

세상의 속도를  
따라잡고 싶다면



# 점프 투 파이썬

박응용 지음(위키독스 운영자)



# 파이썬 날개 달기



05-6 표준 라이브러리



## ■ 파이썬 표준 라이브러리

- 라이브러리(library) = 도서관(원하는 정보를 찾아보는 곳)
- 파이썬 라이브러리는 전 세계의 파이썬 고수들이 만든 유용한 프로그램을 모아 놓은 것
- 파이썬을 설치할 때 자동으로 컴퓨터에 설치됨
- 자주 사용되고 꼭 알아 두면 좋은 라이브러리 소개

## ■ datetime.date

- 연, 월, 일로 날짜를 표현할 때 사용하는 함수

- 1) A 군과 B 양이 2021년 12월 14일부터 만나기 시작했다면 2023년 4월 5일은 둘이 사귀 지 며칠째 되는 날일까?

```
>>> import datetime
>>> day1 = datetime.date(2021, 12, 14)
>>> day2 = datetime.date(2023, 4, 5)
```

함수를 사용하기 전에 해당  
라이브러리를 import하는  
것을 잊지 말자!



```
>>> diff = day2 - day1
>>> diff.days
477 ← 둘이 만난 지 477일째
```

## ■ datetime.date

2) A 군과 B 양이 사귀기 시작한 2021년 12월 14일은 무슨 요일이었을까?

```
>>> day = datetime.date(2021, 12, 14)
>>> day.weekday()
1 ← 2021년 12월 14일은 화요일
```

- 0은 월요일, 1은 화요일, 2는 수요일, ... , 6은 일요일
- 1은 월요일, 2는 화요일, ..., 7은 일요일을 리턴하려면 isoweekday 함수 사용

```
>>> day.isoweekday()
2
```

## ■ time

### 1) time.time( )

- UTC(Universal Time Coordinated, 협정 세계 표준시)를 사용하여 현재 시간을 실수 형태로 리턴
- 1970년 1월 1일 0시 0분 0초를 기준으로 지난 시간을 초 단위로 리턴

```
>>> import time
>>> time.time()
1684983953.5221913
```

### 2) time.localtime( )

- time.time( )이 리턴한 실숫값을 사용해서 연, 월, 일, 시, 분, 초, ...의 형태로 바꾸어 줌

```
>>> time.localtime(time.time())
time.struct_time(tm_year=2023, tm_mon=5, tm_mday=21, tm_hour=16,
tm_min=48, tm_sec=42, tm_wday=1, tm_yday=141, tm_isdst=0)
```

## ■ time

### 3) time.asctime( )

- time.localtime가 리턴한 튜플 형태의 값을 인수로 받아서 날짜와 시간을 알아보기 쉬운 형태로 리턴

```
>>> time.asctime(time.localtime(time.time()))  
'Fri Apr 28 20:50:20 2023'
```

### 4) time.ctime( )

- 항상 현재 시간만을 리턴

```
>>> time.ctime()  
'Fri Apr 28 20:56:31 2023'
```

## ■ time

### 5) time.strftime( )

- 시간에 관계된 것을 세밀하게 표현하는 여러 가지 포맷 코드 제공

```
time.strftime('출력할_형식_포맷_코드', time.localtime(time.time()))
```

#### ■ 예)

```
>>> import time
>>> time.strftime('%x', time.localtime(time.time()))
'05/25/23' ← 현재 설정된 지역의 날짜 출력
>>> time.strftime('%c', time.localtime(time.time()))
'Thu May 25 10:13:52 2023' ← 날짜와 시간 출력
```



## ■ time

### 6) time.sleep( )

- 주로 루프 안에서 많이 사용
- 일정한 시간 간격을 두고 루프를 실행할 수 있음
- 인수는 실수 형태 (예) 1이면 1초, 0.5면 0.5초
- 예) 1초 간격으로 0부터 9까지의 숫자 출력

```
import time
for i in range(10):
    print(i)
    time.sleep(1)
```

## ■ math.gcd

- 최대 공약수(gcd, greatest common divisor)를 쉽게 구하는 함수
- 어린이집에서 사탕 60개, 초콜릿 100개, 젤리 80개를 준비했다. 아이들이 서로 싸우지 않도록 똑같이 나누어 봉지에 담는다고 하면 최대 몇 봉지까지 만들 수 있을까?

```
>>> import math
>>> math.gcd(60, 100, 80)
20
```

- 60, 100, 80의 최대 공약수를 구하면 해결

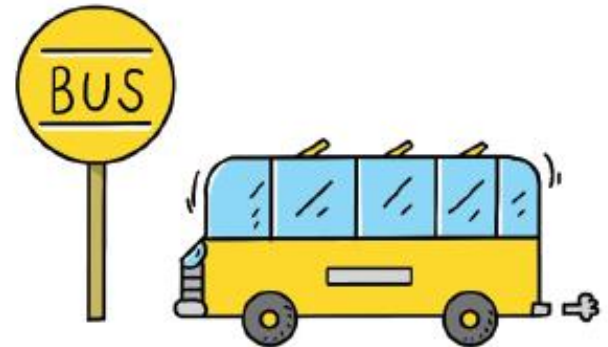


## ■ math.lcm

- 최소 공배수(lcm, least common multiple)를 쉽게 구하는 함수
- 어느 버스 정류장에 시내버스는 15분마다 도착하고 마을버스는 25분마다 도착한다고 한다. 오후 1시에 두 버스가 동시에 도착했다고 할 때 두 버스가 동시에 도착할 다음 시각을 알려면 어떻게 해야 할까?

```
>>> import math  
>>> math.lcm(15, 25)  
75
```

- 15, 25의 최소 공배수를 구하면 해결



## ■ random

- 난수(규칙이 없는 임의의 수)를 발생시키는 모듈

### 1) random.random

- 예) 0.0에서 1.0 사이의 실수 중 난수 값을 리턴

```
>>> import random
>>> random.random()
0.53840103305098674
```

### 2) random.randint

- 3) 예) 1에서 10 사이의 정수 중 난수 값을 리턴

```
>>> random.randint(1, 10)
6
```

## ■ random

- 난수(규칙이 없는 임의의 수)를 발생시키는 모듈

### 3) random.choice

- 입력으로 받은 리스트에서 무작위로 하나를 선택하여 리턴

```
def random_pop(data):  
    number = random.choice(data)  
    data.remove(number)  
    return number
```

### 4) random.sample

- 리스트의 항목을 무작위로 섞음

```
>>> import random  
>>> data = [1, 2, 3, 4, 5]  
>>> random.sample(data, len(data))  
>>> data  
[5, 1, 3, 4, 2]
```

## ■ itertools.zip\_longest

- 같은 개수의 자료형을 묶는 파이썬 내장 함수인 zip 함수와 똑같이 동작
- 전달한 반복 가능 객체(\*iterables)의 길이가 서로 다르다면 긴 객체의 길이에 맞춰 fillvalue에 설정한 값을 짧은 객체에 채울 수 있음
- 예) 유치원생 5명에게 간식을 나누어 주는 코드

```
import itertools

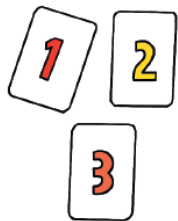
students = ['한민서', '황지민', '이영철', '이광수', '김승민']
snacks = ['사탕', '초콜릿', '젤리']

result = itertools.zip_longest(students, snacks, fillvalue='새우깡')
print(list(result))
```

## ■ itertools.permutation

- 반복 가능 객체 중에서 r개를 선택한 순열을 이터레이터로 리턴하는 함수
  - 이터레이터: 반복 가능한 객체
- 예) 1, 2, 3이라는 숫자가 적힌 3장의 카드에서 2장의 카드를 꺼내 만들 수 있는 2자리 숫자를 모두 구하려면 어떻게 해야 할까?

```
>>> import itertools
>>> list(itertools.permutations(['1', '2', '3'], 2))
[('1', '2'), ('1', '3'), ('2', '1'), ('2', '3'), ('3', '1'), ('3', '2')]
```



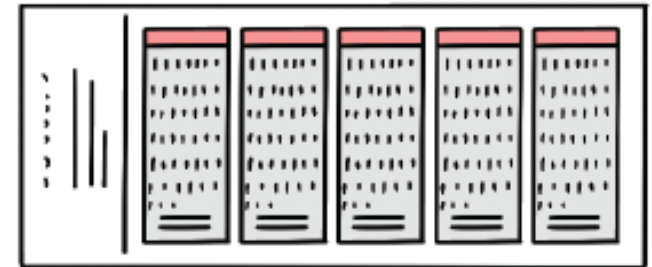
```
>>> for a, b in itertools.permutations(['1', '2', '3'], 2):
...     print(a + b)
...
12
13
21
23
31
32
```

## ■ itertools.combination

- 반복 가능 객체 중에서 r개를 선택한 조합을 이터레이터로 리턴하는 함수
- 예) 1~45 중 서로 다른 숫자 6개를 뽑는 로또 번호의 모든 경우의 수(조합)를 구하고 그 개수를 출력하려면 어떻게 해야 할까?

```
>>> import itertools  
>>> it = itertools.combinations(range(1, 46), 6)
```

```
>>> len(list(itertools.combinations(range(1, 46), 6)))  
8145060
```





## ■ functools.reduce

- 함수(function)를 반복 가능한 객체(iterable)의 요소에 차례대로(왼쪽에서 오른쪽으로) 누적 적용하여 이 객체를 하나의 값으로 줄이는 함수
- 예) 입력 인수 data의 요소를 모두 더하여 리턴

```
import functools

data = [1, 2, 3, 4, 5]
result = functools.reduce(lambda x, y: x + y, data)
print(result)
```

실행 결과

15

- (((1 + 2) + 3) + 4) + 5와 같이 계산

## ■ operator.itemgetter

- 주로 sorted와 같은 함수의 key 매개변수에 적용하여 다양한 기준으로 정렬할 수 있도록 도와주는 모듈
- 예) 학생의 이름, 나이, 성적 등의 정보를 저장한 students 리스트를 나이 순으로 정렬하려면?

```
students = [  
    ("jane", 22, 'A'),  
    ("dave", 32, 'B'),  
    ("sally", 17, 'B'),  
]
```

```
from operator import itemgetter  
  
students = [  
    ("jane", 22, 'A'),  
    ("dave", 32, 'B'),  
    ("sally", 17, 'B'),  
]
```

```
result = sorted(students, key=itemgetter(1))  
print(result)
```

실행 결과

```
[('sally', 17, 'B'), ('jane', 22, 'A'), ('dave', 32, 'B')]
```



## ■ operator.itemgetter

- 예) students 리스트가 딕셔너리일 경우

```
students = [  
    {"name": "jane", "age": 22, "grade": 'A'},  
    {"name": "dave", "age": 32, "grade": 'B'},  
    {"name": "sally", "age": 17, "grade": 'B'},  
]
```

### 실행 결과

```
[{'name': 'sally', 'age': 17, 'grade': 'B'}, {'name': 'jane', 'age': 22, 'grade':  
'A'}, {'name': 'dave', 'age': 32, 'grade': 'B'}]
```

```
from operator import itemgetter
```

```
students = [  
    {"name": "jane", "age": 22, "grade": 'A'},  
    {"name": "dave", "age": 32, "grade": 'B'},  
    {"name": "sally", "age": 17, "grade": 'B'},  
]
```

```
result = sorted(students, key=itemgetter('age'))  
print(result)
```

## ■ shutil

- 파일을 복사(copy)하거나 이동(move)할 때 사용하는 모듈
- 예) 작업 중인 파일을 자동으로 백업하는 프로그램
  - c:\doit\A.txt를 c:\temp\A.txt.bak이라는 이름으로 복사

```
import shutil
```

```
shutil.copy("c:/doit/a.txt", "c:/temp/a.txt.bak")
```

## ■ glob

- 특정 디렉터리 안의 파일 이름을 읽어서 리스트로 리턴
- \*, ? 등 메타 문자를 써서 원하는 파일만 읽어 들일 수 있음
- 예) C:\doit 디렉터리에 있는 파일 중 이름이 mark로 시작하는 파일 모두 찾기

```
>>> import glob
>>> glob.glob("c:/doit/mark*")
['c:/doit\\marks1.py', 'c:/doit\\marks2.py', 'c:/doit\\marks3.py']
```

## ■ pickle

- 객체의 형태를 그대로 유지하면서 파일에 저장하고 불러올 수 있게 하는 모듈

### 1) pickle.dump

- 예) 딕셔너리 객체를 그대로 파일에 저장

```
>>> import pickle
>>> f = open("test.txt", 'wb')
>>> data = {1: 'python', 2: 'you need'}
>>> pickle.dump(data, f)
>>> f.close()
```

### 2) pickle.load

- 예) pickle.dump로 저장한 파일을 원래 딕셔너리 객체 상태 그대로 불러오기

```
>>> import pickle
>>> f = open("test.txt", 'rb')
>>> data = pickle.load(f)
>>> print(data)
{2: 'you need', 1: 'python'}
```

## ■ OS

- 환경 변수나 디렉터리, 파일 등의 OS 자원을 제어할 수 있게 해 주는 모듈

### 1) os.environ

- 현재 시스템의 환경 변수값을 리턴

```
>>> import os
>>> os.environ
environ({'PROGRAMFILES': 'C:\\Program Files', 'APPDATA': ...생략...})
```

```
>>> os.environ['PATH']
'C:\\ProgramData\\Oracle\\Java\\javapath;(...생략...)'
```

## ■ OS

- 환경 변수나 디렉터리, 파일 등의 OS 자원을 제어할 수 있게 해 주는 모듈

### 2) os.chdir

- 현재 디렉터리의 위치 변경

```
>>> os.chdir("C:\\WINDOWS")
```

### 3) os.getcwd

- 현재 자신의 디렉터리 위치를 리턴

```
>>> os.getcwd()  
'C:\\WINDOWS'
```

### 4) os.system

- 시스템 자체의 프로그램이나 기타 명령어를 파이썬에서 호출할 수 있음
- 예) 현재 디렉터리에서 시스템 명령어 dir 실행

```
>>> os.system("dir")
```

### 5) os.popen

- 시스템 명령어를 실행한 결과값을 읽기 모드 형태의 파일 객체로 리턴

```
>>> f = os.popen("dir")
```



## ■ OS

### ■ 기타 유용한 os 관련 함수

함수	설명
os.mkdir(디렉터리)	디렉터리를 생성한다.
os.rmdir(디렉터리)	디렉터리를 삭제한다. 단, 디렉터리가 비어 있어야 삭제할 수 있다.
os.remove(파일)	파일을 지운다.
os.rename(src, dst)	src라는 이름의 파일을 dst라는 이름으로 바꾼다.

## ■ zipfile

- 여러 개의 파일을 zip 형식으로 합치거나 이를 해제할 때 사용하는 모듈
- 예) 'a.txt', 'b.txt', 'c.txt' 3개의 텍스트 파일을 하나로 합쳐 'mytext.zip' 파일로 만들고, 이 파일을 원래의 텍스트 파일 3개로 해제하는 프로그램

```
import zipfile

# 파일 합치기
with zipfile.ZipFile('mytext.zip', 'w') as myzip:
    myzip.write('a.txt')
    myzip.write('b.txt')
    myzip.write('c.txt')

# 해제하기
with zipfile.ZipFile('mytext.zip') as myzip:
    myzip.extractall()
```

## ■ zipfile

- 합친 파일에서 특정 파일만 해제하고 싶을 때

```
# 특정 파일만 해제하기
with zipfile.ZipFile('mytext.zip') as myzip:
    myzip.extract('a.txt')
```

- 파일을 압축하여 묶고 싶을 때

```
# 압축하여 묶기
with zipfile.ZipFile('mytext.zip', 'w', compression=zipfile.ZIP_LZMA, compresslevel=9) as myzip:
    (...생략...)
```

## ■ zipfile

### ■ compression의 4가지 종류

- ZIP\_STORED: 압축하지 않고 파일을 zip으로만 묶는다. 속도가 빠르다.
- ZIP\_DEFLATED: 일반적인 zip 압축으로 속도가 빠르고 압축률은 낮다(호환성이 좋다).
- ZIP\_BZIP2: bzip2 압축으로 압축률이 높고 속도가 느리다.
- ZIP\_LZMA: lzma 압축으로 압축률이 높고 속도가 느리다(7zip과 동일한 알고리즘).

### ■ compressionlevel은 압축 수준을 의미하는 숫자값으로, 1~9를 사용함

- 1은 속도가 가장 빠르지만 압축률이 낮고, 9는 속도는 가장 느리지만 압축률이 높음

## ■ threading

- 초보 프로그래머가 구현하기에는 매우 어려운 기술
- 스레드(thread): 컴퓨터에서 동작하는 한 프로세스 안에서 2가지 또는 그 이상의 일을 동시에 수행할 수 있도록 함

```
import time
import threading ← 스레드를 생성하기 위해서는 threading 모듈이 필요
def long_task():
    for i in range(5):
        time.sleep(1)
        print("working:%s\n" % i)

print("Start")
```

\*뒷장에서 계속

## ■ threading

```
threads = []
for i in range(5):
    # long_task()
    t = threading.Thread(target=long_task) # 스레드를 생성
    threads.append(t)

for t in threads:
    t.start() # 스레드를 실행

for t in threads:
    t.join() # join으로 스레드가 종료될 때까지 대기

print("End")
```

## ■ tempfile

- 파일을 임시로 만들어서 사용할 때 유용

### 1) **tempfile.mkstemp**

- 중복되지 않는 임시 파일의 이름을 무작위로 만들어서 리턴

```
>>> import tempfile
>>> filename = tempfile.mkstemp()
>>> filename
'C:\\WINDOWS\\TEMP\\~-275151-0'
```

## ■ tempfile

- 파일을 임시로 만들어서 사용할 때 유용

### 2) **tempfile.TemporaryFile**

- 임시 저장 공간으로 사용할 파일 객체를 리턴
- 기본적으로 바이너리 쓰기 모드(wb)
- f.close()가 호출되면 파일 객체는 자동으로 사라짐

```
>>> import tempfile  
>>> f = tempfile.TemporaryFile()  
>>> f.close() ← 임시 파일 삭제
```



## ■ traceback

- 프로그램 실행 중 발생한 오류를 추적하고자 할 때 사용하는 모듈

- 예)

```
def a():  
    return 1 / 0  
  
def b():  
    a()  
  
def main():  
    try:  
        b()  
    except:  
        print("오류가 발생했습니다.")  
  
main()
```

실행 결과

오류가 발생했습니다.

## ■ traceback

- 오류가 발생한 위치에 traceback 모듈 적용  
예)

```
import traceback

def a():
    return 1 / 0

def b():
    a()

def main():
    try:
        b()
    except:
        print("오류가 발생했습니다.")
        print(traceback.format_exc())

main()
```

### 실행 결과

오류가 발생했습니다.

Traceback (most recent call last):

File "c:\doit\traceback\_sample.py", line 14, in main

b()

File "c:\doit\traceback\_sample.py", line 9, in b

a()

File "c:\doit\traceback\_sample.py", line 5, in a

return 1/0

ZeroDivisionError: division by zero

## ■ json

- JSON 데이터를 쉽게 처리하고자 사용하는 모듈
- 예) 개인정보를 JSON 형태의 데이터로 만든 myinfo.json 파일을 읽어 딕셔너리로 변환

```
{  
    "name": "홍길동",  
    "birth": "0525",  
    "age": 30  
}
```

```
>>> import json  
>>> with open('myinfo.json') as f:  
...     data = json.load(f)  
...  
>>> type(data)  
<class 'dict'>  
>>> data  
{'name': '홍길동', 'birth': '0525', 'age': 30}
```

## ■ json

- 예) 딕셔너리 자료형을 JSON 형태로 생성

```
>>> import json
>>> data = {'name': '홍길동', 'birth': '0525', 'age': 30}
>>> with open('myinfo.json', 'w') as f:
...     json.dump(data, f)
```

- 예) 파이썬 자료형을 JSON 문자열로 만드는 방법

```
>>> import json
>>> d = {"name": "홍길동", "birth": "0525", "age": 30}
>>> json_data = json.dumps(d)
>>> json_data
'{"name": "\\ud64d\\uae38\\ub3d9", "birth": "0525", "age": 30}'
```

- 예) JSON 문자열을 딕셔너리로 변환

```
>>> json.loads(json_data)
{'name': '홍길동', 'birth': '0525', 'age': 30}
```

## ■ json

- 예) 한글 문자열이 아스키 형태의 문자열로 변경되는 것을 방지하는 방법

```
>>> d = {"name": "홍길동", "birth": "0525", "age": 30}
>>> json_data = json.dumps(d, ensure_ascii=False)
>>> json_data
'{"name": "홍길동", "birth": "0525", "age": 30}'
>>> json.loads(json_data)
{'name': '홍길동', 'birth': '0525', 'age': 30}
```

- 예) 출력되는 JSON 문자열을 보기 좋게 정렬하는 방법

```
>>> d = {"name": "홍길동", "birth": "0525", "age": 30}
>>> print(json.dumps(d, indent=2, ensure_ascii=False))
{
  "name": "홍길동",
  "birth": "0525",
  "age": 30
}
```

- 예) 딕셔너리 외에 리스트나 튜플처럼 다른 자료형도 JSON 문자열로 변경 가능

```
>>> json.dumps([1, 2, 3])
'[1, 2, 3]'
>>> json.dumps((4, 5, 6))
'[4, 5, 6]'
```

## ■ urllib

- URL을 읽고 분석할 때 사용하는 모듈
- 브라우저로 위키독스의 특정 페이지를 읽기 위한 요청

`https://wikidocs.net/페이지_번호(예: https://wikidocs.net/12)`

- 오프라인으로도 읽을 수 있도록 페이지 번호를 입력받아 위키독스의 특정 페이지를 `wikidocs_페이지_번호.html` 파일로 저장하는 함수

```
import urllib.request

def get_wikidocs(page):
    resource = 'https://wikidocs.net/{}'.format(page)
    with urllib.request.urlopen(resource) as s:
        with open('wikidocs_%s.html' % page, 'wb') as f:
            f.write(s.read())
```

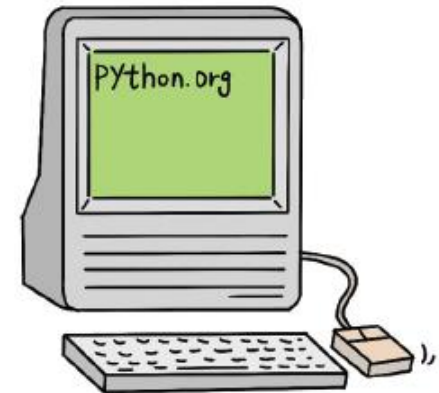
## ■ webbrowser

- 파이썬 프로그램에서 시스템 브라우저를 호출할 때 사용하는 모듈
- 예) `https://python.org` 사이트를 새로운 웹 브라우저로 여는 코드

```
import webbrowser  
  
webbrowser.open_new('http://python.org')
```

- 예) 이미 열린 브라우저로 `https://python.org` 사이트를 여는 경우

```
webbrowser.open('http://python.org')
```





*"Life is too short,  
You need Python!"*

인생은 너무 짧으니,  
파이썬이 필요해!

감사합니다.