 및 함수의 이용
  - 반복
  - 흐름 제어

어떤 복잡한 프로그램이더라도 위 기본 구조는 언제나 같습니다. 즉, 프로그래밍은 언제나 하나의 함수를 만드는 것이며, 이를 만드는 방법은 언어마다 다를지라도 저 구조는 언제나 일정합니다. 이를 꼭 기억하시고 추후 파이썬 외의 다른 언어를 익힐 때도 염두에 두고 학습하시기 바랍니다.

### 3 파이썬 소개

---

#### Implementation 1: Hello World! (*hello.py*)

---

```
1 def main():
2     print('Hello World!')
3
4 main()
```

---

본 단락에서는 파이썬 언어의 기본적인 문법과, 파이썬 코드를 읽는 방법에 대해서 짚고 넘어가고자 합니다. 위에서 언급했듯이 언어를 익히는 것은 다음의 항목들을 학습하는 것입니다.

- 언어의 자료형
  - 언어에서 제공하는 자료형과 그 가공법 이해
  - 언어로 새로운 자료형 생성
- 언어의 문법
  - 변수의 지정
  - 함수의 정의 및 사용
  - 반복
  - 흐름 제어

각 항목을 하나씩 천천히 살펴보겠습니다.

### 3.1 파이썬 데이터 타입

#### 3.1.1 파이썬 언어에서 제공해주는 자료형

파이썬은 기본적으로 다음의 자료형을 지원<sup>67</sup>합니다.

- Boolean Type : 참/거짓 값을 나타내는 타입입니다.
- Numeric Types : 일반적으로 쓰이는 숫자를 나타내는 타입입니다.
- Sequential Types : 배열 형태의 타입입니다.
- Mapping Types : key-value 순서쌍 형태의 타입입니다. 이 타입을 가진 데이터는 특정 key를 받으면 그 key에 맞는 value를 출력해줍니다.<sup>8</sup>

아래에서는 각 타입의 종류와 다양한 연산법에 대해서 알아볼 것입니다.

**Boolean Type** 참(True), 거짓(False)값을 나타냅니다. Boolean 값들은 and, or, not 연산이 가능합니다. 연산의 결과는 아래와 같습니다.

---

#### Implementation 2: Boolean Example (example\_bool1.py)

---

```
1 assert (True and True) == True
2 assert (True and False) == False
3 assert (False and True) == False
4 assert (False and False) == False
5 assert (True or True) == True
6 assert (True or False) == True
7 assert (False or True) == True
8 assert (False or False) == False
9 assert (not True) == False
10 assert (not False) == True
```

---

**Numeric Types** int, float, complex가 있습니다. 각각 정수, 실수<sup>9</sup>, 그리고 복소수를 나타냅니다.

---

#### Implementation 3: Numeric Types (example\_numeric1.py)

---

```
1 a = 1
2 print(type(a)) # <type 'int'>
3 b = 1.0
4 print(type(b)) # <type 'float'>
5 c = 1.0 + 1j # j 알파벳은 그냥 변수지만, 숫자 뒤에 붙어 오면 허수로
       인식합니다 .
6 print(type(c)) # <type 'complex'>
7 print('hello world!')
```

---

파이썬은 일반적인 사칙연산을 지원합니다. 아래에서 어떤 식으로 사칙연산이 사용되는지 볼 수 있습니다.

---

#### Implementation 4: Operations for Numeric Types (example\_numeric2.py)

---

```
1 print(1 + 2) # sum of x and y
```

---

<sup>6</sup>참조 링크 1(위키북스), 참조 링크 2(공식 문서)

<sup>7</sup>여기 리스트된 데이터형이 전부는 아니지만, 중요한 데이터형들이니 잘 알아두시길 권장합니다.

<sup>8</sup>프로그래밍 지식이 있으신 분은 파이썬에서의 dict가 해싱이라고 생각하시면 좋습니다.

<sup>9</sup>차후에 다루겠지만, 정확하게 실수를 나타내는 것은 불가능합니다. 더 정확하게는, 모든 실수를 정확하게 나타내는 것은 불가능합니다. 여기서의 float은 C언어에서의 double과 같다고 보는 것이 정확합니다.

```
2 print(3 - 1) # difference of x and y
3 print(2 * 4) # product of x and y
4 print(3 / 2) # quotient of x and y
5 print(3 // 2) # floored quotient of x and y
6 print(3 % 2) # remainder of x / y
7 print(-1) # x negated
8 print(+1) # x unchanged
9 print(abs(-2)) # absolute value or magnitude of x
10 print(int(3.2)) # x converted to integer
11 print(float(2)) # x converted to floating point
```

---

**Sequential Types** 파이썬에서는 list, tuple, range, string 등의 배열 형태의 데이터형을 지원합니다. 문자열 역시 sequential type의 일종으로 여겨집니다. 문자열 데이터형 string은 중요하기 때문에 따로 조금 더 다루겠습니다. 더 자세한 정보는 공식 Documentation을 참고하시면 됩니다.

---

**Implementation 5:** Sequential Types (*example\_sequence1.py*)

---

```
1 a = [1,2,3,4] # list
2 b = (1,2,3,4) # tuple
3 c = range(10) # range
4 d = 'hello world!' # string
```

---

Sequence 데이터형들은 다음의 연산들을 지원합니다.

- a in d : 배열(d) 안에 특정 원소(a)가 있는지를 검사합니다.
- + : 배열 두 개를 이어서 새로운 배열을 만듭니다. 같은 데이터형이여야 합니다. 예를 들어서,
- d[i] : d의 i번째 원소를 반환합니다.
- d[i:j:k] : d의 i번째 원소부터 j번째 원소까지, k번째 원소마다 선택하여 리스트를 만들어 반환합니다.
- d.index(elem) : d에서 elem이 처음으로 나오는 위치를 반환합니다.

---

**Implementation 6:** Operations for Sequential Types (*example\_sequence2.py*)

---

```
1 from example_sequence1 import *
2
3 print('e' in d) # True
4 print([1,2,3] + [4,5,6]) # [1,2,3,4,5,6]
5 print(d[1]) # 'e'
6 print(d[1:3]) # 'el'
7 print(d[1:6:2]) # 'el '
8 print(d[::-1]) # '!dlrow olleh'
9 print(len(d)) # 12
10 for idx, elem in enumerate(d):
11     print(idx, elem)
```

---

**Mapping Types** key-value 쌍을 저장하는 데이터형으로, 파이썬에서는 dict가 있습니다.

---

**Implementation 7:** Mapping Types (*example\_dict1.py*, line 7-26 omitted)

---

```
1 num2alphabet = \
2     { 1 : 'a',
3      2 : 'b',
```

```
4     3 : 'c',
5     4 : 'd',
6     5 : 'e',
7    26 : 'z', }
```

---

dict는 다음의 연산을 지원합니다.

- d[key] : dict에서 key에 해당되는 value를 반환합니다.
- d[key] = val : dict에서 key에 해당되는 value를 val로 업데이트합니다.
- d.keys() : dict의 key들을 반환합니다.

---

#### Implementation 8: Operations for Mapping Types (example\_dict2.py)

---

```
1 from example_dict1 import *
2
3 a[3] # 'c'
4 a.keys() # [1,2,3,...,26]
5 len(a) # 26
6 a[27]='A'
7 1 in a # True
8 del d[1]
9 1 in a # False
```

---

### 3.1.2 직접 만드는 자료형 : 클래스

본 단락에서는 파일에서 어떻게 원하는 자료형을 정의하는지 살펴보겠습니다.

**클래스란** 클래스는 흔히 과자들과 그 과자들로 찍어낸 과자로 비유됩니다. 클래스는 객체를 만들기 위한 코드 템플릿<sup>10</sup>을 말합니다. 인스턴스는 이 템플릿을 이용하여 만들어진 코드 덩어리로 생각할 수 있습니다. 예<sup>11</sup>를 들어서, 더하기만 가능한 간단한 계산기 코드를 만든다고 가정합시다. 다음과 같이 전역변수를 사용하면 어렵지 않게 구현할 수 있을 것입니다.

---

#### Implementation 9: 더하기만 가능한 계산기 (example\_cal1.py)

---

```
1 result = 0
2
3 def adder(num):
4     global result
5     result += num
6     return result
7
8 print(adder(3))
9 print(adder(4))
```

---

이 때, 각자 다른 계산기를 2개를 만들어서 사용하고자 하면, 다음과 같이 2개의 계산기를 만들어야 할 것입니다.

---

#### Implementation 10: 더하기만 가능한 계산기 2개 (example\_cal2.py)

---

```
1 result1 = 0
```

---

<sup>10</sup><https://www.hackerearth.com/practice/python/object-oriented-programming/classes-and-objects-i/tutorial/>

<sup>11</sup>본 예시는 <https://wikidocs.net/28점프> 투 파일에서 참고하였습니다.

```
2 result2 = 0
3
4 def adder1(num):
5     global result1
6     result1 += num
7     return result1
8
9 def adder2(num):
10    global result2
11    result2 += num
12    return result2
13
14 print(adder1(3))
15 print(adder1(4))
16 print(adder2(3))
17 print(adder2(7))
```

---

이렇게 코드를 작성할 경우, 완벽하게 똑같은 코드를 두 번 작성해야 함을 알 수 있습니다. 따라서 이러한 중복을 피하기 위해서, 파이썬에서는 - 그리고 객체지향적 언어에서는 - 클래스를 제공합니다. 클래스는 다음과 같이 계산기 코드를 템플릿화하여, 재사용이 용이하게 만듭니다.

#### Implementation 11: 더하기 계산기 클래스 (*example\_cal3.py*)

---

```
1 class Calculator:
2     def __init__(self):
3         self.result = 0
4
5     def adder(self, num):
6         self.result += num
7         return self.result
8
9 cal1 = Calculator()
10 cal2 = Calculator()
11
12 print(cal1.adder(3))
13 print(cal1.adder(4))
14 print(cal2.adder(3))
15 print(cal2.adder(7))
```

---

이 때, cal1, cal2는 클래스 Calculator의 인스턴스입니다.

클래스를 사용하는 이유는 개발의 속도와 유지보수의 편의성, 그리고 코드 디자인의 간결성 때문입니다. 잘 구성된 클래스의 경우, 구현하고자 하는 대상과 구현하는 프로그램 소스 코드간 대응이 직관적입니다. 예컨대 회계사용 프로그램을 작성할 경우, 고객/법인/재무재표 등의 클래스를 사용하게 될 것입니다. 이런 경우, 단순 코드 블럭으로 되어있는 것보다 더 직관적으로 프로그램 전체의 구조를 이해할 수 있으며, 이는 개발 속도와 정확도에 긍정적인 영향을 끼칠 것입니다. 프로그램의 유지보수 측면에서도 이와 비슷한 이유로 추후 유지보수가 간결해집니다.

이제 조금 더 자세하게 클래스(특히, 파이썬에서의 클래스에 대해서) 알아보겠습니다.

**파이썬에서의 클래스** 위 단락에서 클래스가 무엇인지, 그리고 왜 필요한지에 대해서 간단하게 살펴보았습니다. 본 단락에서는 클래스를 구성하는 요소를 살펴보고, 이를 정의하는 파이썬 문법에 대해서 알아보고자 합니다. 이해를 돋기 위해서 여기서는 문자열을 다루는 간단한 클래스를 만들어보고자 합

니다. 먼저,

클래스는 속성(attribute)과 메소드(method)로 구성됩니다. 속성은 클래스 속성과 인스턴스 속성으로 분류되며, 메소드는 인스턴스 메소드, 클래스 메소드, 스태틱 메소드로 분류됩니다.<sup>12</sup>

각각의 예시에 대해서 아래 코드에서 살펴보겠습니다. 먼저, 가장 기본적인 부분만을 작성한 코드를 살펴보겠습니다.

---

#### Implementation 12: Making a MyString Class (example\_class1.py)

---

```
1 class MyString:  
2     class_name = 'MyString'  
3     maker = 'Schin'  
4  
5     datum = ''  
6  
7 a = MyString # a is a class  
8 b = MyString() # b is an instance
```

---

위 코드에서 파이썬 클래스의 각 요소들을 설명하면 다음과 같습니다.

- class : 클래스를 정의하는 키워드입니다. 이 키워드 뒤의 단어가 클래스의 이름이 됩니다. 여기서는 MyString입니다.
- maker = 'Schin' : 클래스의 속성을 지정합니다. 여기서는 class\_name, maker, datum의 3개의 속성이 있습니다. 편의상 datum을 우리가 다루고자 하는 문자열로 생각하겠습니다.
- a = MyString : a라는 변수의 값으로 MyString이라는 클래스를 지정합니다.
- b = MyString() : b라는 변수의 값으로 MyString이라는 인스턴스를 지정합니다.

여기서 클래스(a)와 인스턴스(b)의 차이를 살펴보기 위해서 다음 코드를 실행해 보겠습니다.

---

#### Implementation 13: Making a MyString Class (example\_class1.py)

---

```
1 print(a.class_name) # prints 'MyString'  
2 print(b.class_name) # prints 'MyString'  
3 a.class_name = 'new MyString'  
4 print(a.class_name) # prints 'new MyString'  
5 print(b.class_name) # prints 'new MyString'  
6 b.class_name = 'MyString again'  
7 print(a.class_name) # prints 'new MyString'  
8 print(b.class_name) # prints 'MyString again'
```

---

여기서 볼 수 있듯이, 클래스 자체의 변화는 인스턴스에 영향을 주지만 그 역은 성립하지 않습니다. 이는 모든 인스턴스는 클래스의 틀을 따르기 때문입니다. 위에서의 쿠키틀의 예시를 들면, 쿠키 하나를 바꾼다고 쿠키틀이 바뀌지는 않는 것과 같습니다. 반대로, 쿠키틀을 바꾸면 모든 쿠키는 영향을 받게 됩니다.

본격적으로 문자열을 다루기 위해서 b에 우리가 다루고자 하는 문자열을 저장하고자 합니다. 이는 다음과 같은 방식으로 할 수 있습니다.

---

#### Implementation 14: MyClass Attribute (example\_class1.py)

---

```
1 b.datum = 'hello world!'  
2 print(b.datum)
```

---

<sup>12</sup>파이썬의 경우, private method를 지원하기는 하나 완벽하게 private하지는 않습니다. \_\_로 시작하는 속성이나 메소드는 private로 분류되지만, 이것이 완벽하게 클래스 밖에서 은닉되어 있지는 않습니다. 예컨대, 클래스 이름이 Mycls이고 속성이 \_\_attr1일 경우, Mycls\_\_attr1로 접근할 수 있습니다. 더 자세한 내용은 참고 링크를 참고하세요.

이제 이 후에는 어떻게 해야 할까요? 문자열을 저장하는 것만으로는 충분하지 않습니다. 이제 조금 더 복잡한 클래스 문법을 살펴보겠습니다.

---

**Implementation 15: Making a MyString Class (example\_class2.py)**

---

```
1 class MyString:
2     class_name = 'MyString'
3     maker = 'Schin'
4
5     def __init__(self, datum = ''):
6         self.datum = datum
7
8     # instance method
9     def get_first_letter(self):
10        return self.datum[0]
11
12    def most_frequent_word(self):
13        bow = MyString._create_bow(self.datum)
14        res = ''
15        tmp = 0
16        for k in bow.keys():
17            if bow[k] > tmp:
18                res = k
19                tmp = bow[k]
20        return res
21
22    # magic method
23    def __add__(self, other):
24        if isinstance(other, self.__class__):
25            return MyString(datum = self.datum + other.datum)
26        return NotImplemented
27
28    @classmethod
29    def assign_author(cls, author):
30        cls.author = author
31
32    @staticmethod
33    def _create_bow(input_str):
```

---

- 속성(Attribute) : 클래스나 인스턴스가 가지고 있는 정보를 말합니다. 예를 들어서, 문자열의 경우라면 문자열의 내용이나 저자 등이 있을 것입니다.
- 메소드(method) : 클래스나 인스턴스의 상태를 변화시키거나, 새로운 클래스를 만드는 등의 작업을 하는 함수입니다.

위 소스 코드를 통해서 우리는 다음과 같은 일들을 할 수 있습니다.

---

**Implementation 16: MyClass Attribute (example\_class2.py)**

---

```
1 a = MyString('hello')
2 b = MyString('world')
3
4 print(a.get_first_letter())
5
6 c = a+b # MyString('helloworld')
```

```
7 print(c.datum)
8
9 c.assign_author('Shin')
10 print(c.author)
11 print(a.author)
12
13 d = MyString('hello world it is so nice to meet you how are you doing world')
14 print(d.most_frequent_word())
```

---

### 3.2 반복문과 흐름제어

**if-elif-else** 다른 모든 언어와 비슷하게, 파이썬에서도 if-else 문을 지원합니다. 아래와 같은 문법으로 사용됩니다.

---

```
1 if cond1:
2     # when cond1 is True
3 elif cond2:
4     # when cond1 is False and cond2 is True
5 else:
6     # when cond1 is False and cond2 is False
```

---

**for loop** 파이썬에서의 for loop는 임의의 Sequential Type 변수에 대해서, 그 변수 안의 원소를 한번씩 돌게 됩니다. 예를 들어서 아래 코드를 살펴봅시다.

**Implementation 17:** For Loop Example (*example\_for1.py*)

---

```
1 for elem in ['a', 'b', 'c']:
2     print(elem)
```

---

**while loop** 파이썬에서의 while문은 다른 언어에서의 while문과 크게 다르지 않습니다. 아래의 소스 코드를 살펴보면 알 수 있을 것입니다.

**Implementation 18:** While Loop Example (*example\_while1.py*)

---

```
1 i = 0
2 while i<10:
3     print(i)
4     i += 1
```

---

### 3.3 함수의 정의와 이용

#### 3.3.1 함수

파이썬에서 함수는 다음과 같이 정의합니다.

**Implementation 19:** Function Syntax (*example\_function1.py*)

---

```
1 def function(args):
2     return None
```

---

위 소스코드에서 각 항목은 아래와 같은 의미를 가집니다.

- def : 함수 정의 키워드입니다.
- function : 함수 이름을 나타냅니다.
- args : 함수 인자입니다. 아래와 같은 옵션이 있습니다.
  - arg
  - arg\_default : 함수 인자의 기본값을 정해줄 때, =을 이용하여 기본값을 지정해줄 수 있습니다.
  - \*arg\_list : 정해지지 않은 수의 인자를 받고자 할 때, \*을 하나 붙여서 들어온 인자를 배열로 받을 수 있습니다.
  - \*\*arg\_dict : 정해지지 않은 수의 이름이 명시된 인자를 받고자 할 때, \*를 두개 붙여서 들어온 인자들을 dict 형태로 받을 수 있습니다.
- return None : 함수의 결과값으로 return 뒤의 구문을 반환합니다.

아래의 코드<sup>13</sup>를 보면 조금 더 명백해집니다.

---

#### Implementation 20: Function Argument Options (*example\_function2.py*)

---

```
1 def f(a = 0, *args, **kwargs):  
2     print("Received by f(a, *args, **kwargs)")  
3     print("=> f(a=%s, args=%s, kwargs=%s" % (a, args, kwargs))  
4     print("Calling g(10, 11, 12, *args, d = 13, e = 14, **kwargs)")  
5     g(10, 11, 12, *args, d = 13, e = 14, **kwargs)  
6  
7 def g(f, g = 0, *args, **kwargs):  
8     print("Received by g(f, g = 0, *args, **kwargs)")  
9     print("=> g(f=%s, g=%s, args=%s, kwargs=%s)" % (f, g, args, kwargs))  
10  
11 print("Calling f(1, 2, 3, 4, b = 5, c = 6)")  
12 f(1, 2, 3, 4, b = 5, c = 6)
```

---

위 프로그램의 실행 결과는 다음과 같습니다.

---

#### Implementation 21: Output for Function Argument Options

---

```
1 Calling f(1, 2, 3, 4, b = 5, c = 6)  
2 Received by f(a, *args, **kwargs)  
3 => f(a=1, args=(2, 3, 4), kwargs={'c': 6, 'b': 5})  
4 Calling g(10, 11, 12, *args, d = 13, e = 14, **kwargs)  
5 Received by g(f, g = 0, *args, **kwargs)  
6 => g(f=10, g=11, args=(12, 2, 3, 4), kwargs={'c': 6, 'b': 5, 'e': 14, 'd': 13})
```

---

**람다 함수** 람다 함수란 익명함수를 뜻합니다. 이는 람다함수가 변수명을 가질 수 없음을 의미하는 것 이 아닙니다. 예컨대, 아래의 코드에서의 func1, func2는 둘 다 같은 함수(주어진 수에 2를 더하는)이며, func1은 람다식으로 작성되었지만 엄연히 func1이라는 이름을 가지고 있습니다.

---

#### Implementation 22: Lambda Function Example (*example\_lambda1.py*)

---

```
1 func1 = lambda x: x+2  
2  
3 def func2(x):  
4     return x+2
```

---

<sup>13</sup>stackoverflow 질문 : Understanding kwargs in Python 참조

```
5  
6 func3 = lambda x,y,z : x+y+z  
7 func4 = lambda *args : sum(args)
```

---

람다식의 문법은 위 소스 코드에서 볼 수 있듯이 다음과 같이 이루어집니다.

- `lambda` : 람다함수 키워드. 람다함수 뒤의 구문 중 콜론 전에 있는 구문은 함수의 인자를, 뒤는 반환하는 값을 나타낸다.
- `x,y,z(func3)/*args(func4)` : 람다함수의 인자. 쉼표로 구분되며, 상기된 `*args` 등도 똑같이 사용 가능함을 `func4`에서 확인할 수 있다.
- `x+y+z(func3)/sum(args)(func4)` : 람다함수의 반환값.

익명함수가 가지는 이점 중 하나는, 우리가 정수나 문자열을 다루듯이 함수 또한 하나의 변수로 다루고 싶을 때 편리하다는 점입니다. 본 단락에서는 일반화된 정렬 문제에서 어떤 식으로 람다식이 사용 가능한지 보여드리고자 합니다.

어떤 배열을 정렬하는 문제를 생각해 봅시다. 이 때, 어떤 배열의 원소들이 정수라면 정렬 결과에는 이의가 없을 것입니다. 예컨대, 아래의 코드의 마지막 라인에서 `AssertionError`가 나지 않는다면 충분할 것입니다.<sup>14</sup>

---

#### Implementation 23: Inspecting Function Calls (example\_lambda\_sort1.py)

---

```
1 def mysort(lst): # insertion sort  
2     if len(lst) == 1:  
3         return lst  
4     else:  
5         head, tail = lst[0], lst[1:]  
6         tail = mysort(tail)  
7         for idx, elem in enumerate(tail):  
8             if head <= elem:  
9                 return tail[:idx] + [head] + tail[idx:]  
10    return tail + [head]  
11  
12 assert mysort([2,1,3]) == [1,2,3]
```

---

하지만 주어진 리스트가 비교하기 어려운 것들로 되어있는 경우 - 예를 들어서, 숫자 3개짜리 튜플로 되어있는 경우 -에는 어떤 식으로 배열할 수 있을까요? 이를 위해서는 우선 배열의 원소를 서로 비교하기 위한 기준이 필요할 것입니다. 위 코드의 경우 원소간의 비교 기준은 대소관계이며, 8번째 라인 (`head>=elem`)에 이것이 반영되었다고 볼 수 있습니다. 여기서는 이 기준을 세 숫자의 합으로 생각해 봅시다. 그렇다면, 새로운 기준(세 숫자의 합)을 아래와 같이 반영할 수 있을 것입니다.

---

#### Implementation 24: Inspecting Function Calls (example\_lambda\_sort2.py)

---

```
1 def mysort(lst): # insertion sort  
2     if len(lst) == 1:  
3         return lst  
4     else:  
5         head, tail = lst[0], lst[1:]  
6         tail = mysort(tail)  
7         for idx, elem in enumerate(tail):  
8             if sum(head) >= sum(elem):  
9                 return tail[:idx] + [head] + tail[idx:]  
10    assert mysort([(1,2,3), (2,-4,2), (1,3,1)]) == [(1, 2, 3), (1, 3, 1), (2, -4, 2)]
```

---

<sup>14</sup> 물론 실전에서는 더 많은 테스트를 하시는 것을 권장드립니다.

이제 여기서 조금 더 나아가서, 다음과 같은 소스를 생각해 봅시다.

---

**Implementation 25: Inspecting Function Calls (example\_lambda\_sort3.py)**

---

```
1 def compare(l, r): # (1)
2     return sum(l) >= sum(r)
3
4 def mysort(lst, cmp): # (2)
5     if len(lst) == 1:
6         return lst
7     else:
8         head, tail = lst[0], lst[1:]
9         tail = mysort(tail)
10        for idx, elem in enumerate(tail):
11            if cmp(head, elem): # (3)
12                return tail[:idx] + [head] + tail[idx:]
13
14 assert mysort([(1,2,3), (2,-4,2), (1,3,1)], 
15               cmp = compare) == [(1, 2, 3), (1, 3, 1), (2, -4, 2)] # (4)
```

---

여기서, 위 소스를 조금 더 간소화해서 애초에 compare 함수를 def를 쓰지 않고 람다식을 이용하여 저렇게 쓸 수 있습니다.

---

**Implementation 26: Inspecting Function Calls (example\_lambda\_sort4.py)**

---

```
1 def mysort(lst, cmp = lambda x,y: sum(x) >= sum(y)): # (2)
2     if len(lst) == 1:
3         return lst
4     else:
5         head, tail = lst[0], lst[1:]
6         tail = mysort(tail)
7         for idx, elem in enumerate(tail):
8             if cmp(head, elem):
9                 return tail[:idx] + [head] + tail[idx:]
10
11 assert mysort([(1,2,3), (2,-4,2), (1,3,1)], 
12               cmp = compare) == [(1, 2, 3), (1, 3, 1), (2, -4, 2)] # (3)
```

---

**함수의 이용**    함수를 쓰는 방법에는 크게 2가지가 있습니다.

- func(arg) 꼴 : print 같은 함수에서 사용됩니다.
- arg.func(...) 꼴 : list 같은 것에서 append를 쓸 때 이렇게 쓸 수 있습니다.

### 3.4 파이썬 인터프리터의 이해

본 단락에서는 주어진 소스 코드를 파이썬이 계산하는 법을 알아볼 것<sup>15</sup>입니다. 본격적인 설명에 앞서, 변수의 종류에 대해서 설명하겠습니다. 일반적으로 변수에는 다음의 세 종류가 있습니다.

- Bound variable : 어떤 값이나 다른 변수에 의해서 값이 결정되는지 정해진 변수
- Binding variable : Bound variable의 값을 결정하는 변수

---

<sup>15</sup>파이썬 공식 Documentation 참조

- Free variable : Bound되지 않은 변수

이 때, 파이썬 인터프리터<sup>16</sup>는 **bound variable**을 **binding variable**로 대체(substitute) 하여 계산합니다. 만약 계산해야 하는 모든 변수들의 값이 결국 어떤 값(숫자, 문자열 등등)으로 환원되면 그 값을 계산하여 반환하고, 그렇지 않다면 에러를 반환합니다. 따라서 파이썬 인터프리터의 동작을 이해하는 것은 곧 어떤 식으로 변수들이 서로를 bind/bound 하는지를 이해하고, 이를 기반으로 **기계적으로** 변수를 적절한 값으로 대체하여 계산을 수행함을 의미합니다.

Binding이 일어나는 경우는 아래와 같습니다.

**import문의 사용** import문을 사용할 경우, import된 모듈에서의 모든 namespace가 bind됩니다.

**for loop** for loop에서 헤더는 루프 코드블럭 안에서 for loop 헤더에서 선언된 변수를 bind 합니다.

**함수, 클래스의 정의** 함수나 클래스의 정의는 함수나 클래스 이름을 bind하게 됩니다. 예를 들어서 아래 소스코드를 보면, compare이라는 변수는 1번 라인에 의해서 2번 라인에 binding되어, 14번 라인을 거쳐 11번 라인에서 쓰이게 됩니다.

---

**Implementation 27: Binding in Function definition (example\_lambda\_sort3.py)**

---

```
1 def compare(l, r): # (1)
2     return sum(l) >= sum(r)
3
4 def mysort(lst, cmp): # (2)
5     if len(lst) == 1:
6         return lst
7     else:
8         head, tail = lst[0], lst[1:]
9         tail = mysort(tail)
10        for idx, elem in enumerate(tail):
11            if cmp(head, elem): # (3)
12                return tail[:idx] + [head] + tail[idx:]
13
14 assert mysort([(1,2,3), (2,-4,2), (1,3,1)],
15               cmp = compare) == [(1, 2, 3), (1, 3, 1), (2, -4, 2)] # (4)
```

---

**=의 사용** 예를 들어서, 아래의 소스코드에서는 각각 a,b가 bound variable, a가 binding variable, c가 free variable입니다.

---

**Implementation 28: Types of Variables (variables.py)**

---

```
1 a = 1
2 b = a
3 c
```

---

**함수 인자의 binding** 함수 인자 역시 함수가 정의된 곳 내에서의 binding을 야기합니다. 예를 들어서 아래 소스코드를 살펴봅시다.

---

<sup>16</sup>사실 많은 인터프리터가 대부분 이렇게 동작합니다.

**Implementation 29: Inspecting Function Calls (example\_interpreter1.py)**

```
1 def func1(input_num):
2     a = int(input_num[0])
3     b = int(input_num[1])
4     c = int(input_num[2])
5
6
7     return func2(a,b,c,)
8
9 def func2(a,b,c):
10    return 100*c + 10*b + a
11
12 print(func1('123'))
```

---

이 때 출력될 값은 321일 것입니다. 이를 이해하기 위해서 파이썬 인터프리터 안에서 어떤 일이 벌어지는지 한 단계씩 살펴보도록 하겠습니다.

1. line 12 : func1('123')을 호출, 반환된 값을 출력함
2. line 1 : func1 정의된 부분으로 감
3. line 2-7 : 이 부분의 식을 계산하되, input\_num을 '123'으로 대체하여 계산함 (=binding이 일어남: input\_num 이 '123'에 bind됨)
  - (a) line 2-4 : int('123'[0]) == 1 과 같은 계산을 반복
  - (b) line 7 : func2(1,2,3)을 호출, 반환된 값을 반환함
4. line 9 : func2 정의된 부분으로 감
5. line 10 : 100c + 10b + a 계산하되, a,b,c를 각각 1,2,3으로 대체하여 계산함 (=binding이 일어남 : a,b,c가 각각 1,2,3에 bind됨)

## 4 유용한 라이브러리 살펴보기

파이썬에는 기본적으로 유용한 라이브러리가 많이 제공되며, 다른 사람들이 만든 라이브러리도 편하게 사용할 수 있습니다. 본 단락에서는 pip를 이용해서 라이브러리를 설치하는 것과 라이브러리 문서 읽기, 그리고 많이 쓰이는 라이브러리 몇 개를 살펴보도록 하겠습니다.

### 4.1 라이브러리 설치 및 사용하기

사실 파이썬에서 라이브러리란, 남이 짜놓은 파이썬 코드 중 쓰기 좋고 많이 쓰이는 것들을 말합니다. 즉 파이썬 라이브러리는 결국 파이썬 코드이고, 심지어 이 코드를 직접 읽어볼 수도 있습니다. 대개의 경우 파이썬이 설치된 경로(위에서 path에 추가한 경로)에서 Lib 폴더나, Lib/site-packages 폴더에서 이를 찾아볼 수 있습니다.

파이썬에서는 기본적으로 제공하는 라이브러리도 많지만, 아마 앞으로 여러 라이브러리를 깔아서 쓰셔야 할 것입니다. 라이브러리를 깔기 위해서는 파이썬에서 기본적으로 제공되는 pip라는 관리용 프로그램을 사용하는 것이 가장 간편합니다. (대부분의 경우 이것으로 다 됩니다.)

pip는 위에서 설명한 cmd에서 사용합니다. 주 사용처가 인터넷에서 다른 사람들이 작성한 라이브러리를 내려받는 것이므로 꼭 인터넷이 연결되었는지 확인하고 사용하시기 바랍니다. 사용할 때는 pip (기능명) (인자) 식으로 사용합니다. 어떤 기능들을 제공하는지 살펴보겠습니다.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17763]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\user>pip
Usage:
  pip <command> [options]

Commands:
  install          Install packages
  download         Download packages
  uninstall        Uninstall packages
  freeze           Output installed packages in requirements format.
  list              List installed packages
  show              Show information about installed packages.
  check             Verify that installed packages have compatible dependencies.
  config            Manage local and global configuration.
  search             Search PyPI for packages.
  wheel             Built wheels from your requirements.
  hash              Compute hashes of package archives.
  completion        A helper command used for command completion.
  help              Show help for commands.

General Options:
  -h, --help          Show help.
  --isolated         Run pip in an isolated mode, ignoring environment variables and user configuration.
  -v, --verbose       Give more output. Option is additive, and can be used up to 3 times.
  -V, --version      Show version and exit.
  --log <path>        Give a log file path. Option is additive, and can be used up to 3 times (corresponding to WARNING, ERROR, and CRITICAL logging levels).
  --proxy <proxy>    Path to a verbose appending log.
  --retries <retries> Maximum number of retries each connection should attempt (default 5 times).
  --timeout <timeout> Set the socket timeout (float) in seconds (default 5.0).
  --exists-action <action> Default action when a path already exists: (s)witch, (i)gnore, (b)ackup, (a)abort.
  --trusted-host <hostname> Mark this host as trusted, even though it does not have valid or any HTTPS.
  --cert <path>       Path to alternate CA bundle.
  --client-cert <path> Path to client certificate, a single file containing the private key and the certificate in PEM format.
  --cache-dir <dir>   Store the cache data in <dir>
  --no-cache-dir     Disable the cache.
  --disable-pip-version-check Don't periodically check PyPI to determine whether a new version of pip is available for download. Implied with --no-index.
  --no-color          Suppress colored output

C:\Users\user>
```

Figure 2: cmd에서 pip를 띄우는 방법

- **install** : 이름 그대로 라이브러리를 설치하는 용도입니다. 라이브러리를 깔고 싶으면, pip install (라이브러리 설치 이름) 을 이용하면 됩니다. 이 때, 라이브러리의 이름이 꼭 설치할 때 쓰는 이름과 같지는 않습니다. 설치할 때 쓰는 키워드는 보통 라이브러리 홈페이지의 install에 있는 경우가 많으니 이를 찾아보시면 편합니다.

- pip install (라이브러리명)==xx.xx.x : 특정 버전의 라이브러리를 설치합니다. 이게 왜? 하지 만 생각보다 쓸 일이 있습니다. 라이브러리가 업데이트되면 기존의 코드를 쓸 수 없어지는데, 코드를 고치느니 언제나 다시 맞는 버전을 설치하는 것이 편합니다.
- pip install (라이브러리명) -user : 현재 사용중인 유저를 위해서만 설치합니다. 서버를 운영하거나, 여러 명이 공동으로 쓰는 컴퓨터라면 쓸 일이 있습니다.

- **uninstall** : 위 install 명령어와 반대로, 설치된 라이브러리를 삭제합니다.
- **freeze/list** : 설치된 라이브러리를 조회합니다.
- **-version** : 설치된 pip 버전을 조회합니다.

설치된 라이브러리를 사용하고 싶으면 import 문을 이용합니다. 이 때 이용할 수 있는 방법이 여러 가지 있습니다.

- **import (라이브러리명)** : 라이브러리를 사용합니다. 불러온 라이브러리의 함수나 변수를 사용할 시에는 (라이브러리).(사용할 변수/함수명) 형식으로 사용합니다. 예를 들어서, re.match는 re 라이브러리의 match 함수를 의미합니다.
- ... as (별칭) : 라이브러리의 이름이 길거나 할 경우에 매번 타이핑하기가 귀찮기 때문에 이를 이용해서 별칭을 쓸 수 있습니다. 이 때는 위와 비슷하게 사용하되, (별칭).(사용할 변수/함수명)으로 사용합니다.

- `from` (라이브러리명) `import` (함수), (함수), ... : 라이브러리에서 사용할 함수/변수들만을 선택적으로 불러옵니다. 이 경우, 위와는 다르게 바로 (사용할 변수/함수명)으로 사용할 수 있습니다.
- `from` (라이브러리명) `import *` : 그 라이브러리의 모든 변수와 함수를 사용합니다. 위 `import` 문과는 다르게 바로 변수/함수명으로 사용 가능하지만, 기존에 있던 이름과 겹치는 것이 있을 경우 충돌이 일어날 수 있습니다. 예를 들어서, `re` 라이브러리에는 `split`이라는 함수가 있는데, 이는 `string`의 `split` 함수와 이름이 같기 때문에 충돌이 일어납니다. 이 경우 불러온 함수명으로 인식하여 사용합니다.

## 4.2 라이브러리 문서 읽기

대부분의 유명한 라이브러리는 문서화가 대단히 잘 되어 있어, 구글링을 하는 것 이상으로 많은 정보를 얻을 수 있습니다만 의외로 잘 사용되지 않습니다. 다년간의 경험 및 초보일 때의 기억에 따르면, 초보자가 문서를 읽는 것에 몇 가지 어려움이 있기 때문으로 생각됩니다. 따라서 문서를 읽는 틈에 대해서 간단하게 정리해 보도록 하겠습니다. 이 부분은 개인적인 사견이 많이 들어간 부분이기 때문에 정답은 아닙니다만, 제가 생각할 때 도움이 되었거나 제가 주로 하는 방법을 적어보았습니다.

기본적으로 파이썬에 설치되어 있는 라이브러리들은 파이썬 공식 문서화 홈페이지<sup>17</sup>에서 상세한 설명을 제공합니다. 그렇지 않은 라이브러리들의 경우라도 보통은 라이브러리 홈페이지에서 설명을 제공합니다. 구글링 시 (라이브러리 이름) `documentation` 으로 검색하면 보통 나옵니다. 이 중 가능하면 공식 문서나 `readthedocs.io` 사이트의 문서를 보는 것이 좋습니다. 문서의 양이 방대하기 때문에, 전체 문서를 읽는 것은 말이 안되지만 필요한 부분의 경우 정독해보는 것이 좋습니다. 특히, 필요한 라이브러리의 `intro` 부분이 있다면 이를 읽어보는 것은 크게 도움이 됩니다.

예를 들어서 파이썬의 멀티프로세싱 라이브러리를 살펴보겠습니다.

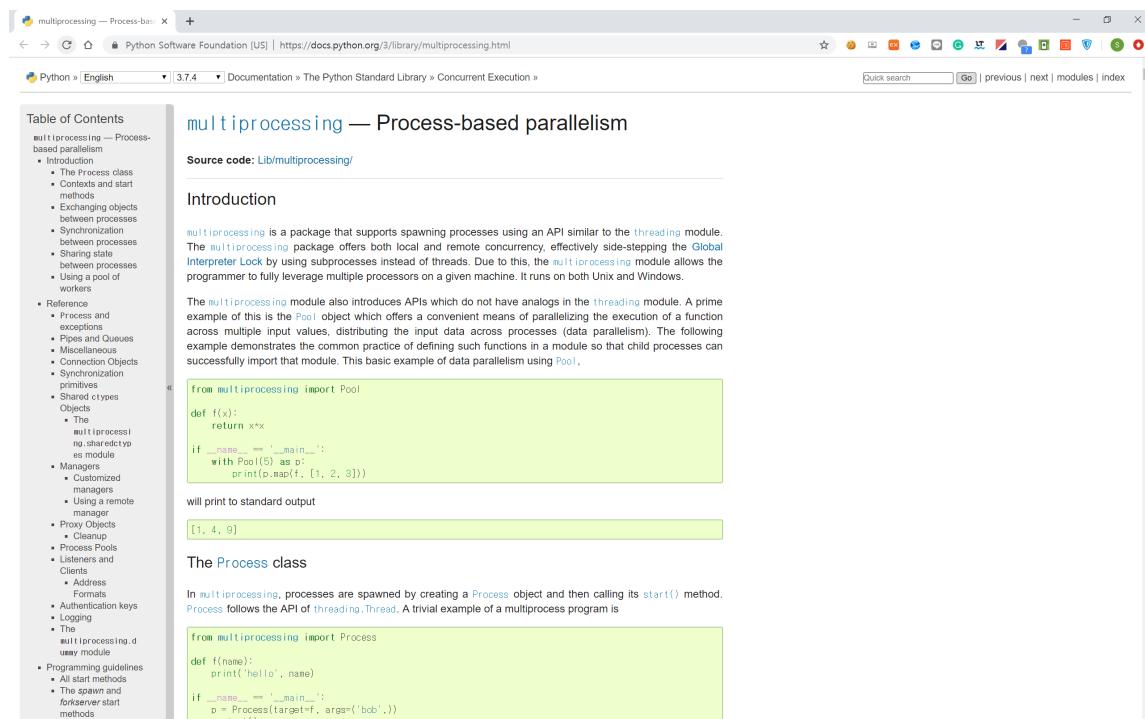


Figure 3: `multiprocessing` 라이브러리 문서 홈페이지

<sup>17</sup><https://docs.python.org/3/library/intro.html>

이 경우, 옆에 있는 table of contents에서 Introduction 부분은 정독하는 것이 좋습니다. 대개의 경우 여기를 정독하는 것으로 라이브러리의 개요를 이해할 수 있게 됩니다. 모든 라이브러리가 저렇게 introduction 부분이 나뉘어져 있는 것은 아니지만, 대개의 경우 라이브러리 전체에서 하고자 하는 내용을 간략히 적어놓은 페이지는 있으니 잘 찾아보시기 바랍니다. Reference 부분은 읽는 것이 아닙니다. Reference는 대개의 경우 사전에 가깝기 때문에, 양이 방대하고 모든 내용을 다 알 필요는 없습니다. 하지만 이 경우에도 각 표제항을 읽는 방법은 필요합니다.

각 표제항은 대부분의 경우, 함수 혹은 클래스에 대한 설명입니다. 대개의 경우 함수에 대한 것을 많이 찾아보게 됩니다. 함수의 경우, input/output, 혹은 줄여서 i/o를 파악하는 것이 가장 중요합니다. 내적 동작 원리의 경우 몰라도 그만인 경우가 많고, 정 필요한 경우에는 그때 다시 자세히 보면 됩니다. 표제항 하나를 살펴보면서 어떤 식으로 읽어가는지 살펴보겠습니다.

#### `re.split(pattern, string, maxsplit=0, flags=0)`

Split *string* by the occurrences of *pattern*. If capturing parentheses are used in *pattern*, then the text of all groups in the pattern are also returned as part of the resulting list. If *maxsplit* is nonzero, at most *maxsplit* splits occur, and the remainder of the string is returned as the final element of the list.

```
>>> re.split(r'WW+', 'Words, words, words.')
['Words', 'words', 'words', '']
>>> re.split(r'(WW+)', 'Words, words, words.')
['Words', '', '', 'words', '', 'words', '', '']
>>> re.split(r'WW+', 'Words, words, words.', 1)
['Words', 'words, words.']
>>> re.split('[a-f]+', '0a3B9', flags=re.IGNORECASE)
['0', '3', '9']
```

If there are capturing groups in the separator and it matches at the start of the string, the result will start with an empty string. The same holds for the end of the string:

```
>>> re.split(r'(WW+)', '...words, words...')
['', '...', 'words', '', 'words', '...', '']
```

That way, separator components are always found at the same relative indices within the result list.

Empty matches for the pattern split the string only when not adjacent to a previous empty match.

```
>>> re.split(r'Wb', 'Words, words, words.')
['', 'Words', '', '', 'words', '', 'words', '']
>>> re.split(r'WW*', '...words...')
['', '', 'w', 'o', 'r', 'd', 's', '', '']
>>> re.split(r'WW*', '...words...')
['', '...', ' ', ' ', 'w', ' ', 'o', ' ', 'r', ' ', 'd', ' ', 's', ' ', ' ', ' ', ' ']
```

*Changed in version 3.1:* Added the optional *flags* argument.

*Changed in version 3.7:* Added support of splitting on a pattern that could match an empty string.

**Figure 4:** *re* 라이브러리의 *split* 함수

함수 입력의 경우, 함수의 정의부에서 볼 수 있습니다. 이 경우, *pattern*, *string*, *maxsplit*, *flags*라는 인자들이 있음을 알 수 있습니다. 각 인자의 뜻은 보통 바로 아래에서 설명합니다. 여기서 볼 수 있듯이, 인자들이 그냥 이름만 쓰여 있는 경우도 있고(*pattern*, *split* 등) 그렇지 않은 경우도 많습니다. 쓰이는 형식은 다음의 4가지입니다.

- 인자 이름 : 무조건 입력되어야 하는 인자입니다.
- (인자 이름) = (값) : 인자를 따로 입력하지 않을 경우, = 뒤의 값이 기본값으로 들어갑니다.

- \*args : 인자들의 갯수가 정해져 있지 않음을 의미합니다. 이 때 들어가는 인자들 각각은 이름이 따로 없습니다. 본 문서의 함수 단락을 참고하길 바랍니다. 꼭 키워드가 args는 아니지만 주로 args를 사용하며, 어떤 경우에도 앞에 \*가 하나 붙습니다.
- \*\*kargs : \*args와 비슷한데, 인자들 각각에 이름이 붙습니다. 이 역시 위 단락의 함수 부분의 설명을 참고하면 좋습니다. 역시 \*\*가 붙는 것은 고정이지만, kargs 키워드는 고정이 아닙니다.

출력의 경우는 명시하는 형식이 정해져 있지는 않기 때문에, 표제항을 읽으면서 살펴보아야 합니다. 대부분의 경우 첫 단락을 i/o를 설명하는데 쓰기 때문에, 이를 참고하는 것이 좋습니다.

많이 쓰이는 함수들의 경우는 위 표제항의 예시처럼 사용 방법 예시가 나와 있는 경우가 많습니다. 역시 가장 참고하기 좋은 예시이고 가장 신뢰성이 높으므로 적극적으로 활용합시다.

끝으로 일반적인 팁 몇 가지를 적어보겠습니다.

- 검색보다 지인에게 질문하는 것이 좋습니다. 하지만 질문하기 전에 질문을 명확하게 하기 위해서 검색을 해 보아야 합니다.
- 검색은 구글에서, 영어로 합니다. 한국어로 되어있는 문서들도 많습니다만, 대개의 경우 질이 좋지 않거나 검증되지 않은 경우가 많습니다.
- pdf 같은 파일보다 공식 사이트가 신뢰성이 높습니다. 공식 사이트의 경우 모든 버전에 대한 설명을 담고 있기 때문에 그렇습니다.
- stackoverflow 사이트를 애용합니다. 대부분의 질문은 여기에 올라와 있습니다. 다만, 버전과 내 컴퓨터에서 돌아갈지는 언제나 확인할 수 없습니다.

## 4.3 라이브러리 살펴보기

이제 많이 쓰이는 라이브러리 몇 개를 살펴보고 이를 활용해 보겠습니다. 여기서는 os/pickle, re, beautifulsoup/requests, multiprocessing 등을 살펴보고, 이를 이용해서 네이버 뉴스 크롤러를 짜 보도록 하겠습니다.

### 4.3.1 os/pickle 라이브러리

os라는 이름에서 알 수 있듯이, 운영체제에서 하는 일들을 주로 많이 합니다. 운영체제라는 말이 어렵다면, 보통의 윈도우 탐색기라고 생각하시면 됩니다. 디렉토리를 만들거나 탐색하는 것이 주 사용처입니다. 많이 쓰이는 함수들을 살펴보고, 예제 몇 개를 풀어보도록 하겠습니다. 또, 본 단락에서는 file을 다루는 것도 같이 살펴보도록 하겠습니다. 이는 os 라이브러리의 일부는 아니지만, 같이 쓰이는 일이 비일비재하기 때문에 같이 보는 것이 유용하기 때문입니다.

본 단락에서는 pickle 라이브러리 역시 살펴봅니다. pickle 라이브러리는 파일 오브젝트를 말 그대로 '절여서' 파일로 저장해주는 라이브러리입니다. 매번 오브젝트를 새로 만드는 것이 번거로운 경우가 많아서 역시 os 라이브러리와 같이 많이 사용됩니다.

**os 라이브러리** 컴퓨터에서 파일 시스템은 보통 수형도로 이해할 수 있습니다. 이렇게 볼 때, 그 수형도의 잎들을 우리는 파일이라고 하고, 그 사이 노드들을 폴더라고 합니다. 즉, 폴더는 안에 파일이나 다른 폴더를 담을 수 있지만 파일은 그렇지 못합니다. 수형도의 맨 위를 root라고 합니다. 보통 윈도우의 경우 드라이브가 파일 경로의 끝입니다.

파일 각각의 위치는 수형도상에서, 어떤 시작점에서 어떤 경로를 따라 파일에 도달하는지로 나타냅니다. 이 때, 시작점이 root인 경우 파일의 위치를 절대경로라고 하고, 시작점이 다른 파일이나 폴더인 경우 상대위치라고 합니다. 이 때, 경로는 각 폴더들의 이름을 어떤 구분자로 이어준 문자열입니다. 이 문자열은 운영체제에 따라서 다른데, 보통은

을 이용하게 됩니다. 수형도 상에서의 부모는 항상 ..으로 가리키게 됩니다. 자매는 따로 가리키는 표현법이 없습니다.

위와 같은 파일시스템에 접근하기 위해서 파일에서는 os라는 모듈을 사용합니다. 주로 많이 쓰이는 함수들은 아래와 같습니다.

- os.sep : 운영체제에서 쓰이는 구분자입니다.
- os.getcwd() : 현재 실행되는 경로를 말합니다.
- os.chdir(somepath) : 실행 경로를 somepath로 옮깁니다.
- os.path.isdir(somepath) : somepath가 폴더이면 True, 아니면 False를 리턴합니다.
- os.path.isfile(somepath) : somepath가 파일이면 True, 아니면 False를 리턴합니다.
- os.listdir(somepath) : somepath에 들어있는 폴더와 파일들을 리턴합니다.

**파일 다루기** 파일에서는 파일을 직관적으로 다룰 수 있습니다. 파일은 우선 열릴 수 있습니다. 이 때, 읽기 전용으로 읽을 수도 있고, 쓰기 위해서 읽을 수도 있고, 뒤에 덧붙이는 식으로 읽을 수도 있습니다. 이를 mode라고 합니다.

정확히는 다음과 같이 파일을 열 수 있습니다.

---

#### Implementation 30: Opening a file (example\_fileio.py)

---

```
1 f = open('filename', 'rb', encoding = 'utf-8')
2 with open('filename', 'rb', encoding = 'utf-8') as f:
3     # do sth...
```

---

각각 인자들을 조금 더 살펴보겠습니다.

- 첫 인자 : 열 파일의 경로입니다.
- 두번째 인자 : 파일을 열 때의 모드입니다. 모드는 (r|w|a)(b)?(+)? 형식이며, 각자의 의미는 다음과 같습니다.
  - (r|w|a) : 파일을 읽기 전용/쓰기/덧붙이기 모드로 엽니다. 쓰기 모드의 경우, 기존 내용을 다 지워버리기 때문에 주의해야 합니다.
  - (b)? : 파일에 바이트 형식으로 열지, 문자열 형식으로 열지를 결정합니다.
  - (+)? : 주어진 경로에 파일이 없는 경우, +가 있으면 그 파일을 만듭니다. +가 없는데 파일 경로에 그 파일이 없으면 에러를 리턴합니다.
- encoding : 파일을 열 때 인코딩을 지정합니다. 일반적으로 한글은 utf-8 혹은 euc-kr을 사용합니다. 보통은 utf-8이 권장됩니다.

파일을 연 후에는 읽거나 쓸 수 있습니다. 파일을 읽을 때는 다음의 함수들을 사용합니다.

- (파일 변수명).read() : 파일 전체 내용을 한번에 읽습니다. 열 때 바이트로 열었으면 바이트 객체를 리턴하고, 문자열로 열었으면 문자열을 리턴합니다.
- (파일 변수명).readline() : 파일의 현재 줄을 읽습니다. 이 때, 한 줄을 읽고 나면 한번 연 파일에서는 다음 줄로 돌아가지 못합니다.
- (파일 변수명).readlines() : 파일을 한 줄 한 줄 읽습니다. 이 때 역시 읽고 난 줄을 다시 읽을 수 없습니다.

파일에 쓰는 것은 대단히 간단합니다. 파일에 쓸 때는 (파일 변수명).write(쓸 내용)으로 쓸 수 있습니다. 다만, 열 때 w모드 혹은 a모드로 열어야만 쓸 수 있습니다.

**pickle** 라이브러리 파이썬 상에서 만든 수많은 오브젝트들은 파이썬 실행이 끝나면 사라집니다. 때로는 이런 오브젝트들을 저장해서, 매번 새로 만들 필요가 없으면 좋겠다는 생각이 들 수 있습니다. 이럴 때를 위해서 pickle이라는 라이브러리가 만들어졌습니다.

pickle에서 알아야 할 함수는 딱 두개입니다.

- `pickle.dump(var, dumpto)` : var에 저장된 파이썬 오브젝트를 dumpto 파일에 저장합니다. 이 때 dumpto는 경로가 아니라 파일입니다. 즉, `open(...)`으로 열린 파일이여야 하고, `wb/wb+`모드로 열어야만 합니다. pickle은 언제나 오브젝트를 바이트 형태로 만들어서 저장하기 때문입니다.
- `pickle.load(loadfrom)` : loadfrom 파일을 읽어서 이를 파이썬 오브젝트로 만듭니다. pickle로 저장된 파일만 가능하며, 열 때 `rb`로 열어야만 합니다.

### 4.3.2 re 라이브러리

re는 파이썬에서 정규표현식을 다루는 라이브러리입니다. 정규표현식 자체에 대해서는 대부분 익숙하지겠지만, 그래도 한번 더 짚고 넘어가보도록 하겠습니다. 파이썬 re 라이브러리는 사실상 정규표현식 그대로이기 때문에 정규표현식 자체를 이해하는 것이 더 중요하기 때문입니다.

정규표현식은 텍스트를 나타내기 위한 일종의 메타언어입니다. 정의가 조금 복잡한데, 다음의 규칙들만 기억하면 대부분의 경우 문제 없이 쓸 수 있습니다. 대개의 경우 많이 쓰이는 것들은 외우게 되지만, 외우지 않더라도 cheat sheet들을 참고하면 됩니다.

정규표현식 식 자체는 글자와, 그 글자들의 조합으로 이루어집니다. 따라서 정규표현식을 배운다는 것은 각 글자들을 나타내는 일종의 메타글자와, 그들의 조합법을 배우는 것입니다. 조합법은 quantifier를 통해서 정의됩니다.

- 메타글자

- `. :` 엔터를 제외한 모든 글자
  - `a :` 글자 a. 여기서 a는 알파벳 a만을 뜻하는 것이 아니고, 일반적인 모든 문자를 뜻합니다. '가'나 'Z' 같은 문자들을 다 포함합니다.
  - `ab :` 문열 ab. 위 글자와 같이 일반 문자들이 붙어 있으면 그 문자열을 의미합니다.
  - `a|b :` a 혹은 b.
- `a-x :` 알파벳 순으로 a부터 x까지의 모든 글자 중 하나를 의미합니다.

- 조합법 : 여기서

-

### 4.3.3 BeautifulSoup/requests 라이브러리

실습으로 갈음합니다.

### 4.3.4 multiprocessing 라이브러리

이 라이브러리의 경우, 전산적 지식이 상당히 필요하지만 알아두면 대단히 유용합니다. 특히, 좋은 컴퓨터를 보유하고 있을수록 컴퓨터의 성능을 100% 뽑아 쓸 수 있기 때문에 유용합니다. 본 단락에서는 아주 간단한 병렬처리의 설명과 이용법만 최대한 간단하게 예제에서 보고 넘어가도록 하겠습니다.

#### 4.3.5 xlrd/xlwt 라이브러리

엑셀 xlsx/xls 파일을 만들고 읽는 라이브러리입니다. 따로 소개하지는 않습니다만 있다는 것을 알고 있는 것이 좋습니다. 사실상 문과에서 파이썬을 배워서 할 수 있는 가장 좋은 것 중 하나가 엑셀 다루기이기 때문에...

#### 4.3.6 예제들

- 어떤 폴더 src를, 주어진 위치 dest에 복사하되
  - src 폴더 안의 폴더 구조만 복사

**Implementation 31:** Opening a file (*example\_fileio.py*)

---

```
1 import os
2 from shutil import copyfile
3
4 def create_dir(path):
5     if not os.path.exists(path):
6         os.mkdir(path)
7
8 def copy_directory(src, dest):
9     """Copy directory in src to dest directory.
10    """
11    create_dir(dest)
12
13    for elem in os.listdir(src):
14
15        src_path = os.path.join(src, elem)
16        dest_path = os.path.join(dest, elem)
17
18        if os.path.isdir(src_path):
19            copy_directory(os.path.join(src, elem),
20                           os.path.join(dest, elem),
21                           folder_only = folder_only)
22
23        elif not folder_only:
24            copyfile(os.path.join(src, elem),
25                     os.path.join(dest, elem),)
26
27 copy_directory('docs', 'newdoc_folder')
```

---

- src 폴더 안의 모든 폴더와 파일을 복사
- src 폴더 안의 파일 중 .py 파일만 같은 위치에 복사
- src 폴더 안의 파일 중 확장자가 없는 파일을 다 .py 파일로 만들어 복사
- src 폴더 안의 파일들의 경로와 이름을 적은 txt 파일을 생성
- 네이버 뉴스 크롤링
  - daterange 함수를 이용해서 뉴스를 얻고 싶은 기간의 날짜들을 얻음
  - 각 날짜별로 얻은 기사들을 저장할 폴더를 만듬

- 각 날짜별로 긁어야 할 기사의 링크의 목록을 얻는 함수를 작성
- 얻은 링크들에서 기사를 긁어옴
- (optional, open problem) 얻은 기사들을 정제함
- (optional, open problem) 언론사/기자/... 등등의 기준으로 저장하는 폴더를 만들고 저장
- (optional, open problem) 얻은 기사들을 정제하여 class로 만들어서 저장
- 얻은 결과를 폴더에 저장함