

# 파이썬



printed by SuPY



## 머릿글

프로그램이라는 용어의 정의는 일의 처리 방법과 순서를 가리키는 것이다. 컴퓨터에서 프로그램 작성에 사용되는 도구가 프로그램 언어이다. 프로그램 언어는 컴퓨터가 발명된 이후로 지금까지 여러 가지가 만들어지고 사라졌다. 원래 컴퓨터는 그들만의 언어-기계어가 있는데 당연히 사람은 기계가 아니므로 그들의 언어에는 익숙하지 못하다. 따라서 사람의 말에 가까운 언어가 필요했는데 사람의 언어가 가까울수록 기계의 입장에서는 효율적이지 못하였다.

대표적인 프로그램 언어 C언어는 그런 입장에서 가장 적절한 언어이다. 따라서 지금까지 만들어진 프로그래밍 언어 중에서 가장 사랑받고 활용된 언어이다. 그렇지만 C언어를 제대로 활용하기 위해서는 C언어에 익숙 해줘야 한다. 사실 소수의 전문가(?)를 제외하고는 C언어를 익숙하게 사용하여 활용하는 사람은 드물다. 언어로 공부할 뿐 일상 생활에서 이용하지 못하고 있다. 물론 소수의 사람을 제외하고서는 말이다.

프로그램이 일상 생활에서 이용할 요구가 제시되었을 때 어느 한 의인(義人:귀도 반 로섬)이 나타나 우리에게 던져준 프로그램 언어가 파이썬이다.

파이썬은 여러 가지 용도로 활용되고 있다. 과학적인 수치 연산과 데이터베이스, 데이터 분석과 사물인터넷, 인공지능(AI), 머신 런닝 등 수없이 많다. 차라리 활용되지 못하는 분야를 찾는 것이 쉽다. 모바일이나 시스템 분야 등 몇몇 분야가 있다. 그것은 파이썬이 가지고 있는 약점이 있기 때문이다. 파이썬의 약점은 속도가 느리다는 것이다. 그러나 우리가 가지고 있는 개인 컴퓨터의 속도가 빨라지고 우리가 생활에 활용되는 것이 다른 언어에 비해 느리다고 해도 사실 몇분 정도의 차이도 아니다. 단지 몇 밀리 초 차이일 뿐이다. 인간 입장에서는 별 차이가 없다.

파이썬을 옹호하는 집단에서는 이것을 개선하려고 노력하고 있다. 그래서 그 성과가 서서히 나타나고 있다. 이는 파이썬은 오픈 소스(open source)로서 누구나 프로그램 개선에 참여할 수 있기 때문이다.

교육계에서는 프로그램 교육에서 C언어에서 파이썬으로 급격히 전환되고 있다. 이는 배우기 쉬울 뿐만 아니라 간단 명료함, 라즈베리와 같은 여러 플랫폼에 적용이 가능하기 때문이다. 파이썬은 대표적인 인터프리터 언어이고 그 중에서도 가장 쉬운 언어라는 명성을 갖고 있다. 이와 동시에 데이터 사이언스 분야에서는 가장 강력한 언어이기도 하다. 그래서 미국에서는 대학생들이 학과를 불문하고 파이썬으로 코딩을 배우고 있고, 유명한 애플사의 컴퓨터에는 모두 이 파이썬이 기본적으로 설치되어 판매된다.

이 책에는 파이썬을 우리 생활에 쉽게 활용하기 위한다는 목표를 가지고 나아갈 것이다. 가능한 한 중고등학생, 대학생, 일반인까지 생활에 활용할 수 있는 결과를 얻기 위해 노력할 것이다. 그러나 처음 시작하는 우리는 유치원생일 뿐이다.

프로그램 예제는 반드시 입력해 실행하여 보기 바란다. RPA 와 같이 실생활에 유용한 프로그램을 소개하려고 노력하였다.

코로나가 우리 일상을 휩쓸고 있다. 코로나를 극복하고 활동적인 삶이 다시 찾아오기를 기원한다.

---

## 차 례

1. 파이썬 개요.....	1
2. 기본 문법.....	8
3. 조건문.....	20
4. 반복문.....	25
5. 리스트.....	35
6. 튜플과 딕셔너리 셋.....	48
7. 함수.....	65
8. 클래스와 예외 처리.....	76
9. 기본 모듈과 패키지.....	81
10. 파일 처리.....	92
11. 크롤링.....	111
12. RPA.....	122
13. folium.....	137

---

# CHAP 1. 파이썬 개요

```
31
32 self.file = None
33 self.fingerprints = set()
34 self.logdupes = True
35 self.debug = debug
36 self.logger = logging.getLogger(__name__)
37 if path:
38     self.file = open(os.path.join(path, "requests.log"),
39                     "a")
40     self.file.seek(0)
41     self.fingerprints.update(os.listdir(path) + [os.path.basename(self.file.name)])
42
43 @classmethod
44 def from_settings(cls, settings):
45     debug = settings.getboolean("DEBUG", False)
46     return cls(job_dir(settings), debug)
47
48 def request_seen(self, request):
49     fp = self.request_fingerprint(request)
50     if fp in self.fingerprints:
51         return True
52     self.fingerprints.add(fp)
53     if self.file:
54         self.file.write(fp + os.linesep)
55
56 def request_fingerprint(self, request):
57     return request_fingerprint(request)
```

## 1. 파이썬

우리에게 파이썬을 가져다 준 의인(義人) 귀도 반 로섬(Guido van Rossum)은 오픈 소스로 누구나 사용할 수 있고 개발할 수 있는 언어 파이썬(Python)을 개발하였다.

파이썬은 "가장 아름다운 하나의 답이 존재한다" 를 기본 철학으로 하고 있다. PEP 20 에서 제시된 파이썬 기본 철학(The Zen of Python)에 자세히 나열되어 있다.

1. 아름다운 것이 추한 것보다 낫다. (Beautiful is better than ugly.)
2. 명시적인 것이 암시적인 것보다 낫다. (Explicit is better than implicit.)
3. 간결한 것이 복잡한 것보다 낫다. (Simple is better than complex.)
4. 복잡한 것이 복잡한 것보다 낫다. (Complex is better than complicated.)
5. 수평적인 것이 내포된 것보다 낫다. (Flat is better than nested.)
6. 여유로운 것이 밀집한 것보다 낫다. (Sparse is better than dense.)
7. 가독성은 중요하다. (Readability counts.)
8. 특별한 경우들은 규칙을 어길 정도로 특별하지 않다. (Special cases aren't special enough to break the rules.)
9. 허나 실용성은 순수성을 이긴다. (Although practicality beats purity.)
10. 오류는 절대로 조용히 지나가지 않는다. (Errors should never pass silently.)
11. 명시적으로 오류를 감추려는 의도가 아니라면. (Unless explicitly silenced.)
12. 모호함을 대할때, 이를 추측하려는 유혹을 거부하라. (In the face of ambiguity, refuse the temptation to guess.)
13. 명확한, 그리고 가급적이면 유일한 하나의 방법은 항상 존재한다. (There should be one--and preferably only one --obvious way to do it.)
14. 비록 그 방법이 처음에는 명확해 보이지 않을지라도[10]. (Although that way may not be obvious at first unless you're Dutch.)
15. 지금 행동에 옮기는 것이 아예 안하는 것보다는 낫다. (Now is better than never.)
16. 비록 아예 안하는 것이 지금 \*당장\* 하는 것보다 나을때도 많지만. (Although never is often better than \*right\* now.)
17. 구현 결과를 설명하기 쉽지 않다면, 그것은 나쁜 아이디어이다. (If the implementation is hard to explain, it's a bad idea.)
18. 구현 결과를 설명하기 쉽다면, 그것은 좋은 아이디어일지도 모른다. (If the implementation is easy to explain, it may be a good idea.)
19. 네임스페이스를 사용하는 것은 완전 좋은 생각이다! (Namespaces are one honking great idea -- let's do more of those!)

파이썬의 높은 생산성은 그 무엇과도 비교할 수 없는 파이썬만의 특징이다. 전 세계의 모든 프로그래밍 언어 중에서 파이썬 정도의 낮은 난이도를 가지면서, 범용성을 갖추고, 파이썬 수준의 프로그램 개발 속도를 따라잡을 수 있는 언어가 없다.

## 2. 파이썬의 설치와 입력

### 2.1 파이썬 설치

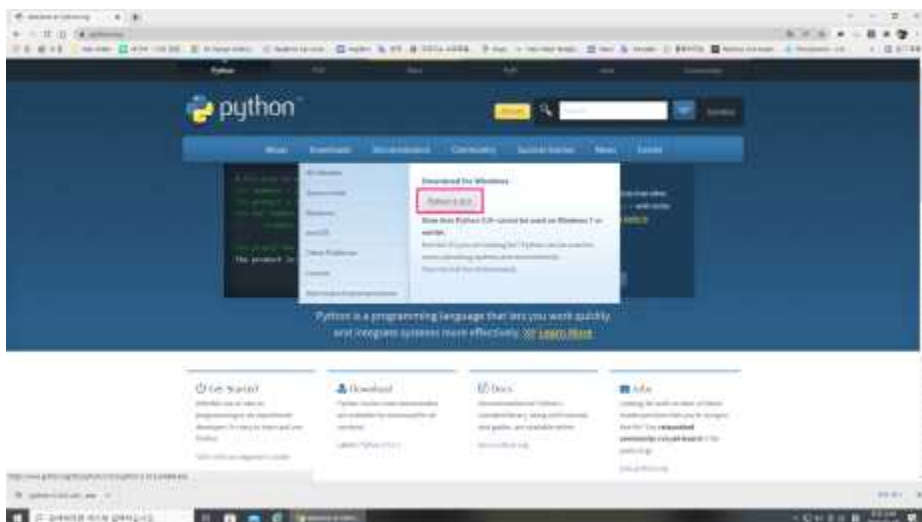
인터프리터 언어이다. (한 줄씩 번역하며 바로 실행하는 방식) 공짜이다.

[www.python.org](http://www.python.org) 에서 최신버전을 무료로 다운 가능하다. 전세계의 프로그래머들이 참여하여 지속적으로 기능을 개선하고 발전시키고 있다.(open source 원본프로그램 코드 즉 소프트웨어의 설계도를 공개하고 있음)

[www.python.org](http://www.python.org)를 접속한다.



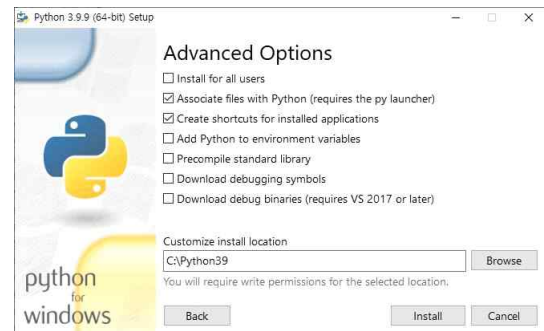
Download Python 3.9.9 다운 후 실행




Add Python 3.9 to Path check  
Customize installation



Advanced Options  
c:\Python39  
Install  
close



현재 이 글을 쓰는 시점에서 파이썬의 버전은 3.10.2이다. 최신 버전을 다운로드 받아도 되나 버그가 없는 안정화 버전 3.9.9를 다운로드 받아 실행하자. 본 교재의 예제 프로그램도 가능한한 3.9.9에 맞추어 작성되었으나 버전별로 실행이 안 될 수도 있으니 Jupyter Notebook으로 실행해보거나 인터넷을 검색하여 보자.

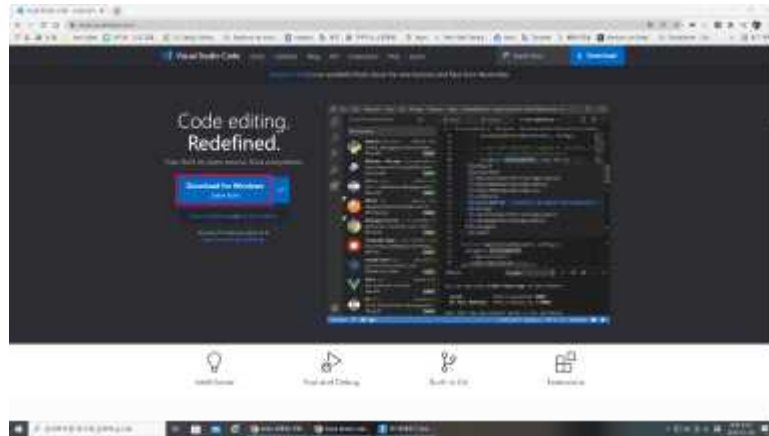
윈도우에서  + R 키를 눌러 [실행] 창을 연 후 [열기] 입력 상자에 'cmd'를 입력하고 [확인]을 클릭해 명령 프롬프트를 실행한다.

명령 프롬프트에서 'python -version' 또는 'python -V'를 입력하여 설치한 파이썬의 버전을 확인한다.



## 2.2 Visual studio code 설치

파이썬에서 사용하는 IDLE(Integrated Development and Learning Environment)는 여러가지가 있다. 그중에서 가장 많이 사용하는 Visual Studio code를 설치하여 보자. 우선 [code.visualstudio.com](https://code.visualstudio.com)을 접속하여 보자



Download for windows를 클릭하여 다운로드한 후 실행하여 보자.

계약에 동의함  
다음  
다음

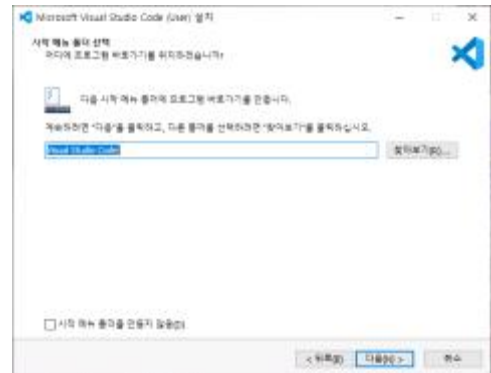


바탕 화면 바로 가기 만들기 Check  
다음 설치 마침

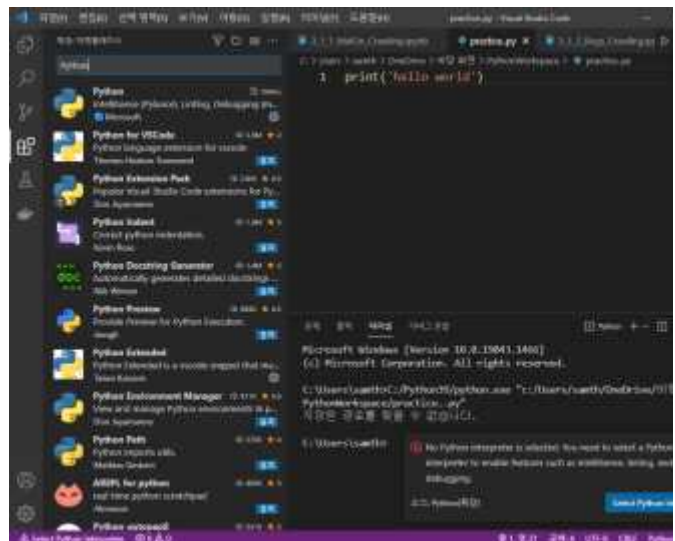


바탕화면에 PythonWorkspace 라는 폴더를 생성하고  
Visual studio code 실행한 후 Open Folder 메뉴에서 PythonWorkspace 선택하고  
New file 아이콘 Click 한 후 helloworld.py를 작성한다.

바탕 화면 바로 가기 만들기 Check  
다음 설치 마침



왼쪽 확장 extension(📦) 아이콘 클릭  
Python 선택  
install 설치



마찬가지로 Korea Language 선택 install(설치)한다.

Windows system 제어판 시스템(Windows 시작 버튼 설정(기어모양))  
오른쪽 고급시스템 설정 - 고급 - 환경변수 - 시스템변수 새로만들기  
path 디렉터리 찾아보기  
PythonWorkspace 지정하기 명칭 path

## 2.3 파이썬 입력

파이썬에서 공백은 중요한 역할을 한다. 사실, 한 행의 앞에 붙어있는 공백이 정말 중요하다. 이것을 `_들여쓰기_`라 부른다. 한 논리적 명령행의 앞에 붙어있는 공백 (빈칸 혹은 탭)은 논리적 명령행의 들여쓰기 단계를 의미하며, 이것은 한 명령의 범위를 구분하는 데 사용된다.

잘 적용된 경우

```
score = [100, 90, 70, 80]
for n in score:
    print(n)
```

잘 못 적용된 경우

```
score = [100, 90, 70, 80]
for n in score:
    print(n)

i = 10
    x = [ 10, 29, 40]
print(i)
```

문장 앞의 `#` 기호는 주석으로 설명을 달 때 사용한다. `#` 뒤의 내용은 프로그램 실행에 관여하지 않는다. `"""` 와 같이 따옴표를 사용하면 문장 전체가 주석이 되거나 '나' 처럼 인용 기호로 쓰인다.

```
# 파이썬 프로그램 설치
```

```
"""파이썬은 여러 가지 용도로 활용되고 있다. 과학적인 수치 연산과 데이터베이스, 데이터 분석과 사물인터넷, 인공지능(AI), 머신러닝 등 수없이 많다. """
```

**표준 입출력**

```
import sys
print("Korea", "Japan")
print("Korea", "Japan", sep=",")
print("Korea", "Japan", sep=",", end="?")
print("Korea", "Japan", sep=",", end="\n")
print("Korea" + "Japan")
print("Korea", "Japan", file=sys.stdout)
print("Korea", "Japan", file=sys.stderr)

# 시험 성적
scores = {"국어":70, "영어":50, "수학":100}
for subject, score in scores.items():
    print(subject, score)
scores = {"국어":70, "영어":50, "수학":100}
for subject, score in scores.items():
    print(subject.ljust(8), str(score).rjust(4), sep=":")

# 0001, 0002, 0003 .....
for num in range(1, 21):
    print("대기번호 : "+str(num).zfill(4))

answer = input("아무 값이나 입력하세요 : ") # 항상 문자열 형태로 입력
print(type(answer))
print("입력하신 값은 " + answer + "입니다.")
```

**=== 다양한 출력 포맷**

```
print("{0: >10}".format(500))
print("{0: >+10}".format(500))
print("{0: >+10}".format(-500))
print("{0: _<10}".format(500))
print("{0:,}".format(100000000))
print("{0:+,}".format(100000000))
print("{0:+,}".format(-100000000))
print("{0: ^>+30,}".format(100000000000))
print("{0: ^<+30,}".format(100000000000))
print("{0:f}".format(5/3))
print("{0:.2f}".format(5/3))
```

## CHAP 2. 기본 문법



## 1. 상수

상수(常數:constant)는 용어 그대로 변하지 않는 수이다. 우리가 상수를 이야기 할 때는 앞으로 나올 변수와 같이 수학적인 개념과 약간의 차이가 있다. 컴퓨터의 기억 장치에서 기억되는 데이터를 말한다. 쉽게 생각하면 프로그램 상에서 10이라는 데이터가 있다. 그 데이터는 프로그래머가 의도적으로 변경하지 않는 한 그 값을 유지하고 있다. 데이터는 정수나 실수 같이 숫자가 될 수도 있고 문자도 될 수 있다.

상수는 기억장치 메모리에 기억되어 있는데 인간이 알기 위해서는 print라는 명령을 사용한다, 다른 언어를 시작할 때 처음 사용하는 예는 파이썬에서는 단 한 줄로 표시된다. 이는 곧 상수 문자열을 출력하는 명령이다.

```
print( "Hello, World!" )
```

C언어에서는 다음과 같다.

```
#include<stdio.h>
main(void) {
    print( "Hello, World" )
}
```

### 1.1 정수형 상수

소숫점이 없는 수로서 파이썬은 정수형을 미리 선언하지 않는다.

ex) 123, 10, 0, -27 ....

실수나 숫자 형태의 문자를 정수로 바꾸고자 할 때는 int() 함수를 사용한다.

ex) int(4.5), int('57')

### 1.2 실수형 상수

소숫점이 있거나 지수인 수

ex) 3.14, 0.12, 0., -27.45

```
print(15)
print(9 + 6)
```

constant\_1.py

정수나 숫자 형태의 문자를 정수로 바꾸고자 할 때는 float() 함수를 사용한다.  
ex) float(4), float('57.4')

수식 연산자 : 수식 연산을 하기 위한 연산자이다.

수식 연산자	설 명
+	더하기
-	빼기
*	곱하기
/	나누기
//	나눗셈의 몫
%	나눗셈의 나머지
**	거듭제곱

표 1. 수식연산자

상수를 이용하여 계산기처럼 출력할 때 파이썬 Shell 모드에서 사용하거나 IDLE나 Visual Studio Code와 같은 편집기 모드에서 사용할 수 있다. Shell 모드에서는 print를 사용하지 않아도 되나 편집기 모드나 Shell 모드에서 상수를 출력할 때 print 명령을 사용할 수 있으니 가능한 한 편집 모드와 Shell 모드 동시에 사용하는 print를 사용한다.

```
print(10 + 4)
print(10 - 4)
print(10 * 4)
print(10 / 4)
print(10 // 4)
print(10 % 4)
print(10 ** 4)
```

constant\_2.py

파일명으로 3\_1\_constant2.py로 저장하고 반드시 실행해 보기로 한다. 앞으로 실습이 필요할 때는 파일명을 변경하여 저장하여 실행하거나 전의 프로그램을 지우고 실행한다. 위 프로그램의 실행 결과중에서 print(10/4)의 결과는 무엇인가. 파이썬은 정수와 정수의 계산에서 C언어와 같이 정수가 출력되지 않는다.

지수는 다음과 같이 표현한다.

```
print(10e4)
print(type(10e4))
print(3.1415e2)
print(12.2 - 12.1)
```

constant\_3.py

10e4는  $10 \times 10^4$ 이다. 지수는 실수이다. 상수의 형을 알고자 할 때는 type 함수를 사용한다. 마지막 명령에서 결과가 0.1이 아닌 오류가 나타나는 데 컴퓨터의 한계에 따른 정밀도의 오류이다. 무시하여도 되는 근삿값이지만 정확하게 하고자 한다면 Decimal 라이브러리를 사용한다.

```
from decimal import Decimal
print(Decimal('12.2') - Decimal('12.1'))
```

constant\_4.py

### 1.3 복소수

수학에서는 복소수를 많이 사용하는데 허수 표현에서 i 대신 j를 사용한다. 복소수를 이용하여 벡터값을 구할 수 있다.

```
a = (2 + 1j)
print(a.real) # 실수
print(a.imag) # 허수
print(abs(a)) # 크기
```

constant\_5.py

복소수 a에서 허수부 j는 반드시 1을 사용하여 1j로 표현한다. 1을 사용하지 않으면 변수가 되기 때문이다. #는 주석으로 프로그램의 실행에 참여하지 않아 설명하고자 할 때 사용한다.

### 1.4 문자열

문자열은 문자를 순차적으로 나열한 것을 말한다. 작은따옴표(' '), 큰 따옴표(" ")를 사용하며 문자와 문자열을 구분하지 않는다.

```
words = "Happy"
letter = 'Birthday!!'
print(words + letter)
print("="*20)
```

constant\_6.py

문자열은 연산자 +와 \*를 사용한다. +는 두 문자열을 합하여 주며 \*는 숫자 만큼 반복을 하여 준다. print("="\*20)은 숫자만큼 =를 반복 출력하여 출력을 구분할 때 사용한다.



## 1.5 논리상수

논리 상수는 참(True)과 거짓이 있다. 논리 상수는 제어문에서 참과 거짓을 판별할 때 많이 사용된다.

특히 제어문에서 참이나 거짓의 어느 값을 취할 수 있는 상수이다. True 또는 False를 어느 논리 변수에 할당하면 그 논리 변수의 값은 각각 참 또는 거짓이 된다.

파이썬 연산자순위

```
1순위 : ( )
2순위 : *, /, %(나머지)
3순위 : +, -
4순위 : ==, !=, <, <=, >, >=
5순위 : not
6순위 : and
7순위 : or
```

출력시 분리자를 둘 수 있다.

```
a = 'Korea'
b = 'Japan'
c = 'China'
d = 'Rusia'
print(a, b, c, d, sep = ',')
```

constant\_7.py

이어서 출력하고자 하면 end = ''를 사용하면 된다.

```
a = 'Korea'
b = 'Japan'
print(a, end = '')
print(b)
```

constant\_8.py

## 1.6 문자열 관련 함수

### 1) join()

문자열을 특정 글자로 합쳐준다.

```
ex) str1='KBS'
    str2=', '
    station = str2.join(str1)
    print(station)
```

## 2) split()

특정 글자로 문자열을 나누어 리스트를 만든다.

```
ex) s = "KBS MBC SBS"
    print(s.split())
    s2 = "서울->대전->대구->부산"
    city = s2.split("->")
    print(city)
    for c in city:
        print(c, "찍고", end = ' ')
```

## 3) strip()

문자열 앞 뒤의 공백을 지운다. 파싱을 할 때 주로 사용한다.

```
ex) str1=' Welcome to Korea '
    greeting = str1.strip()
    print(greeting)
```

## 4) replace()

특정 글자를 다른 글자로 바꾸어준다.

```
ex) str1=' Welcome=to=Korea '
    greeting = str1.replace('=',' ')
    print(greeting)
```

## 5) in, not in

in : 문자열 안에 해당 문자열이 있는지 확인한다. 있으면 True, 없으면 False

not in : 문자열 안에 해당 문자열이 없는지 확인한다. 없으면 True, 있으면 False

```
ex) station = 'MBC' in 'KBS, MBC, SBS'
    print(station)
```

```
station = 'TBS' not in 'KBS, MBC, SBS'
print(station)
```

6) count(), index(), find()

count() : 해당 문자가 몇 번 쓰였는지 확인해 줌

index() : 해당 문자의 위치를 확인해 줌

find() : "찾을 문자" 혹은 "찾을 문자열"이 존재하는지 확인하고, 찾는 문자가 존재한다면 해당 위치의 index를 반환해주고 찾는 문자가 존재하지 않는다면 -1을 반환한다. 만약에 찾는 문자나 문자열이 여러 개 있다면 맨 처음 찾은 문자의 index를 반환한다.

```
ex) text = 'Welcome to Korea'
print(text.count('e'))
print(text.index('e'))
print(text.find('o'))
```

7) capitalize(), lower(), upper()

capitalize() : 첫 글자를 대문자로 만든다.

lower() : 대문자를 소문자로 만든다.

upper() : 소문자를 대문자로 만든다.

```
ex) text = 'kBs, mBc, sBs'
print(text.capitalize())
print(text.lower())
print(text.upper())
```

8) startswith(), endswith()

startswith() : 문자의 시작에 문자(열)이 있는지 확인한다. 있으면 True 없으면 False를 반환한다.

endswith() : 문자의 끝에 문자(열)이 있는지 확인한다. 있으면 True 없으면 False를 반환한다.

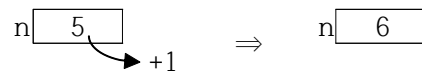
```
ex) text = 'happy.csv'
print(text.startswith('ha'))
print(text.endswith('.csv'))
```

## 2. 변수

### 2.1 변수

변수는 기억 장소의 이름이다. 이 장소는 임시로 혹은 영구적으로 어떤 값을 기억해두기 위해 사용된다. 변하는 수, 프로그램이 진행되면서 새로운 값으로 바꾸어 저장하거나 기억된 값을 꺼내 사용할 수 있다. 역시 수학적인 개념과 약간의 차이가 있다. 변수에 필수적으로 사용하는 '=' 또한 수학에서 말하는 같다는 개념으로 생각하면 안 된다. 나중에 기술하겠지만 '='는 대입연산자이다. 수학에서는  $n = n + 1$ 이 불가능하겠지만 변수를 사용하는 연산자에서는 가능하다. 따라서 우변에는 상수, 변수, 수식, 함수 등의 표현이 모두 가능하지만, 좌변에는 변수만 올 수 있다.

즉 변수  $n$ 에 1을 더해서 다시  $n$ 에 대입(기억)시키라는 의미이다.



```
n = 5
n = n + 1
print(n)
```

variable\_1.py

변수의 정의에서 기억 장소의 이름이라 했기에 이름을 짓는다는 규칙이 있다. 프로그래밍에서 시간 대부분은 각종 이름을 짓는 일에 소요된다. 변수, 상수, 함수, 객체 등등 프로그램에 나오는 알 수 없는 단어들은 모두 프로그래머가 필요에 의해 맘대로 만들어 낸 이름들이라고 생각하면 된다. 이 알 수 없는 단어들이 난무하게 되면 당연히 프로그램이 이해하기 어렵게 되고 작성자 자신도 자신이 짠 프로그램이 뭘 하라는 것인지 모르는 상태가 되고 만다. 이를 방지하기 위해 일정한 규칙과 의미 있는 단어를 사용해야만 프로그램의 가독성이 높아지고 이것이 좋은 프로그램의 첫 번째 덕목이다.

첫째, 예약어는 사용할 수 없다.

```
import keyword
print(keyword.kwlist)
```

variable\_2.py

출력 결과는 아래와 같은데, 우리가 변수명으로 사용할 수 없는 단어들이다. 또한 기본적으로 우리가 배워야 할 명령어기도 하다.

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
```

'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

또한 후에 기술하겠지만 모듈이나 라이브러리에서 사용하는 명령어도 사용할 수 없다. 그러나 모듈이나 라이브러리에서 사용하는 명령어, 함수 등은 전부 기억하고 암기할 필요는 없다. 필요하다면 참고 서적이나 인터넷에서 검색하면 그만이다. 중요하지 않은 것이 아니라 그만큼 파이썬에서 사용하는 모듈이나 라이브러리가 무수히 많기 때문이다.

둘째, 숫자는 첫 글자로 사용할 수 없다.

셋째, 공백을 사용할 수 없다. 따라서 공백 대신 `_`을 사용하는 것이 좋다.

ex) `news_file`, `open_list`

넷째, 공백을 사용할 수 없다.

마지막으로 가장 중요한 점은 파이썬에서는 명령문도 대소문자를 구분하여 사용하듯이 변수명도 당연히 구분한다. 즉, 명령 `print`를 `PRINT`로 쓸 수 없듯이 변수명 `apple`과 `Apple`, `APPLE`는 서로 다른 변수이다.

```
wc = 4
yc = 4
wc = wc + 1
yc += 1
print(wc, yc)
```

variable\_3.py

파이썬에서 변수는 형을 구별하지 않고 기억한다. 다른 언어는 정수형이고 실수형이고 문자형이고 구분하고 기억한다. 동적 타입 언어로 변수의 자료형을 지정하지 않고 선언하는 것만으로 값을 지정할 수 있다. 자료형은 우변의 값 기억되는 데이터의 형에 따라 결정된다. 변수만 보고서는 기억된 데이터의 형을 알 수 없다. 따라서 변수의 형을 알기 위해서는 `type()`이라는 함수를 사용한다.

```
s=100
k="korea"
print(type(s), type(k))
```

variable\_4.py

## 2.2 메소드

파이썬은 객체지향 프로그램이다. 객체는 프로그램에서 사용되는 숫자, 문자열, 함수 등을 말한다. 객체에 관해서는 나중에 자세히 설명하겠지만 어떤 내용(객체)의 성격을 바꾸는 방법으로 메소드를 사용한다. 예를 들면 소문자란 객체를 대문자로 대문자 객체를 소

문자로 바꾸고자 할 때 메소드를 사용한다. 점(.)과 괄호를 사용한다.

```
print("abc".upper())  
print("DEF".lower())
```

method\_1.py

위의 예에서 소문자 "abc"로 이루어진 객체는 대문자 "ABC"로 바뀌고 "DEF"는 소문자 "def"로 바뀐다. 메소드는 객체를 기술하고 그 뒤에 점을 찍은 후 메소드를 기술하고 괄호를 사용한다.

### 2.3 여러 변수 지정하기

한 번에 여러 변수를 지정할 수 있다. 아래와 같이 입력할 경우 두 변수 a 와 b 에는 모두 같은 값 10이 할당된다.

```
a = b = 10  
print(a, b)
```

point\_1.py

아래와 같이 입력할 경우에는 두 변수 a, b에 각각 10과 20이 할당된다.

```
a , b = 10, 20  
print(a, b)
```

point\_2.py

아래와 같은 방식으로 두 변수의 값을 서로 맞바꿀 수도 있다.

```
a , b = 10, 20  
b, a = a, b  
print(a, b)
```

point\_3.py

### 3. 연산자

#### 3.1 연산 순서

다음에 주어진 수식을 암산으로 계산하여 보아라.

$$4 + 2 * 3 - 6 - 2^3$$

계산값은 프로그램을 입력하고 실행하면 정답을 알려줄 것이다. 단순한 계산식이지만 계산값이 틀린 사람도 있을 것이다. 이는 암산을 못하는 것이 아니라 계산 순서를 헷갈려 하기 때문이다. 무조건 앞에서부터 계산하여 오류가 발생할 수 있다. 모든 계산에는 순서가 있다. 더군다나 명령을 주어 일을 시키는 프로그램에서는 계산 순서가 정해지지 않으면 혼란이 올 수 있다.

약속된 계산 순서는 다음과 같다.

괄호  $\Rightarrow$  함수  $\Rightarrow$  거듭제곱  $\Rightarrow$  곱셈과 나눗셈  $\Rightarrow$  덧셈과 뺄셈 (동일 순서는 왼쪽부터)

이 순서를 기억하고 프로그램을 실행하여 계산해보라.

```
comp = 4 + 2 * 3 - 6 - 2**3
print(comp)
```

operater\_1.py

```
print("화씨 100도는 섭씨 몇도인지 아십니까 ")
fahrenheit = float(input("화씨 온도를 입력하세요: "))
celsius = 5 / 9 * (fahrenheit - 32)
print("섭씨 온도", celsius)
```

operater\_2.py

#### 3.2 대입 연산자

대입 연산자는 오른쪽에 있는 값이나 식을 왼쪽에 있는 변수에 기억시키는 연산자이다.

대입 연산자	설 명
=	오른쪽 값을 왼쪽에 기억(대입)
+=	변수에 어떤 수를 더한 값을 다시 변수에 기억(대입)
-=	변수에 어떤 수를 뺀 값을 다시 변수에 기억(대입)
*=	변수에 어떤 수를 곱한 값을 다시 변수에 기억(대입)
/=	변수에 어떤 수를 나눈 값을 다시 변수에 기억(대입)
//=	변수에 어떤 수를 나눈 몫을 다시 변수에 기억(대입)
%=	변수에 어떤 수를 나눈 나머지를 다시 변수에 기억(대입)
**=	변수에 어떤 수를 거듭제곱한 값을 다시 변수에 기억(대입)

표 2. 대입연산자

대입 연산자를 사용한 수식 `wc += 1`은 `wc = wc + 1`과 같은 식이다.

```
a=9
b=5
print( a + b )    # 더하기
print( a - b )    # 빼기
print( a * b )    # 곱하기
print( a / b )    # 나누기
print( a // b )   # 나눈 몫
print( a % b )    # 나눈 나머지
```

operator\_3.py

`s += t` 과 같이 우변에 변수를 사용하는 경우는 `s = s + t` 와 같은 식으로 프로그램에서 누적(累積)을 구할 때 사용한다. 쉽게 말하면 여러 수의 합을 구하거나, 곱을 구하는 명령으로 쓰인다.

```
변수1 = 변수1 + 상수1  # ex) n = n + 1
변수1 = 변수1 + 변수2  # ex) s = s + n
```

### 3.3 비교연산자

수식 연산자는 상수 항목 표1에서 이미 배웠다. 비교 연산자는 다음 표와 같다.

비교연산자	의 미
==	같다.
!=	같지 않다
>	크다
<	작다
>=	크거나 같다
<=	작거나 같다

표 3. 비교연산자



비교 연산자 ==는 같다는 것을 의미한다. 대입 연산자 =와는 다른 연산자이다. 수학식에서는 ==가 없기 때문에 잘 구별해서 사용해야 한다.

### 3.4 논리연산자

여러 논리 상수를 사용하여 연산을 할 수 있다. 수학의 부울 연산과 동일하다.

논리연산자	의 미
not	논리 부정
and	논리 곱
or	논리 합

표 4. 논리연산자

```
logic1 = True
logic2 = False
logic = logic1 and logic2
print(logic)
```

operator\_4.py

### 3.5 비트연산자

2진수로 이루어진 데이터끼리 연산을 할 수 있다.

비트연산자	의 미
&	비트 단위 AND 연산
	비트 단위 OR 연산
^	비트 단위 XOR
~	비트 단위 NOT
<<	좌측 shift
>>	우측 shift

표 5. 비트연산자

```
bita = 4 # 0100
bitb = 7 # 0111
bit1 = bita & bitb
print(bit1)
bitc = 0b0100
bitd = 0b0111
bit2 = bitc & bitd
print(bit2)
```

operator\_5.py

여기서 정수 4와 7은 이진수로 각각 0100, 0111 이므로 비트 단위로 결과는 이진수 0100 이다.

```

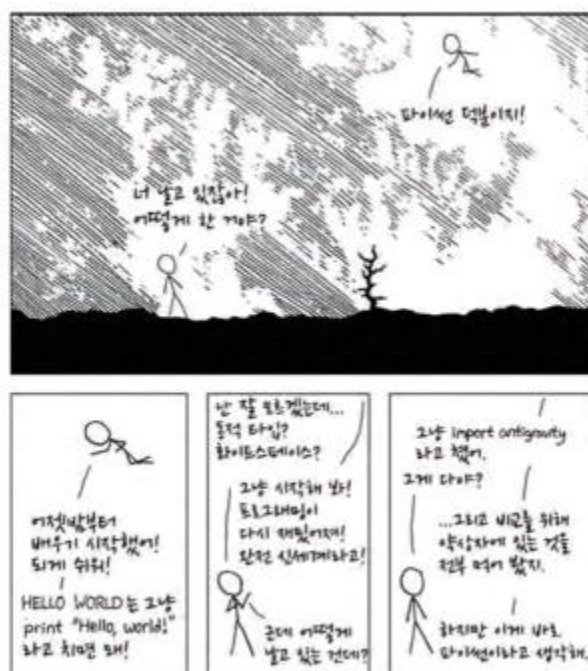
      0100
    & 0111
    -----
      0100
  
```

```

loga = True
logb = False
logc = loga & logb
logd = loga and logb
print(bin(loga)) # True 값을 2진수로
print(bin(logb)) # false 값을 2진수로
print(logc)
print(logd)
  
```

operator\_6.py

비트연산자 ~, &, | 는 논리연산자 not, and, or와 똑같이 사용하는데 논리값 True와 False를 2진수로 바꾸면 1과 0이기 때문이다. 위 코드에서 bin( )함수는 정수를 2진수로 바꾸는 함수이다.



## CHAP 3. 조건문



## 1. 제어문

프로그램의 정의를 다시 한번 상기해보자. 프로그램은 어떤 문제를 해결하기 위해 컴퓨터에게 주어지는 처리 방법과 순서를 기술한 일련의 명령문의 집합체이다. 보통은 우리가 기술한 대로 위에서 아래로 실행된다. 하지만 우리는 일의 순서를 바꾸어 다른 명령을 실행할 필요도 있다. 7시에 일어나서 준비하고 등교하는 것이 명령 프로그램이라면 우리는 7시라는 시간을 확인할 필요가 있다. 눈을 뜨니 6시라면 자리에 누워 다시 잠자리에 들 것이다. 컴퓨터도 마찬가지이다. 우리가 설정한 조건이 아니라면 계속 실행하다가 조건에 맞으면 실행을 바꾸면 된다. 제어문에는 알게 모르게 조건이 따른다. 조건 없이 프로그램 스스로 실행을 변경하지는 않기 때문이다.

제어문에는 if, for, while 등이 있다.

**if 조건식 :**

└─문장1

└─문장2

└─....

**else :**

└─문장I

└─문장II

└─....

제어문을 사용할 때는 Tab 키를 사용하여 들여쓰기에 주의해야 한다.

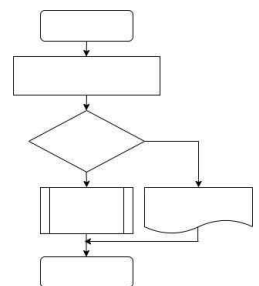
flowchart(순서도, 흐름도, 플로우차트, 플로차트)

프로그램이나 작업의 진행 흐름을 나타낸 도표를 말한다.

상자와 화살표로 명령의 순서를 보여주는 도표로 알고리즘, 프로세스를 표현하기에 적합함

프로그램 논리의 흐름, 업무 처리 과정을 표현하는 데 사용한다.

각 단계를 여러 가지 모양의 상자로 표현하고, 그 사이를 화살표로 연결하여 흐름을 나타낸다.



## 2. if문

if문은 조건을 주면 판단을 해서 실행을 결정하는 명령문이다. if문에는 if, if~else, if~elif~else 문이 있다.

조건문 1이 참이면 문장1, 2..를 수행하고 제일 아래쪽으로 이동한다.

조건문 1이 거짓이면 조건문 2를 조사해서 참이면 문장a, b..를 수행하고 제일 아래쪽으로 이동한다.

조건문 2도 거짓이면 이후에 나오는 elif 조건문을 차례대로 검사하며 동일 방식으로 진행한다.

모두 거짓이면 문장A, B...를 수행하고 내려간다.

elif나 else문이 불필요하면 기술하지 않는다.

여러 개의 elif로 추가의 조건을 검사 할 수 있다. 각각 참인 경우가 발생하면 해당 블록만을 수행하고 if문 맨 아래로 이동한다. 어떤 경우든 한 가지 경우만 실행하게 되며 else가 없으면 실행할 문장이 없을 수도 있다.

들여쓰기가 각 문장의 소속을 표시하므로 정확히 해야 한다.

if문 일반형식

if 조건문1:

    수행할 문장1

    수행할 문장2

    ...

elif 조건문2:

    수행할 문장a

    수행할 문장b

    ...

....

else:

    수행할 문장A

    수행할 문장B

    ...

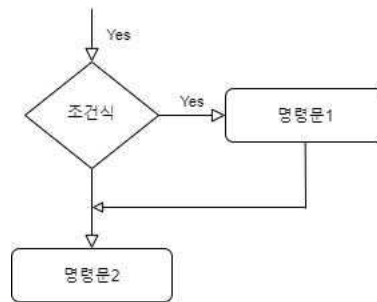
### 2.1 기본 if문

if 조건식:

    명령문1

    명령문2

조건식을 만족하면 명령문1을 실행하고 만족하지 않으면 if문 다음 명령문2를 실행하는 명령문이다.



두수의 차를 구하는 프로그램을 구해보자.

```

a, b = 5, 10 # a는 5 b는 10
if a < b:
    b, a = a, b
print('두 수의 차:', a-b)
  
```

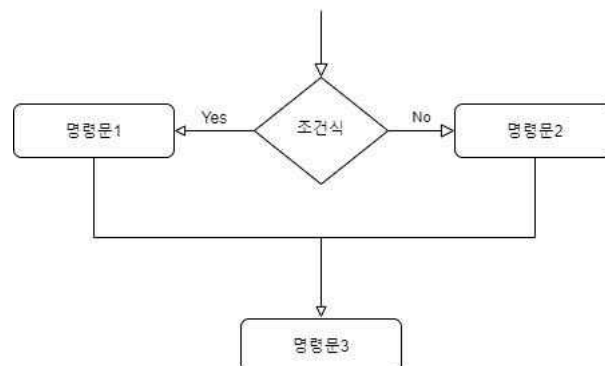
if\_1.py

a가 b보다 작으므로 a 값과 b 값을 바꾸고 두수를 빼서 값의 차를 구하는 프로그램이다.

## 2.2 if else문

```

if 조건식:
    명령문1
else:
    명령문2
명령문3
  
```



조건식을 만족하면 명령문1을 수행하고 만족하지 않으면 명령문2를 수행하는 제어문이다.

```
age = 20
if age >= 20:
    print('입장 가능')
else:
    print('입장 불가')
```

if\_2.py

나이가 20살 이상이면 '입장 가능'을 출력하고 20살보다 작으면 '입장 불가'를 출력하는 프로그램이다.

### 2.3 if elif else문

파이썬의 명령문에서는 case나 switch문이 없는데 if ~ elif ~ else문을 사용하여 구현할 수 있다.

```
if 조건식1:
    명령문1
elif 조건식2:
    명령문2
....
else:
    명령문n
```

```
score = int(input("점수: "))
if score >= 90:
    grade = '수'
elif score >= 80:
    grade = '우'
elif score >= 70:
    grade = '미'
elif score >= 60:
    grade = '양'
else:
    grade = '가'
print(score, "는", grade, "입니다")
```



if\_3.py

점수를 입력하면 수우미양가를 판별해주는 프로그램이다. input( )은 사용자로부터 입력을 받아 문자열로 기억시키는 함수이다. 따라서 문자열을 숫자로 바꿔줘야 하는데 int( )가 그 역할을 한다.

컴퓨터 화면의 해상도를 알려고 한다면 다음 명령을 입력하여 실행해 보기 바란다.

```
import pyautogui
width, height = pyautogui.size()
print(width, height)
```

#### 언어 사용 순위

	Jan 2020	Jan 2021	Change	Programming Language	Rating	Change
1	3		▲	 Python	13.52%	+1.86%
2	1		▼	 C	12.64%	-4.94%
3	2		▼	 Java	10.06%	-1.30%
4	6		▲	 C++	8.29%	+0.73%
5	5		▲	 C#	5.60%	+1.73%
6	8		▲	 Visual Basic	5.74%	+0.50%
7	7		▲	 JavaScript	2.00%	-0.11%
8	11		▲	 Assembly language	1.55%	+0.21%
9	12		▲	 SQL	1.60%	+0.19%
10	13		▲	 Swift	1.41%	-0.02%
11	9		▼	 PHP	1.40%	-0.60%
12	9		▼	 R	1.35%	-0.63%
13	14		▲	 Go	1.04%	-0.37%
14	19		▲	 Delphi/Object Pascal	0.99%	+0.30%



## CHAP 4. 반복문



## 1. for문

for문은 반복 실행 명령문이다. 사람은 단순한 일의 반복을 하면 지루해하고 실수가 발생하기 쉽다. 찰리 채플린의 모던타임즈란 영화에서 주인공은 하루 종일 나사만 조립하다가 착란을 일으킨다. 인간들에게 반복 작업에서 해방시켜 준 것이 컴퓨터이다. 컴퓨터는 반복 작업을 지루해하지도 않고 실수도 하지 않는다.

반복문은 데이터를 모아 놓은 리스트 같은 자료에서 하나하나씩 끌어내어 실행한다. 반면에 다른 언어는 초기값, 최종값, 증가값을 정하고 반복 실행한다. C언어는 아래와 같이 반복 실행문을 실행한다.

```
#include<stdio.h>
main(void) {
    int i;
    for(i=1;i<=10;i++)
    {
        print( "%d:" , i);
    }
```

하지만 파이썬은 아래와 같은 형식을 사용한다.

for 변수 range() 함수:  
명령문

for 변수 <리스트, 튜플, 딕셔너리>:  
명령문

for 변수 문자열:  
명령문

제어 변수는 반복 횟수를 관리하는 변수이고 뒤의 리스트에 들어있는 값을 앞에서부터 차례대로 넣어가면서 아래의 들여쓰기 된 문장들을 반복 수행하다가 리스트의 맨 끝값까지 다 넣고 수행했으면 반복 구간을 빠져나와 아래로 내려간다.

이전에 미리 반복 횟수만큼의 리스트를 만들어 놓고 for 문을 사용하는 것이 번거로우므로 리스트를 만드는 range 함수를 많이 사용한다.

range 함수는 인수를 한 개만 쓸 수도 있고 두 개를 쓸 수도 있고 세개를 쓸 수도 있다. 한 개만 쓰면 0부터 인수 바로 앞의 값까지의 정수를 리스트로 만들어 준다. 두 개

를 쓰면 시작 값과 끝값으로 사용한다. 끝값은 바로 앞의 값까지를 말하고 자신은 포함되지 않는다. 세 개를 쓰면 시작 값, 끝값, 간격으로 사용한다. 즉 시작 값에 간격값 만큼씩 증가된 값들이 만들어져서 끝값을 넘지 않는 최댓값까지 리스트가 만들어진다.

먼저 1부터 10까지 나열해보자.

```
for cnt in range(11):  
    print(cnt)
```

for\_1.py

마지막 값은 11을 빼고 10까지 나열된다. 리스트를 사용하여 1에서 15까지 출력할 수 있는데 range를 사용하면 간단히 처리할 수 있다.

```
a=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]  
for cnt in a:  
    print(cnt)  
for cnt in range(1,16):  
    print(cnt)
```

for\_2.py

리스트를 사용하여 점수를 출력하여 보자.

```
scores = [70, 80, 85, 90, 100, 70, 80, 90]  
for score in scores:  
    print(score)  
print('————')  
for cnt in range(len(scores)):  
    scores[cnt] = scores[cnt]+2  
    print(scores[cnt])  
print(scores)
```

for\_3.py

딕셔너리도 for문으로 출력해보자.

```
dict = {'kor':50, 'eng':76, 'mat':90, 'sci':80, 'com':55}
for cnt in dict:
    print(cnt)
```

for\_4.py

딕셔너리에서는 키 만 출력된다.

키뿐만 아니라 값도 출력되기 위해서는 enumerate( ) 함수를 사용하면 된다. 나중에 다시 배우겠지만 리스트와 인덱스와 값을, 딕셔너리에서 키와 값을 출력할 때 enumerate( ) 함수를 사용한다. for 문을 enumerate와 같이 사용하면, 각 항목의 값과 인덱스 값을 동시에 알 수 있다. cnt 부분은 value 항목의 인덱스 값으로 출력된다.

```
dict = {'kor':50, 'eng':76, 'mat':90, 'sci':80, 'com':55}
for cnt, value in enumerate(dict):
    print(cnt, value, dict[value])
```

for\_5.py

```
dict = {'kor':50, 'eng':76, 'mat':90, 'sci':80, 'com':55}
for dt in dict.values():
    print(dt, end=" ")
```

for\_6.py

items라는 메소드를 사용할 수도 있다.

```
dict = {'kor':50, 'eng':76, 'mat':90, 'sci':80, 'com':55}
for subject, score in dict.items():
    print(subject, score)
```

for\_7.py

문자열도 사용할 수 있다.

```
nation = "Republic of Korea"
for cnt in nation:
    print(cnt)
```

for\_8.py

출력 결과는 위에서 아래로 출력이 되는 데 이어서 출력하려면 end=' '를 조건으로 달아야 한다. 간격을 띄어쓰기를 위해 ' ' 안에 공백을 둔다.

```
scores = [60, 90, 75, 80, 100, 60, 90, 80]
for score in scores:
    print(score, end = ' ')
```

for\_9.py

공백뿐만 아니라 기호와 이스케이프 코드를 같이 사용할 수 있다.

이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 문자 조합이다. 주로 출력물을 보기 좋게 정렬하는 용도로 사용한다. 몇 가지 이스케이프 코드를 정리하면 다음과 같다.

이스케이프 코드	설 명
\n	문자열에서 줄을 바꿀 때 사용
\t	문자열에서 탭 간격을 줄 때 사용
\\	\\(백스페이스)를 출력하고자 할 때 사용
\'	'(작은따옴표)를 출력하고자 할 때 사용
\"	"(큰 따옴표)를 출력하고자 할 때 사용
\r	캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동)
\f	폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동)

표 6. 이스케이프 코드

```
scores = [60, 90, 75, 80, 100, 60, 90, 80]
for score in scores:
    print(score, end=',')
for score in scores:
    print(score, end='Wt')
```

for\_10.py

리스트에서 '.py'로 끝나는 원소를 골라서 리스트를 구성하려고 한다면 다음 프로그램과 같이 리스트 안에서 for 문을 사용하면 된다.

```
files = ['pro.py', 'sample.txt', 'play.py', 'music.py', 'help.txt']
file_list = [file for file in files if file.endswith(".py")]
print(file_list)
```

for\_11.py

급여를 입력하면 화폐 또는 동전의 개수를 줄여서 지급하려고 한다. 각 화폐의 개수를 출력하는 프로그램을 작성하여 보자.

```
money = [50000, 10000, 5000, 1000, 500, 100, 50, 10, 5, 1]
count = []
pay = int(input("급여 : "))
for cnt, value in enumerate(money):
    count.append(pay // money[cnt])
    pay = pay % money[cnt]
    print(value, "원", "원 : ", count[cnt])
```

for\_12.py

```
for dan in range(1,10):
    for i in range(2,10):
        print('%d x %d = %2d'%(dan, i, dan*i))
    print()
```

for\_13.py

구구단을 1단부터 9단까지 모두 출력한다.

중첩된 for 루프로 먼저 dan에 1이 들어간 다음 안쪽의 for문을 만나 i가 1부터 9까지 차례대로 들어가며 안쪽의 프린트 문이 반복되어 1단을 출력한다. i가 9까지 모두 사용되면 안쪽의 for 루프가 끝나고 다시 바깥쪽 for문 까지 올라온다. dan에 2가 들어간 다음 안쪽 for문이 다시 반복되며 2단을 찍는다. 이런 식으로 3,4...9단까지 모두 찍게 된다.

마지막의 빈 프린트문의 들여 쓰기 위치는 문장의 소속을 안쪽 for 루프와 바깥쪽 for루프 사이에 위치함을 의미한다. 즉 안쪽 for루프를 다 수행하고 바깥쪽 for루프로 올라가기 전에 빈 프린트문을 수행하여 단과 단 사이에 한 줄을 띄우게 된다.

구구단 표를 만들자면 다음과 같다.

```
for i in range(1,10):
    for dan in range(2,10):
        print('%d x %d = %2d'%(dan, i, dan*i), end=" ")
    print()
```

for\_14.py

## 2. while문

while문은 조건이 만족하는 한 반복 수행된다. 따라서 반복 수행문에서 빠져나올 상황을 만들어 주어야 한다. 그렇지 않으면 무한 반복한다. 물론 Ctrl키와 c키를 동시에 눌러 빠져 나올 수 있기는 하다. 조건문은 참, 거짓을 판단할 수 있는 명제 형태로 표현되며 참일 경우만 뒤에 들여쓰기 되어 있는 문장들을 수행하게 된다. 들여쓰기는 문장의 소속관계를 표현하는 중요한 사항으로 while 문장은 자신보다 안쪽으로 들여 쓴 문장들을 반복 구간으로 판단한다. 한 칸이라도 어긋나 있으면 안 되므로 정확하게 맞추어야 하며 탭키를 사용하여 맞추는 것이 좋다.

while 은 다양한 경우 섬세하게 반복 조건을 표현하여 사용할 수 있는 강력한 명령이지만 조건의 변화에 영향을 주는 모든 문장을 추적해보아야 얼마나 반복되는지를 알 수 있어서 프로그램을 이해하기 어렵고 작성하기도 까다로운 단점이 있다. 실제 프로그램에서는 정해진 일정 횟수만큼 무조건 반복해야 하는 경우가 많으므로 이를 위한 간편한 명령이 필요해졌다. for 문이 그 답이다.

while 조건:

명령문

1부터 100까지의 정수합을 구해보자.

```
sum = 0
x = 1
while x <= 100:
    sum = sum + x
    x = x + 1
print(sum)
```

while\_1.py

다음은 동화를 가지고 while문을 활용하여 보자. 어느날 부잣집에 일꾼이 와서 30일 동안 일을 할 테니 첫날은 한냥, 다음날은 전날의 2배를 달라고 했다. 과연 일꾼은 30일 후에 얼마를 받을 수 있는가?

아마 옛날에 컴퓨터가 있으면 부잣집 주인을 이런 계약을 하지 않았을 것이다. 여기서 파이썬의 장점은 매우 큰 수도 지수로 변경하지 않는다는 것이다. while 문을 변경하여 어느 정도의 크기까지 출력이 되는지 확인하여 보자. 실제로 정수가 어느 정도까지 출력 되는지 가늠이 되지 않는다. 다른 언어 특히, C 같은 언어는 어느 정도의 범위를 벗어나면 지수값으로 변한다. 그런데 문제는, 돈은 지수값으로 값을 표현할 수는 없다. 물론 지수값으로 표현될 정도로 큰돈을 다루는가는 차후 문제이겠지만 아무튼 이는 파이썬의 큰 장점이 아닐 수 없다.

```
total = 1
days = 1
print(days, " : ", total)
while days < 30: # 30을 변경해보자
    days = days + 1
    total = total + total * 2
    print(days, " : ", total)
```

while\_2.py

while 문을 메뉴를 구성하는 데도 사용된다.

```
fruit = ['바나나', '사과', '복숭아']
print("1. 바나나")
print("2. 사과")
print("3. 복숭아")
print("0. 종료")
while True:
    key = int(input("당신의 선택은? "))
    if key != 0:
        print(fruit[key-1], "을 선택하셨습니다")
    else:
        break
```

while\_3.py

while 문에 조건으로 True를 사용했기 때문에 무한 반복되는데 선택 문자를 입력받아 0이 아니면 과일을 선택하고 0이면 break 문으로 반복문에서 빠져나온다. break은 조건문이나 반복문을 수행하다가 해당 영역을 벗어날 때 사용하고 continue는 반복문의 처음으로 제어를 이동한다.

1에서 100까지 홀수 값만 출력하여 보자.

```
i = 0
while i < 100:
    i += 1
    if i % 2 == 0:
        continue
    print(i)
```

while\_4.py

짝수 값을 출력하기 위해서는 4번째 줄 대입연산자 ==를 !=로 바꾸기만 하면 될 것이



다.

```
i=1
while i <= 10:
    print(i, '번 출력되었습니다')
    i += 1
print('프로그램이 끝났습니다')
```

while\_5.py

맨처음 i에 1을 넣고 시작(초기값)한다.  $i \leq 10$  은 i에 있는 값이 10보다 작거나 같은가? 를 묻음. 참이면 아래의 들여쓰기 된 구간을 실행하고 다시 올라와서 반복, 거짓이면 들여쓰기가 끝난 부분까지 그냥 통과한다.

$i += 1$  은  $i = i + 1$  을 축약해서 표현한 식이다. 현재 i의 값을 꺼내서 1을 더한 다음 다시 i에 저장하라는 뜻. 즉 1만큼 원래의 값을 증가시키게 된다.

`print(i, '번 출력되었습니다')` print문은 콤마(,)로 출력할 값들을 분리하면서 필요한 만큼 여러 개의 인수를 사용할 수 있다. i에 있는 값을 출력하고 뒤에 ' '로 묶인 문자열을 출력하게 된다.

`print('프로그램이 끝났습니다')` 이 문장은 위쪽의 while과 같은 칸에서 시작되었으므로 while의 반복 구간 밖에 있음을 알 수 있다. 즉 while의 조건문이 거짓이 되면 여기까지 그냥 내려오게 된다. (이 문장을 실행할 차례가 된다) 프로그램은 특별한 제어 명령이 아닌 한 위에서 아래로 차례대로 수행하게 되어 있다.

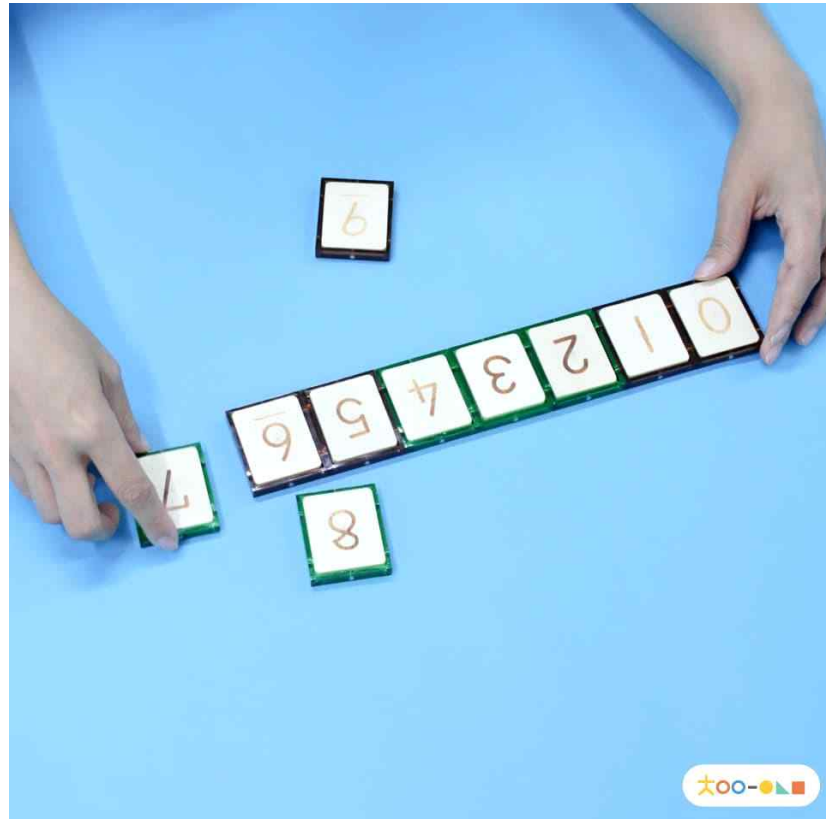
메뉴는 딕셔너리로 각 항목을 출력하고 key 값으로 선택하여 함수를 실행하면 된다. 파이썬으로 여러 프로그램을 import로 불러와 실행하는 경우가 아니라면 위 프로그램으로 메뉴를 구성하여 보자. while True로 키 입력이 이루어지기 전까지 메뉴를 출력하다 키 입력을 받으면 함수를 실행하는 프로그램 구성이다.

```
msg = {
    '1': '데이터 추가',
    '2': '데이터 수정',
    '3': '데이터 삭제',
    '4': '데이터 출력',
    '0': '종료'
}

def menu1():
    print('함수 1입니다')
def menu2():
    print('함수 2입니다')
def menu3():
    print('함수 3입니다')
def menu4():
    print('함수 4입니다')
while True:
    for key in msg:
        print(key+'.', msg[key])
    ans=input("선택할 메뉴는 ? : ")
    print(msg[ans])
    if ans == '1':
        menu1()
    elif ans == '2':
        menu2()
    elif ans == '3':
        menu3()
    elif ans == '4':
        menu4()
    else:
        print('프로그램을 종료합니다')
        break
```

while\_6.py

# CHAP 5. 리스트



## 1. 리스트(list)

### 1.1 리스트 정의와 활용

여러 개의 데이터를 하나의 객체로 묶어서 표현하는 데이터 유형을 말함(=배열, array). 파이썬에서 리스트는 여타 변수와 같이 형을 구별하지 않는다.

다른 언어와 달리 리스트의 값들은 동일한 형으로만 이루어지지 않는다. 따라서 파이썬에서는 배열이라는 말을 쓰지 않는다. 한마디로 데이터들을 나열(list)하는 것이다. 데이터들은 [ ]로 묶고 값들은 ‘,’로 구분한다.

[값1, 값2, 값3, 값4.....]

ex) [35, "Korea", True], [apple, banana, peach, grape]

리스트는 여러 요소들을 갖는 집합(컬렉션)으로 새로운 요소를 추가하거나 갱신, 삭제하는 일이 가능하다. 파이썬의 리스트는 동적배열(Dynamic Array)로서 자유롭게 확장할 수 있는 구조를 갖는다. 리스트는 그 안의 요소(element)들은 그 값을 자유롭게 변경할 수 있는 데이터 타입이다. 각각의 요소를 호출하려면 리스트 이름 뒤에 각괄호 안에 번호를 적어 호출한다. 주의할 것은 첨자는 항상 0부터 시작하므로 리스트안의 개체 숫자보다 하나 작은 값까지만 호출이 가능하다.

파이썬에서는 리스트를 활용하면 효과적으로 프로그래밍을 할 수 있다. 만약 내가 첫 날 시험에서 받은 성적을 합계를 내고 싶다면 파이썬에서는 단 두줄이면 가능하다.

```
first_list = [70, 80, 85, 90, 100]
print(sum(first_list))
```

list\_1.py

### 2.1 리스트 값 생성과 삽입, 추가

배열에서도 각각의 위치의 값을 구할 수 있듯이 각 위치를 지정할 수 있다. 위 프로그램에서의 첫 번째 값의 위치를 나타내는 수의 값은 0이다. 모든 리스트의 위치는 0부터 시작된다.

first\_list

0	1	2	3	4
70	80	85	90	100

리스트에서 첫 번째 3번째 값을 호출하여 출력하고자 한다면 다음과 같이 하면 된다.

```
first_list = [70, 80, 85, 90, 100]
print(first_list[0], first_list[2])
```

list\_2.py

다음날 두 과목의 시험을 더 치렀다고 하자. 리스트는 다른 리스트와 결합할 수 있다.

```
first_list = [70, 80, 85, 90, 100]
second_list = [70, 100]
tot_list = first_list + second_list
print(tot_list)
```

list\_3.py

물론 append 함수를 사용하여 첫날 본 성적 점수 리스트에 추가할 수 있다.

```
first_list = [70, 80, 85, 90, 100]
first_list.append(70)
first_list.append(100)
print(first_list)
```

list\_4.py

여기서 .append는 메소드로서 특정한 값을 추가하는 기능을 수행하는 함수이다. 물론 한꺼번에 리스트의 값을 추가할 수도 있다.

```
first_list = [70, 80, 85, 90, 100]
first_list.extend([70, 100])
print(first_list)
```

list\_5.py

첫 번째 날 네 번째 과목 시험이 문제에 이상이 있어 점수가 상향되었다면 다음과 같이 수정할 수 있다.

```
first_list = [70, 80, 85, 90, 100]
first_list[3] = 95
print(first_list)
```

list\_6.py

두 번째 날 시험 성적에서 첫 과목 성적 다음에 새로운 과목 성적을 삽입하여 보자.

```
second_list = [70, 90]
second_list.insert(1, 80)
print(second_list)
```

list\_7.py

### 1.3 리스트 값 삭제

전체 성적에서 네 번째 값을 리스트에서 삭제하여 보자.

```
total_list = [70, 80, 85, 90, 100, 70, 80, 90]
del total_list[3]
print(total_list)
```

list\_8.py

위 프로그램에서 `del total_list[3]` 명령을 `total_list.pop(3)`으로 수정하여 실행하여 보자. 아마 같은 결과를 얻을 것이다.

전체 성적에서 90점인 성적을 삭제하여 보자.

```
total_list = [70, 80, 85, 90, 100, 70, 80, 90]
total_list.remove(90)
print(total_list)
```

list\_9.py

90점인 성적 중에서 첫 번째 성적만 삭제되는 것을 볼 수 있다.

전체 리스트에서 마지막 성적을 삭제해보자. 간단한 프로그램에서는 원하는 값의 위치 (여기서는 8번째)를 알 수 있지만 모른다고 가정한다면 `pop()`을 사용하는 것이 최선의 방법일 것이다.

```
total_list = [70, 80, 85, 90, 100, 70, 80, 90]
total_list.pop()
print(total_list)
```

list\_10.py

전체 성적 리스트에서 내가 본 과목의 수와 가장 낮은 점수와 높은 성적순으로 성적을 나열해보자.

```
total_list = [70, 80, 85, 90, 100, 70, 80, 90]
print(len(total_list))
total_list.sort()
print(total_list)
total_list.reverse()
print(total_list)
```

list\_11.py

#### 1.4 리스트의 나열값 생성

일련된 값을 가지는 리스트를 만들어 보자. 예를 들면 아래와 같은 리스트를 간단히 만들어 보자.

```
[ 0, 1, 2, 3, 4, 5]
[num for num in range(6)]
```

물론 위와 같은 간단한 리스트를 만든다고 하면 직접 나열된 값을 입력하면 된다. 그러나 숫자가 많아지면 간단한 방법으로 리스트를 만들 수 있다.

```
range(6)
```

실제로 파이썬에서 확인해보자.

```
kbs = list(range(6))
print(kbs)
```

list\_12.py

여기서 range 괄호 안의 값 인수는 kbs 리스트의 최댓값보다 1이 큰 것을 알 수 있다. 리스트의 마지막 값은 range 인수 값에서 -1을 한 것이다. list() 함수는 나열된 숫자를 리스트로 만들어 주는 함수이다.

다시 한번 0에서 100까지 나열한 리스트를 만들어 보자.

```
mbc = list(range(101))
```

우리가 이전에 사용한 sum 함수를 사용하여 0에서 100(실제로는 1에서 100)까지의 합을 구할 수 있다.

```
mbc = list(range(101))
print(sum(mbc))
```

list\_13.py

range의 인수 값을 변경하여 증가하는 값을 가진 리스트를 만들 수 있다.

range(시작값, 종료값, 증가값)

range 함수를 사용하여 1에서 100까지 홀수 합과 짝수 합을 구하여 보자. 종료값의 인수는 리스트에서 가지는 값보다 1인 큰 것을 상기하여야 한다.

```
mbc1 = list(range(1, 101, 2))
print(sum(mbc1))
mbc2 = list(range(2, 101, 2))
print(sum(mbc2))
```

list\_14.py

## 1.5 문자열과 리스트

사실 문자열을 list() 함수를 사용하여 하나의 리스트로 바꿀 수 있다. 문자열을 리스트로 바꿔서 여러 가지로 활용할 수 있다. 우선 문자열을 리스트로 만들어 보자.

```
nation = list("Republic of Korea")
print(nation)
```

list\_15.py

만약 주민등록 번호를 입력받았을 경우 남자인지 여자인지 확인하는 방법을 알아보자. 다음과 같은 임의의 주민등록 번호가 있다고 하자.

```
030224-3xxxxxx
980730-2xxxxxx
```

주민등록의 8번째 문자가 1이나 3일 경우 남자, 2이나 4일 경우 여자이다. 앞으로 배우겠지만 if 문을 사용한다면 간단하게 성별을 출력할 수 있다.



```
juno = "030224-3xxxxxx"
jumin = list(juno) # 문자를 리스트로 만들기
s_no = jumin[7]    # 리스트 성별 문자
if s_no == '1' or s_no == '3':
    print("남자")
else:
    print("여자")
```

list\_16.py

리스트는 파이썬에서 매우 중요한 역할을 한다. 잘 익혀두면 파이썬으로 할 수 있는 것이 무궁무진할 것이다.

```
test = input("문자열 입력 : ")
lt = []
lt = test.split()
print(lt)
word_list = [lt.count(p) for p in lt]
print(word_list)
print(dict(zip(lt, word_list)))
```

list\_17.py

문자열을 입력할 때 ‘사과 바나나 파인애플 사과 바나나’ 등과 같이 단어 사이에 공백을 두고 입력하면 간단하게 리스트가 만들어지는 것을 볼 수 있다. split() 메소드가 문자열을 분리하여 리스트로 만들어 준다.

```
from random import randint
N = 10
a = []
total = 0
for i in range(N):
    a.append(randint(0, 9))
    print(a[i], end=' ')
    total += a[i]
average = total / N
print(average)
for i in a:
    if i > average:
        print(i, end=' ')
```

list\_18.py

리스트값이 양수이면 1, 음수이면 -1 로 바꾸어 (0은 그대로) 새로운 리스트에 값을 넣는 프로그램이다. 원래의 리스트 값은 보존해야 한다.

```
array = [10, -15, 3, 8, 0, 9, -6]
co_arr = []
for i in range(len(array)):
    if array[i] > 0:
        co_arr.append(1)
    elif array[i] < 0:
        co_arr.append(-1)
    else:
        co_arr.append(0)
print(array)
print(co_arr)
```

list\_19.py

리스트를 사용하여 2차원 배열처럼 값을 불러올 수도 있다.

```
lol = [ [1, 2, 3], [4, 5], [6, 7, 8, 9]]
print(lol[0])
print(lol[2][1])
for sub in lol:
    for item in sub:
        print(item, end=' ')
    print()
```

list\_20.py

1	2	3	
4	5		
6	7	8	9

list lol

```
list1 = [1, 2, 3, 4, 5]
list2 = [10, 11]
list1.extend(list2)
list3=list1+list2
print(list1)
print(list3)
```

list\_21.py

## 1.6 lambda 사용

한줄로 리스트를 계산식을 사용하여 구성하는 방법으로 lambda 함수와 filter 함수가 사용된다.

### lambda

lambda 변수, 표현식

```
target = [1,2,3,4,5,6,7,8,9,10]
a_list = list(filter(lambda x: x % 2 == 0, target))
b_list = [x * 2 for x in range(1,6)]
print(a_list)
print(b_list)
```

list\_22.py

```
# 1. 일반 함수 버전
def plus_two(x):
    return x + 2
result1 = list(map(plus_two, [1, 2, 3, 4, 5]))
print(result1)
# 2. 람다 함수 버전
result2 = list(map((lambda x: x + 2), [1, 2, 3, 4, 5]))
print(result2)
```

list\_23.py

```
r = list(map(lambda a,b: a+b, [1,2,3], [10,20,30]))
print(r) # [11, 22, 33]
```

list\_24.py

lambda는 런타임에 생성해서 사용할 수 있는 익명 함수이다. 쓰고 버리는 일시적인 함수다. 함수가 생성된 곳에서만 필요하다. 즉, 간단한 기능을 일반적인 함수와 같이 정의해두고 쓰는 것이 아니고 필요한 곳에서 즉시 사용하고 버릴 수 있다.

```
# 기존 함수
def minus_one(a):
    return a-1
```

# 람다 함수

```
lambda a: a-1
```

# 람다 함수 호출 방법 1. 함수 자체를 호출

```
print((lambda a : a-1)(10))
```

# 람다 함수 호출 방법 2. 변수에 담아서 호출

```
minus_one_2 = lambda a: a-1
```

```
print(minus_one_2(100))
```

# 람다 함수에서 if문 사용

# 기존 함수

```
def is_positive_number(a):
```

```
    if a>0:
```

```
        return True
```

```
    else:
```

```
        return False
```

# 람다 함수

```
lambda a: True if a>0 else False
```

# 람다 함수 호출(1)

```
print((lambda a : True if a>0 else False)(-2))
```

# 람다 함수 호출(2)

```
is_positive_number = lambda a : True if a>0 else False
```

```
print(is_positive_number(2))
```

### **map()**

리스트로부터 원소를 하나씩 꺼내서 함수를 적용한 다음, 그 결과를 새로운 리스트에 기억시킨다.

```
num1=[1,2,3]
num2=[4,5,6]
result=list(map(lambda x, y:x+y, num1, num2))
print(result)
```

list\_25.py

```
def half(s):
    return s / 2
score = [ 45, 89, 72, 53, 94 ]
for s in map(half, score):
    print(s, end = ' ', )
```

list\_26.py

### filter()

리스트에 들어있는 원소들을 함수에 적용시켜서 결과가 참인 값을 리스트로 변환할 수 있도록 함

```
def is_neg(x):
    return x < 0
my_list = [10, -1, 20, -3, -2.5, 30]
print(list(filter(is_neg, my_list)))
```

list\_27.py

```
# 1. 일반 함수 버전
def is_even(x):
    return x % 2 == 0
result1 = list(filter(is_even, range(10)))
print(result1)
# 2. 람다 함수 버전
result2 = list(filter((lambda x: x % 2 == 0), range(10)))
print(result2)
```

list\_28.py

# map 함수= 기존 리스트를 수정하여 새로운 리스트를 만들 때

# - 사용 방법 map(함수, 순서가 있는 자료형)  
print(list(map(int,['3','4','5','6'])))

items =[' 로지텍 마우스 ', ' 애플키보드 ']

# 1) for 사용  
for i in range(len(items)):  
 items[i] = items[i].strip()  
print(items)

```
# 2) map 사용
def strip_all(x):
    return x.strip()
items = list(map(strip_all, items))
print(items)

# 3) lambda 사용
items = list(map(lambda x : x.strip(), items))
print(items)

# 2. filter 함수
# 기존 리스트에서 조건에 만족하는 요소만 뽑고 싶을 때

# filter(함수, 순서가 있는 자료형)
def func(x):
    return x<0
print(list(filter(func, [-3, -2, 0, 5, 7])))

# 리스트의 길이가 3이하인 문자들만 필터링
animals = ['cat', 'tiger', 'dog', 'bird', 'monkey']

# 1) for 사용
result = []
for i in animals:
    if len(i) <=3:
        result.append(i)
print(result)

# 2) filter 사용
def word_check(x):
    return len(x) <=3

result = list(filter(word_check, animals))
print(result)
```

### # 3) lambda 사용

```
result = list(filter(lambda x: len(x) <= 3, animals))
print(result)
```

### zip()

zip() 함수는 2개 이상의 시퀀스 형 자료를 한데 묶어서 튜플을 원소로 구성하는 시퀀스로 만들어 주는 함수이다.

```
numbers = [1, 2, 3]
letters = ["A", "B", "C"]
total=list(zip(numbers, letters))
print(total)
for pair in zip(numbers, letters):
    print(pair)
```

list\_29.py

```
for number, upper, lower in zip("12345", "ABCDE", "abcde"):
    print(number, upper, lower)
```

list\_30.py

```
list_zip = list(zip("abc", "def"))
print(list_zip)
Number = [1,2,3,4]
Name = ['hong', 'park', 'yoon', 'kim']
dic = {}
for i in range(len(Number)) :
    dic[Number[i]] = Name[i]
print(dic)
```

list\_31.py

리스트 안에 다른 리스트를 원소로 넣을 수 있다.

```
a_list = [2, 5, 10]
b_list = a_list
c_list = a_list[:]    # member-by-member 복사 수행
a_list[2] = 20
print('a_list : ',a_list, 'b_list : ', b_list,'c_list : ', c_list)
```

list\_32.py

```
mylist = [[10, 20, 30], [40, 50, 60], [70, 80, 90, 100]]
def flatten(the_list):
    return [one_item
            for one_sublist in the_list
            for one_item in one_sublist]
print(flatten(mylist))
```

list\_33.py

### 1.7 슬라이싱

리스트에서 일정 범위의 값을 선택할 때는 슬라이싱(slicing)을 사용한다. 슬라이싱은 시작 인덱스: 끝 인덱스 형식을 사용하는데, 시작 인덱스의 값은 포함하지만, 끝 인덱스의 값은 포함하지 않는다.

```
x = ['a', 'b', 'c', 'd', 'e']
print(x[1])
print(x[-1])
print(x[0:3])
print(x[3:])
print(x[:3])
```

list\_34.py

시작 인덱스:끝 인덱스:간격 의 형식으로 간격을 표시할 수도 있다.

```
x = [5, 10, 15, 20, 25]
print(x[:2])
print(x[1:4:2])
print(x[::-1])
```

list\_35.py

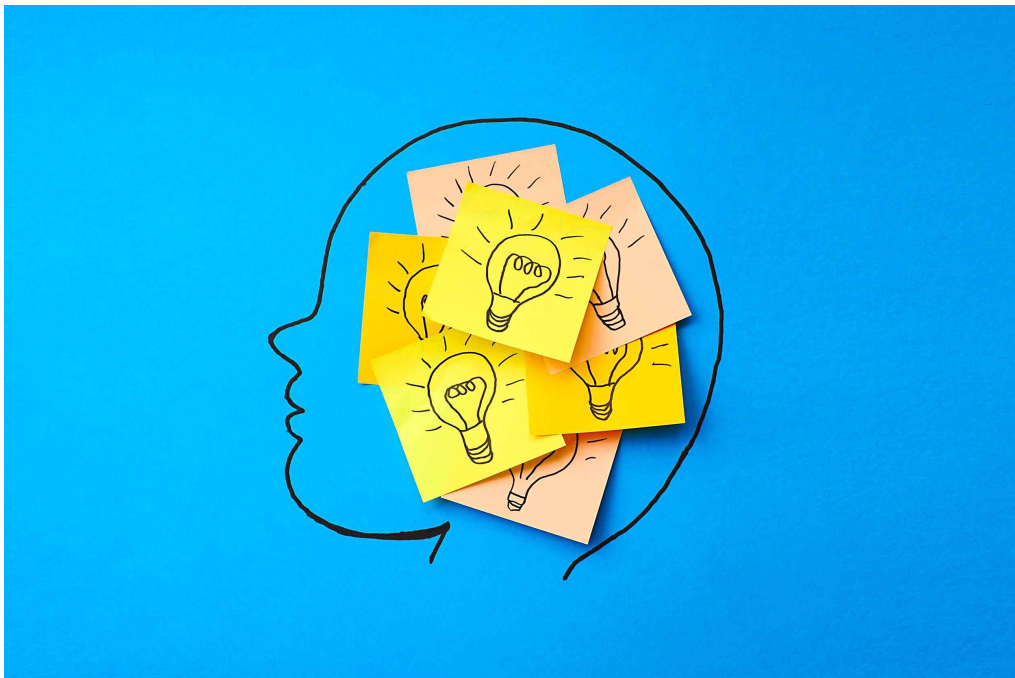
문자열은 리스트로 슬라이싱을 할 수 있다.

```
file = "20171224-104830.jpg"
print("촬영 날짜 : " + file[4:6] + "월" + file[6:8] + "일")
print("촬영 시간 : " + file[9:11] + "시" + file[11:13] + "분")
print("확장자 : " + file[-3:])
print("역순 : " + file[::-1])
```

list\_36.py



# CHAP 6. 튜플, 딕셔너리, 셋



## 1. 튜플(tuple)

튜플은 [ ]대신 ( ) 기호를 사용하는 것 외에 성격이 리스트와 동일하다. 단지 튜플의 원소는 추가, 변경, 삭제를 할 수 없다. 따라서 프로그램 내에서 변경해서는 안 되는 중요한 데이터를 나열할 때 초기값 처럼 사용한다.

튜플은 위치마다 정해진 의미가 있는 경우에 사용된다. 만약 평면상에서 좌표를 두 개의 수로 나타낸다고 해보자. 이 경우 0번 값은 가로 좌표, 1번 값은 세로 좌표와 같은 식으로 정해진 의미가 있다. 여기에 값을 추가하거나 뺄 필요가 없으므로 이 경우엔 튜플을 쓴다.

만약 좌표를 리스트로 나타낼 수도 있다. 하지만 프로그램이 복잡해지면 리스트는 실수로 여기에 값을 추가하거나 뺄 수도 있다. 그러면 이 좌표에 가로와 세로 좌표가 모두 들어있을지 아니면 가로 좌표만 있을지 또는 가로와 세로 외에 다른 무엇이 들어 있을지 장담할 수가 없게 된다.

튜플은 형태가 고정되어 있으므로 좀 더 안심하고 쓸 수 있다. 또한 추가/삭제/수정 등에 대비할 필요가 없으므로 리스트보다 약간 더 효율적인 내부 구조로 되어 있다.

튜플의 형태는 다음과 같다.

```
t1 = ()  
t2 = (5, )  
t3 = (5, 6, 4, 5)
```

여기서 값이 하나만 있는 튜플은 (5, ) 콤마 하나 꼭 붙여야 한다는 것이다.

```
first_tuple = (70, 80, 85, 90, 100)  
first_tuple[3] = 95  
print(first_tuple)
```

tuple\_1.py

위의 프로그램을 실행하면 에러가 발생하고 값을 수정할 수 없게 된다. 그래도 튜플의 값을 수정하고자 한다면 튜플을 리스트로 바꾸고 값을 수정한 후 다시 튜플로 변경하면 된다.

```
first_tuple = (70, 80, 85, 90, 100)  
imsi = list(first_tuple)  
imsi[3] = 95  
first_tuple = tuple(imsi)  
print(first_tuple)
```

tuple\_2.py

튜플은 괄호를 생략하여도 하나로 묶어서 기억된다.

```
scores = 90, 85, 70
kor, eng, mat = scores
print(eng)
```

tuple\_3.py

두 기억 장소의 값을 바꿀(swap) 때 사용할 수 있다.

```
a = 45
b = 80
b, a = a, b
print(a)
print(b)
```

tuple\_4.py

만약에 튜플을 사용하지 않는다면, 알고리즘에 의해 다음과 같이 프로그램을 해야 한다.

```
a = 45
b = 80
c = a
a = b
b = c
print(a)
print(b)
```

tuple\_5.py

튜플을 사용하는 것이 훨씬 직관적이고 이해하기 쉽다.

튜플의 값이 하나일 때는 값 뒤에 콤마(,)을 붙여서 사용한다. 콤마를 사용하지 않으면 하나의 값인 상수가 되기 때문이다.

```
score1 = (76)
print(type(score1))
score2 = (84,)
print(type(score2))
```

tuple\_6.py

```
tuplex = 4, 8, 3, 4, 7, 3
print(tuplex)
print(tuplex.count(4))
print(tuplex[1])
```

tuple\_7.py

튜플도 리스트와 같이 값을 기억시킬 수는 없지만, 원소값을 출력시킬 수 있다. 마지막 라인의 출력값은 8이 된다.

```
import math
def circle(r):
    lenc = 2 * math.pi * r
    arec = math.pi * r ** 2
    return lenc, arec
a, b = circle(2)
print(a, b) # 튜플
```

tuple\_8.py

## 2. 딕셔너리

딕셔너리는 특정 키에 특정 값을 연결하여 사용한다. 키와 값은 ':'으로 구분하며 전체는 { }로 묶는다. 사전(dictionary)은 키(key)와 그에 대응하는 값을 짝 지워놓은 자료구조다. 사전이 단어에 해당하는 뜻을 찾기 위한 책인 것을 생각해보면 이런 이름이 붙은 이유를 짐작할 수 있을 것이다.

사전은 키와 값을 콜론(:)으로 짝짓고, 이들을 중괄호({})로 감싸서 표시한다.

{키 1 : 값 1, 키 2: 값 2, 키 3 : 값 3 . . . }

딕셔너리의 값의 추가와 수정은 리스트나 튜플보다 편리하다. 키 값이 존재하는가에 따라 수정과 추가를 할 수 있다.

```
scores = {'국어':80, '영어':70, '수학':100}
scores['과학'] = 85 # 키값이 존재하지 않아 값이 추가
print(scores)
scores['영어'] = 80 # 키값이 존재하여 값이 수정됨
print(scores)
```

dictionary\_1.py

update 메소드를 통하여 딕셔너리를 전체적으로 추가할 수 있다.

```
dict = {'경기도':'수원', '강원도':'원주', '경상북도':'안동'}
dict2 = {'전라남도':'무안', '충청북도':'청주', '강원도':'춘천'}
dic.update(dict2)
print(dict)
```

dictionary\_2.py

"B" 값은 dict 과 dict1에 중첩되므로 나중 값으로 바뀐다.

```
dict = {"A": 1, "B": 2}
dict1 = {"B": 3, "C": 4}
dict |= dict1 # version 3.9
print(dict)
```

dictionary\_3.py

딕셔너리의 값의 제거와 삭제는 list와 유사하다. 단 리스트는 값의 위치나 값 자체로 삭제를 하나, 딕셔너리는 키를 기준으로 삭제를 한다.

```
scores = {'국어':80, '영어':70, '수학':100, '과학':85}
del scores['영어']
print(scores)
scores.pop('수학')
print(scores)
```

dictionary\_4.py

사전의 키를 모두 확인하고 싶다면, .keys() 메소드를 부른다.  
사전의 값을 모두 확인하고 싶다면, .values() 메소드를 부른다.  
사전의 키와 값을 짝지어 보고 싶다면, .items() 메소드를 부른다.

```
scores = {'국어':80, '영어':70, '수학':100, '과학':85}
print(scores.keys())
a_list=list(scores.keys())
print(a_list)
print(scores.values())
b_list=list(scores.values())
print(b_list)
print(scores.items())
c_list=list(scores.items())
print(c_list)
b_list=list(scores.values())
```

dictionary\_5.py

딕셔너리에서 키와 값이 있는지를 확인하려면 in을 사용한다.

- 키 in 딕셔너리: 딕셔너리에 키가 있는지 확인한다.
- 값 in 딕셔너리.values(): 딕셔너리에 값이 있는지 확인한다.
- 딕셔너리.get(키) : 딕셔너리 키의 값을 가져온다.

```
scores = {'국어':80, '영어':70, '수학':100, '과학':85}
print('영어' in scores)
print(scores.get('영어'))
print(75 in scores.values())
```

dictionary\_6.py

리스트와 인덱스와 값을, 딕셔너리에서 인덱스와 키 값을 출력할 때 enumerate( ) 함수를 사용한다. for 문을 enumerate와 같이 사용하면, 인덱스 값과 키 값을 동시에 알

수 있다. i 부분은 value 항목의 인덱스 값으로 출력된다. 키 값에 해당하는 값은 dict[value]로 출력할 수 있다.

만약 인덱스의 시작 값을 바꾸고 싶다면, 원하는 시작 값을 넘겨줄 수 있다. 예시로 인덱스값을 1로 시작하게 바꿔보자.

```
# list
lt = [50, 76, 90, 80, 55]
for i, value in enumerate(lt):
    print(i, value)
for i, value in enumerate(lt, 1): # 시작값
    print(i, value)
print('='*20)
# dictionary
dict = {'kor':50, 'eng':76, 'mat':90, 'sci':80, 'com':55}
for i, value in enumerate(dict):
    print(i, value, dict[value])
```

dictionary\_7.py

items() 메소드를 사용하여 값을 구할 수도 있다.

```
dict = { '한국': '서울', '일본': '동경', '미국': '워싱턴' }
print(dict.items())
print(dict.keys())
print(dict.values())
for i, j in dict.items():
    print(i, '-', j)
for i in dict.keys():
    print(i)
for i in dict.values():
    print(i)
```

dictionary\_8.py

dict={x : x \* x for x in range(1,n+1)} 값은 x의 거듭제곱의 값을 딕셔너리로 만든 것이다. dict.values()는 딕셔너리 값을 리스트로 만든 것이다.

```
n=int(input("숫자 입력 : "))
dict={x:x*x for x in range(1,n+1)}
print(dict)
print(sum(dict.values()))
total=0
for i in dict:
    total += dict[i]
print(total)
```

dictionary\_9.py

튜플을 사용하여 딕셔너리를 생성할 수 있다.

```
tuplex = ((2, 'w'), (3, 't'))
dicts = dict((y,x) for x, y in tuplex)
print(dicts)
```

dictionary\_10.py

딕셔너리를 삭제할 때는 del를 사용한다.

```
del 딕셔너리[키]
```

딕셔너리의 값의 최대값과 최소값을 구할 수 있다.

```
mydict = {'a':1, 'b':2, 'c':3, 'd':4}
for key, value in mydict.items():
    print(key, value)
key=input("삭제할 키를 입력 : ")
if key in mydict:
    del mydict[key]
print(mydict)
key_max = max(mydict.keys(), key=(lambda k: mydict[k]))
key_min = min(mydict.keys(), key=(lambda k: mydict[k]))
print(key_min, '~', key_max, ':', end=' ')
print(mydict[key_min], '~', mydict[key_max])
```

dictionary\_11.py

딕셔너리를 입력할 때는 물론 프로그램 내에서도 입력할 수 있지만 콘솔로 입력이 가능하다.



두 개의 리스트를 하나의 딕셔너리로 만들 때는 zip 함수를 사용한다. 앞의 리스트는 키가 되고 뒤의 리스트는 키에 해당되는 값이 된다.

```
yoil = ["월", "화", "수", "목", "금", "토", "일"]
food = ["갈비탕", "순대국", "칼국수", "삼겹살"]
menu = dict(zip(yoil, food))
print(menu)
for y, f in menu:
    print("%s요일 메뉴 : %s" % (y, f))
```

dictionary\_12.py

외부에서 키와 값을 입력받아 딕셔너리를 만드는 프로그램은 아래와 같다.

```
keys=[]
values=[]
n=int(input("딕셔너리 원소 수 입력 : "))
print("키 입력")
cnt = 0
for x in range(n):
    cnt += 1
    print(cnt, '번', end=' ')
    element=input("키(문자) : ")
    keys.append(element)
print("값 입력")
cnt = 0
for x in range(n):
    cnt += 1
    print(cnt, '번', end=' ')
    element=input("값(숫자) : ")
    values.append(element)
result=dict(zip(keys, values))
print("딕셔너리는")
print(result)
```

dictionary\_13.py

셋을 이용하여 키 값에 해당하는 리스트를 값으로 만들 수 있다. 다음 프로그램의 결과부터 말한다면 다음과 같다.

```
{'x': [1, 2, 3], 'y': [4, 5, 6]}
```

```
l = [("x",1), ("x",2), ("x",3), ("y",4), ("y",5), ("y",6)]
d={}
for a, b in l:
    d.setdefault(a, []).append(b)
print(d)
```

dictionary\_14.py

딕셔너리에 관심을 가지려면 인터넷을 검색한 후 많은 예제를 접해봐야 할 것이다.  
split 함수를 사용하여 단어의 개수를 구하는 프로그램을 작성하여 보자.

```
s = 'I am what I am and that is all that I am'
wrd_list = s.split()
print(wrd_list)
hist_dict = {}
for wrd in wrd_list:
    hist_dict[wrd] = hist_dict.get(wrd, 0) + 1
print(hist_dict)
```

dictionary\_15.py

딕셔너리를 비울 때는 clear 메소드를 사용한다.

```
a_dict = {'test':0}
a_dict.clear()
print(a_dict)
```

dictionary\_16.py

zip() 함수를 이용하여 리스트를 딕셔너리로 만들 수 있다.

```
keys = [1, 2, 3]
values = ["A", "B", "C"]
diction = dict(zip(keys, values))
print(diction)
ks=dict(zip(["year", "month", "date"], [2001, 1, 31]))
print(ks)
```

dictionary\_17.py

함수를 사용하여 키와 값을 바꿀 수 있다.

```
dict = {'a':1, 'b':2, 'c':3, 'd':4}
def flipped_dict(a_dict):
    return {
        value : key
        for key, value in a_dict.items()
    }
print("Dict:",dict)
print("Flipped Dict:",flipped_dict(d))
```

dictionary\_18.py

함수는 추후 배우겠지만 딕셔너리 dict의 키와 값이 바뀐 것을 볼 수 있다.

```
# 딕셔너리
dict = {'a': 123123, 'kim': 'blockdmask', 'b': "blog", 'c': 3333,
123: 'name'}
# key 출력
print(dict.keys())
# key 하나씩 받기
for k in dict.keys():
    print(f'key : {k}')
# values 출력
print(dict.values())
# value 하나씩 받기
for v in dict.values():
    print(f'value : {v}')
# items 출력
print(dict.items())
# items 하나씩 출력
for m in dict.items():
    print(f'한쌍씩 : {m}')
    print(f'Key : {m[0]}')
    print(f'Value : {m[1]}')
```

dictionary\_19.py

### 3. 셋(set)

셋은 집합을 말한다. 집합은 순서가 없고 중복된 값을 가지지 않는다. 집합은 중괄호(`{ }`)로 값을 감싸 만든다. 기호는 사전과 비슷하지만, 키와 값의 짝이 없다는 점이 다르다.

```
{값1, 값2, 값3....}
```

셋의 값은 중복된 값을 허용하지 않는다. 만약 겹치는 항목이 집합에 들어 있다면, 반복되는 항목은 제거된다.

```
subject = {'국어', '영어', '수학', '영어'}  
print(subject)
```

set\_1.py

셋도 리스트와 같은 방법으로 값을 추가하거나 수정할 수 있다. 단 동일한 값을 하나 이상 추가할 수 없다. 셋의 값들은 동일한 값이 없는 고유한 값으로 존재한다.

```
subject = { '국어', '영어', '수학' }  
subject.add('과학')      # 추가  
subject.add('영어')      # 추가 안됨  
subject.remove('수학')   # 제거  
print(subject)  
subject.update({'생물', '역사', '영어'})  
print(subject)
```

set\_2.py

수학에서의 함수와 동일하게 사용할 수 있다.

```
a = { 12, 14, 16, 18, 20, 22 }  
b = { 13, 16, 19, 22, 25 }  
print("교집합", a & b)  
print("합집합", a | b)  
print("차집합", a - b)  
print("차집합", b - a)  
print("배타적 차집합", a ^ b)
```

set\_3.py

```
seta =set(['red', 'green'])
setr = seta.copy()
print(setr)
```

set\_4.py

difference, discard, intersection 메소드를 사용해보자.

```
a_set = {1, 2, 3, 4}
b_set = {3, 4, 5, 6}
c = a_set.difference(b_set)
print(c) # {1, 2} 출력
print(b_set.difference(a_set)) # {5, 6} 출력
```

set\_5.py

'참외' 값을 버려 보자. discard() 메소드를 사용하면 된다.

```
a_set = {'사과', '바나나', '참외'}
a_set.discard('참외')
print(a_set) # {'사과', '바나나'} 출력
```

set\_6.py

intersection() 메소드는 셋 a\_set과 b\_set 의 교집합 즉 겹치는 값으로 셋이 설정된다. union, intersetion, differnce 메소드는 각각 합집합(|), 교집합(&), 차집합(-)에 해당하는 메소드이다.

```
s1 = {1, 2, 3, 4, 5}
s2 = {4, 5, 6, 7}
print(s1 | s2) # 합집합
s2.union(s1)
print(s1 & s2) # 교집합
s1.intersection(s2)
print(s1 - s2)
s1.difference(s2) # 차집합
```

set\_7.py

## 4. 자료 구조의 변경

파이썬은 동적 타입 언어로 변수의 자료형을 지정하지 않고 선언하는 것만으로 값을 지정할 수 있다.

이렇게 선언한 변수의 자료형은 코드가 실행되는 시점에 결정된다. 이는 파이썬의 장점이자 단점이 될 수 있는데 리스트와 튜플 셋 등 자료형을 쉽게 바꿀 수 있다.

```
const_a = 100
const_b = 3.14
const_c = float(const_a)
const_d = int(const_b)
print(const_a, const_b, const_c, const_d)
```

type\_1.py

```
menu = {"커피", "우유", "주스"}
print(menu, type(menu))
menu=list(menu)
print(menu, type(menu))
menu = tuple(menu)
print(menu, type(menu))
menu = set(menu)
print(menu, type(menu))
```

type\_2.py

## 5. 축약

리스트, 딕셔너리, 세트는 이미 존재하는 순회형 자료를 기반으로 새로운 리스트, 딕셔너리, 세트를 생성할 수 있다. 이를 각각 리스트 축약, 딕셔너리 축약, 세트 축약이라 한다.

### 5.1 리스트 축약

대괄호([])로 묶은 표현식과 for 문을 통해 간단하게 리스트를 생성할 수 있으며, 필요에 따라 조건문을 넣을 수도 있다.

표현식은 변수 혹은 변수의 객체를 생성하는 식이 온다.

[표현식 for 변수 in 순회형 <if 불린-표현식>]

```
a_list = [ i for i in range(1,11)]
b_list = [str(i) for i in range(1,11)]
print(a_list)
print(b_list)
c_list=[]
for i in range(1,11):
    c_list.append(i)
print(c_list)
d_list=[]
for i in range(1,11):
    d_list.append(str(i))
print(d_list)
```

comprehension\_1.py

여기서 리스트 a\_list와 b\_list가 축약형 리스트이다. 반복문을 사용하면 c\_list와 d\_list로 구현할 수 있는데 이를 처음 한 줄로 구현할 수 있다.

```
tmp = []
for 변수 in 순회형:
    tmp.append(표현식)
```



[표현식 for 변수 in 순회형]

다음 예는 조건문이 있는 리스트 축약 형식으로 작성해서 실행한 결과를 보여준다.

```
e_list=[str(i) for i in range(1,11) if i % 2 == 1]
print(e_list)
```

comprehension\_2.py

```
tmp = []
for 변수 in 순회형:
    if 불린-표현식:
        tmp.append(표현식)
```



```
[표현식 for 변수 in 순회형 if 불린-표현식]
```

2000년에서 2200년 사이의 윤년을 구하려면 다음과 같이 축약 기법을 사용하면 된다.

```
yun_year = [year for year in range(2000, 2201)
             if year % 4 == 0 and year % 100 != 0 or year % 400 == 0]
print(yun_year)
```

comprehension\_2.py

확장자가 xlsx, csv인 파일을 구하여 보자.

```
f_list= [file for file in os.listdir() if
         file.lower().endswith(('.xlsx', '.csv'))]
print(f_list)
```

comprehension\_3.py

축약 기법도 중첩으로 사용할 수 있다.

```
e_list=[f'{i}*{j}={i*j}' for i in range(2,10) for j in range(1,10)]
print(e_list)
```

comprehension\_4.py

## 5.2 딕셔너리 축약

딕셔너리 축약도 리스트 축약처럼 간결한 구문으로 딕셔너리를 생성하는 방법이다.

```
{키-표현식:매핑값-표현식 for 변수 in 순회형 [if 불린-표현식]}
```



```
a_dict={i:i*2 for i in range(1,10)}  
print(a_dict)
```

comprehension\_5.py

한글과 영문의 키를 바꿔 보자.

```
koreng = dict(사과='apple', 블루베리='blueberry',  
              딸기='strawberry', 키위='kiwi', 바나나='banana',  
              포도='grape', 자두='plum')  
print(koreng)  
engkor = {v: k for k, v in koreng.items()}  
print(sorted(engkor.items()))
```

comprehension\_6.py

### 5.3 세트 축약

세트 축약이란 앞서 설명한 리스트 축약이나 딕셔너리 축약처럼 간결한 구문으로 세트를 생성하는 방법이다. 사용 방법은 리스트 축약과 비슷하다.

{표현식 for 변수 in 순회형 [if 불린-표현식]}

```
a_set={abs(i) for i in range(-9, 10)}  
print(a_set)
```

comprehension\_7.py

#### = exe 만들기

pip install pyinstaller # 1회 실행

pyinstaller .\practice.py

소스 마지막 line에 key=input() 추가 화면에 보이게 함

pyinstaller --onefile sam.py

```
import sys
# 0은 일요일, 1은 월요일...
y = int(input('연도는? >> '))
m = int(input('월은? >> '))
d = int(input('일은? >> '))
yoil = ['일', '월', '화', '수', '목', '금', '토']
yoil_e = ['Sunday', 'Monday', 'Wednesday', 'Thursday', 'Friday', 'saturday']
yoil_h = ['日', '月', '火', '水', '木', '金', '土']
y0 = y - (14 - m) // 12
x = y0 + y0//4 - y0//100 + y0//400
m0 = m + 12 * ((14 - m) // 12) - 2
d0 = (d + x + (31*m0)//12) % 7
print(str(y)+"년"+str(m)+"월"+str(d)+"일은",
      yoil[d0], "(", yoil_e[d0], yoil_h[d0], ") 요일입니다.")
```

comprehension\_8.py

```
list1 = [1, 2, 3, 4]
list2 = ['a', 'b', 'c', 'd']
list3 = ['あ', 'い', 'う', 'え']
for tp in zip(list1, list2, list3):
    print(tp)
a = [0, 1, 2, 3, 4]
b1 = a[:]      #b1 = [0, 1, 2, 3, 4] => 전체 복사
b2 = a[2:]     #b2 = [2, 3, 4]      => a[2] 포함
b3 = a[:2]     #b3 = [0, 1]        => a[2] 불포함
print(b1,b2,b3)
c1 = a[-4:]    #c1 = [0, 1, 2, 3, 4] => 전체 복사
c2 = a[:-1]    #c2 = [0, 1, 2, 3]  => a[-1] 불포함
c3 = a[-3:-1]  #c3 = [2, 3]
c4 = a[-2:-3]  #c4 = []            => 빈 리스트
print(c1, c2, c3, c4)
d1 = a[::-1]   #d1 = [4, 3, 2, 1, 0] => 뒤집기
d2 = a[:2]     #d2 = [0, 2, 4]      => 짝수만
d3 = a[1::2]   #d3 = [1, 3]        => 홀수만
print(d1, d2, d3)
```

comprehension\_7.py

## CHAP 7. 함수



## 1. 함수

### 1.1 정의

프로그램에서 반복하는 작업이 필요할 경우 함수의 이름만 적어주면 함수 안에 들어 있는 명령들이 실행된다. 이처럼 함수는 반복되는 코드를 정의하여 필요할 때 불러서 사용하는 것을 말한다. 기존 파이썬에서 `int( )`, `sum( )` 등 만들어져 있는 내장 함수가 있고 사용자가 필요 때문에 만들어진 함수가 있다. 사용자가 함수를 호출할 때 함수에게 넘겨주는 값이 있는데 이를 인수라하고 함수에서는 넘겨받아 매개변수로 활용한다. 그리고 처리한 값을 다시 호출한 곳으로 돌려주기도 한다.

수학에서 말하는 함수와는 비슷하지만 약간 다르다. 입력과 출력이 일정한 관계를 맺고 있다는 점은 같다. 함수를 만든 목적은 두 가지인데 첫째는 큰 프로그램을 짤 때, 복잡하고 세세한 일을 하나의 블록으로 묶어 따로 만들 수 있으면 여기에 적당한 이름을 붙여 함수로 정의를 해 두고 필요할 때 불러 쓰도록 하면 프로그램의 전체적인 윤곽을 먼저 만들고 여럿이 일을 나누기 편하게 하기 위함이다.

둘째 목적은 비슷한 일을 자주 반복하게 될 때 이를 따로 함수로 정의하여 두고 바뀌어야 할 부분만 인수로 전달하도록 하여 프로그램 작성을 쉽게 하기 위함이다. 즉 함수를 잘 활용해서 일을 나눠야 쉽고 좋은 프로그램을 작성할 수 있다.

#### 인수(매개변수 `argument`, `parameter`, ):

함수가 일을 수행하는데 필요한 값을 호출하는 곳으로부터 전달받는 값을 말한다. 호출하는 곳에서는 실제 전달할 수 있는 값을 적어야 하고, 받는 함수 쪽에는 이를 받아 저장할 수 있는 변수(매개변수) 이름이 나온다.

```
# 위치 매개 변수
def my_func(a, b):
    print(a, b)
my_func(2, 3)

# 기본 매개 변수
def post_info(title, content='내용 없음'):
    print('제목:', title)
    print('내용:', content)

# 키워드 매개 변수
def post_info(title, content):
    print('제목:', title)
    print('내용:', content)
post_info(content='없어요', title='여자친구 만드는 방법')

# 위치 가변 매개 변수
```

```
def print_fruits(*args):
    for arg in args:
        print(arg)
print_fruits('apple', 'orange', 'mango', 'grape')

# 키워드 가변 매개 변수
def comment_info(**kwargs):
    for key, value in kwargs.items():
        print(f'{key} : {value}')
comment_info(name='파린이', content='정말 감사합니다!')
```

### 함수의 정의(define)

함수가 해야 할 일을 작성한 부분으로 함수 호출 전에 나와야 한다. def 로 시작하며 다음 문장들은 들여쓰기로 함수의 범위를 표시한다. 매개변수들은 함수를 호출한 곳에서 넘겨주는 인수들을 받아서 함수 내부에서 사용하기 위한 변수이름들이고 이 변수들은 함수 내부에서만 존재하며 함수가 호출하면 만들어지고 함수가 실행을 마치고 호출된 곳으로 돌아 갈 때(리턴 return) 없어진다.

### 함수의 호출(function call)

함수를 호출하여 실행하는 부분으로 함수 이름 뒤에 괄호를 열고 전달할 인수 값 들을 콤마로 분리하며 적는다. 전달할 인수 가 없어도 빈 괄호를 써야 한다. 리턴 값은 함수를 호출한 그 자리로 돌아오며 리턴 값이 두 개 이상이면 튜플 형태로 돌아온다. 함수의 호출은 수식의 일부로 사용될 수 있다. 또는 다른 함수를 호출하는 인수의 일부로 들어갈 수 있다. 결과를 =연산자를 이용해 변수로 받을 수도 있다. 리턴값이 없거나 있어도 활용하지 않을 수도 있다.(입출력 등 함수가 실제 행한 일에 목적을 두고 결과가 필요치 않을 경우, ex. print문)

## 1.2 내장 함수

파이썬의 내장 함수는 다음과 같다. 파이썬의 버전에 따라 추가되기도 한다.

abs()	all()	any()	ascii()	bin()
bool()	breakpoint()	bytearray()	bytes()	callable()
chr()	classmethod()	compile()	complex()	delattr()
dict()	dir()	divmod()	eval()	enumerate()
exec()	filter()	float()	format()	frozenset()
getattr()	globals()	hasattr()	hash()	help()
hex()	id()	input()	int()	isinstance()
issubclass()	iter()	len()	list()	locals()
map()	max()	memoryview()	min()	next()
object()	oct()	open()	ord()	pow()
print()	property()	range()	repr()	reversed()
round()	set()	setattr()	slice()	sorted()
staticmethod()	str()	sum()	super()	tuple()
type()	vars()	zip()	__import__()	

표5 내장함수

이미 enumerate(), input(), int(), len(), list(), sum(), print() 등은 이미 사용하여 보았을 것이다.

그 외 몇 가지 중요한 내장 함수만 알아보기로 하자.

### abs()

절대값을 구하는 함수이다.

ex) abs(-4.5) # 4.5, abs(2+2j) # 2.8284271247461903 복소수 벡터의 크기

### any(), all()

all()은 원소가 모두 존재할 때 참 any()는 하나만 존재하여도 참을 반환한다.

```
adult = [True, False, True, False ]
```

```
print(any(adult))
```

```
print(all(adult))
```

### bin(), oct(), hex()

정수를 2진수로 바꿔줌

ex) bin(7) # 0b111, bin(-5) # -0b101, hex(37) # 0x25, oct(12) # 0o14

oct()는 정수를 8진수로 변환, hex()는 16진수로 변환하는 함수이다.

### bool()

논리값, 즉 True 또는 False 중 하나를 돌려준다.

ex) bool(1) # True, bool(0) # False,

`bool(a & b)` # a가 True이고 b가 False이면 결과는 False

### **chr(), ord()**

`chr()`은 문자 코드에 해당하는 문자를 돌려줌. `ord()`는 문자에 해당하는 코드를 알려줌

ex) `chr(65)` # 'A'    `ord('A')` # 65

### **eval()**

매개변수로 받은 식을 문자열로 받아서, 실행하는 함수

ex) `a = 100`  
`print(eval('3 * a + 55'))`

```
test0 = 1
test1 = 2
test2 = 3
test3 = 4
test4 = 5
for i in range(5):
    print(eval('test'+str(i)))
for i in range(5):
    total += eval('test'+str(i))
```

function\_1.py

### **exec()**

기본적으로 문자열로 된 코드를 입력으로 받아 그 코드를 실행만 시키고, 아무것도 return하지 않는 함수

```
a=10
exec("b = a + 20")
print(b)
```

function\_2.py

### **int(), float()**

`int()`는 문자 형태의 숫자나 실수를 정수로 `float()`는 실수로 바꿔주는 함수

ex) `int('43')` # 43, `int(23.7)` # 23, `float(3)` # 3.0, `float('32.7')` # 32.7

### **len()**

객체의 길이를 구해 준다. 문자열일 경우 문자의 수, 리스트일 경우는 원소의 수를 구한다.

ex) `len('Republic of Korea')` # 17, `len([23, 34, 35])` # 3

### **list(), tuple()**

주어진 숫자들을 리스트와 튜플로 변경하여 준다.

ex) `list((10, 34))` # 튜플을 리스트로 [10, 34],  
`tuple([10, 34])` # 리스트를 튜플로 (10, 34)

### **max(), min(), sum()**

주어진 숫자들(리스트)의 최대값, 최소값, 합을 구하는 함수

ex) `max([12, 34, 17])` # 34, `min([12, 34, 17])` # 12, `sum([12, 34, 17])` # 63

```
score = [75, 95, 60, 86]
print(max(score))
print(min(score))
print(sum(score))
```

function\_2.py

### **pow()**

어떤 수의 거듭제곱을 구한다.

ex) `pow(10, 2)` # 100, `pow(10, -3)` # 0.001

### **round()**

주어진 수의 반올림 값을 구한다.

ex) `round(3.14)` # 3, `round(3.14159, 3)` # 3.142

```
print(round(3.14159))
print(round(3.14159, 3))
```

function\_3.py

### **type()**

데이터의 형을 알려준다.

ex) `type([10, 23, 34])` # <class list>, `type(True)` # <class bool>

### **print()**

데이터를 출력한다. 아래와 같이 다양한 출력 양식이 있으므로 익히기 바란다.



```
citys = {'Seoul':5034, 'Suwon':761, 'Gwangju':1020, 'Busan':315}
for city, value in citys.items():
    print("도시:",city,"인원:",value)
print('-'*30)
for city, value in citys.items():
    print("도시: {0}  인원: {1}".format(city, value))
print('-'*30)
for city, value in citys.items():
    print(f"도시: {city}  인원: {value}")
print('-'*30)
for city, value in citys.items():
    print("도시:",city.ljust(8),"인원:",str(value).rjust(5))
print('-'*30)
for city, value in citys.items():
    print("도시: {0:8s}  인원: {1:,}".format(city, value))
print('-'*30)
for city, value in citys.items():
    print("도시: {0}  인원: {1:.2f}".format(city, value))
print('-'*30)
for city, value in citys.items():
    print(f"도시: {city.ljust(8)}  인원: {str(value).rjust(5)}")
```

function\_4.py

ljust()와 rjust()는 왼쪽과 오른쪽으로 정렬해주는 메소드이다. 숫자에 ‘,’ 표시를 하고 왼쪽과 오른쪽 정렬을 하면 다음과 같다. 실제로 정돈되게 출력하려면 다음 방식을 참고하여 사용하면 된다.

```
citys = {'Seoul':5034, 'Suwon':761, 'Gwangju':1020, 'Busan':315}
space=" "*30
for city, value in citys.items():
    city=city+space[:8-len(city.encode('euc-kr'))] # (한글!=영문)글자수
    value = f"{value:,}".rjust(5)
    print(f"도시: {city}  인원: {value}")
```

function\_5.py

한글과 영문자를 혼용해 쓸 경우 한글 한글자는 영문 두 글자이므로 encode = 'euc-kr'로 글자 수를 계산해서 그 글자만큼 빈칸을 채워 열을 맞추는 방법이다. 이 방법을 사용해서 열을 맞춰 출력하는 방법을 사용하면 된다.

많이 사용되는 내장 함수를 나열해보았다. 그 외의 내장 함수는 아래 사이트에서 확인하고 추후 필요할 때 학습하기로 한다.

<https://docs.python.org/ko/3.9/library/functions.html>

```
citys = {'Seoul':5034, 'Suwon':761, 'Gwangju':1020, 'Busan':315}
for city, value in citys.items():
    city = f"{city}".ljust(8)
    value = f"{value:,}".rjust(5)
    print(f"도시: {city} 인원: {value}")
```

function\_6.py

```
# 파이썬 변수 출력 예
age = 30
color = "노랑"
print(f"나는 {age}살이며, {color}색을 좋아해요.")
```

## 2. 사용자 함수

사용자가 직접 만들어 사용하는 함수이다.

매개변수는 함수에 입력으로 전달된 값을 받는 변수이다. 인수 또는 인자는 함수를 호출할 때 전달하는 입력값이다.

입력값	결과값	사용 방법	예시
유	유	결과값 변수=함수명(인수1, 인수2...)	<pre>def plus(a, b):     hap=a+b     return hap a=plus(3,4) print(a)</pre>
무	유	결과값 변수=함수명()	<pre>def insa():     return '안녕' a=insa() print(a)</pre>
유	무	함수명(인수1, 인수2...)	<pre>def plus(a,b):     print(a+b) print(plus(3, 4))</pre>
무	무	함수명()	<pre>def insa():     print('안녕') insa()</pre>

표 6 사용자 함수

```
open_account()
def open_account():
    print("새로운 계좌가 생성되었습니다.")
function_7.py
```

```
def fruit():
    print("과일")
    print("사과")
fruit()
fruit()
function_8.py
```

```
def total(a, b):
    s= a+b
    return s
print(total(3, 5))
print(total(12, 34))
function_9.py
```

```
def get_qr(a, b):  
    return a // b, a % b  
x = 19  
y = 4  
quo, rem = get_qr(x, y)  
print("몫 : ", quo, "나머지 : ", rem)
```

function\_10.py

일반적으로 함수의 인수 개수는 정해져 있다. 하지만 필요시 인수 개수를 정하지 않고 함수를 사용할 수 있다. args는 임의로 정한 변수명으로, 얼마든지 다른 변수명으로 변경할 수 있다.

```
def add_sum(*args):  
    result=0  
    for i in args:  
        result=result + i  
    return result  
result1 = add_sum(1, 2, 3)  
result2 = add_sum(1, 2, 3, 4, 5)  
print(result1, result2)
```

function\_9.py

```
def calcstep(**args):  
    begin = args['begin']  
    end = args['end']  
    step = args['step']  
    sum = 0  
    for num in range(begin, end + 1, step):  
        sum += num  
    return sum  
print("3 ~ 10 =", calcstep(begin = 3, end = 10, step = 1))  
print("1 ~ 10 =", calcstep(step = 1, end = 10, begin = 1))
```

function\_11.py

함수 안에서 또 다른 함수를 호출할 수 있다. 파이썬은 다음과 같이 함수 안에 함수를 정의할 수 있습니다. 함수 안쪽에 정의된 함수를 내부 함수라고 부르고, 바깥쪽의 함수를 외부 함수라고 부릅니다.

```
def calc(op,a,b):  
    op(a,b)  
def add(a, b):  
    print(a+b)  
def multi(a,b):  
    print(a*b)  
calc(add,1,2)  
calc(multi,3,4)
```

function\_12.py

```
def outer():  
    def inner():  
        print("inner")  
    return inner  
f = outer()  
f()
```

function\_13.py

함수에서 같은 함수를 호출하는 것을 재귀 호출이라 하는데 간단히 팩토리얼 값을 구할 수 있다.

```
def factorial(n):  
    if n > 1:  
        return n * factorial(n-1)  
    else:  
        return 1  
print(factorial(5))
```

function\_14.py

재귀 호출로 1에서 100까지의 합을 구할 수 있다.

```
def sum(n):  
    if n == 0:  
        return 0  
    return n + sum(n-1)  
print(sum(100))
```

function\_15.py

```
def sum_numbers(numbers):  
    return sum(int(x)  
                for x in numbers.split()  
                if x.isdigit() )  
print(sum_numbers('10 20 30 abcd efgh 40 50'))  
function_16.py
```

### 스코핑 룰(scoping rule)

변수들은 만들어진 위치에 따라 통용되는 범위가 결정된다. 메인 프로그램(함수 밖)에서 만들어진 것은 프로그램 전체 어디서나 사용 가능하고 함수 안에서 만들어진 변수는 그 함수 안에서만 사용할 수 있다.

```
x=100  
def fun1(a):  
    return a+x  
print(fun1(20))  
def fun2(a):  
    x=200  
    return a+x  
print(fun2(20))
```

function\_17.py

```
## g, h 는 전역 이름  
g = 10  
h = 5  
def f(a):          # a는 지역  
    h = a + 10    # h는 지역, 새로 정의했으므로  
    b = a + g     # b도 지역, , g는 전역  
    return b
```

function\_18.py

# CHAP 8. 클래스와 예의 처리



## 1. 클래스

### 1.1 클래스의 개념

객체는 크게 객체 자체가 가질 수 있는 정보와 그러한 정보를 기반으로 객체가 수행할 수 있는 행동을 통해서 정의된다. 이러한 정보와 행동을 정의할 수 있게 해주는 일종의 틀이 클래스이며, 클래스를 통해 객체를 만드는 작업을 통해 객체지향 프로그램을 완성할 수 있다.

### 2.1 클래스의 정의

파이썬에서 클래스는 역시 '객체'로 일종의 함수이다. 즉, 파이썬 클래스는 결과값(output)을 갖는 함수(function)의 일종으로 단지, 정의시 'def'가 아니라 'class'란 키워드를 사용하고, 하나 이상의 내장 함수들을 갖게 되어 있으며, 클래스의 함수실행 결과값이 다른 아닌 인스턴스(instance)가 된다는 점 등에서 다른 함수와 다른 특색이 있을 뿐이다.

파이썬의 클래스는 서로 관련이 깊은 정보와 행동을 같이 묶을 수 있는 수단을 제공하는 일종의 설계도라고 볼 수 있다. 클래스 자체만으로는 할 수 있는 일이 없으나 클래스를 통해 객체를 실체화하는 과정을 통해서 실제 정보를 담은 객체를 만들고 이러한 객체들의 관계를 적절히 조율하며 객체 지향 프로그래밍을 수행할 수 있으므로 파이썬을 이용한 객체지향 프로그래밍에서 클래스는 필수적인 존재라고 할 수 있다.

```
class 클래스명 :  
    변수A  
    변수B  
    ...  
def 메소드명() :  
    문장1  
    문장2  
    ...  
def 메소드명() :  
    문장I  
    문장II  
    ...
```



```
class Animal :
    name = '고양이'
    def sound(self) :
        print('냐옹~')
cat = Animal()
print(cat.name)
cat.sound()
```

class\_1.py

```
class FourCal:
    def setdata(self, first, second): # 초기자
        self.first = first
        self.second = second
    def add(self): # 메서드는 클래스 안에서 정의된 함수
        result = self.first + self.second
        return result
a = FourCal()
c=a.setdata(4,2) # instance
print(a.first)
print(a.second)
print(a.add())
```

class\_2.py

### 1.3 클래스 상속

이미 만들어진 클래스에 클래스를 확장하는 방식으로 클래스를 정의하는 경우 사용한다.

```
class Member :
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def showMember(self) :
        print('이름 : %s'%self.name)
        print('나이 : %d'%self.age)
mem1 = Member('황재호',30)
mem1.showMember()
```

class\_3.py

```
class Student :
    total = 0
    avg= 0.0
    def __init__(self,name,kor,eng,math):
        self.name = name
        self.kor = kor
        self.eng = eng
        self.math = math
    def getSum(self):
        self.total = self.kor + self.eng + self.math
        return self.total
    def getAvg(self):
        self.avg = self.total/3
        return self.avg
s1 = Student('홍지영',90,90,100)
print(' 이름 : %s '%s1.name)
print(' 합계 : %d '%s1.getSum())
print(' 평균 : %.1f'%s1.getAvg())
```

class\_4.py

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is', self.name)
p = Person('Swaroop')
p.say_hi()
```

class\_5.py

## 2. 예외 처리

소프트웨어 실행 시 문제가 발생하여 멈추는 경우가 있다. 숫자를 계산해야 하는데 문자로 계산하든지 리스트의 범위를 벗어나서 호출한다든지 여러 상황에서 예외가 발생할 수 있는데 이를 처리하는 경우를 말한다. 아래 명령을 하나하나 실행해보자.

```
print(23/0)
print(speed*3)
print("8"+2)
```

ZeroDivisionError, NameError, TypeError 등이 예외의 원인이 된다. 파이썬에서 발생하는 예외를 알아보자.

예외	원인
ImportError	import문이 모듈을 로드하지 못할 때 발생함
ModuleNotFoundError	모듈을 찾을 수 없을 때 발생. 모듈을 설치해줘야 함
IndexError	리스트등의 인덱스가 범위를 벗어날 때 발생
KeyError	딕셔너리 키가 발견되지 않을 때 발생
NameError	변수를 찾을 수 없을 때 발생
ZeroDivisionError	어떤 수를 0으로 나눌려고 할 때 발생
TypeError	데이터의 형이 일치하지 않을 때 발생
SyntaxError	문법적인 예외가 발생

```
try:
    num = int(input("정수 입력 "))
    print("원의 둘레:", 2 * 3.14 * num)
    print("원의 넓이:", 3.14 * num * num)
except Exception as exception:
    print("type( ):", type(exception))
    print("exception:", exception)
```

except\_1.py

정수를 입력하지 않고 문자(a, b, c등)를 입력하면 예외 처리가 되어 다음과 같이 출력될 것이다.

```
정수 입력 a
type( ): <class 'ValueError'>
exception: invalid literal for int() with base 10: 'a'
```

정수를 입력하지 않아 예외 처리(ValueError)가 발생한 것이다.

```
score = [96, 82, 77, 36]
try:
    num = int(input("정수 입력 "))
    print(f"인덱스 {num}번 값: {score[num]}")
except Exception as exception:
    print("type(exception):", type(exception))
    print("exception:", exception)
```

except\_2.py

이번에는 리스트 색인 범위를 벗어난 숫자 9를 입력 예외 처리(IndexError)가 발생한 것이다.

정수 입력 9

type(exception): <class 'IndexError'>

exception: list index out of range

들여쓰기를 잘못된 경우이다.

```
score = [96, 82, 77, 36]
try:
    num = int(input("정수 입력 "))
    print(f"인덱스 {num}번 값: {score[num]}")
except Exception as exception:
    print("type(exception):", type(exception))
    print("exception:", exception)
```

except\_3.py

print(f"인덱스 {num}번 값: {score[num]}")

IndentationError: unexpected indent

# CHAP 9. 기본 모듈과 패키지



## 1. 모듈

파이썬 프로그램은 다양한 모듈이 존재한다. 여러 가지 기능을 하는 모듈에서 필요한 함수나 클래스를 불러와서 사용하면서 여러 가지 프로그램을 작성할 수 있다. 모듈은 pip로 터미널에서 설치해야 하는데 ()안에 모듈 설치 명령을 기술했다.

### 1.1 기본 모듈

#### 1) os 모듈

os 모듈은 운영체제(os)에서 제공하는 기능을 파이썬에서 수행할 수 있게 해주는 모듈이다.

<https://docs.python.org/ko/3/library/os.html>

```
import os
data = os.listdir(".")
print(data)
for d in data:
    print(d)
    print("directory : " + str(os.path.isdir(d)))
    print("file : " + str(os.path.isfile(d)))
```

module\_1.py

```
import os
# 현재 디렉터리
dir = str(os.getcwd())
print(dir)
# 서브 디렉터리
path = dir+'/eprog'
files=[file for file in os.listdir(path) if file.endswith(".py")]
cnt=0
for file in files:
    cnt += 1
    print(' %3d : ' % cnt, file)
```

module\_2.py

#### os.getcwd()

현재 작업 디렉터리를 나타내는 문자열을 반환한다.

#### os.listdir(path)

path에 의해 주어진 디렉터리에 있는 항목들의 이름을 담고 있는 리스트를 반환한다.

## 2) 문자 찾기

### find(찾을문자, 찾기시작할위치)

find는 문자열중에 특정 문자를 찾고 위치를 반환해준다. 없을경우 -1을 리턴

### startswith(시작하는문자, 시작지점)

startswith는 문자열이 특정 문자로 시작하는지 여부를 알려준다. true나 false를 반환한다.

### endswith(끝나는문자, 문자열의시작, 문자열의끝)

endswith는 문자열이 특정 문자로 끝나는지 여부를 알려준다. true나 false를 반환한다.

```
words='Repulic of Korea'
print(words.find('o')) # 위치(0부터시작)
print(words.startswith('R'))
print(words.startswith('i'))
print(words.startswith('i',5))
print(words.endswith('rea'))
print(words.endswith('c',0,9))
print(words.endswith('c',0,7))
```

module\_3.py

## 3) 경로 지정

### 상대 경로 지정

현재 폴더를 기준으로 지정하여 표시하는 방법

폴더와 파일의 위치 예

(현재폴더)

```
root---Python39---news.txt
|
+---Doc-----python39.chm
+---Temp-----sample.txt

./news.txt
./Doc/python399.chm
../Temp/sample.txt
```

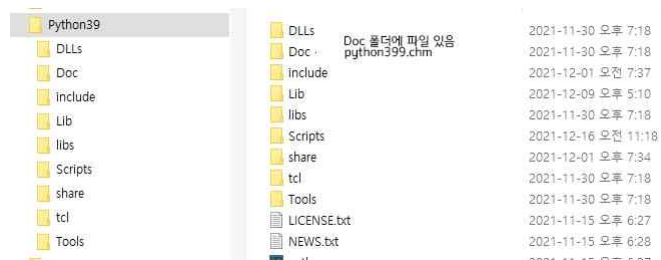
## 절대 경로 지정

root 디렉터리부터 시작하여 경로를 표시하는 방법

c:/Python39/news.txt

c:/Python39/Doc/python399.chm

c:/Temp/sample.txt



## 4) time 모듈

시간을 다룰 수 있는 라이브러리

현재 시각 구하기

```
time.ctime()
```

지정한 초 수 동안 일시 정지하기

```
time.sleep(초 수)
```

## 5) datetime 모듈

모듈은 날짜와 시간을 조작하는 클래스를 제공한다.

<https://docs.python.org/ko/3/library/datetime.html>



```
from datetime import datetime
daydict = {0: '월요일', 1: '화요일', 2: '수요일', 3: '목요일', 4: '금요일',
          5: '토요일', 6: '일요일'}
current_time = datetime.now()
utc_time = datetime.utcnow() # 세계 협정시
current_date = datetime.today()
yoi1 = daydict[current_date.weekday()]
print(current_time, "UTC:", utc_time)
print(current_date.ctime())
print(f'Year : {current_time.year}')
print(f'Month : {current_time.month}')
print(f'Day : {current_time.day}')
print(f'Hour : {current_time.hour}')
print(f'Minute : {current_time.minute}')
print(f'Second : {current_time.second}')
print(current_date.strftime("%Y년 %m월 %d일"), "(" + yoi1 + ")")
```

module\_4.py

### weekday()

정수로 요일을 반환한다. 월요일은 0이고 일요일은 6이다.

### isoweekday()

정수로 요일을 반환한다. 월요일은 1이고 일요일은 7이다.

### strftime(format)

명시적인 포맷 문자열로 제어되는, 날짜를 나타내는 문자열을 반환한다.

다음 프로그램은 입력한 날에서 오늘이 며칠째인지 알려주는 프로그램이다.

```
import datetime
today = datetime.date.today()
t_year = int(input("년 : "))
t_month = int(input("월 : "))
t_day = int(input("일 : "))
target_day = datetime.date(t_year, t_month, t_day)
values = today - target_day
print("오늘은", target_day, "에서", values.days, "일째 입니다.")
```

module\_5.py

이를 응용하여 100일 200일 ..1,000일 기념일을 알려주는 프로그램이다.

```
from datetime import date, timedelta, datetime
y=int(input('년:'))
m=int(input('월:'))
d=int(input('일:'))
dd=datetime(y,m,d)
print(dd)
print( f'기념일은 {dd.year}년 {dd.month}월 {dd.day}일' )
for i in range(1,11):
    nd=dd+timedelta(days=i*100-1) # 100의 배수
    print( f'{i*100}일 {nd.year}년 {nd.month}월 {nd.day}일' )
```

module\_6.py

timedelta()

두 date, time 또는 datetime 인스턴스 간의 차이를 마이크로초 해상도로 나타내는 기간을 나타내는 함수로 인수로 days=정수를 써서 장수 날만큼의 이후 날짜를 구할 수 있다.

6) random 모듈

임의의 수 난수를 발생시키는 모듈이다.

<https://docs.python.org/ko/3/library/random.html>

```
import random
items = [1, 2, 3, 4, 5, 6, 7]
print(random.random())
print(random.uniform(1, 10))
print(random.randint(1, 10))
print(random.randrange(0, 101, 2))
print(random.choice('abcdefghij'))
print(random.sample(items, 3))
random.shuffle(items)
print(items)
```

module\_7.py

random.random()

0부터 1사이의 랜덤 실수를 리턴한다.

`random.uniform(1, 10)`

2개의 숫자 사이의 랜덤 실수를 리턴한다.

`random.randint(1, 10)`

2개의 숫자 사이의 랜덤 정수를 리턴한다.

`random.randrange(0, 101, 2)`

`range(start, stop, step)` 함수로 만들어지는 정수 중에 하나를 랜덤하게 리턴한다.

`random.choice('abcdefghij')`

랜덤하게 하나의 원소를 선택한다.

`random.sample([1, 2, 3, 4, 5], 3)`

랜덤하게 여러 개의 원소를 선택한다.

`random.shuffle(items)`

원소의 순서를 랜덤하게 바꿈

임의의 숫자를 제시하여 숫자를 맞추는 프로그램이다.

```
import random
number = random.randint(1,100)
count = 0
while True:
    guess = int(input('숫자를 입력하세요(1-100) :'))
    count += 1
    if guess == number:
        print('정답입니다')
        print(f"{count}번째에 맞추었습니다.")
        break
    elif guess > number:
        print('더 작은 수 입니다')
    else:
        print('더 큰 수 입니다.')
```

module\_8.py

```
import random
def position(number, title, column):
    print(title)
    students = list(range(1, number+1))
    random.shuffle(students)
    pos = ""
    for i in range(0, number):
        if students[i] < 10:
            pos = pos + "0" + str(students[i]) + "번Wt "
        else:
            pos = pos + str(students[i]) + "번Wt "
        if (i+1) % column == 0:
            pos = pos + "Wn"
    print(pos)
position(20, title="3-2반", column=5)
```

module\_9.py

#### 외부 모듈의 설치

외부 모듈을 터미널에서 pip 명령으로 설치할 수 있다. 우선 무조건 아래 명령으로 pip 명령을 업그레이드 해보자.

```
pip install --upgrade pip
```

만약 프로그램을 실행시켰을 때 다음과 같은 메시지가 출력되면

```
ModuleNotFoundError: No module named 'openpyxl'
```

다음과 같이 ' '안의 모듈을 설치한다.

```
pip install openpyxl
```

#### 7) sys 모듈

인터프리터에 의해 사용되거나 유지되는 일부 변수와 인터프리터와 강하게 상호 작용하는 함수에 대한 액세스를 제공한다.

<https://docs.python.org/ko/3/library/sys.html?highlight=sys#module-sys>

```
import os
import sys
def work(view):
    print('입력 인덱스 :')
    view=sys.argv
    for i, name in enumerate(view):
        print(i,":", name)
    print("\n파일명 :")
    files = os.listdir(view[1])
    print(files)
if __name__ == '__main__':
    work(sys.argv)
else:
    print('다른 모듈에서 불러옴')
```

module\_10.py

실행은 터미널에서 python 1.2\_module5.py data를 입력하여 실행한다. 인덱스는 0은 프로그램명 1.2\_module5.py이고 인덱스 1은 폴더명은 data이다. 폴더명은 인덱스 원소인 sys.argv[1]이 된다. 1.2\_module5.py는 sys.argv[0]이다. 물론 data는 존재하는 폴더명이다. 폴더가 존재하지 않으면 data 폴더를 만들거나 존재하는 폴더 c:\python3를 대신 입력하면 된다.

```
python 1.2_module5.py c:\python39
```

또다른 sys 모듈의 사용 예이다. 어떤 결과가 나오는지 반드시 확인하기 바란다.

```
if __name__ == '__main__':
    work(sys.argv)
else:
    print('다른 모듈에서 불러옴')
```

모든 파이썬 모듈은 속성을 가지고 있다. 만약 그 이름이 'main'일 경우, 이것은 모듈이 사용자로부터 직접 실행된 것임을 의미하므로 이에 맞는 적절한 처리를 해줄 수 있다. 모든 모듈은 이름을 갖고 있으며, 모듈 내에 포함된 명령을 통해 모듈의 이름을 알아올 수 있다. 이 속성은 현재 모듈이 불러들여져서 사용되고 있는지 아니면 인터프리터에서 곧바로 실행된 것인지를 구분하는데 편리하게 사용될 수 있다. 이전에 알아보았지만, 모듈 내부의 코드는 모듈이 첫 번째로 불러들여졌을 때 실행되게 된다. 이러한 속성을 통해 모듈이 외부로부터 불러들여졌을 때와 곧바로 실행되었을 때 각각 다르게 처

리하도록 할 수 있다. 이를 위해 name 속성을 사용한다.

#### 8) math 모듈

수학에서 사용하는 함수를 제공한다.

<https://docs.python.org/ko/3/library/math.html#module-math>

`math.fabs(x)`

x의 절대값을 구한다.

`math.factorial(x)`

x의 팩토리얼 값을 구한다.

`math.gcd(*interger)`

정수의 최대 공약수를 구한다.

`math.exp(x)`

e의 x 거듭제곱 값을 구한다.

`math.log(x[, base])`

x의 자연로그 값을 구한다. 밑수는 base

`math.log10(x)`

밑수 10인 로그를 반환한다.

`math.pow(x, y)`

x의 y 거듭제곱 값을 구한다.

`math.sqrt()`

제곱근을 구한다.

```
import math
print(math.sin(math.radians(45)))
print(math.sqrt(2))
print(math.factorial(5))
```

module\_11.py

삼각 함수 값 (일반적인 각도 45°는 radian으로 변환하여 구해야 한다)  
math.sin(x) ex) math.cos(math.radians(45))

math.cos(x)

math.tan(x)

아크 삼각함수 값

math.asin(x)

math.acos(x)

math.atan(x)

각도 변환

math.degrees(x)

각도 x를 라디안에서 도(degree)로 변환합니다.

math.radians(x)

각도 x를 도(degree)에서 라디안으로 변환합니다.

함수	설명
math.pi	원주율
math.e	자연상수
abs()	절대값 계산 함수(내장함수)
round()	반올림 계산 함수(내장함수)
math.trunc()	버림 계산 함수
math.factorial()	팩토리얼 계산 함수
math.degrees()	라디안을 입력받아 도를 계산
math.radians()	도를 입력받아 라디안을 계산
math.cos()	입력된 라디안에 대한 코사인 값을 계산
math.sin()	입력된 라디안에 대한 사인 값을 계산
math.tan()	입력된 라디안에 대한 탄젠트 값을 계산
math.acos()	cos()의 역함수
math.asin()	sin()의 역함수
math.atan()	tan()의 역함수
math.pow()	제곱 연산
math.sqrt()	제곱근 연산
math.log()	첫 번째 매개변수의 로그
math.log10()	밑수가 10인 로그

math.copysign(3.14, -3)

두 번째 인자의 부호를 가져와 첫 번째 인자의 부호에 적용하는 함수이다.

math.factorial(5)

팩토리얼 함수로써, 1부터 주어진 인자의 값까지 모두 곱한 값

math.gcd(6, 8)      두 수의 최대 공약수

math.log(10, 10)

math.log(a, b)는 로그 함수이며 b를 밑으로 하는 log a에 대한 로그 값

## 파이썬 모듈(라이브러리) 종류와 설치 방법

모듈명	설치 방법
설명	
BeautifulSoup	pip install beautifulsoup4
Python 크롤링에 필요한 라이브러리	
folium	pip install folium
지도 시각화 도구 라이브러리	
numpy	pip install numpy
벡터, 행렬 등 수치 연산을 수행하는 선형 대수 패키지	
matplotlib	pip install -U matplotlib
그래프를 그리고 시각화 하기 모듈	
openpyxl	pip install openpyxl
엑셀 파일을 불러 오고 데이터를 입력하는 모듈	
pandas	pip install pandas
데이터 분석과 처리를 하는 라이브러리	
PyAutoGui	pip install pyautogui
컴퓨터 입력 장치 마우스와 키보드의 입력을 제어할 수 있는 라이브러리	
PyPDF2	pip install pypdf2
pdf 파일 추출 병합하는 라이브러리	
requests	pip install requests
파이썬으로 HTTP 호출하는 라이브러리	
selenium	pip install selenium
Python 크롤링에 필요한 라이브러리	
pyinstaller	pip install pyinstaller
파이썬 실행 파일을 만드는 라이브러리	
win32com	pip install pywin32
Python을 이용하여 ActiveX 함수를 호출할 수 있도록 도와주는 라이브러리	
xlswriter	pip install XlsxWriter
엑셀 데이터 쓰기 라이브러리	
xlwings	pip install xlwings
엑셀 업무 자동화를 위한 라이브러리	

pip 라이브러리 업그레이드 pip install --upgrade pip



## CHAP 10. 파일 처리



## 1. 파일 쓰기

기억 데이터를 저장하고 불러오려면 텍스트 등 파일로 저장하고 읽어들이야 한다. 여러 가지 파일을 쓰고 읽는 방법을 배워보자.

### 1) 텍스트 파일

텍스트 파일을 작성하려면 open 함수를 써야 한다.

```
file = open("score.txt", "w", encoding="utf8")
file.write("국어 : 60")
file.write("영어 : 70\n")
print("수학 : 70", file=file)
print("전공 : 80", file=file)
file.write("수학 : 50")
file.close()
```

file\_1.py

print 함수나 write 메소드를 사용해야 한다. 줄을 바꾸려면 file.write("영어 : 70\n")와 같이 이스케이프 문자를 추가해야 한다.

파일을 읽어 보자.

```
file = open("score.txt", 'r', encoding="utf8")
str = file.read()
print(str)
file.close()
```

file\_2.py

'r' 은 읽기 모드, 'w'은 쓰기 모드이고 인코딩 종류는 "utf8"(유니코드)이다. 윈도우 10은 유니코드를 쓰기 때문에 지정하여 주는 것이 좋다.

### 2) 마크업

마크업이란 어떠한 표시(주로 <>)를 뜻하며 html 문서는 마크업으로 이루어진 언어의 한 종류다. 결국 XML은 정해진 태그 외에 확장해서 태그를 선언할 수 있으므로 문서 및 데이터의 구조를 표현할 수 있다는 것이다. XML은 텍스트 형태이므로 텍스트 파일 읽기 쓰기과 연관이 깊다.

## 2. 엑셀 파일

### 1) openpyxl 모듈

openpyxl은 엑셀 파일을 읽고 쓰기를 Python으로 할 수 있는 일종의 모듈이다. 예를 들어 아래와 같은 조작이 가능하다.

- 셀 번호를 지정해서 이미지나 문자열을 입력할 수 있다.
- Excel 파일의 시트 데이터를 복사해서 다른 시트에 붙여넣기 할 수 있다.
- 시트를 추가하거나 삭제할 수 있다.

다만 이 모듈을 사용하기 위해서 주의해야 할 점이 한 가지가 있는데 바로 확장자가 ".xlsx"여야 한다는 것이다.

파이썬은 무료 프로그램이므로 상용 프로그램인 MS 오피스나 한글 오피스를 다룰 수 있는 장점이 있다.

### 2) openpyxl

터미널에서 아래와 같이 입력함으로써 모듈을 설치할 수 있다.

```
pip install openpyxl
```

### 3) excel 파일 생성

sheet 이름이 "성적"인 ex\_data.xlsx 파일을 생성한다.

```
from openpyxl import Workbook
wb = Workbook() # 새 워크북 생성
ws = wb.active # 현재 활성화된 sheet 가져옴
ws.title = "성적" # sheet 의 이름을 변경
wb.save("ex_data.xlsx")
wb.close()
```

excel\_1.py

### 4) excel 파일 읽기

아래와 같은 xlsx 파일을 읽어보자.



	A	B	C	D	E	F	G	H
1	학번	이름	국어	영어	수학	과학	한국사	
2	3101	홍길동	72	86	90	70	68	
3	3102	김고은	64	88	82	76	84	
4	3103	신성일	48	58	60	56	46	

```
from openpyxl import load_workbook
wb = load_workbook("ex_data.xlsx")
ws = wb.active
for row in ws.iter_rows(min_row=1):
    for cell in row:
        print(cell.value, end=' ')
    print("")
```

excel\_2.py

```
from openpyxl import load_workbook
wb = load_workbook('./ex_file/sjdata.xlsx', data_only=True)
ws = wb['sung11']
for cols in ws.iter_cols():
    for cell in cols:
        print(cell.value, end='Wt ')
    print()
print()
for rows in ws.iter_rows():
    for cell in rows:
        print(cell.value, end='Wt ')
    print()
```

excel\_2.py

출력 양식을 매끄럽게 한 성적 출력 프로그램이다.

```
from openpyxl import load_workbook
wb = load_workbook("ex_data.xlsx", data_only=True)
ws = wb.active
cw = " "*30
tabno = [5, 9, 5, 5, 5, 5, 5] # 글자 크기 간격
print(" "*12, "파이썬 고등학교 성적")
print("-"*50)
for k, row in enumerate(ws.iter_rows(min_row=1)):
    for i, cell in enumerate(row):
        ct = len(str(cell.value).encode('euc-kr'))
        print(str(cell.value)+cw[:tabno[i]-ct], end=' ')
    print()
    if (k) % 5 == 0: # 5줄
        print("-"*50)
```

excel\_3.py

## 5) excel 파일 쓰기

```

from openpyxl import Workbook
wb=Workbook()
ws=wb.active
ws.title="성적"
ws.cell(row=1, column=1).value="파이썬"
ws['A2'].value=90
wb.save("s_data.xlsx")
wb.close()

```

excel\_4.py

엑셀 워크시트명을 "성적"으로 바꾸고 1행, 1열 셀은 "파이썬"으로 A2 셀은 90으로 쓰는 프로그램이다. ws.cell(row=1, column=1).value="파이썬"은 위치를 for 문 등으로 바꾸어 쓰기 편하다. ws['A2'].value=90 다음 줄에 아래 문장을 삽입하고 실행하여 보자.

```

for cnt in range(65,70):
    ws[chr(cnt)+"3"]= 100

```

chr(65)는 영문자 'A' 이므로 65, 66, 67, 68, 69는 각각 'A', 'B', 'C', 'D', 'E' 이므로 셀 주소는 'A3', 'B3', 'C3', 'D3', 'E3'이 된다.

위 프로그램을 실행할 때 당연히 엑셀 파일을 불러온 상태에서 실행하면 오류가 발생한다. 또한 가끔 한셀로 저장한 파일을 사용하려고 할 때도 오류가 발생하기도 하는데 가능한 Microsoft 엑셀을 사용하기로 하자.

## 6) 엑셀 파일에서 계산식 사용하기

openpyxl에서는 SUM이나 COUNT 같은 간단한 계산식을 사용할 수 있다. openpyxl에서 사용할 수 있는 계산식은 다음과 같다.

함수	설명
SUM(cell1:cell2)	셀 범위에 들어 있는 숫자의 합
PRODUCT(cell1:cell2)	셀 범위에 들어 있는 숫자의 곱
AVERAGE(cell1:cell2)	셀 범위에 들어 있는 숫자의 평균
QUOTIENT(num1,num2)	숫자 num1을 num2로 나눈 몫을 반환
MOD(num1,num2)	숫자 num1을 num2로 나눈 나머지를 반환
COUNT(cell1:cell2)	셀 범위에 들어 있는 숫자의 수

표 엑셀 파일 계산 함수

또한 파일을 불러오는 명령에서 data\_only=True을 조건으로 하는 이유는 문자가 출력되지 않고 계산 값이 출력되도록 하기 위해서이다.

간단하게 셀의 위치를 함수로 표현하면 다음과 같다.

cxy(행수, 열수)

cxy(1, 1)이면 1열 (전체행의수+1) 행이므로 샘플 파일은 전체 11줄이므로 'A12'이 된다. ws[cxy(1,1)].value="합계" 명령으로 A12 셀에 '합계'라는 문자를 입력한다.

엑셀 셀의 내용을 정렬하려고 한다면 alignment 메소드를 사용하면 된다. 우선 프로그램의 앞에 from openpyxl.styles import Alignment 명령을 추가하여 엑셀 양식 모듈을 추가한다. 그리고 정렬하고자 하는 셀에 추가하면 된다.

```
ws['A12']="합계"
```

```
ws['A12'].alignment = Alignment(horizontal='center', vertical='bottom')
```

```
from openpyxl import load_workbook
wb = load_workbook("ex_data.xlsx", data_only=True)
ws = wb.active
m_row = ws.max_row # 최대 행 수
m_col = ws.max_column # 최대 열 수
def cxy(x, y):
    cs = chr(64+x)+str(m_row+y)
    return cs
ws.merge_cells(cxy(1,1)+'':cxy(2,1))
ws.merge_cells(cxy(1,2)+'':cxy(2,2))
ws.merge_cells(cxy(1,3)+'':cxy(2,3))
ws[cxy(1,1)].value="합계"
ws[cxy(1,2)].value="응시"
ws[cxy(1,3)].value="평균"
for j in range(3, m_col+1):
    ws[cxy(j,1)].value='=SUM(' +chr(64+j)+'2'+':'+cxy(j,0)+' )'
    ws[cxy(j,2)].value='=COUNT(' +chr(64+j)+'2'+':'+cxy(j,0)+' )'
    ws[cxy(j,3)].value='=AVERAGE(' +chr(64+j)+'2'+':'+cxy(j,0)+' )'
wb.save("ex_data.xlsx")
wb.close()
```

excel\_5.py

horizontal은 수평 정렬, vertical은 수직정렬로 right, left, center, bottom, top의 속성값을 지정하면 된다. merge\_cells 지정한 셀범위를 병합하는 메소드이다.

사실 openpyxl의 스타일은 매우 복잡하고 사용하기 불편하다. 그렇지만 보고서는 단정한(?) 양식을 요구하기도 한다. 따라서 미리 엑셀 파일 양식을 만들어 놓고 셀에 값만 입력하므로서 해결하기도 한다. 우선 다음 인터넷을 참조하기를 바란다.

<https://openpyxl.readthedocs.io/en/stable/styles.html#cell-styles-and-named-styles>

다음 프로그램은 ex\_data.xlsx를 스타일을 설정한 '성적통지표.xlsx' 양식 파일을 불러서 개개인의 데이터를 입력한 후 imsi 폴더에 성적\_홍길동.xlsx 형식으로 저장하는 프로그램이다. 세세한 성적통지표의 양식이 지정되었을 것이다.

리스트에서 person[1]은 이름을 지정하므로 이름을 기준으로 여러 개의 파일이 만들어 졌을 것이다. imsi 폴더는 프로그램이 존재하는 폴더 하위 디렉터리에 미리 만들어 놓아야 한다. 절대 경로 지정과 상대 경로 지정을 참조하여 임의의 폴더에 저장할 수 있다.

```
from openpyxl import load_workbook
wb = load_workbook(filename = "ex_data.xlsx", data_only = True)
ws = wb[wb.sheetnames[0]]
tmp = []
for i in range(2, ws.max_row+1):
    tmp.append([])
    for cell in ws[i]:
        tmp[-1].append(cell.value)
for person in tmp:
    wb2 = load_workbook(filename = "성적통지표.xlsx")
    ws2 = wb2[wb2.sheetnames[0]]
    for i in range(1, len(person)+1):
        name=str(person[0])
        ws2['D4'] = name[:1]
        ws2['F4'] = name[1:2]
        ws2['H4'] = int(name[2:4])
        ws2['K4'] = person[1]
        ws2['A7'] = person[2]
        ws2['C7'] = person[3]
        ws2['E7'] = person[4]
        ws2['I7'] = person[5]
        ws2['K7'] = person[5]
    wb2.save("./imsi/성적_"+str(person[1])+".xlsx")
```

excel\_6.py

폴더 내의 다음과 같이 동일 양식의 여러개의 엑셀 파일이 있다면

	A	B	C
1	학번	이름	동아리명
2	1101	강원수	독서반
3	1102	김석훈	음악감상반
4	1103	김고은	과학반
5	1104	정은정	독서반
6	1105	박정태	영어회화반

+

	A	B	C
1	학번	이름	동아리명
2	1201	강호동	다큐멘터리반
3	1202	정석민	음악감상반
4	1203	김소영	과학반
5	1204	박윤기	독서반
6	1205	윤지민	컴퓨터반

.....=

	A	B	C
1	학번	이름	동아리명
2	1101	강원수	독서반
3	1102	김석훈	음악감상반
4	1103	김고은	과학반
5	1104	정은정	독서반
6	1105	박정태	영어회화반
7	1201	강호동	다큐멘터리반
8	1202	정석민	음악감상반
9	1203	김소영	과학반
10	1204	박윤기	독서반
11	1205	윤지민	컴퓨터반

다음 프로그램으로 보기와 같이 결합할 수 있을 것이다.

```
import os
from openpyxl import Workbook
from openpyxl import load_workbook
path='./test_folder'
list_dir = os.listdir(path)
wb2 = Workbook()
ws2 = wb2.active
cnt = 0
student = 0
for filename in list_dir:
    if ".xlsx" not in filename:
        continue
    wb = load_workbook(path+'/'+filename)
    ws = wb.active
    cnt += 1
    if cnt==1:
        num=1
    else:
        num=2
    for row in ws.iter_rows(min_row=num):
        data=[]
        student+=1
        for cell in row:
            data.append(cell.value)
        ws2.append(data)
print(cnt,"개의 파일",student-1,"명을 처리했습니다")
wb2.save("./out/total_collect.xlsx")
```

excel\_7.py

다시 아래 이 파일을 프로그램으로 동아리반 별로 분류할 수 있다. 1.1\_file7.py와 1.1\_file8.py는 프로그램 문법에 신경 쓰지 않아도 된다. 물론 원리를 알면 좋지만 단지 프로그램의 용도만 알고 활용만 하여도 될 것이다.

파이썬을 활용하여 RPA(Robotic Process Automation)을 할 수 있는 것이 큰 장점이다. 파이썬은 단순한 반복 작업을 프로그램 한번 실행만으로 복잡한 일을 처리할 수 있다. 물론 엑셀에서는 매크로나 vba로 위와 같은 일을 처리할 수 있지만 파이썬이 훨씬 편리하다.



```
from openpyxl import *
def categoryList(col : int)-> list:
    temp_list = [r[col].value for r in ws]
    del temp_list[0]
    temp_set = set(temp_list)
    name_list = list(temp_set)
    return name_list
def categoryData(col, name):
    total_list=[]
    for r in ws.rows:
        temp_list=[]
        cell_num = len(r)
        if r[col].value == name:
            for n in range(0,cell_num):
                temp_list.append(r[n].value)
        if temp_list != []:
            total_list.append(temp_list)
    return total_list
def writeExcel(name, total_list):
    ws = wb.create_sheet(name)
    i=1
    for data in total_list:
        data_length = len(data)
        for n in range(0, data_length):
            ws.cell(row=i, column=n+1).value = data[n]
        i=i+1
path = "./out/total_collect.xlsx"
wb = load_workbook(path)
ws = wb.active
pb = wb.sheetnames
col = 2 # 기준 열번호
category = categoryList(col)
for name in category:
    total_list = categoryData(col, name)
    writeExcel(name, total_list)
wb.save("./out/hr_result.xlsx")
```

excel\_8.py

openpyxl

엑셀 파일로 출력하기

```
df.to_excel("파일명.xlsx")
```

엑셀 파일로 출력하기(인덱스를 삭제)

```
df.to_excel("파일명.xlsx", index=False)
```

엑셀 파일로 출력하기(시트명을 지정)

```
df.to_excel("파일명.xlsx", sheet_name="시트명")
```

여러 개의 시트를 하나의 엑셀 파일로 출력하기

```
with pd.ExcelWriter("파일명.xlsx") as writer:
```

```
    df1.to_excel(writer, sheet_name="시트명 1")
```

```
    df2.to_excel(writer, sheet_name="시트명 2")
```

엑셀 파일 읽어 들이기

```
df = pd.read_excel("파일명.xlsx")
```

엑셀 파일 읽어 들이기(여러 개의 시트로부터)

```
df = pd.read_excel("파일명.xlsx", sheet_name="시트명")
```



### 3. Pandas

1) pandas 모듈 (pip install pandas)

데이터 조작 및 분석을 위한 Python 프로그래밍 언어용으로 작성된 소프트웨어 라이브러리이다. 특히 숫자 테이블과 시계열을 조작하기 위한 데이터 구조와 연산을 제공한다. 이 라이브러리는 터미널에서 pip install pandas로 설치해야 한다.

임포트하기 : import pandas as pd

(엑셀 파일인 경우 csv 대신 excel 파일명.xlsx)

csv 파일 읽어 들이기 : df = pd.read\_csv("파일명.csv")

csv 파일 읽어 들이기(헤더 없이 1행부터 데이터를 읽는다)

df = pd.read\_csv("파일명.csv", header=None)

csv 파일 읽어 들이기(3행째에 헤더가 있는 데이터를 읽는다)

df = pd.read\_csv("파일명.csv", header=2)

csv 파일 읽어 들이기(1열을 인덱스로 하여 읽는다)

df = pd.read\_csv("파일명.csv", index\_col=0)

csv 파일 읽어 들이기(인코딩이 euc-kr인 파일을 읽는다)

df = pd.read\_csv("파일명.csv", encoding="euc-kr")

ZIP으로 압축된 csv 파일 읽어 들이기(ZIP으로 압축된 채로 읽을 수 있다)

df = pd.read\_csv("파일명.zip")

csv 파일로 출력하기 : df.to\_csv("파일명.csv")

csv 파일로 출력하기(인덱스를 삭제) : df.to\_csv("파일명.csv", index=False)

csv 파일로 출력하기(인덱스와 헤더를 삭제)

df.to\_csv("파일명.csv", index=False, header=False)

DataFrame을 새로 만들기(빈 데이터) : df = pd.DataFrame()

DataFrame을 새로 만들기(2열, 2행 데이터)

df = pd.DataFrame({"열 1":["행 1","행 2"], "열 2":["행 1","행 2"]})

DataFrame의 칼럼(열명)을 구하기 : print(df.columns.values)

DataFrame의 인덱스를 구하기 : print(df.index.values)

한 열의 데이터 구하기 : print(df["열명"])

여러 열의 데이터 구하기 : print(df[["열명 1","열명 2"]])

한 행의 데이터 구하기 : df.loc[행 번호]

여러 행의 데이터 구하기 : df.loc[[행 번호1,행 번호2]]

하나의 데이터 구하기 : df.loc[행 번호][\"열명\"]

1행 추가하기 : df.loc[2] = [\"행 3A\",\"행 3B\"]

1열 추가하기 : df[\"열 C\"] = [\"행 1C\",\"행 2C\",\"행 3C\"]

지정한 열 삭제하기 : df = df.drop(\"열명\",axis=1)

지정한 행 삭제하기 : df = df.drop(행 번호,axis=0)

조건에 맞는 데이터 추출하기(90보다 큰 데이터를 추출) : `df = df[df["열명"] >= 90]`

집계(최댓값, 최솟값, 평균값, 중앙값, 합계 등)를 한 결과 구하기

```
df["열명"].max(), df["열명"].min(), df["열명"].mean(), df["열명"].median(),
df["열명"].sum()
```

데이터 정렬하기(오름차순: 작은 값이 먼저 나옴) : `df = df.sort_values("열명")`

데이터 정렬하기(내림차순: 큰 값이 먼저 나옴) :

```
df = df.sort_values("열명",ascending=False)
```

여러 열의 집계 결과를 새로운 열에 넣기 : `df["새 열"] = df["열 1"] - df["열 2"]`

행과 열을 바꾸기 : `df.T`

데이터를 파이썬 리스트로 만들기 : `df.values`

```
import pandas as pd
dict_data = {'a': 1, 'b': 2, 'c': 3}
sr = pd.Series(dict_data)
print(type(sr))
print('\n')
print(sr)
```

pandas\_1.py

```
import pandas as pd
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9],
             'c3':[10,11,12], 'c4':[13,14,15]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
print('\n')
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf = df.reindex(new_index)
print(ndf)
print('\n')
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf2 = df.reindex(new_index, fill_value=0)
print(ndf2)
```

pandas\_2.py

```
import pandas as pd
temp = pd.Series([-20, -10, 10, 20])
print(temp)
temp = pd.Series([-20, -10, 10, 20], index=['Jan', 'Feb', 'Mar', 'Apr'])
print(temp)
```

pandas\_3.py

```
import pandas as pd
data = {
    '이름' : ['최민식', '안성기', '김혜수', '송강호', '전지현', '장동건',
    '설경구', '엄정화'],
    '성별' : ['남', '남', '여', '남', '여', '남', '남', '여'],
    '나이' : [59, 70, 51, 54, 40, 49, 54, 52],
    '키' : [177, 175, 170, 180, 173, 181, 178, 164],
    '몸무게' : [70, 72, 50, 80, 52, 68, 70, 47],
    '데뷔' : [1990, 1957, 1985, 1991, 1997, 1992, 1996, 1992],
    '영화편수' : [33, 71, 39, 36, 12, 21, 41, 24],
    '학력' : ['학사', '학사', '석사', '대학중퇴', '학사', '학사', '학사', '고졸']
}
print(data)
df = pd.DataFrame(data)
print(df)
print(df['이름'])
print(df['키'])
print(df[['이름', '키']])
list=[no for no in range(1,9)]
print(list_no)
df = pd.DataFrame(data, columns=['이름', '성별', '키'], index=list)
print(df)
```

pandas\_4.py

```
import pandas as pd
data = {
    '이름' : ['최민식', '안성기', '김혜수', '송강호', '전지현', '장동건',
             '설경구', '엄정화'],
    '성별' : ['남', '남', '여', '남', '여', '남', '남', '여'],
    '나이' : [59, 70, 51, 54, 40, 49, 54, 52],
    '키' : [177, 175, 170, 180, 173, 181, 178, 164],
    '몸무게' : [70, 72, 50, 80, 52, 68, 70, 47],
    '데뷔' : [1990, 1957, 1985, 1991, 1997, 1992, 1996, 1992],
    '영화편수' : [33, 71, 39, 36, 12, 21, 41, 24],
    '학력' : ['학사', '학사', '석사', '대학중퇴', '학사', '학사', '학사', '고졸']
}
list=[no for no in range(1,9)]
df = pd.DataFrame(data, index=list)
df.index.name="순번"
print(df)
print(df.reset_index())
df2=df.set_index('이름')
print(df2)
print(df2.sort_index())
print(df2.sort_index(ascending=False))
```

pandas\_5.py

```
import pandas as pd
# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'이름' : ['서준', '우현', '인아'],
              '수학' : [ 90, 80, 70],
              '영어' : [ 98, 89, 95],
              '음악' : [ 85, 95, 100],
              '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)
print(df)
print('Wh')
df = df.transpose()
print(df)
print('Wh')
df = df.T
print(df)
```

pandas\_6.py

```
import pandas as pd
import openpyxl
data = {
    '이름' : ['최민식', '안성기', '김혜수', '송강호', '전지현', '장동건',
             '설경구', '엄정화'],
    '성별' : ['남', '남', '여', '남', '여', '남', '남', '여'],
    '나이' : [59, 70, 51, 54, 40, 49, 54, 52],
    '키' : [177, 175, 170, 180, 173, 181, 178, 164],
    '몸무게' : [70, 72, 50, 80, 52, 68, 70, 47],
    '데뷔' : [1990, 1957, 1985, 1991, 1997, 1992, 1996, 1992],
    '영화편수' : [33, 71, 39, 36, 12, 21, 41, 24],
    '학력' : ['학사', '학사', '석사', '대학중퇴', '학사', '학사', '학사', '고졸']
}
list=[no for no in range(1,9)]
df = pd.DataFrame(data, index=list)
df.index.name="순번"
df.to_csv('movie.csv', encoding='euc-kr')
df.to_csv('movie.txt', sep='Wt')
df.to_excel('movie.xlsx', sheet_name="movie", engine='openpyxl')
df.to_excel('movie1.xlsx', sheet_name="movie",
            index=False, engine='openpyxl')
```

pandas\_7.py

```
import pandas as pd
df = pd.read_excel('movie.xlsx', index_col='순번')
print(df)
print(df.describe())
print(df.info())
print(df.head())
print(df.head(4))
print(df.tail())
print(df.tail(3))
print(df.values)
print(df.columns)
print(df.index)
print(df.shape)
print(df.values[3])
print(df.values[3][0])
print(df['키'])
print(df['키'].describe())
print(df['키'].min())
print(df['키'].max())
print(df['키'].nlargest())
print(df['키'].nlargest(3))
print(df['키'].mean())
print(df['키'].sum())
print(df['학력'].count())
print(df['학력'].unique())
print(df['학력'].nunique())
```

pandas\_8.py

```
import pandas as pd
df = pd.read_excel('movie.xlsx', index_col='순번')
print(df.columns)
print(df.columns[0])
print(df.columns[2])
print(df[df.columns[-1]])
print(df['데뷔'][0:5])
print(df[['이름', '키']][:3])
print(df[3:])
```

pandas\_9.py



```
import pandas as pd
df = pd.read_excel('movie.xlsx', index_col='순번')
print(df.loc[1])
print(df.loc[5])
print(df.loc[1, '몸무게'])
print(df.loc[[1, 2], '몸무게'])
print(df.loc[[1, 2], ['이름', '키']])
print(df.loc[1:5, '이름':'나이'])
print(df.iloc[0])
print(df.iloc[4])
print(df.iloc[0:5])
print(df.iloc[0, 1])
print(df.iloc[4, 3])
print(df.iloc[[0, 1], 0])
print(df.iloc[[0, 1], [3, 4]])
print(df.iloc[0:5, 3:8])
print(df.iloc[0:4])
```

pandas\_10.py

```
import pandas as pd
df1 = pd.read_excel('학생시험성적0.xlsx', sheet_name = '1차시험',
    index_col='학생')
df2 = pd.read_excel('./data/학생시험성적1.xlsx', sheet_name = '2차시험',
    index_col='학생')
excel_writer3 = pd.ExcelWriter('학생시험성적4.xlsx', engine='openpyxl')
df1.to_excel(excel_writer3, index=False, sheet_name='중간고사')
df2.to_excel(excel_writer3, index=False, sheet_name='기말고사')
df3 = pd.concat([df1, df2], axis=1)
del df3['평균']
df3['국어평균']=(df1['국어']+df2['국어'])/2
df3['영어평균']=(df1['영어']+df2['영어'])/2
df3['수학평균']=(df1['수학']+df2['수학'])/2
df3['평균합계']=df3['국어평균'] + df3['영어평균'] + df3['수학평균']
df3['순위'] = df3['평균합계'].rank(ascending=False)
df3.to_excel(excel_writer3, index='학생', sheet_name='총합')
excel_writer3.save()
```

pandas\_11.py

## 역대 박스 오피스

<https://www.kobis.or.kr/kobis/business/stat/boxes/findFormerBoxOfficeList.do>

KOBIS\_역대\_박스오피스(통합전산망\_집계\_기준)\_2021-12-26.xlsx을 kovis.xlsx로 바꾸고 실행해봄

순위	영화명	개봉일	해출액	관객수	스크린수	대표국적	국적	배급사
1	명량	2014-07-30	135,748,398,910	17,613,682	1,587	한국	한국	(주)씨제이엔터테인먼트
2	극한직업	2019-01-23	139,647,979,516	16,264,944	1,970	한국	한국	(주)씨제이엔터테인먼트
3	신과함께-죄와 벌	2017-12-20	115,698,854,137	14,410,754	1,912	한국	한국	롯데엔터테인먼트
4	국제시장	2014-12-17	110,913,409,630	14,257,115	980	한국	한국	(주)씨제이엔터테인먼트
5	아벤저스 엔드게임	2019-04-24	122,182,694,160	13,934,592	2,835	미국	미국	윌트디즈니컴퍼니코리아 유한책임회사
6	겨울왕국 2	2019-11-21	114,810,421,450	13,747,792	2,648	미국	미국	윌트디즈니컴퍼니코리아 유한책임회사
7	아바타	2009-12-17	128,447,097,523	13,624,328	912	미국	미국	주식회사 테라온엔터테인먼트, 이십세기폭스코리아(주)
8	배트맨	2015-06-05	105,168,155,250	13,414,009	1,064	한국	한국	(주)씨제이엔터테인먼트
9	리틀	2006-07-27	0	13,019,740	167	한국	한국	(주)쇼박스
10	도둑들	2012-07-25	93,065,508,500	12,983,330	1,072	한국	한국, 홍콩	(주)쇼박스
11	7번방의 선물	2013-01-23	91,431,914,670	12,811,206	787	한국	한국	(주)엑스엔터테인먼트, 뉴트럴(NUEW)
12	완벽한	2015-07-22	98,463,132,781	12,705,700	1,519	한국	한국	(주)쇼박스
13	달리던	2018-05-23	106,983,620,359	12,555,894	1,311	미국	미국	윌트디즈니컴퍼니코리아 유한책임회사
14	왕의 남자	2012-09-13	88,900,208,769	12,319,542	810	한국	한국	(주)씨제이엔터테인먼트
15	왕의 남자	2005-12-20	0	12,302,831	94	한국	한국	(주)시네마서비스
16	신과함께-망고 연	2018-08-01	102,660,146,909	12,274,996	2,235	한국	한국	롯데엔터테인먼트, 롯데엔터테인먼트
17	역시올정사	2017-06-02	95,855,737,149	12,188,684	1,806	한국	한국	(주)쇼박스
18	태극기 휘날리며	2004-02-05	0	11,746,135	110	한국	한국	(주)다자소프트(주)쇼박스
19	부산행	2016-07-20	93,179,283,048	11,565,479	1,788	한국	한국	(주)엑스엔터테인먼트, 뉴트럴(NUEW)
20	해운대	2009-07-22	81,934,638,201	11,453,338	753	한국	한국	CI ENM

## 참고

크롤링 연습 사이트

<https://www.pythonscraping.com/pages/page3.html>

```

import pandas as pd
import openpyxl
# list_no = [num for num in range(1,201)]
#print(list_no)
df = pd.read_excel('kobis.xlsx', header=6, index_col='순위', usecols='A:H')
yoil=['월', '화', '수', '목', '금', '토', '일']
def yoilm(nal):
    return yoil[nal]+'요일'
print(df)
print("="*100)
print('관객수 평균 : ',df['관객수'].mean(), '관객수 합계 : ', df['관객수'].sum())
print("매출액 평균 : ",df['매출액'].mean(), "매출액 합계 : ", df['매출액'].sum())
print("스크린수평균:", df['스크린수'].mean(),"상영횟수평균:", df['상영횟수'].mean())
print("="*100)
print(df.groupby('대표국적').sum())
print("="*100)
print(df.groupby('대표국적').count())
print("="*100)
pd.to_datetime(df['개봉일'])
df['연도']=df['개봉일'].dt.year
print(df.groupby('연도').count())
print("="*100)
df['월'] = df['개봉일'].dt.month #월
print(df.groupby('월').count())
# df['day'] = df['개봉일'].dt.day #일
print("="*100)
df['요일'] = df['개봉일'].dt.dayofweek
df['요일'] = df['요일'].apply(yoilm)
print(df.groupby('요일').count())
print("="*100)
print(df)
df.to_csv('boxoffice.csv', encoding="utf-8-sig")
df.to_excel('boxoffice.xlsx', engine="openpyxl")

```

pandas\_9.py

xlsxwriter는 openpyxl과 같이 많이 사용된다. 같은 작업을 했을 때 실행 속도가 openpyxl 보다 빠르다. 단 xlsxwriter는 엑셀 파일(.xlsx)을 불러오지 못하는 치명적인 단점이 있다. 따라서 pandas나 openpyxl과 같이 혼합하여 사용한다.

## 패키지 활용하기

```
import xlswriter
패키지 불러오기
wb = xlswriter.Workbook("exam.xlsx")
```

```
import xlswriter as xlw
패키지 이름 변경하기
wb = xlw.Workbook("exam.xlsx")
```

```
from xlswriter import Workbook
특정 패키지 불러오기
wb = Workbook("exam.xlsx")
```

```
from xlswriter import *
모든 패키지 불러오기
wb = Workbook("exam.xlsx")
```

### win32com 모듈 설치하기

파이썬을 통해서 윈도우 상의 응용프로그램들을 제어할 수 있는 라이브러리이다.

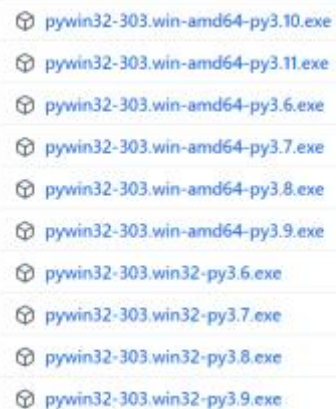
<https://github.com/mhammond/pywin32/releases>

64 bit

amd64-py버전

32bit

win32-py버전



pywin32-303.win-amd64-py3.10.exe
pywin32-303.win-amd64-py3.11.exe
pywin32-303.win-amd64-py3.6.exe
pywin32-303.win-amd64-py3.7.exe
pywin32-303.win-amd64-py3.8.exe
pywin32-303.win-amd64-py3.9.exe
pywin32-303.win32-py3.6.exe
pywin32-303.win32-py3.7.exe
pywin32-303.win32-py3.8.exe
pywin32-303.win32-py3.9.exe

# CHAP 11. 크롤링



## 1. 정적 웹크롤링

request / BeautifulSoup

이번 시간에는 정적 웹페이지의 데이터를 가져오는 '정적 웹크롤링'에 대해 배워보자.

정적 웹 크롤링은 아주 간단합니다.

1단계. 원하는 웹 페이지의 html문서를 싹 긁어온다.

2단계. 긁어온 html 문서를 파싱(Parsing)한다.

3단계. 파싱한 html 문서에서 원하는 것을 골라서 사용한다.

위의 3단계를 수행하기 위해 사용되는 패키지를 하나씩 살펴보겠습니다.

### 1. 정적 웹크롤링 관련 패키지

#### 1단계. requests

html 문서를 가져올 때 사용하는 패키지이다. requests는 사용자 친화적인 문법을 사용하여 다루기 쉬우면서 안정성이 뛰어나다고 한다.

그래서 파이썬 기본 라이브러리에 포함된 urllib 패키지보다 자주 사용된다.

설치 방법

VS code의 터미널창에서 아래와 같이 입력하면 된다.

만약 anaconda를 사용하신다면 pip 대신 conda를 입력해도 된다.

```
pip install requests
```

```
# requests 패키지 가져오기
import requests
# 가져올 url 문자열로 입력
url = 'https://www.naver.com'
# requests의 get함수를 이용해 해당 url로 부터 html이 담긴 자료를 받아옴
response = requests.get(url)
# 우리가 얻고자 하는 html 문서가 여기에 담기게 됨
html_text = response.text
```

## R.text와 R.content

### ① 개요

Python3에서 불과 3~4년 전만 하더라도 urllib를 사용하여 HTTP 자원에 접근하는 코드가 많았지만, 이제는 거의다 requests 모듈을 사용하는 추세이다. 그만큼 requests 모듈의 편리성과 확장성이 Pythonic하다고 볼 수 있다. requests 모듈에서 HTTP Response에 접근하는 방법은 크게 두가지가 있다. 하나는 r.text이고 하나는 r.content이다.

※ 이 글에서 설명하는 내용은 다음과 같은 코드가 실행된 상태라고 가정한다.

```
import requests
```

```
url = 'https://www.example.com/'
```

```
r = requests.get(url)
```

### ② r.text

r.text의 경우엔 HTTP Request를 보낸 URL에서 readable한 내용을 가져올 때 사용한다. r.text를 사용하는 경우에 r.encoding 이라는 리소스를 사용할 수가 있는데, 이는 requests 모듈 자체에서 Request를 바탕으로 받은 Response를 자동으로 encoding을 추측하고 해당 인코딩 값으로 디코드시킨 값이 바로 r.text이다. 따라서 r.text의 경우 이미 requests 모듈내에서 한 번 가공된 값이라고 볼 수 있다.

### ③ r.content

r.content의 경우에도 gzip이나 deflate 알고리즘에 의해 압축된 HTTP Response를 unzip했다는 점에서 가공되었다고 볼 수 있다. 하지만 r.text는 r.encoding으로 디코드되었지만 r.content는 디코드되지 않은 점에서 차이가 있다. 결론적으로 r.content에는 HTTP Response의 body 부분이 bytes 상태로 그대로 저장되어 있다. 따라서 이미지 등의 미디어 파일을 다른 서버로부터 받고 싶을 때는 r.content를 사용해야 한다. r.text의 경우엔 디코드된 상태이기 때문에 절대로 원본 파일일 수가 없다.

### ④ r.content를 사용해야 하는 경우

r.text의 경우 문서의 인코딩 방식을 자동으로 추측하고 그에 맞게 디코딩해준다는 점에서는 굉장히 편리하다. 그렇지만 다음과 같은 경우를 보면 인코딩 에러가 나타난다.

```
# coding: utf-8
import requests
from bs4 import BeautifulSoup

def main():
    url = 'https://docs.python.org/ko/3/library/weakref.html'
    r = requests.get(url)
    soup = BeautifulSoup(r.text, 'html.parser')
    print(r.encoding)
    print(soup.prettify())

if __name__ == '__main__':
    main()
```

만약 `https://docs.python.org` 만을 파싱하는 경우라고 한다면 `r.encoding = 'utf-8'` 을 사용하여 해결할 수 있겠으나, 만약 `docs.python.org` 외의 다른 웹사이트도 파싱하는 경우라면? `r.encoding = 'utf-8'` 을 사용하여 `r.text`를 가져오는 경우는 너무나 비범용적이다. 따라서 이 경우 `r.content`를 사용하면 한글 데이터를 보존할 수 있게 된다.

## 2단계. BeautifulSoup4

BeautifulSoup4 패키지는 매우 길고 정신없는 html 문서를 잘 정리되고 다루기 쉬운 형태로 만들어 원하는 것만 쓱쓱 가져올 때 사용한다.

이 작업을 파싱(Parsing)이라고도 부른다.

### 설치 방법

마찬가지로 VS code의 터미널창에서 아래와 같이 입력하면 된다.

```
pip install beautifulsoup4
```

### 사용 방법

방금 전 Requests 패키지로 받아온 html 문서를 파싱해야하므로, 이전 코드 블록을 실행해야 한다.



```
# BeautifulSoup 패키지 불러오기
# 주로 bs로 이름을 간단히 만들어서 사용함
from bs4 import BeautifulSoup as bs
# html을 잘 정리된 형태로 변환
html = bs(html_text, 'html.parser')
```

### 3단계. BeautifulSoup4 (find /select)

이제 마지막 3단계이다.

사실 1, 2단계는 느끼셨겠지만 url만 수정하면 아주 쉽게 응용할 수 있을 정도로 쉽다. 즉, 정적 웹 크롤링의 핵심은 필요한 정보의 위치와 구조를 파악해서 쓱쓱 가져오는 3단계이다. 크롤링을 원하는 사이트가 생겼을 때에 능수능란하게 코딩하기 위해서는 꼭 이 스킬을 확실히 익혀야 한다.

#### 1) find( ), find\_all( ) 함수

BeautifulSoup 패키지에서 find 함수는 아래와 같이 12개가 있다.

하지만 우리가 알면 되는 것은 딱 2개이다. - find( ): 하나만 찾는 것 - find\_all( ): 모두 다 찾는 것이다. 주의할 것은 find로 찾는 것이 여러 개 라면 가장 첫 번째 것만 가져온다는 것이다.

```
find
find_all
find_all_next
find_all_previous
find_next
find_next_sibling
find_next_siblings
find_parent
find_parents
find_previous
find_previous_sibling
find_previous_siblings
```

괄호( ) 안에는 html의 태그(tag)나 속성(attribute)이 들어간다. 웹 크롤링 관련 코드를 보다보면 find 함수를 종종 사용한 경우를 보는데, 그 코드를 이해할 목적으로만 가볍게 알고 넘어가시는 것을 추천한다. 왜냐하면 CSS 선택자 개념을 사용하는 select( ) 함수를 사용하는 것이 훨씬 직관적이기 때문이다.

```
# 목표 태그 예)
<p class = "para">코딩유치원</p>
<div id = "zara">코딩유치원</div>
# 태그 이름으로 찾기
soup.find('p')
# 태그 속성(class)으로 찾기 - 2가지 형식
```

soup.find(class\_='para') #이 형식을 사용할 때는 class 다음에 언더바\_를 꼭 붙여주어야 한다.

```
soup.find(attrs = {'class':'para'})
# 태그 속성(id)으로 찾기 - 2가지 형식
soup.find(id='zara')
soup.find(attrs = {'id':'zara'})
# 태그 이름과 속성으로 찾기
soup.find('p', class_='para')
soup.find('div', {'id' = 'zara'})
```

## 2) select( ), select\_one( ) 함수

select( )는 find\_all( )과 같은 개념이고, select\_one( )은 find( )와 같은 개념이다. 다만, select( ) 함수는 괄호( )안에 CSS 선택자라는 것을 넣어서 원하는 정보를 찾는 것이 특징이다.

```
# a태그의 class 속성명이 news_tit인 태그
soup.select_one('a.news_tit')
soup.select('a.news_tit')
for i in titles:
    title = i.get_text() print(title)
```

```
ex = '''
<html>
  <head>
    <title>html연습</title>
  </head>
  <body>
    
    <h1>시장가서 사야할 과일 목록
      <div> <p id='fruits1' class='name1' title='바나나'>바나나
        <span class = 'price'>3000원</span>
        <span class = 'count'>10개</span>
        <span class = 'store'>바나나가게</span>
        <a href =
'https://www.banana.com'>banana.com</a>
      </p>
      </div>
      <div> <p id='fruits2' class='name2' title='체리'>체리
        <span class = 'price'>100원</span>
        <span class = 'count'>50개</span>
        <span class = 'store'>체리가게</span>
        <a href =
'https://www.cherry.com'>banana.com</a>
      </p>
      </div>
      <div> <p id='fruits3' class='name3' title='오렌지'>오렌지
        <span class = 'price'>500원</span>
        <span class = 'count'>20개</span>
        <span class = 'store'>오렌지가게</span>
        <a href =
'https://www.orange.com'>orange.com</a>
      </p>
      </div>
    </body>
  </html>'''
```

1) find( ) 함수 - 주어진 조건을 만족하는 첫 번째 태그 값만 가져오기

```
ex) soup.find('p')
    soup.find('p', align='center')
```

2) find\_all() 함수 - 해당 태그가 여러개 있을 때 한꺼번에 모두 가져오기

```
ex) soup.find_all('p')
    soup.find(['p', 'img'])
```

3) select() 함수 사용하기

```
soup = BeautifulSoup(ex, 'html.parser')
```

(1) select('태그이름')

```
ex) soup.select('p')
```

(2) select('.클래스명')

```
ex) soup.select('.name1')
```

(3) select('상위태그 > 하위태그 > 하위태그')

```
ex) soup.select('div > p > span')
```

위 명령은 여러 태그가 단계적으로 있을 때 아주 요긴하게 사용하는 방법이다. 부등호 앞 뒤로 공백이 반드시 들어가야 한다는 점 주의해야 한다.

그런데 위와 같이 특정 태그가 여러 개 있는 경우 한꺼번에 모두 나와서 보기가 번거롭다? 이때는 앞의 과정 중 리스트에 배웠던 인덱싱 관련 기능을 함께 사용하면 된다.

아래 명령을 보자.

```
ex) soup.select('div > p > span')[0]
    soup.select('div > p > span')[0]
```

(4) select('상위태그.클래스명 > 하위태그.클래스명')

```
ex) soup.select('p.name1 > span.store')
```

(5) select('#아이디명')

```
ex) soup.select('#fruits1')
```

(6) select('#아이디명 > 태그명.클래스명')

```
ex) soup.select('#fruits1 > span.store')
```

(7) select('태그명[속성1=값1]')

```
ex) soup.select('a[href]')
```

```
soup.select('a[href]')[0]
```

ind( ) 함수와 find\_all( ) 함수와 select( ) 함수를 이용해서 태그를 찾아냈다. 그런데 정작 우리에게 필요한 것은 화면에 보여지는 텍스트 내용이 가장 중요하다. 태그를 찾았지만 태그의 콘텐츠(문장이나 이미지 등)를 가져오지 못하면 무용지물이다. 그래서 지금부터 태그내의 문장을 가져오는 방법을 알아보도록 하자.

```
txt = soup.fond('p')
print(txt.string)
```

위 그림을 보면 p태그가 실제로는 3개가 존재하지만 find( ) 함수를 사용하면 제일 먼저 나오는 p 태그를 가져온다는 사실 기억하는가?

그래서 1개만 가져와서 txt 변수에 담았다. 그리고 p태그를 찾은 후에 객체에서 string을 가져 왔다. string은 태그의 문장을 가지고 있다.

태그에 포함된 문장만을 가지고 올 때 자주 사용되니까 잘 기억해 두자. 그런데 위의 string을 사용하면 한 번에 한 문장 밖에 가져오지 못한다.

그렇다면 태그 안에 존재하는 여러 개의 문장을 한꺼번에 가져오려면 어떻게 해야 할까? 바로 아래와 같은 방법을 사용하시면 된다.

```
ex) txt2 = soup.find_all('p')
    for i in txt2:
        print(i.string)
```

이번에는 get\_text( ) 함수를 사용하는 방법을 알아보자.

이 함수도 태그 내의 텍스트 데이터를 추출할 때 아주 많이 사용한다.

```
ex) txt2 = soup.find_all('p')
    for i in txt2:
        print(i.get_text())
```

## 크롤링 주요 라이브러리

### requests

인터넷상의 페이지 정보, 이미지 파일 등을 구할 수 있는 라이브러

설치 방법 : pip install requests

임포트하기 : import requests

웹 페이지 구하기 : response = requests.get(URL)

페이지의 글자가 깨지지 않도록 하기 :

```
response.encoding = response.apparent_encoding
```

구한 페이지의 문자 데이터 구하기 : `response.text`

구한 페이지의 바이너리 구하기 : `response.content`

구한 페이지의 URL 구하기 : `response.url`

구한 페이지의 스테이터스 코드 구하기 : `response.status_code`

구한 페이지의 리스폰스 헤더 구하기 : `response.headers`

## BeautifulSoup

HTML을 해석해 스크래핑할 수 있는 라이브러리

설치 방법 : `pip install beautifulsoup4`

임포트하기 : `from bs4 import BeautifulSoup`

HTML 해석하기 : `soup = BeautifulSoup(html.content, "html.parser")`

태그를 찾아 요소 꺼내기 : `soup.find("태그명")`

모든 태그를 찾아 요소를 리스트로 꺼내기 : `soup.find_all("태그명")`

id로 찾아 요소 꺼내기 : `soup.find(id="id명")`

class로 찾아 요소 꺼내기 : `soup.find(class_="class명")`

요소의 속성값 꺼내기 : `요소.get("속성명")`

## urllib

URL에 액세스하거나 URL을 처리할 수 있는 라이브러리

설치 필요 없음(표준 라이브러리)

임포트하기 : `import urllib`

상대 URL을 절대 URL로 변환하기 : `parse.urljoin(베이스 URL, 조사할 URL)`

테스트로 실습해 보자. 웹에서는 `url = "https://site "` 로 주소를 지정하고 `html2 = requests.get(url).text` 로 텍스트로 불러와서 클로링 하면 된다.

```
from bs4 import BeautifulSoup
# 테스트용 HTML 코드
html2 = """
<html>
  <head>
    <title>작품과 작가 모음</title>
  </head>
  <body>
    <h1>책 정보</h1>
    <p id="book_title">토지</p>
    <p id="author">박경리</p>
    <p id="book_title">태백산맥</p>
    <p id="author">조정래</p>
    <p id="book_title">감옥으로부터의 사색</p>
    <p id="author">신영복</p>
  </body>
</html>
"""

# url = "https://site "
# html2 = requests.get(url).text
soup2 = BeautifulSoup(html2, "lxml")
print('===== 타이틀 =====')
print(soup2.title)
print('===== 본문 =====')
print(soup2.body)
print(soup2.body.h1)
print('===== p 태그 =====')
print(soup2.find_all('p'))
print('===== p 태그 id: "book_title" 첫번째 =====')
print(soup2.find('p', {"id": "book_title"}))
print('===== p 태그 id: "book_title" 모두 =====')
book_titles = soup2.find_all('p', {"id": "book_title"})
print(book_titles)
print('===== p 태그 id: "author" 모두 ===== ')
print(soup2.find_all('p', {"id": "author"}))
print('===== 전체 인쇄 =====')
authors = soup2.find_all('p', {"id": "author"})
for book_title, author in zip(book_titles, authors):
    print(book_title.get_text() + ' / ' + author.get_text())
```

parsing\_1.py

```

name = input('BeautifulSoup 교재 _ 검색 항목 : ')
from bs4 import BeautifulSoup as bs
import requests
url = 'https://search.naver.com/search.naver?where=news&sm=tab_jum&query='+name
response = requests.get(url)
html_text = response.text
html = bs(html_text, 'html.parser')

a = html.select("p") # 태그명으로 검색

# 속성값(class name)으로 검색 <a class="news_tit">
b = html.select(".news_tit")
# print(b)
print('=' * 100)
for tb in b:
    print(tb.text)

# 태그와 속성값(class name)을 조합 하여 검색 <div class="news_cluster">
d = html.select("div.news_cluster")
# print(d)
print('=' * 100)
for td in d:
    print(td.text)

# 태그와 속성값(class name)을 조합 하여 검색 <a class="news_tit">
e = html.select("a.news_tit")
print('=' * 100)
for te in e:
    print(te.text)

f = html.select("a.news_tit")
print('=' * 100)
for tf in f:
    title = tf.get_text()
    print(title)

g = html.select("a.news_tit")
for tg in g:
    href = tg.attrs['href']
    print(href)

```

parsing\_2.py



## 2. 동적 웹크롤링 관련 패키지

### selenium

#### 1.1 동적 웹 페이지 클로링

다양한 사용자와 상호 작용하며 콘텐츠 내용이 계속 바뀌는 웹 페이지를 동적 웹 페이지라하며 selenium과 chrome 드라이버를 사용한다. 라이브러리를 설치하기 위해서 터미널 창에서 다음과 같이 입력한다.

```
pip install selenium
```

#### 1.2 사용 방법

프로그램에서 라이브러리를 불러오기 위해 다음과 같이 코딩한다.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time
```

라이브러리를 불러오고 크롬드라이버를 실행해야 한다.

```
driver = webdriver.Chrome( )
```

최신 버전에서는 직접 드라이버를 호출할 필요가 없어 크롬드라이버의 위치를 지정할 필요가 없다. 단, 크롬 브라우저는 최신 버전이 설치되어야한다. 항상 크롬 브라우저가 최신의 버전으로 업그레이드 될 수 있도록 설정하여야 한다. 자주 최신 버전으로 바뀌므로 브라우저 설정 - Chrome에서 최신 버전인지 확인해야 한다.

#### 1.3 페이지 이동

크롬 드라이버를 실행하고 원하는 페이지로 이동하기 위해서는 driver.get 함수를 사용한다.

```
driver.get('http://news.naver.com/')
time(3)
```

time 함수는 원하는 페이지를 불러오기 위해서는 시간이 걸리는데 시간 여유를 주기 위해서이다.

## 1.4 원하는 HTML 인덱싱하기

HTML을 인덱싱하기 위해서는 find\_element와 find\_elements 함수를 사용해야 한다.

driver.find\_element(<로케이터>, '내용')

로케이터	설명
By.TAG_NAME	<p>, <div> 등과 같은 태그명으로 검색
By.ID	태그의 id 속성값을 이용해서 검색
By.CLASS_NAME	태그의 클래스 속성값을 이용하여 검색
By.CSS_SELECTOR	css 선택자로 검색
By.XPATH	태그의 경로로 검색
By.NAME	태그의 name 값으로 검색
By.LINK_TEXT	링크 텍스트 값으로 검색
By.PARTIAL_LINK_TEXT	링크 텍스트의 자식 텍스트 값으로 검색

<p class="title">HTML 구조</p>

태그명 : p

속성명 : class

속성값 : "title", HTML 구조

ex) <div id="header">

## 1.5 자료 추출 구분 문자열 확인

크롬 브라우저의 우측 개발자 도구(단축키 F12) 커서 모양의 아이콘 클릭

찾고자 하는 대상 클릭

왼쪽 ...을 클릭 메뉴에서 copy를 선택 필요한 것을 선택한후 Ctrl+V로 확인

## CHAP 12. RPA



## RPA

Robotic Process Automation의 약자. 현재는 데이터 입력등의 단순 반복 업무 프로세스의 자동화에 주로 적용되고 있지만, 향후 AI, 머신러닝등의 기술이 발전, RPA와 결합한다면 자료 분석 및 Solution 제시 등의 영역까지도 가능(?)할 수 있다.

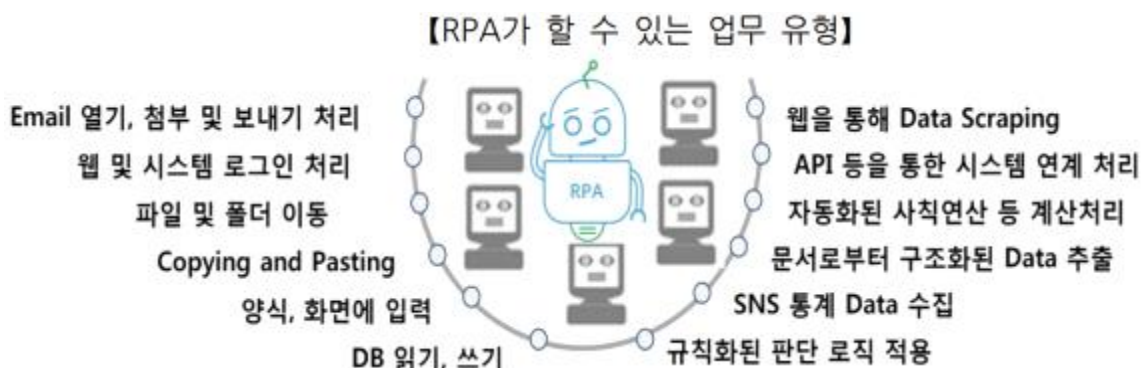
기존의 Robot이 공장 생산 라인의 실제적 기계였다면 RPA는 Software로 구현된 사무직을 위한 Robot으로 이해하면 쉽다.

RPA는 작업 시간을 다음의 예에서 보듯이 엄청나게 단축하여 준다.



주 52시간 근무제 도입으로 인력 운영의 어려움이 가중되며 RPA 관심이 증가했다.

- RPA(Robotic Process Automation)는 단순하고 반복적인 사무업무를 Software 프로그램으로 자동화하는 것이다
- 국내에서도 2016년 이후 은행, 보험, 카드사 등 금융권의 계약관리, 보험증권 처리, 정보조회 등 Back-Office 업무 중심으로 RPA 도입 시작했다.
- '17년 이후 주 52시간 근무제 도입이 가시화되며 제조, 물류, 공공 등 저 산업 영역에서의 RPA에 대한 관심이 확산됐다.



#### □ RPA로 자동화되고 있는 주요 업무 유형

- 초기 금융권 Back-office 업무에 주로 도입, 점차 제조·유통으로 확산
- 제조의 경우 일 단위로 이루어지는 재무·회계, 수·출입, 판매 분야의 서류작성 및 ERP 등 DB 등록 업무를 중심으로 도입되고 있음
- 유통은 재고관리, 판매실적 기록, 작업보고서 등의 업무에 주로 도입

분야별 RPA 도입 업무 사례

분야	주요 업무
금융	<ul style="list-style-type: none"> <li>• 비대면 계좌 승인 및 거부 처리 자동화</li> <li>• 신분증 위조 검증 자동화</li> <li>• 외부사이트 식용등급 조회 및 엑셀 보고서 작성 자동화</li> <li>• 펀드 매매기준 데이터 시스템 업로드 자동화</li> <li>• 전자공시 정보조회 및 DART 편집/엑셀 보고서 작성</li> <li>• 보험증권 서류 작성 및 등록 자동화</li> <li>• 고객 등기우편 발송 결과 취합 및 시스템 등록 자동화</li> <li>• 법인카드, 출장비, 매입 세금계산서 처리 자동화</li> </ul>
제조	<ul style="list-style-type: none"> <li>• 거래처 등록 및 견적 대사, 식용정보 관리</li> <li>• 대금대사 및 계정 조정, 분개방 기업, Reconciliation</li> <li>• 자재, 생산관리 물자표(BOM) 데이터 조회 및 ERP 등록</li> <li>• 물품 대금 및 작업비 청구서 프로세스 자동화</li> <li>• 판매코드 기준 데이터 집계 자동화</li> <li>• 선적문서, 수출입면장 데이터 조회 및 ERP 입력 자동화</li> <li>• 급여명세서, 복리후생/근태 관리 문서 작성 자동화</li> <li>• 법인카드, 출장비, 매입 세금계산서 처리 자동화</li> </ul>
유통	<ul style="list-style-type: none"> <li>• 재고관리 입력 및 승인 프로세스 자동화</li> <li>• POS 데이터 입력, 작업 보고서 입력 자동화</li> <li>• 제품, 수출입 선적 서류 처리 및 ERP 입력 자동화</li> <li>• 일/월 마감 업무 처리 자동화</li> <li>• 법인카드, 출장비, 매입 세금계산서 처리 자동화</li> </ul>

출처: 한국IBM('18), LGCNS('18), EY한영('18)

#### □ RPA 적용이 용이한 업무의 유형

- Rule Based의 표준화된 반복업무, 수작업 오류가 많고 시간이 필요한 업무 등이 주요 대상

<ul style="list-style-type: none"> <li>• Rule Based의 반복업무</li> <li>• 정형 Data를 다루는 영역</li> <li>• 프로세스가 정의되고 표준화된 분야</li> </ul>	<ul style="list-style-type: none"> <li>• 고정된 시스템이나 웹을 통해 정보가 연결되는 분야</li> <li>• 많은 인력과 시각이 투입되는 업무</li> <li>• 수작업 오류가 나기 쉬운 분야</li> </ul>
---	---

## 1. csv 파일 엑셀 xlsx 파일 변환

```
from openpyxl import Workbook
import os
import csv
path="./test_folder/csv"
file_list = os.listdir(path)
file_list = [file for file in file_list if file.endswith(".csv")]
print(file_list)
for file_name in file_list:
    fn = path+'/'+file_name
    wb = Workbook()
    ws = wb.active
    with open(fn, 'r') as f:
        for row in csv.reader(f):
            ws.append(row)
    wb.save('./out/'+file_name[0:-4]+'.xlsx')
    wb.close()
```

rpa\_1.py

csv 파일을 excel의 xlsx 파일로 변환하는 프로그램이다. 파이썬은 한글 오피스의 한셀 파일에서는 작동되지 않는 경우가 가끔 있다. excel과 한셀 파일의 파일 구조가 다르기 때문이다. 물론 어느 파일이나 한셀로나 엑셀로 불러올 때 차이가 없지만 실제 데이터 성분은 차이가 있다. 엑셀 프로그램이 있다면 문제가 없지만 한셀 프로그램만 있다면 이 프로그램을 사용하면 된다.

대상 파일 : ./test\_folder/csv 내 모든 csv 파일

생성 파일 : ./out 내 xlsx 확장자 파일

```
import os
f_list=[file for file in os.listdir() if file.lower().endswith(('.xlsx', '.csv'))]
print(f_list)
```

## 2. 엑셀 xlsx 파일 한 Sheet 취합

```
import os
from openpyxl import Workbook
from openpyxl import load_workbook
path='./test_folder/club'
list_dir = os.listdir(path)
wb2 = Workbook()
ws2 = wb2.active
cnt = 0
student =0
for filename in list_dir:
    if ".xlsx" not in filename:
        continue
    wb = load_workbook(path+'/'+filename)
    ws = wb.active
    cnt += 1
    if cnt==1:
        num=1
    else:
        num=2
    for row in ws.iter_rows(min_row=num):
        data=[]
        student+=1
        for cell in row:
            data.append(cell.value)
        ws2.append(data)
print(cnt, "개의 파일", student-1, "명을 처리했습니다")
wb2.save("./out/total_collect.xlsx")
```

rpa\_2.py

여러개의 엑셀 파일 데이터를 모아서 total\_collect.xlsx에 취합하는 프로그램이다.

대상 파일 : ./test\_folder/club 내 모든 xlsx 파일

생성 파일 : total\_collect.xlsx

### 3. 엑셀 xlsx 항목별 Sheet 분류

```
from openpyxl import *
def categoryList(col : int)-> list:
    temp_list = [r[col].value for r in ws]
    del temp_list[0]
    temp_set = set(temp_list)
    name_list = list(temp_set)
    return name_list
def categoryData(col, name):
    total_list=[]
    for r in ws.rows:
        temp_list=[]
        cell_num = len(r)
        if r[col].value == name:
            for n in range(0,cell_num):
                temp_list.append(r[n].value)
        if temp_list != []:
            total_list.append(temp_list)
    return total_list
def writeExcel(name, total_list):
    sht = wb.create_sheet(name)
    i=1
    for data in total_list:
        data_length = len(data)
        for n in range(0, data_length):
            sht.cell(row=i, column=n+1).value = data[n]
        i=i+1
path = "./out/total_collect.xlsx"
wb = load_workbook(path)
ws = wb.active
pb= wb.sheetnames
col = 2 #열번호(엑셀의 A열이 0번이다. 2은 C열을 의미)
category = categoryList(col)
for name in category:
    total_list = categoryData(col, name)
    writeExcel(name, total_list)
wb.save("./out/collect_result.xlsx")
```

rpa\_3.py

rpa1.py에서 생성한 파일을 col = 2 값으로 분류할 항목을 구분할 것이다. 결과적으로 항목별로 시트가 생성될 것이다.

대상 파일 : total\_collect.xlsx

생성 파일 : collect\_result.xlsx



#### 4. 엑셀 xlsx 파일 여러 Sheet 취합

```
import glob
import win32com.client
excel = win32com.client.Dispatch("Excel.Application")
excel.Visible = False
wb_new = excel.Workbooks.Add()
filepath='C:\sampleW*.xlsx'
list_filepath = glob.glob(filepath, recursive=True)
for filepath in list_filepath:
    wb = excel.Workbooks.Open(filepath)
    wb.Worksheets("Sheet1").Copy(Before=wb_new.Worksheets("Sheet1"))
wb_new.SaveAs("C:\sampleW통합 문서.xlsx")
excel.Quit()
```

rpa\_4.py

여러개의 xlsx 파일을 한 파일 여러 시트로 취합할 때 사용한다.

#### 5. 파일 복사

```
# pip install pytest-shutil
import os
import shutil
dir = str(os.getcwd())
path = dir+"/eprog"
print(path)
files=[file for file in os.listdir(path) if file.endswith(".py")]
for file in files:
    print(file)
    shutil.copy(path+"/"+file, 'i:/py')
```

rpa\_5.py

#### 6. 엑셀 행(row) 삭제

```
import openpyxl as op
wb = op.load_workbook("./data/row_col.xlsx")
ws = wb.active
del_row=[2,5,7,9,13] # 행 지정
cnt=0
for row in del_row:
    delr = row - cnt
    cnt=cnt+1
    ws.delete_rows(delr,1)
wb.save("./out/row_delete_result.xlsx")
```

rpa\_6.py

## 7. 엑셀 xlsx 파일 여러 항목 별 취합

```
import openpyxl as op
import os
def saveCellAddress()->list:
    num = int(input('입력받을 셀수 : '))
    save_cell = []
    save_head = []
    for i in range(1,num+1):
        print(str(i)+"번째 입력 : ")
        head = input(' 헤더 이름 입력 ex) 이름, 국어 : ')
        cell = str(input(' 셀 주소 입력 ex) A1, B1 : '))
        save_head.append(head)
        save_cell.append(cell)
    return save_head, save_cell
def saveData(path : str, head_list, cell_list : list):
    excellist = os.listdir(path)
    excellist = [file for file in excellist if file.endswith(".xlsx")]
    dic = {}
    for cell in cell_list:
        dic[cell] = ''
    cnt=0
    for cell in cell_list:
        templist = [head_list[cnt]]
        cnt = cnt+1
        for file in excellist:
            wb = op.load_workbook(path + "/" + file)
            ws = wb.active
            templist.append(ws[cell].value)
            dic[cell] = templist
    return dic
def writeExcel(dic):
    wb = op.Workbook()
    ws = wb.active
    row_num=1
    col_num=1
    for key in dic.keys():
        for data in dic[key]:
            ws.cell(row=row_num, column = col_num).value = data
            row_num = row_num + 1
            col_num = col_num+1
        row_num = 1
    wb.save("./out/dicion_Result.xlsx")
if __name__ == "__main__":
    path = "./test_folder/makexlsX"
    head, cell = saveCellAddress()
    dic = saveData(path, head, cell)
    writeExcel(dic)
```

rpa\_7.py

## 8. 엑셀 xlsx 파일로 보고서 작성

```
from openpyxl import load_workbook
wb = load_workbook(filename = "./ex_file/ex_data.xlsx", data_only = True)
ws=wb[wb.sheetnames[0]]
tmp = []
for i in range(2, ws.max_row+1):
    tmp.append([])
    for cell in ws[i]:
        tmp[-1].append(cell.value)
for person in tmp:
    wb2 = load_workbook(filename = "./ex_file/성적통지표.xlsx")
    ws2 = wb2[wb2.sheetnames[0]]
    print("Wr",person[1], end=' ')
    for i in range(1,len(person)+1):
        name=str(person[0])
        ws2['D4'] = name[:1]
        ws2['F4'] = name[1:2]
        ws2['H4'] = int(name[2:4])
        ws2['K4'] = person[1]
        ws2['A7'] = person[2]
        ws2['C7'] = person[3]
        ws2['E7'] = person[4]
        ws2['I7'] = person[5]
        ws2['K7'] = person[5]
    wb2.save("./out/성적_"+str(person[1])+".xlsx")
```

rpa\_8.py

## 9. 빈 파일 생성

```
from openpyxl import Workbook
wb = Workbook()
ws = wb.active
ws.title = "sung11"
wb.save("./out/blank_data.xlsx")
wb.close()
```

rpa\_9.py

## 10. 엑셀 xlsx 파일로 pdf 파일 작성과 병합

```
import win32com.client
import openpyxl as op
import os
from PyPDF2 import PdfFileMerger, PdfFileReader

def excelInfo(filepath):
    excel_list = os.listdir(filepath)
    result = []
    for file in excel_list:
        wb = op.load_workbook(filepath+"/"+file)
        ws_list = wb.sheetnames
        filename = file.replace(".xlsx", "")
        for sht in ws_list:
            temp_tuple = (filepath+"/"+file, filename, sht)
            result.append(temp_tuple)
    return result

def transPDF(fileinfo, savepath):
    excel = win32com.client.Dispatch("Excel.Application")
    i=0
    for info in fileinfo:
        wb = excel.Workbooks.Open(info[0])
        ws = wb.Worksheets(info[2])
        ws.Select()
        wb.ActiveSheet.ExportAsFixedFormat(0, savepath+"/"+str(i+1)
        + "_" + info[1] + "_" + info[2] + ".pdf")
        i=i+1
        wb.Close(False)
        excel.Quit()

def PDFMerger(path):
    pdf_merge = PdfFileMerger()
    pdflist = os.listdir(path)
    for pdf in pdflist:
        pdf_merge.append(PdfFileReader(open(path+"/"+pdf, 'rb')))
    pdf_merge.write("c:/output/result.pdf")

if __name__ == "__main__":
    filepath = "c:/sample"
    pdfpath = "c:/output"
    excelinfo = excelInfo(filepath)
    transPDF(excelinfo, pdfpath)
    PDFMerger(pdfpath)
```

rpa\_10.py

## 11. 폴더 내 모든 파일 인쇄

```
import os
folder_name = "./imsi"
file_list = os.listdir(folder_name)
for file_name in file_list:
    os.startfile(folder_name + "/" + file_name, "print")
```

rpa\_11.py

```
import glob
for filename in glob.glob('*.Wdata*.xlsx'):
    #print(filename)
    file_list = filename.split('WW')
    #print(file_list)
    print(file_list[2])
print(file_list[0]+'WW'+file_list[1])
```

## 12. 텍스트 파일을 csv 파일로 취합

```
import os
directory = "./test_folder/personal_info"
outfile_name = "./out/merged_text.csv"
out_file = open(outfile_name, 'w')
input_files = os.listdir(directory)
headers = []
outfile_has_header = False
for filename in input_files:
    if ".txt" not in filename:
        continue
    file = open(directory + "/" + filename)
    contents = []
    for line in file:
        if ":" in line:
            splits = line.split(":")
            contents.append(splits[-1].strip())
            if len(contents) > len(headers):
                headers.append(splits[0].strip())
    if not outfile_has_header:
        header = ", ".join(headers)
        out_file.write(header)
        outfile_has_header = True
    new_line = ", ".join(contents)
    out_file.write("\n" + new_line)
    file.close()
out_file.close()
```

rpa\_12.py

### 13. 엑셀 파일 별 sheet 계산

```
import pandas as pd
df1 = pd.read_excel('학생시험성적0.xlsx', sheet_name = '1차시험',
    index_col='학생')
print(df1)
df2 = pd.read_excel('학생시험성적1.xlsx', sheet_name = '2차시험',
    index_col='학생')
print(df2)
excel_writer3 = pd.ExcelWriter('학생시험성적4.xlsx', engine='openpyxl')
df1.to_excel(excel_writer3, index='학생', sheet_name='중간고사')
df2.to_excel(excel_writer3, index='학생', sheet_name='기말고사')
df3 = pd.concat([df1,df2], axis=1)
del df3['평균']
df3['국어평균']=(df1['국어']+df2['국어'])/2
df3['영어평균']=(df1['영어']+df2['영어'])/2
df3['수학평균']=(df1['수학']+df2['수학'])/2
df3['평균합계']=df3['국어평균'] + df3['영어평균'] + df3['수학평균']
df3['순위'] = df3['평균합계'].rank(ascending=False)
print(df3)
df3.to_excel(excel_writer3, index='학생', sheet_name='총합')
excel_writer3.save()
```

rpa\_13.py

#### 오름차순, 내림차순

ascending : 오름차순      순위 : rank(ascending=True)

ex) 1, 20, 25, 40, 43, 57

KBS, MBC, SBS

desending : 내림차순      순위 : rank(ascending=False)

ex) 57, 43, 40, 25, 20, 1

SBS, MBC, KBS

## 14. 같은 양식 여러 엑셀 파일을 한번에 수정하기

```
import os
import openpyxl
path = "./imsi/"
output = "./out/"
file_list = os.listdir(path)
print(file_list)
num = int(input("몇개 바꿀지 정수로 입력 : "))
for i in range(1, num+1):
    print(str(i)+"번째 설정:")
    globals()['cell_position'+str(i)] = str(input("바꿀 위치 : "))
    globals()['cell_value'+str(i)] = str(input("바꿀 문자열 : "))
    print("\n")
for file_name_raw in file_list:
    file_name = path + file_name_raw
    out_name = output + file_name_raw
    print(file_name)
    wb = openpyxl.load_workbook(filename=file_name)
    ws = wb.active
    for i in range(1, num+1):
        print(file_name_raw + "의 "+str(i)+"번째 변경")
        cell_position = globals()['cell_position'+str(i)]
        cell_value = globals()['cell_value'+str(i)]
        ws[cell_position].value = cell_value
        print(cell_position+"의 값을("+cell_value+")로 입력합니다.")
    print("\n")
    # wb.save(file_name_raw) #저장
    wb.save(out_name) #저장
    wb.close()
```

rpa\_14.py

## 전역변수

- 프로그램 내 함수 밖에서 선언됨
- 프로그램 전체에 공유되지만 함수 내에서는 수정할 수 없다.

## 예제

```
global a
a = 3
def test():
    global a
    b = 2
    return a + b
print(test())
print(a)
```

## 15 여러 폴더 안에 있는 파일명 한번에 바꾸기

```
import os
#여러 폴더가 있는 경로
folderpath = "./test_folder/picture"
#폴더명 받아오기
folderlist = os.listdir(folderpath)
#폴더명 리스트 출력해보기
print(folderlist)
i=0 #파일명 변경하기 위한 넘버링 변수
#폴더리스트를 for문을 통해 반복
for fname1 in folderlist:
    #해당 test 폴더(1,2,3,4) 위치 설정
    current_folder = folderpath + "/" + fname1
    #각 test폴더(1,2,3,4) 안의 파일명 받아오기
    filelist = os.listdir(current_folder)
    print("현재 폴더명 : ", fname1)
    #각 폴더명의 파일리스트를 다시 for문을 통해 반복
    for fname2 in filelist:
        i = i+1
        print(fname2+" 를 photo_"+str(i)+".jpg로 파일명을 변경")
        os.rename(current_folder + "/" + fname2 ,
current_folder+"/"++"photo_"+str(i)+".jpg")
rpa_15py
```



## 16. 여러 엑셀 파일 각 시트를 pdf변환하기

```
import win32com.client
import openpyxl as op
import os
def excelInfo(filepath):
    excel_list = os.listdir(filepath)
    result = [] #빈 리스트 생성
    for file in excel_list: #엑셀파일명 리스트를 for문을 통해 반복
        wb = op.load_workbook(filepath+"/"+file)
        ws_list = wb.sheetnames
        filename = file.replace(".xlsx","")
        for sht in ws_list: #시트명 리스트를 for문을 통해 반복
            temp_tuple = (filepath+"/"+file, filename, sht)
            result.append(temp_tuple) #위 튜플을 빈리스트에 추가
    #print(result) #결과 출력
    return result # 튜플로 이루어진 리스트 리턴

def transPDF(fileinfo, savepath):
    excel = win32com.client.Dispatch("Excel.Application")
    i=0 #파일명 중복을 방지하기 위한 인덱싱 번호
    #excelinfo를 입력받아 for 문 실행
    for info in fileinfo:
        wb = excel.Workbooks.Open(info[0])
        ws = wb.Worksheets(info[2]) #튜플의 2번째 요소는 시트명임.
        ws.Select() #위 설정한 시트 선택
        wb.ActiveSheet.ExportAsFixedFormat(0,
savepath+"/"+str(i)+"_"+info[1]+"_"+info[2]+".pdf")
        i=i+1
        wb.Close(False)
        excel.Quit() # excel application 닫기

filepath = r"C:\Users\wsamth\OneDrive\바탕 화면\PythonWorkspace
WeprogWtest_folder\wsamil"
pdfpath = r"C:\Users\wsamth\OneDrive\바탕 화면\PythonWorkspace
WeprogWtest_folder\Woutput"
excelinfo = excelInfo(filepath)
transPDF(excelinfo, pdfpath)
```

rpa\_16.py

## 17. 엑셀 데이터 종류별 sheet 자동 분류하기(한파일)

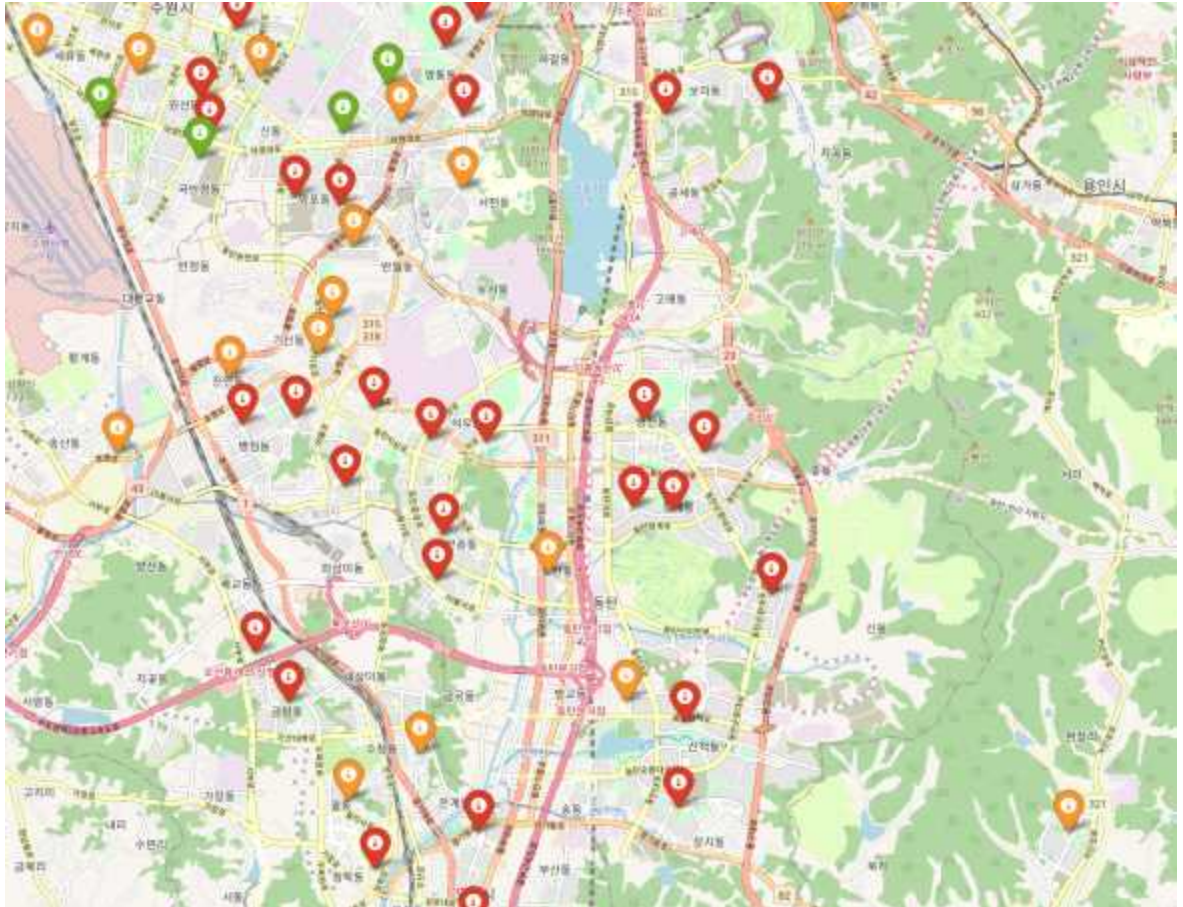
```

import openpyxl as op
from openpyxl import load_workbook
def categoryList(col : int)-> list:
    temp_list = [r[col].value for r in ws]
    del temp_list[0] #제목행 제거
    temp_set = set(temp_list) #중복을 제거하기 위해 list를 set으로 변환
    name_list = list(temp_set) #다시 set을 리스트로 변환
    return name_list #위 리스트를 리턴
def categoryData(col, name):
    total_list=[] #temp_list를 전부 담을 토털 리스트(결과 리턴값)
    for r in ws.rows: # 엑셀 시트에서 data가 있는 행을 반복
        temp_list=[] #한 줄의 데이터를 임시로 담을 리스트 초기화
        cell_num = len(r)
        if r[col].value == name:
            for n in range(0,cell_num):
                temp_list.append(r[n].value)
        if temp_list != []:
            total_list.append(temp_list)
    return total_list
#카테고리별 시트를 생성하여 데이터를 모두 입력한다.
def writeExcel(name, total_list):
    sht = wb.create_sheet(name)
    i=1 #각 시트 행에 접근하기 위한 인덱스
    for data in total_list: #total_list 반복
        data_length = len(data)
        for n in range(0, data_length):
            sht.cell(row=i, column=n+1).value = data[n]
        i=i+1 #행을 바꾸기 위한 인덱스 숫자 증가
path = "./ex_file/rawdata.xlsx" # 원본 엑셀파일 경로
wb = op.load_workbook(path)
ws = wb.active #활성화시트 ws로 정의
col = 1 #열번호(엑셀의 A열이 0번이다. 1은 B열을 의미)
category = categoryList(col)
print(category)
for name in category: #위 category 결과 List를 반복
    tota_llist = categoryData(col, name)
    writeExcel(name, tota_llist)
wb.save("./out/분류_결과파일.xlsx")

```

rpa\_11.py

# CHAP 13. folium



## folium

데이터를 지도 위에 표시할 수 있는 라이브러리

설치 방법 : `pip install folium`

임포트 하기 : `import folium`

지정한 지점의 지도 만들기

```
m = folium.Map([위도, 경도], zoom_start=확대 배율)
```

마커 추가하기

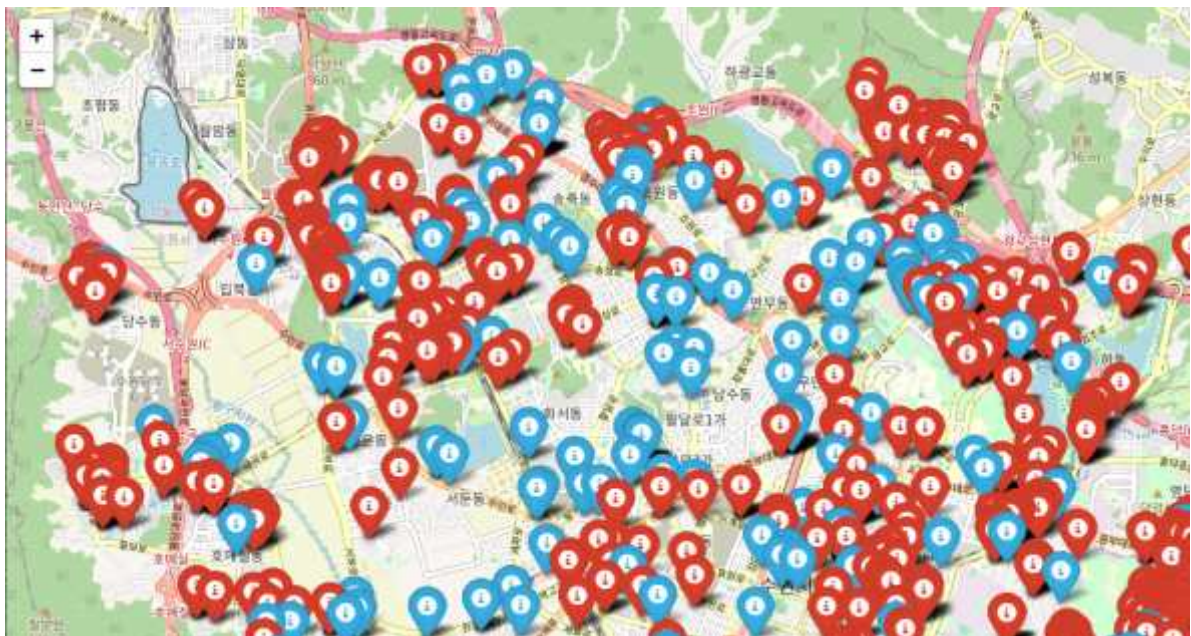
```
folium.Marker([위도, 경도]).add_to(m)
```

마커 추가하기(툴팁 포함)

```
folium.Marker([위도, 경도], tooltip="문자열").add_to(m)
```

지도를 표시할 HTML 파일 쓰기

```
m.save("파일명.html")
```



빨간색 마커는 이용 불가능한 곳이고 파란색은 공공 구역이므로 사용가능한 곳이다.

```
import pandas as pd
import folium
folium.__version__
pd.set_option('display.max_columns', 30)
# 충전소 리스트를 데이터프레임 변환
df = pd.read_excel('./data/electric_carstation.xlsx', index_col='연번')
print(df[['충전소명', '이용자제한', '충전기상태', '충전기타입']])
print("="*90, "\n")
# # 수원 지도 만들기
station_map = folium.Map(location=[37.27, 127.03], zoom_start=13)
# # 위치정보를 Marker로 표시
for name, lat, lng, stat in zip(df.충전소명, df.위도, df.경도, df.이용자제한):
    if stat == '이용자 제한 없음':
        col = 'blue'
    else:
        col = 'red'
    folium.Marker([lat, lng],
                  icon=folium.Icon(color=col, icon='info-sign'),
                  popup=name).add_to(station_map)
# 지도를 HTML 파일로 저장하기
station_map.save('./out/전기충전소.html')
print("\n")
```

folium\_1.py

파이썬 유치원

2022.02.11.

교육인 PySU