ESTSOFT WASSUP

AI 서비스 기획

맹광국 강사

ggmaeng@gmail.com

EST

파이썬 문법
(예외처리와 내장함수, 라이브러리)

위키독스 점프투파이썬

EST

예외 처리

프로그램을 만들다 보면 수없이 많은 오류를 만나게 된다. 물론 오류가 발생하는 이유는 프로그램이 잘못 동작하는 것을 막기 위한 파이썬의 배려이다. 이번에는 파이썬에서 오류를 처리하는 방법에 대해서 알아보자.

오류는 언제 발생하는가?

EST

오류를 처리하는 방법을 공부하기 전에 어떤 상황에서 오류가 발생하는지 한번 알아보자. 오타를 입력했을 때 발생하는 구문 오류 같은 것이 아닌 실제 프로그램에서 자주 발생하는 오류를 중심으로 살펴보자.

먼저 존재하지 않는 파일을 사용하려고 시도했을 때 발생하는 오류이다.

```
>>> f = open("나없는파일", 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: '나없는파일'
```

위 예에서 볼 수 있듯이 없는 파일을 열려고 시도하면 FileNotFoundError 오류가 발생한다.

이번에는 0으로 다른 숫자를 나누는 경우를 생각해 보자. 이 역시 자주 발생하는 오류이다.

```
>>> 4 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

4를 0으로 나누려고 하니 ZeroDivisionError 오류가 발생한다.

마지막으로 1가지 예를 더 들어 보자. 다음 오류는 정말 빈번하게 일어난다.

```
>>> a = [1, 2, 3]
>>> a[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

a[3]은 a의 네 번째 요솟값을 가리키는데, a 리스트에는 값이 3개밖에 없으므로([1, 2, 3]) 값을 얻을 수 없다. 따라서 IndexError 오류가 발생한다. 파이썬은 이런 오류가 발생하면 프로그램을 중단하고 오류 메시지를 보여 준다.

오류 예외 처리 기법

EST

try-except 문

다음은 오류를 처리하기 위한 try-except 문의 기본 구조이다.

```
try:
    ...
except [발생오류 [as 오류변수]]:
    ...
```

try 블록 수행 중 오류가 발생하면 except 블록이 수행된다. 하지만 try 블록에서 오류가 발생하지 않는다면 except 블록은 수행되지 않는다.

except 구문을 자세히 살펴보자.

```
except [발생오류 [as 오류변수]]:
```

위 구문을 보면 []를 사용하는데, 이 기호는 괄호 안의 내용을 생략할 수 있다는 관례적인 표기법이다. 즉, except 구문은 다음 3가지 방법으로 사용할 수 있다.

1. try-except만 쓰는 방법

```
try:
    ...
except:
    ...
```

이 경우에는 오류의 종류에 상관없이 오류가 발생하면 except 블록을 수행한다.

2. 발생 오류만 포함한 except 문

```
try:
    ...
except 발생오류:
    ...
```

이 경우는 오류가 발생했을 때 except 문에 미리 정해 놓은 오류와 동일한 오류일 경우에만 except 블록을 수행한다는 뜻이다.

3. 발생 오류와 오류 변수까지 포함한 except 문

```
try:
    ...
except 발생오류 as 오류변수:
    ...
```

이 경우는 두 번째 경우에서 오류의 내용까지 알고 싶을 때 사용하는 방법이다.

이 방법의 예를 들어 보면 다음과 같다.

```
# try_except.py
try:
    4 / 0
except ZeroDivisionError as e:
    print(e)
```

위처럼 4를 0으로 나누려고 하면 ZeroDivisionError가 발생하여 except 블록이 실행되고 오류 변수 e에 담기는 오류 메시지를 출력할 수 있다. 출력되는 오류 메시지는 다음과 같다.

```
division by zero
```

오류 예외 처리 기법

EST

try-finally 문

try 문에는 finally 절을 사용할 수 있다. finally 절은 try 문 수행 도중 예외 발생 여부에 상관없이 항상 수행된다. 보통 finally 절은 사용한 리소스를 close해야 할 때 많이 사용한다.

다음 예를 살펴보자.

```
# try_finally.py
try:
    f = open('foo.txt', 'w')
    # 무언가를 수행한다.

    (... 생략 ...)

finally:
    f.close()  # 중간에 오류가 발생하더라도 무조건 실행된다.
```

foo.txt 파일을 쓰기 모드로 연 후 예외 발생 여부에 상관없이 항상 파일을 닫아 주려면 try-finally 문을 사용하면 된다.

오류 예외 처리 기법

EST

여러 개의 오류 처리하기

try 문 안에서 여러 개의 오류를 처리하려면 다음과 같이 사용해야 한다.

```
try:
    ...
except 발생오류1:
   ...
except 발생오류2:
   ...
```

즉, 0으로 나누는 오류와 인덱싱 오류를 다음과 같이 처리할 수 있다.

```
# many_error.py
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("인덱싱 할 수 없습니다.")
```

a는 2개의 요솟값을 가지고 있으므로 a[3]이 IndexError를 발생시켜 "인덱싱할 수 없습니다."라는 문자열을 출력할 것이다. 인덱싱 오류가 먼저 발생했으므로 `4 / 0`에 따른 Zero DivisionError 오류는 발생하지 않는다.

오류 예외 처리 기법

EST

앞에서 알아본 것과 마찬가지로 오류 메시지도 다음과 같이 확인할 수 있다.

```
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError as e:
    print(e)
except IndexError as e:
    print(e)
```

프로그램을 실행하면 'list index out of range'라는 오류 메시지가 출력될 것이다.

다음과 같이 ZerroDivisionError와 IndexError를 함께 처리할 수도 있다.

```
try:
    a = [1,2]
    print(a[3])
    4/0
except (ZeroDivisionError, IndexError) as e:
    print(e)
```

2개 이상의 오류를 동일하게 처리하기 위해서는 위와 같이 괄호를 사용하여 함께 묶어 처리하면 된다.

오류 예외 처리 기법

EST

try-else 문

try 문에는 다음처럼 else 절을 사용할 수도 있다.

```
try:
    ...
except [발생오류 [as 오류변수]]:
    ...
else:  # 오류가 없을 경우에만 수행
    ...
```

try 문 수행 중 오류가 발생하면 except 절, 오류가 발생하지 않으면 else 절이 수행된다.

다음은 try 문에 else 절을 사용한 간단한 예제이다.

```
# try_else.py
try:
    age=int(input('나이를 입력하세요: '))
except:
    print('입력이 정확하지 않습니다.')
else:
    if age <= 18:
        print('미성년자는 출입금지입니다.')
    else:
        print('환영합니다.')
```

만약 '나이를 입력하세요: '라는 질문에 숫자가 아닌 다른 값을 입력하면 오류가 발생하여 '입력이 정확하지 않습니다.'라는 문장을 출력한다. 오류가 없을 경우에만 else 절이 수행된다.

오류 회피하기

EST

코드를 작성하다 보면 특정 오류가 발생할 경우 그냥 통과시켜야 할 때가 있다. 다음 예를 살펴보자.

```
# error_pass.py
try:
    f = open("나없는파일", 'r')
except FileNotFoundError:
    pass
```

try 문 안에서 FileNotFoundError가 발생할 경우, pass를 사용하여 오류를 그냥 회피하도록 작성한 예제이다.

오류 일부러 발생시키기

EST

이상하게 들리겠지만, 프로그래밍을 하다 보면 종종 오류를 일부러 발생시켜야 할 경우도 생긴다. 파이썬은 raise 명령어를 사용해 오류를 강제로 발생시킬 수 있다.

예를 들어 Bird 클래스를 상속받는 자식 클래스는 반드시 fly라는 함수를 구현하도록 만들고 싶은 경우(강제로 그렇게 하고 싶은 경우)가 있을 수 있다. 다음 예를 살펴보자.

```
# error_raise.py
class Bird:
    def fly(self):
        raise NotImplementedError
```

Bird 클래스를 상속받는 자식 클래스는 반드시 fly 함수를 구현해야 한다는 의지를 보여 준다. 만약 자식 클래스가 fly 함수를 구현하지 않은 상태로 fly 함수를 호출한다면 어떻게 될까?

> NotImplementedError는 파이썬에 이미 정의되어 있는 오류로, 꼭 작성해야 하는 부분이 구현되지 않았을 경우 일부러 오류를 발생시키기 위해 사용한다.

```
class Eagle(Bird):
    pass

eagle = Eagle()
eagle.fly()
```

Eagle 클래스는 Bird 클래스를 상속받았다. 그런데 Eagle 클래스는 fly 메서드를 오버라이딩하여 구현하지 않았다. 따라서 eagle 객체의 fly 메서드를 수행하는 순간 Bird 클래스의 fly 메서드가 수행되어 NotImplementedError가 발생한다.

오류 일부러 발생시키기

EST

```
Traceback (most recent call last):
  File "...", line 33, in <module>
    eagle.fly()
  File "...", line 26, in fly
    raise NotImplementedError
NotImplementedError
```

> 상속받는 클래스에서 메서드를 재구현하는 것을 '메서드 오버라이딩'이라고 한다.

NotImplementedError가 발생하지 않게 하려면 다음과 같이 Eagle 클래스에 fly 함수를 구현해야 한다.

```
class Eagle(Bird):
    def fly(self):
        print("very fast")

eagle = Eagle()
eagle.fly()
```

위 예처럼 fly 함수를 구현한 후 프로그램을 실행하면 오류 없이 다음 문장이 출력된다.

```
very fast
```

예외 만들기

프로그램을 수행하다가 특수한 경우에만 예외 처리를 하려고 종종 예외를 만들어서 사용한다. 이번에는 직접 예외를 만들어 보자.

예외는 다음과 같이 파이썬 내장 클래스인 Exception 클래스를 상속하여 만들 수 있다.

```
# error_make.py
class MyError(Exception):
    pass
```

그리고 별명을 출력하는 함수를 다음과 같이 작성해 보자.

```
def say_nick(nick):
    if nick == '바보':
        raise MyError()
    print(nick)
```

그리고 다음과 같이 say_nick 함수를 호출해 보자.

```
say_nick("천사")
say_nick("바보")
```

저장한 후 프로그램을 실행해 보면 다음과 같이 "천사"가 한 번 출력된 후 MyError가 발생한다.

```
천사
Traceback (most recent call last):
  File "...", line 11, in <module>
    say_nick("바보")
  File "...", line 7, in say_nick
    raise MyError()
__main__.MyError
```

예외 만들기

이번에는 예외 처리 기법을 사용하여 MyError 발생을 예외 처리해 보자.

```
try:
    say_nick("천사")
    say_nick("바보")
except MyError:
    print("허용되지 않는 별명입니다.")
```

프로그램을 실행하면 다음과 같이 출력된다.

```
천사
허용되지 않는 별명입니다.
```

만약 오류 메시지를 사용하고 싶다면 다음처럼 예외 처리를 하면 된다.

```
try:
    say_nick("천사")
    say_nick("바보")
except MyError as e:
    print(e)
```

하지만 프로그램을 실행해 보면 `print(e)` 로 오류 메시지가 출력되지 않는 것을 확인할 수 있다. 오류 메시지를 출력했을 때 오류 메시지가 보이게 하려면 오류 클래스에 다음과 같은 `__str__` 메서드를 구현해야 한다. `__str__` 메서드는 `print(e)` 처럼 오류 메시지를 print 문으로 출력할 경우에 호출되는 메서드이다.

```
class MyError(Exception):
    def __str__(self):
        return "허용되지 않는 별명입니다."
```

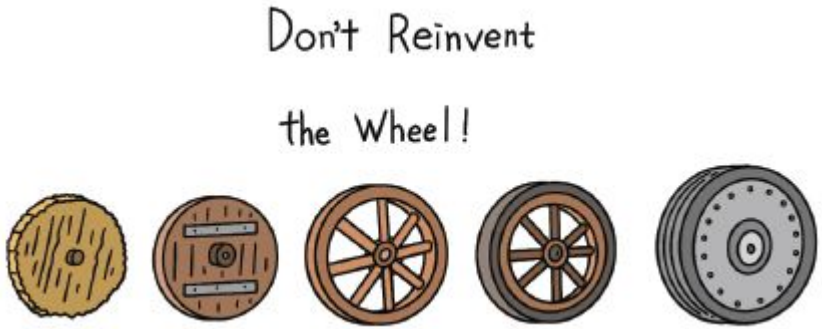
프로그램을 다시 실행해 보면 "허용되지 않는 별명입니다."라는 오류 메시지가 출력되는 것을 확인할 수 있을 것이다.

내장 함수

지금까지 파이썬으로 프로그래밍하기 위해 알아야 하는 것들을 대부분 공부했다. 이제 여러분은 원하는 프로그램을 직접 만들 수 있을 것이다. 하지만 그 전에 먼저 여러분이 만들려는 프로그램이 이미 만들어져 있는지 살펴보기 바란다.

물론 공부를 위해서라면 누군가 만들어 놓은 프로그램을 또 만들 수도 있다. 하지만 그런 목적이 아니라면 이미 만들어진 것을 다시 만드는 것은 불필요한 행동이다. 그리고 이미 만들어진 프로그램은 테스트 과정을 수없이 거쳤기 때문에 충분히 검증되어 있다. 따라서 무엇인가 새로운 프로그램을 만들기 전에는 이미 만들어진 것들, 그중에서도 특히 파이썬 배포본에 함께 들어 있는 파이썬 라이브러리를 살펴보는 것이 매우 중요하다.

내장함수 란?



라이브러리를 살펴보기 전에 파이썬 내장(built-in) 함수를 먼저 살펴보자. 우리는 이미 몇 가지 내장 함수를 배웠다. print, del, type 등이 바로 그것이다. 이러한 파이썬 내장 함수는 파이썬 모듈과 달리 import가 필요하지 않기 때문에 아무런 설정 없이 바로 사용할 수 있다.

이 책에서는 활용 빈도가 높고 중요한 함수를 중심으로 알파벳 순서대로 간략히 정리했다. 파이썬으로 프로그래밍을 하기 위해 이들 함수를 지금 당장 모두 알아야 하는 것은 아니므로 가벼운 마음으로 천천히 살펴보자.

내장 함수

EST

abs

abs(x)는 어떤 숫자를 입력받았을 때 그 숫자의 절댓값을 리턴하는 함수이다.

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```

all

all(x)는 반복 가능한 데이터 x를 입력값으로 받으며 이 x의 요소가 모두 참이면 True, 거짓이 하나라도 있으면 False를 리턴한다.

> 반복 가능한 데이터란 for 문에서 사용할 수 있는 자료형을 의미한다. 리스트, 튜플, 문자열, 딕셔너리, 집합 등이 있다.

다음 예를 살펴보자.

```
>>> all([1, 2, 3])
True
```

리스트 [1, 2, 3]은 모든 요소가 참이므로 True를 리턴한다.

```
>>> all([1, 2, 3, 0])
False
```

리스트 [1, 2, 3, 0] 중에서 요소 0은 거짓이므로 False를 리턴한다.

```
>>> all([])
True
```

만약 all의 입력 인수가 빈 값인 경우에는 True를 리턴한다.

> 자료형의 참과 거짓은 02-7을 참고하기 바란다.

any

any(x)는 반복 가능한 데이터 x를 입력으로 받아 x의 요소 중 하나라도 참이 있으면 True를 리턴하고 x가 모두 거짓일 때만 False를 리턴한다. 즉, all(x)의 반대로 작동한다.

다음 예를 살펴보자.

```
>>> any([1, 2, 3, 0])
True
```

리스트 [1, 2, 3, 0] 중에서 1, 2, 3이 참이므로 True를 리턴한다.

```
>>> any([0, ""])
False
```

리스트 [0, ""]의 요소 0과 ""은 모두 거짓이므로 False를 리턴한다.

```
>>> any([])
False
```

만약 any의 입력 인수가 빈 값인 경우에는 False를 리턴한다.

chr

chr(i)는 유니코드 숫자 값을 입력받아 그 코드에 해당하는 문자를 리턴하는 함수이다.

> 유니코드는 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준 코드이다

```
>>> chr(97)
'a'
>>> chr(44032)
'가'
```

내장 함수

EST

dir

dir은 객체가 지닌 변수나 함수를 보여 주는 함수이다. 다음 예는 리스트와 딕셔너리가 지닌 함수(메서드)를 보여 주는 예이다. 우리가 02장에서 살펴본 함수들을 구경할 수 있다.

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```

divmod

divmod(a, b)는 2개의 숫자 a, b를 입력으로 받는다. 그리고 a를 b로 나눈 몫과 나머지를 튜플로 리턴한다.

```
>>> divmod(7, 3)
(2, 1)
```

몫을 구하는 연산자 `//`와 나머지를 구하는 연산자 `%`를 각각 사용한 결과와 비교해 보자.

```
>>> 7 // 3
2
>>> 7 % 3
1
```

eval

eval(expression)은 문자열로 구성된 표현식을 입력으로 받아 해당 문자열을 실행한 결괏값을 리턴하는 함수이다.

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```

내장 함수

EST

enumerate

enumerate는 '열거하다'라는 뜻이다. 이 함수는 순서가 있는 데이터(리스트, 튜플, 문자열)를 입력으로 받아 인덱스 값을 포함하는 enumerate 객체를 리턴한다.

> 보통 enumerate 함수는 for 문과 함께 사용한다.

다음 예를 살펴보자.

```
>>> for i, name in enumerate(['body', 'foo', 'bar']):
...     print(i, name)
...
0 body
1 foo
2 bar
```

인덱스 값과 함께 body, foo, bar가 순서대로 출력되었다. 즉, enumerate를 for 문과 함께 사용하면 자료형의 현재 순서index와 그 값을 쉽게 알 수 있다.

for 문처럼 반복되는 구간에서 객체가 현재 어느 위치에 있는지 알려 주는 인덱스 값이 필요할 때 enumerate 함수를 사용하면 매우 유용하다.

내장 함수

EST

filter

filter란 '무엇인가를 걸러 낸다'라는 뜻으로, filter 함수도 이와 비슷한 기능을 한다.

```
filter(함수, 반복_가능한_데이터)
```

filter 함수는 첫 번째 인수로 함수, 두 번째 인수로 그 함수에 차례로 들어갈 반복 가능한 데이터를 받는다. 그리고 반복 가능한 데이터의 요소 순서대로 함수를 호출했을 때 리턴값이 참인 것만 묶어서(걸러 내서) 리턴한다.

다음 예를 살펴보자.

```
# positive.py
def positive(l):
    result = []
    for i in l:
        if i > 0:
            result.append(i)
    return result

print(positive([1,-3,2,0,-5,6]))
```

[실행 결과]

```
[1, 2, 6]
```

위에서 만든 positive는 리스트를 입력으로 받아 각각의 요소를 판별해서 양수 값만 리턴하는 함수이다.

filter 함수를 사용하면 위 내용을 다음과 같이 간단하게 작성할 수 있다.

```
# filter1.py
def positive(x):
    return x > 0

print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
```

[실행 결과]

```
[1, 2, 6]
```

`filter(positive, [1, -3, 2, 0, -5, 6])`은 [1, -3, 2, 0, -5, 6]의 각 요솟값을 순서대로 positive 함수에 적용하여 리턴값이 참인 것만 묶어서 리턴한다. 즉, 1, 2, 6 요소만 `x > 0` 문장에 참이 되므로 [1, 2, 6]이라는 결괏값이 출력된다.

> list 함수는 filter 함수의 리턴값을 리스트로 출력하기 위해 사용했다.

이 예제는 lambda를 사용하면 더욱 간단해진다.

```
>>> list(filter(lambda x: x > 0, [1, -3, 2, 0, -5, 6]))
[1, 2, 6]
```

내장 함수

EST

hex

hex(x)는 정수를 입력받아 16진수(hexadecimal) 문자열로 변환하여 리턴하는 함수이다.

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

id

id(object)는 객체를 입력받아 객체의 고유 주솟값(레퍼런스)을 리턴하는 함수이다.

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

위 예의 3, a, b는 고유 주솟값이 모두 135072304이다. 즉, 3, a, b가 모두 같은 객체를 가리키고 있다.

만약 id(4)라고 입력하면 4는 3, a, b와 다른 객체이므로 당연히 다른 고유 주솟값이 출력된다.

```
>>> id(4)
135072292
```

input

`input([prompt])` 는 사용자 입력을 받는 함수이다. 입력 인수로 문자열을 전달하면 그 문자열은 프롬프트가 된다.

> []는 괄호 안의 내용을 생략할 수 있다는 관례 표기법이라는 것을 기억하자.

```
>>> a = input()
hi
>>> a
'hi'
>>> b = input("Enter: ")
Enter: hi
>>> b
'hi'
```

int

int(x)는 문자열 형태의 숫자나 소수점이 있는 숫자를 정수로 리턴하는 함수이다. 만약 정수가 입력되면 그대로 리턴한다.

```
>>> int('3')
3
>>> int(3.4)
3
```

int(x, radix)는 radix 진수로 표현된 문자열 x를 10진수로 변환하여 리턴한다. 예를 들어 2진수로 표현된 '11'의 10진수 값은 다음과 같이 구할 수 있다.

```
>>> int('11', 2)
3
```

16진수로 표현된 '1A'의 10진수 값은 다음과 같이 구할 수 있다.

```
>>> int('1A', 16)
26
```

내장 함수

EST

isinstance

isinstance(object, class) 함수는 첫 번째 인수로 객체, 두 번째 인수로 클래스를 받는다. 입력으로 받은 객체가 그 클래스의 인스턴스인지를 판단하여 참이면 True, 거짓이면 False를 리턴한다.

```
>>> class Person: pass
...
>>> a = Person()
>>> isinstance(a, Person)
True
```

위 예는 a 객체가 Person 클래스에 의해 생성된 인스턴스라는 것을 확인시켜 준다.

```
>>> b = 3
>>> isinstance(b, Person)
False
```

b는 Person 클래스로 만든 인스턴스가 아니므로 False를 리턴한다.

len

len(s)는 입력값 s의 길이(요소의 전체 개수)를 리턴하는 함수이다.

```
>>> len("python")
6
>>> len([1,2,3])
3
>>> len((1, 'a'))
2
```

list

list(iterable)은 반복 가능한 데이터를 입력받아 리스트로 만들어 리턴하는 함수이다.

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

list 함수에 리스트를 입력하면 똑같은 리스트를 복사하여 리턴한다.

```
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```

내장 함수

EST

map

map(f, iterable)은 함수(f)와 반복 가능한 데이터를 입력으로 받는다. map은 입력받은 데이터의 각 요소에 함수 f를 적용한 결과를 리턴하는 함수이다.

다음 예를 살펴보자.

```
# two_times.py
def two_times(numberList):
    result = []
    for number in numberList:
        result.append(number*2)
    return result

result = two_times([1, 2, 3, 4])
print(result)
```

two_times는 리스트를 입력받아 리스트의 각 요소에 2를 곱해 리턴하는 함수이다. 실행 결과는 다음과 같다.

```
[2, 4, 6, 8]
```

위 예제는 map 함수를 사용하여 다음처럼 바꿀 수 있다.

```
>>> def two_times(x):
...     return x*2
...
>>> list(map(two_times, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

이 예제를 해석해 보자. 먼저 리스트의 첫 번째 요소인 1이 two_times 함수의 입력값으로 들어가고 `1 * 2`의 과정을 거쳐서 2가 된다. 다음으로 리스트의 두 번째 요소인 2가 `2 * 2`의 과정을 거쳐 4가 된다. 따라서 결괏값은 이제 [2, 4]가 된다. 총 4개의 요솟값이 모두 수행되면 [2, 4, 6, 8]이 된다. 이것이 map 함수가 하는 일이다.

> map 함수의 결과를 리스트로 출력하기 위해 list 함수를 사용했다. map 함수는 map 객체를 리턴한다.

앞의 예는 lambda를 사용하여 다음처럼 간략하게 만들 수 있다.

```
>>> list(map(lambda a: a*2, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

내장 함수

EST

max

max(iterable)은 인수로 반복 가능한 데이터를 입력받아 그 최댓값을 리턴하는 함수이다.

```
>>> max([1, 2, 3])
3
>>> max("python")
'y'
```

min

min(iterable)은 max 함수와 반대로, 인수로 반복 가능한 데이터를 입력받아 그 최솟값을 리턴하는 함수이다.

```
>>> min([1, 2, 3])
1
>>> min("python")
'h'
```

oct

oct(x)는 정수를 8진수 문자열로 바꾸어 리턴하는 함수이다

```
>>> oct(34)
'0o42'
>>> oct(12345)
'0o30071'
```

내장 함수

EST

open

`open(filename, [mode])` 은 '파일 이름'과 '읽기 방법'을 입력받아 파일 객체를 리턴하는 함수이다. 읽기 방법(mode)을 생략하면 기본값인 읽기 모드(r)로 파일 객체를 만들어 리턴한다.

mode	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

b는 w, r, a와 함께 사용한다. 예를 들어 rb는 '바이너리 읽기 모드'를 의미한다.

```
>>> f = open("binary_file", "rb")
```

ord

ord(c)는 문자의 유니코드 숫자 값을 리턴하는 함수이다.

> ord 함수는 chr 함수와 반대로 동작한다.

```
>>> ord('a')
97
>>> ord('가')
44032
```

내장 함수

EST

pow

pow(x, y)는 x를 y제곱한 결괏값을 리턴하는 함수이다.

```
>>> pow(2, 4)
16
>>> pow(3, 3)
27
```

range

range([start,] stop [,step])은 for 문과 함께 자주 사용하는 함수이다. 이 함수는 입력받은 숫자에 해당하는 범위 값을 반복 가능한 객체로 만들어 리턴한다.

인수가 하나일 경우

시작 숫자를 지정해 주지 않으면 range 함수는 0부터 시작한다.

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

인수가 2개일 경우

입력으로 주어지는 2개의 인수는 시작 숫자와 끝 숫자를 나타낸다. 단, 끝 숫자는 해당 범위에 포함되지 않는다는 것에 주의하자.

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
```

인수가 3개일 경우

세 번째 인수는 숫자 사이의 거리를 말한다.

```
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

round

`round(number [,ndigits])` 는 숫자를 입력받아 반올림해 리턴하는 함수이다.

> `[,ndigits]` 는 ndigits가 있을 수도 있고, 없을 수도 있다는 의미이다.

```
>>> round(4.6)
5
>>> round(4.2)
4
```

다음과 같이 실수 5.678을 소수점 2자리까지만 반올림하여 표시할 수 있다.

```
>>> round(5.678, 2)
5.68
```

round 함수의 두 번째 인수는 반올림하여 표시하고 싶은 소수점의 자릿수(ndigits)를 의미한다.

sorted

sorted(iterable)는 입력 데이터를 정렬한 후 그 결과를 리스트로 리턴하는 함수이다.

```
>>> sorted([3, 1, 2])
[1, 2, 3]
>>> sorted(['a', 'c', 'b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3, 2, 1))
[1, 2, 3]
```

리스트 자료형에도 sort 함수가 있다. 하지만 리스트 자료형의 sort 함수는 리스트 객체 그 자체를 정렬만 할 뿐, 정렬된 결과를 리턴하지는 않는다.

내장 함수

EST

str

str(object)는 문자열 형태로 객체를 변환하여 리턴하는 함수이다.

```
>>> str(3)
'3'
>>> str('hi')
'hi'
```

sum

sum(iterable)은 입력 데이터의 합을 리턴하는 함수이다.

```
>>> sum([1,2,3])
6
>>> sum((4,5,6))
15
```

tuple

tuple(iterable)은 반복 가능한 데이터를 튜플로 바꾸어 리턴하는 함수이다. 만약 입

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
```

type

type(object)는 입력값의 자료형이 무엇인지 알려 주는 함수이다.

```
>>> type("abc")
<class 'str'>
>>> type([ ])
<class 'list'>
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'>
```

zip

`zip(*iterable)` 은 동일한 개수로 이루어진 데이터들을 묶어서 리턴하는 함수이다.

> 여기서 사용한 `*iterable` 은 반복 가능한 데이터를 여러 개 입력할 수 있다는 의미이다.

다음 예제로 사용법을 확인해 보자.

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

EST

표준 라이브러리

이제 파이썬 프로그래밍 능력을 높여 줄 더 큰 날개를 달아 보자. 전 세계의 파이썬 고수들이 만든 유용한 프로그램을 모아 놓은 것이 바로 파이썬 표준 라이브러리이다. '라이브러리'는 '도서관'이라는 뜻 그대로 원하는 정보를 찾아보는 곳이다. 모든 라이브러리를 다 알 필요는 없고 어떤 일을 할 때 어떤 라이브러리를 사용해야 한다는 정도만 알면 된다. 이를 위해 어떤 라이브러리가 존재하고 어떻게 사용하는지 알아야 한다. 자주 사용되고 꼭 알아 두면 좋은 라이브러리를 중심으로 하나씩 살펴보자.

* 파이썬 표준 라이브러리는 파이썬을 설치할 때 자동으로 컴퓨터에 설치된다.

(sys, re모듈은 고급으로 미포함)

표준 라이브러리

EST

datetime.date는 연, 월, 일로 날짜를 표현할 때 사용하는 함수이다.

만약 A 군과 B 양이 2021년 12월 14일부터 만나기 시작했다면 2023년 4월 5일은 둘이 사귄 지 며칠째 되는 날일까? 아울러 사귀기 시작한 2021년 12월 14일은 무슨 요일이었을까? datetime.date 함수를 사용하면 이 문제를 쉽게 해결할 수 있다.

연, 월, 일로 다음과 같이 datetime.date 객체를 만들 수 있다.

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

이처럼 연, 월, 일을 인수로 하여 2021년 12월 14일에 해당하는 날짜 객체는 day1, 2023년 4월 5일에 해당하는 날짜 객체는 day2로 생성하였다. 이렇게 날짜 객체를 만들었다면 두 날짜의 차이는 다음과 같이 뺄셈으로 쉽게 구할 수 있다.

```
>>> diff = day2 - day1
>>> diff.days
477
```

day2에서 day1을 빼면 datetime 모듈의 timedelta 객체가 리턴된다. 이 객체를 diff 변수에 대입하고 이 diff 변수를 이용하여 두 날짜의 차이를 쉽게 확인해 봤다.

요일은 datetime.date 객체의 weekday 함수를 사용하면 쉽게 구할 수 있다.

```
>>> day = datetime.date(2021, 12, 14)
>>> day.weekday()
1
```

0은 월요일을 의미하며 순서대로 1은 화요일, 2는 수요일, …, 6은 일요일이 된다. 이와 달리 월요일은 1, 화요일은 2, …, 일요일은 7을 리턴하려면 다음처럼 isoweekday 함수를 사용하면 된다.

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

2021년 12월 14일은 화요일이므로 isoweekday()를 사용하면 화요일을 뜻하는 2가 리턴된다. weekday()를 사용하면 1이 리턴된다.

표준 라이브러리

EST

time

시간과 관련된 time 모듈에는 함수가 매우 많다. 그중 가장 유용한 몇 가지만 알아보자.

time.time

time.time()은 UTC(universal time coordinated, 협정 세계 표준시)를 사용하여 현재 시간을 실수 형태로 리턴하는 함수이다. 1970년 1월 1일 0시 0분 0초를 기준으로 지난 시간을 초 단위로 리턴해 준다.

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

time.localtime

time.localtime은 time.time()이 리턴한 실숫값을 사용해서 연, 월, 일, 시, 분, 초, ... 의 형태로 바꾸어 주는 함수이다.

```
>>> 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.asctime

time.asctime은 time.localtime가 리턴된 튜플 형태의 값을 인수로 받아서 날짜와 시간을 알아보기 쉬운 형태로 리턴하는 함수이다.

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

time.ctime

`time.asctime(time.localtime(time.time()))` 은 간단하게 time.ctime()으로 표시할 수 있다. ctime이 asctime과 다른 점은 항상 현재 시간만을 리턴한다는 점이다.

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

표준 라이브러리

EST

time.strftime

strftime 함수는 시간에 관계된 것을 세밀하게 표현하는 여러 가지 포맷 코드를 제공한다.

```
time.strftime('출력할 형식 포맷 코드', time.localtime(time.time()))
```

다음은 time.strftime을 사용하는 예이다.

```
>>> import time
>>> time.strftime('%x', time.localtime(time.time()))
'05/25/23'
>>> time.strftime('%c', time.localtime(time.time()))
'Thu May 25 10:13:52 2023
```

포맷코드	설명	예
%a	요일의 줄임말	Mon
%A	요일	Monday
%b	달의 줄임말	Jan
%B	달	January
%c	날짜와 시간을 출력함.	Thu May 25 10:13:52 2023
%d	일(day)	[01,31]
%H	시간(hour): 24시간 출력 형태	[00,23]
%I	시간(hour): 12시간 출력 형태	[01,12]
%j	1년 중 누적 날짜	[001,366]
%m	달	[01,12]
%M	분	[01,59]
%p	AM or PM	AM
%S	초	[00,59]
%U	1년 중 누적 주(일요일 시작)	[00,53]
%w	숫자로 된 요일	[0(일), 6(토)]
%W	1년 중 누적 주(월요일 시작)	[00,53]
%x	현재 설정된 지역에 기반한 날짜 출력	05/25/23
%X	현재 설정된 지역에 기반한 시간 출력	17:22:21
%Y	연도 출력	2023
%Z	시간대 출력	대한민국 표준시
%%	문자 %	%
%y	세기 부분을 제외한 연도 출력	01

표준 라이브러리

EST

time.sleep

time.sleep 함수는 주로 루프 안에서 많이 사용한다. 이 함수를 사용하면 일정한 시간 간격을 두고 루프를 실행할 수 있다. 다음 예를 살펴보자.

```
# sleep1.py
import time
for i in range(10):
    print(i)
    time.sleep(1)
```

위 예는 1초 간격으로 0부터 9까지의 숫자를 출력한다. time.sleep 함수의 인수는 실수 형태를 쓸 수 있다. 즉 1이면 1초, 0.5면 0.5초가 되는 것이다.

점프 투 파이썬

인수 없이 time 함수 사용하기

time.localtime, time.asctime, time.strftime 함수는 다음처럼 입력 인수 없이 사용할 수 있다. 입력 인수 없이 사용할 경우 현재 시각을 기준으로 함수가 수행된다.

```
>>> time.localtime()
time.struct_time(tm_year=2023, tm_mon=6, tm_mday=18, tm_hour=17, tm_min=1, tm_sec=4, tm_wday=6, tm_yday=
169, tm_isdst=0)
>>> time.asctime()
'Sun Jun 18 17:01:09 2023'
>>> time.strftime('%c')
'Sun Jun 18 17:01:30 2023'
```

표준 라이브러리

EST

math.gcd

math.gcd 함수를 이용하면 최대 공약수(gcd, greatest common divisor)를 쉽게 구할 수 있다.

> - math.gcd 함수는 파이썬 3.5 버전부터 사용할 수 있다.
> - 공약수란 두 수 이상의 여러 수의 공통된 약수를 말하며 공약수 중 가장 큰 수를 최대 공약수라고 말한다. 예를 들어 30과 15의 약수는 1, 3, 5, 15, 최대 공약수는 15이다.

어린이집에서 사탕 60개, 초콜릿 100개, 젤리 80개를 준비했다. 아이들이 서로 싸우지 않도록 똑같이 나누어 봉지에 담는다고 하면 최대 몇 봉지까지 만들 수 있을까? 단, 사탕, 초콜릿, 젤리는 남기지 않고 모두 담도록 한다.

이 문제는 60, 100, 80의 최대 공약수를 구하면 바로 해결된다. 즉, 똑같이 나눌 수 있는 봉지 개수가 최대가 되는 수를 구하면 된다.

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

> 파이썬 3.9 버전부터는 math.gcd에 여러 개의 인수를 입력할 수 있지만, 3.9 미만 버전에서는 2개까지만 허용된다.

math.gcd() 함수로 최대 공약수를 구했더니 20이었다. 따라서 최대 20봉지를 만들 수 있다. 각 봉지에 들어가는 사탕, 초콜릿, 젤리의 개수는 다음과 같이 전체 개수를 최대 공약수 20으로 나누면 구할 수 있다.

```
>>> 60/20, 100/20, 80/20
(3.0, 5.0, 4.0)
```

따라서 한 봉지당 사탕 3개씩, 초콜릿 5개씩, 젤리 4개씩 담으면 된다.

표준 라이브러리

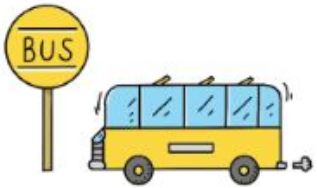
EST

math.lcm

math.lcm은 최소 공배수(lcm, least common multiple)를 구할 때 사용하는 함수이다.

> - math.lcm() 함수는 파이썬 3.9 버전부터 사용할 수 있다.
> - 최소 공배수란 두 수의 공통 배수 중 가장 작은 수를 말한다. 예를 들어 3과 5의 최소 공배수는 15이다.

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?



이 문제는 15와 25의 공통 배수 중 가장 작은 수, 즉 최소 공배수를 구하면 바로 해결된다.

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

math.lcm 함수를 사용하여 최소 공배수 75를 구했다. 따라서 두 버스가 동시에 도착할 다음 시각은 75분 후인 오후 2시 15분이다.

표준 라이브러리

EST

random

random은 난수(규칙이 없는 임의의 수)를 발생시키는 모듈이다. 먼저 random과 randint 함수에 대해 알아보자.

다음은 0.0에서 1.0 사이의 실수 중에서 난수 값을 리턴하는 예를 보여 준다.

```
>>> import random
>>> random.random()
0.5384010330509867 4
```

다음 예는 1에서 10 사이의 정수 중에서 난수 값을 리턴해 준다.

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

다음 예는 1에서 55 사이의 정수 중에서 난수 값을 리턴해 준다.

```
>>> random.randint(1, 55)
43
```

random 모듈을 사용해서 재미있는 함수를 하나 만들어 보자.

```
# random_pop.py
import random
def random_pop(data):
    number = random.randint(0, len(data)-1)
    return data.pop(number)

if __name__ == "__main__":
    data = [1, 2, 3, 4, 5]
    while data:
        print(random_pop(data))
```

[실행 결과]

```
2
3
1
5
4
```

앞에서 만든 random_pop 함수는 리스트의 요소 중에서 무작위로 하나를 선택하여 꺼낸 다음 그 값을 리턴한다. 물론 꺼낸 요소는 pop 메서드에 의해 사라진다.

random_pop 함수는 random 모듈의 choice 함수를 사용하여 다음과 같이 좀 더 직관적으로 만들 수도 있다.

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

random.choice 함수는 입력으로 받은 리스트에서 무작위로 하나를 선택하여 리턴한다.

리스트의 항목을 무작위로 섞고 싶을 때는 random.sample 함수를 사용하면 된다.

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

random.sample 함수에서 두 번째 인수인 `len(data)` 는 무작위로 추출할 원소의 개수를 의미한다. 만약 `random.sample(data, 3)` 과 같이 사용한다면 data 리스트에서 무작위로 3개를 추출하여 리턴할 것이다.

표준 라이브러리

EST

itertools.zip_longest

`itertools.zip_longest(*iterables, fillvalue=None)` 함수는 같은 개수의 자료형을 묶는 파이썬 내장 함수인 zip 함수와 똑같이 동작한다. 하지만 `itertools.zip_longest()` 함수는 전달한 반복 가능 객체(`*iterables`)의 길이가 서로 다르다면 긴 객체의 길이에 맞춰 fillvalue에 설정한 값을 짧은 객체에 채울 수 있다.

예시로 유치원생 5명에게 간식을 나누어 주고자 다음과 같은 파이썬 코드를 작성해 보자.

```
# itertools_zip.py
students = ['한민서', '황지민', '이영철', '이광수', '김승민']
snacks = ['사탕', '초컬릿', '젤리']

result = zip(students, snacks)
print(list(result))
```

간식의 개수가 유치원생보다 적으므로 이 파이썬 코드를 실행하면 다음과 같은 결과가 나온다.

```
[('한민서', '사탕'), ('황지민', '초컬릿'), ('이영철', '젤리')]
```

students와 snacks의 요소 개수가 다르므로 더 적은 snacks의 개수만큼만 zip()으로 묶게 된다.

students의 요소 개수가 snacks보다 많을 때 그만큼을 '새우깡'으로 채우려면 어떻게 해야 할까? 이럴 때 요소 개수가 많은 것을 기준으로 자료형을 묶는 itertools.zip_longest()를 사용하면 된다. 부족한 항목은 None으로 채우는데, 다음처럼 fillvalue로 값을 지정하면 None 대신 다른 값으로 채울 수 있다.

```
# itertools_zip.py
import itertools

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

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

실행 결과는 다음과 같다.

```
[('한민서', '사탕'), ('황지민', '초콜릿'), ('이영철', '젤리'), ('이광수', '새우깡'), ('김승민', '새우깡')]
```

표준 라이브러리

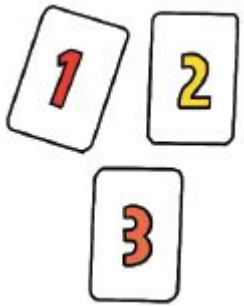
EST

itertools.permutation

`itertools.permutations(iterable, r)` 은 반복 가능 객체 중에서 r개를 선택한 순열을 이터레이터로 리턴하는 함수이다.

> 이터레이터란 반복 가능한 객체를 의미한다.

1, 2, 3이라는 숫자가 적힌 3장의 카드에서 2장의 카드를 꺼내 만들 수 있는 2자리 숫자를 모두 구하려면 어떻게 해야 할까?



조합을 사용하는 함수

3장의 카드에서 순서에 상관없이 2장을 고르는 조합은 다음처럼 itertools.combinations()를 사용하면 된다.

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

[1, 2, 3]이라는 3장의 카드 중 순서에 상관없이 2장을 뽑는 경우의 수는 모두 3가지이다(조합).

- 1, 2
- 2, 3
- 1, 3

하지만 이 문제에서는 2자리 숫자이므로 이 3가지에 순서를 더해 다음처럼 6가지가 된다(순열).

- 1, 2
- 2, 1
- 2, 3
- 3, 2
- 1, 3
- 3, 1

이 순열은 itertools.permutations()를 사용하면 간단히 구할 수 있다.

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

따라서 만들 수 있는 2자리 숫자는 다음과 같이 모두 6가지이다.

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

표준 라이브러리

EST

itertools.combination

`itertools.combinations(iterable, r)` 은 반복 가능 객체 중에서 r개를 선택한 조합을 이터레이터로 리턴하는 함수이다.

1~45 중 서로 다른 숫자 6개를 뽑는 로또 번호의 모든 경우의 수(조합)를 구하고 그 개수를 출력하려면 어떻게 해야 할까?

다음과 같이 itertools.combinations()를 사용하면 45개의 숫자 중 6개를 선택하는 경우의 수를 구할 수 있다.

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

`itertools.combinations(range(1, 46), 6)` 은 1~45의 숫자 중에서 6개를 뽑는 경우의 수를 이터레이터로 리턴한다.

이터레이터 객체를 루프를 이용하여 출력하면 아마 끝도 없이 출력될 것이다. 궁금하다면 직접 실행해 봐도 좋다.

```
>>> for num in it:
...     print(num)
...
(1, 2, 3, 4, 5, 6)
(1, 2, 3, 4, 5, 7)
(1, 2, 3, 4, 5, 8)
(1, 2, 3, 4, 5, 9)
(1, 2, 3, 4, 5, 10)
(1, 2, 3, 4, 5, 11)
(1, 2, 3, 4, 5, 12)
(1, 2, 3, 4, 5, 13)
...
```

하지만 순환하여 출력하지 않고 이터레이터의 개수만 세려면 다음과 같이 하면 된다.

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

선택할 수 있는 로또 번호의 가짓수는 8,145,060이다.

> 여러분이 반드시 로또에 당첨되길 희망한다면 서로 다른 번호로 구성한 8,145,060장의 로또를 사면 된다. 게임 1번에 1,000원이라고 할 때 그 금액은 무려 81억 4,506만 원이다.

점프 투 파이썬

중복 조합을 사용하는 함수

만약 로또 복권이 숫자 중복을 허용하도록 규칙이 변경된다면 경우의 수는 몇 개가 될까?

> 중복이 허용된다는 것은 당첨 번호가 [1, 2, 3, 4, 5, 5]처럼 5가 2번 이상 나와도 되고 [1, 1, 1, 1, 1, 1]처럼 1이 6번 나와도 된다는 의미이다.

같은 숫자를 허용하는 중복 조합은 itertools.combinations_with_replacement()를 사용하면 된다.

```
>>> len(list(itertools.combinations_with_replacement(range(1, 46), 6)))
15890700
```

당연히 중복을 허용하지 않을 때보다 훨씬 많은 경우의 수가 리턴되는 것을 확인할 수 있다.

표준 라이브러리

EST

functools.reduce

`functools.reduce(function, iterable)` 은 함수(function)를 반복 가능한 객체(iterable)의 요소에 차례대로(왼쪽에서 오른쪽으로) 누적 적용하여 이 객체를 하나의 값으로 줄이는 함수이다.

다음은 입력 인수 data의 요소를 모두 더하여 리턴하는 add 함수이다.

```
def add(data):
    result = 0
    for i in data:
        result += i
    return result

data = [1, 2, 3, 4, 5]
result = add(data)
print(result)  # 15 출력
```

functools.reduce()를 사용하여 마찬가지로 동작하는 코드를 작성하려면 어떻게 해야 할까? functools.reduce()를 사용한 코드는 다음과 같다.

```
# reduce_test.py
import functools

data = [1, 2, 3, 4, 5]
result = functools.reduce(lambda x, y: x + y, data)
print(result)  # 15 출력
```

functools.reduce()를 사용하면 reduce()에 선언한 람다 함수를 data 요소에 차례대로 누적 적용하여 다음과 같이 계산한다.

```
((((1+2)+3)+4)+5)
```

따라서 앞서 본 add 함수와 동일한 역할을 하게 된다.

점프 투 파이썬

functools.reduce()로 최댓값 구하기

functools.reduce 함수로 최댓값도 구할 수 있다.

```
num_list = [3, 2, 8, 1, 6, 7]
max_num = functools.reduce(lambda x, y: x if x > y else y, num_list)
print(max_num)  # 8 출력
```

[3, 2, 8, 1, 6, 7] 요소를 차례대로 reduce()의 람다 함수로 전달하여 두 값 중 큰 값을 선택하고 마지막에 남은 최댓값을 리턴한다.

> 최솟값은 `functools.reduce(lambda x, y: x if x < y else y, num_list)` 로 구할수 있다.

표준 라이브러리

EST

operator.itemgetter

operator.itemgetter는 주로 sorted와 같은 함수의 key 매개변수에 적용하여 다양한 기준으로 정렬할 수 있도록 도와주는 모듈이다.

예를 들어 학생의 이름, 나이, 성적 등의 정보를 저장한, 다음과 같은 students 리스트가 있다고 가정해 보자.

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

students 리스트에는 3개의 튜플이 있으며 각 튜플은 순서대로 이름, 나이, 성적에 해당하는 데이터로 이루어졌다. 이 리스트를 나이순으로 정렬하려면 어떻게 해야 할까?

이 문제는 다음처럼 sorted 함수의 key 매개변수에 itemgetter()를 적용하면 쉽게 해결할 수 있다.

```
# itemgetter1.py
from operator import itemgetter

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

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

이 파일을 실행하여 출력해 보면 다음과 같이 나이 순서대로 정렬한 것을 확인할 수 있다.

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

itemgetter(1)은 students의 아이템인 튜플의 두 번째 요소를 기준으로 정렬하겠다는 의미이다. 만약 itemgetter(2)와 같이 사용한다면 성적순으로 정렬한다.

이번에는 students의 요소가 다음처럼 딕셔너리일 때를 생각해 보자.

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

딕셔너리일 때도 마찬가지로 age를 기준으로 정렬해 보자. 이때도 마찬가지로 itemgetter()를 적용하면 된다. 단, 이번에는 `itemgetter('age')` 처럼 딕셔너리의 키를 사용해야 한다. `itemgetter('age')` 는 딕셔너리의 키인 age를 기준으로 정렬하겠다는 의미이다.

```
# itemgetter2.py
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)
```

출력 결과는 다음과 같이 age 순으로 정렬된 것을 확인할 수 있다.

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

표준 라이브러리

EST

operator.attrgetter()

students 리스트의 요소가 튜플이 아닌 Student 클래스의 객체라면 다음처럼 attrgetter()를 적용하여 정렬해야 한다.

```
# attrgetter1.py
from operator import attrgetter

class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

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

result = sorted(students, key=attrgetter('age'))
```

`attrgetter('age')` 는 Student 객체의 age 속성으로 정렬하겠다는 의미이다. 이와 마찬가지로 `attrgetter('grade')` 와 같이 사용하면 성적순으로 정렬한다.

표준 라이브러리

EST

shutil

shutil은 파일을 복사(copy)하거나 이동(move)할 때 사용하는 모듈이다.

작업 중인 파일을 자동으로 백업하는 기능을 구현하고자 `c:\doit\a.txt`를 `c:\temp\a.txt.bak`이라는 이름으로 복사하는 프로그램을 만들고자 한다. 어떻게 만들어야 할까? `c:\doit` 디렉터리에 a.txt를 만드는 중이며 백업용 `c:\temp` 디렉터리는 이미 만들었다고 가정한다.

다음은 shutil을 사용한 방법이다.

```
# shutil_copy.py
import shutil

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

점프 투 파이썬

shutil.move로 삭제 기능 만들기

휴지통으로 삭제하는 기능을 구현하고자 `c:\doit\a.txt` 파일을 `c:\temp\a.txt`로 이동하려면 다음과 같이 코드를 작성해야 한다.

```
# shutil_move.py
import shutil

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

표준 라이브러리

EST

glob

가끔 파일을 읽고 쓰는 기능이 있는 프로그램을 만들다 보면 특정 디렉터리에 있는 파일 이름 모두를 알아야 할 때가 있다. 이럴 때 사용하는 모듈이 바로 glob이다.

디렉터리에 있는 파일들을 리스트로 만들기 - glob(pathname)

glob 모듈은 디렉터리 안의 파일들을 읽어서 리턴한다. `*`, `?` 등 메타 문자를 써서 원하는 파일만 읽어 들일 수도 있다. 다음은 `c:/doit` 디렉터리에 있는 파일 중 이름이 mark로 시작하는 파일을 모두 찾아서 읽어들이는 예이다.

> `?`는 1자리 문자열, `*`은 임의의 길이의 문자열을 의미한다.

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

pickle

pickle은 객체의 형태를 그대로 유지하면서 파일에 저장하고 불러올 수 있게 하는 모듈이다. 다음 예는 pickle 모듈의 dump 함수를 사용하여 딕셔너리 객체인 data를 그대로 파일에 저장하는 방법을 보여 준다.

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

다음은 pickle.dump로 저장한 파일을 pickle.load를 사용해서 원래 있던 딕셔너리 객체(data) 상태 그대로 불러오는 예이다.

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

위 예에서는 딕셔너리 객체를 사용했지만, 어떤 자료형이든 저장하고 불러올 수 있다.

표준 라이브러리

EST

os

os 모듈은 환경 변수나 디렉터리, 파일 등의 OS 자원을 제어할 수 있게 해 주는 모듈이다.

내 시스템의 환경 변숫값을 알고 싶을 때 - os.environ

시스템은 제각기 다른 환경 변숫값을 가지고 있는데, os.environ은 현재 시스템의 환경 변숫값을 리턴한다. 다음을 따라 해 보자.

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

이 결괏값은 필자의 시스템 정보이다. os.environ은 환경 변수에 대한 정보를 딕셔너리 형태로 구성된 environ 객체로 리턴한다. 자세히 보면 여러 가지 유용한 정보를 찾을 수 있다.

리턴받은 객체는 다음과 같이 호출하여 사용할 수 있다. 다음은 필자 시스템의 PATH 환경 변수 내용이다.

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

디렉터리 위치 변경하기 - os.chdir

os.chdir를 사용하면 다음과 같이 현재 디렉터리의 위치를 변경할 수 있다.

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

디렉터리 위치 돌려받기 - os.getcwd

os.getcwd는 현재 자신의 디렉터리 위치를 리턴한다.

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

시스템 명령어 호출하기 - os.system

시스템 자체의 프로그램이나 기타 명령어를 파이썬에서 호출할 수도 있다. `os.system("명령어")`처럼 사용한다. 다음은 현재 디렉터리에서 시스템 명령어 dir을 실행하는 예이다.

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

실행한 시스템 명령어의 결괏값 돌려받기 - os.popen

os.popen은 시스템 명령어를 실행한 결괏값을 읽기 모드 형태의 파일 객체로 리턴한다.

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

읽어 들인 파일 객체의 내용을 보기 위해서는 다음과 같이 하면 된다.

```
>>> print(f.read())
```

이 밖에 유용한 os 관련 함수는 다음과 같다.

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.

표준 라이브러리

EST

zipfile

zipfile은 여러 개의 파일을 zip 형식으로 합치거나 이를 해제할 때 사용하는 모듈이다.

다음과 같은 3개의 텍스트 파일이 있다고 가정해 보자.

```
a.txt
b.txt
c.txt
```

이 3개의 텍스트 파일을 하나로 합쳐 'mytext.zip'이라는 파일을 만들고 이 파일을 원래의 텍스트 파일 3개로 해제하는 프로그램을 만들려면 어떻게 해야 할까?

zipfile.ZipFile()을 사용하여 해결해 보자.

```
# zipfile_test.py
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 객체의 write() 함수로 개별 파일을 추가할 수 있고 extreactall() 함수를 사용하면 모든 파일을 해제할 수 있다.

합친 파일에서 특정 파일만 해제하고 싶다면 다음과 같이 extract() 함수를 사용하면 된다.

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

만약 파일을 압축하여 묶고 싶은 경우에는 compression, compresslevel 옵션을 사용할 수 있다.

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

compression에는 4가지 종류가 있다.

- 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 알고리즘으로 알려져 있다).

compressionlevel은 압축 수준을 의미하는 숫자값으로, 1~9를 사용한다. 1은 속도가 가장 빠르지만 압축률이 낮고, 9는 속도는 가장 느리지만 압축률이 높다.

표준 라이브러리

EST

threading

스레드 프로그래밍은 초보 프로그래머가 구현하기에는 매우 어려운 기술이다. 여기에 잠시 소개했으므로 눈으로만 살펴보고 넘어가자.

컴퓨터에서 동작하고 있는 프로그램을 프로세스(process)라고 한다. 보통 1개의 프로세스는 1가지 일만 하지만, 스레드(thread)를 사용하면 한 프로세스 안에서 2가지 또는 그 이상의 일을 동시에 수행할 수 있다.

간단한 예제로 설명을 대신하겠다.

```
# thread_test.py
import time

def long_task():  # 5초의 시간이 걸리는 함수
    for i in range(5):
        time.sleep(1)  # 1초간 대기한다.
        print("working:%s\n" % i)

print("Start")

for i in range(5):  # long_task를 5회 수행한다.
    long_task()

print("End")
```

long_task는 수행하는 데 5초의 시간이 걸리는 함수이다. 위 프로그램은 이 함수를 총 5번 반복해서 수행하는 프로그램이다. 이 프로그램은 5초가 5번 반복되므로 총 25초의 시간이 걸린다.

하지만 앞에서 설명했듯이 스레드를 사용하면 5초의 시간이 걸리는 long_task 함수를 동시에 실행할 수 있으므로 시간을 줄일 수 있다.

표준 라이브러리

다음과 같이 프로그램을 수정해 보자.

```
# thread_test.py
import time
import threading  # 스레드를 생성하기 위해서는 threading 모듈이 필요하다.

def long_task():
    for i in range(5):
        time.sleep(1)
        print("working:%s\n" % i)

print("Start")

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

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

print("End")
```

이와 같이 프로그램을 수정하고 실행해 보면 25초 걸리던 작업이 5초 정도에 수행되는 것을 확인할 수 있다. threading.Thread를 사용하여 만든 스레드 객체가 동시 작업을 가능하게 해 주기 때문이다.

하지만 프로그램을 실행해 보면 "Start"와 "End"가 먼저 출력되고 그 이후에 스레드의 결과가 출력되는 것을 확인할 수 있다. 그리고 프로그램이 정상 종료되지 않는다. 우리가 기대하는 것은 "Start"가 출력되고 그다음에 스레드의 결과가 출력된 후 마지막으로 "End"가 출력되는 것이다.

이 문제를 해결하기 위해서는 프로그램을 다음과 같이 수정해야 한다.

```
# thread_test.py
import time
import threading

def long_task():
    for i in range(5):
        time.sleep(1)
        print("working:%s\n" % i)

print("Start")

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

for t in threads:
    t.start()

for t in threads:
    t.join()  # join으로 스레드가 종료될때까지 기다린다.

print("End")
```

스레드의 join 함수는 해당 스레드가 종료될 때까지 기다리게 한다. 따라서 위와 같이 수정하면 우리가 원하던 출력을 보게 된다.

표준 라이브러리

EST

tempfile

파일을 임시로 만들어서 사용할 때 유용한 모듈이 바로 tempfile이다. `tempfile.mkstemp()` 는 중복되지 않는 임시 파일의 이름을 무작위로 만들어서 리턴한다.

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

`tempfile.TemporaryFile()` 은 임시 저장 공간으로 사용할 파일 객체를 리턴한다. 이 파일은 기본적으로 바이너리 쓰기 모드(wb)를 갖는다. `f.close()` 가 호출되면 이 파일은 자동으로 삭제된다.

```
>>> import tempfile
>>> f = tempfile.TemporaryFile()
>>> f.close()
```

표준 라이브러리

EST

traceback

traceback은 프로그램 실행 중 발생한 오류를 추적하고자 할 때 사용하는 모듈이다.

다음과 같은 코드를 작성하여 실행해 보자.

```
# traceback_test.py
def a():
    return 1/0

def b():
    a()

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

main()
```

프로그램 실행 결과는 다음과 같다.

```
오류가 발생했습니다.
```

main() 함수가 시작되면 b() 함수를 호출하고 b() 함수에서 다시 a() 함수를 호출하여 1을 0으로 나누므로 오류가 발생하여 "오류가 발생했습니다."라는 메시지를 출력했다.

> 이렇게 간단한 프로그램이 아니라 복잡한 파이썬 코드라면 어디에서 어떤 오류가 발생하는지 알기 어렵다.

이때 이 코드에서 오류가 발생한 위치와 원인을 정확히 판단할 수 있도록 코드를 업그레이드하려면 어떻게 해야 할까?

오류가 발생한 위치에 다음과 같이 traceback 모듈을 적용해 보자.

```
# traceback_test.py
import traceback

def a():
    return 1/0

def b():
    a()

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

main()
```

오류가 발생한 위치에 `print(traceback.format_exc())` 문장을 추가했다. traceback 모듈의 format_exc()는 오류 추적 결과를 문자열로 리턴하는 함수이다. 이렇게 코드를 수정하고 다시 프로그램을 실행하면 다음과 같이 출력될 것이다.

```
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
```

오류 추적을 통해 main() 함수에서 b() 함수를 호출하고 b() 함수에서 다시 a() 함수를 호출하여 `1 / 0`을 실행하려 하므로 0으로 나눌 수 없다는 ZeroDivisionError가 발생했다는 것을 로그를 통해 정확하게 확인할 수 있다.

표준 라이브러리

EST

json은 JSON 데이터를 쉽게 처리하고자 사용하는 모듈이다.

다음은 개인정보를 JSON 형태의 데이터로 만든 myinfo.json 파일이다.

[파일명: myinfo.json]

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

인터넷으로 얻은 이 파일을 읽어 파이썬에서 처리할 수 있도록 딕셔너리 자료형으로 만들려면 어떻게 해야 할까?

JSON 파일을 읽어 딕셔너리로 변환하려면 다음처럼 json 모듈을 사용하면 된다.

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

표준 라이브러리

JSON 파일을 읽을 때는 이 예처럼 `json.load(파일_객체)` 를 사용한다. 이렇게 load() 함수는 읽은 데이터를 딕셔너리 자료형으로 리턴한다. 이와 반대로 딕셔너리 자료형을 JSON 파일로 생성할 때는 다음처럼 `json.dump(딕셔너리, 파일_객체)` 를 사용한다.

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

이번에는 파이썬 자료형을 JSON 문자열로 만드는 방법에 대해서 알아보자.

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

딕셔너리 자료형을 JSON 문자열로 만들려면 json.dumps() 함수를 사용하면 된다. 그런데 딕셔너리를 JSON 데이터로 변경하면 '홍길동'과 같은 한글 문자열이 코드 형태로 표시된다. 왜냐하면 dump(), dumps() 함수는 기본적으로 데이터를 아스키 형태로 저장하기 때문이다. 유니코드 문자열을 아스키 형태로 저장하다 보니 한글 문자열이 마치 깨진 것처럼 보인다.

그러나 JSON 문자열을 딕셔너리로 다시 역변환하여 사용하는 데는 전혀 문제가 없다. JSON 문자열을 딕셔너리로 변환할 때는 다음처럼 json.loads() 함수를 사용한다.

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

표준 라이브러리

한글 문자열이 아스키 형태의 문자열로 변경되는 것을 방지하는 방법도 있다.

```
>>> 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}
```

이처럼 `ensure_ascii=False` 옵션을 사용하면 된다. 이 옵션은 데이터를 저장할 때 아스키 형태로 변환하지 않겠다는 뜻이다.

출력되는 JSON 문자열을 보기 좋게 정렬하려면 다음처럼 indent 옵션을 추가하면 된다.

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

그리고 딕셔너리 외에 리스트나 튜플처럼 다른 자료형도 JSON 문자열로 바꿀 수 있다.

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

표준 라이브러리

EST

urllib

urllib은 URL을 읽고 분석할 때 사용하는 모듈이다.

브라우저로 위키독스의 특정 페이지를 읽으려면 다음과 같이 요청하면 된다.

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

그러면 오프라인으로도 읽을 수 있도록 페이지 번호를 입력받아 위키독스의 특정 페이지를 `wikidocs_페이지_번호.html` 파일로 저장하는 함수는 어떻게 만들어야 할까?

URL을 호출하여 원하는 리소스를 얻으려면 urllib 모듈을 사용해야 한다.

```
# urllib_test.py
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())
```

get_wikidocs(page)는 위키독스의 페이지 번호를 입력받아 해당 페이지의 리소스 내용을 파일로 저장하는 함수이다. 이 코드에서 보듯이 `urllib.request.urlopen(resource)` 로 s 객체를 생성하고 `s.read()` 로 리소스 내용 전체를 읽어 이를 저장할 수 있다. 예를 들어 `get_wikidocs(12)` 라고 호출하면 `https://wikidocs.net/12` 웹 페이지를 `wikidocs_12.html` 라는 파일로 저장한다.

표준 라이브러리

EST

webbrowser

webbrowser는 파이썬 프로그램에서 시스템 브라우저를 호출할 때 사용하는 모듈이다.

개발 중 궁금한 내용이 있어 파이썬 문서를 참고하려 한다. 이를 위해 https://python.org 사이트를 새로운 웹 브라우저 로 열려면 코드를 어떻게 작성해야 할까?

파이썬으로 웹 페이지를 새 창으로 열려면 webbrowser 모듈의 open_new() 함수를 사용해야 한다.

```
# webbrowser_test.py
import webbrowser

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

이미 열린 브라우저로 원하는 사이트를 열고 싶다면 다음처럼 open_new() 대신 open()을 사용하면 된다.

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

외부 라이브러리

파이썬 설치 시 기본으로 설치되는 라이브러리를 '파이썬 표준 라이브러리'라고 한다. 이번에 소개하는 외부 라이브러리는 파이썬 표준 라이브러리가 아니므로 사용하려면 먼저 pip 도구를 이용하여 설치해야 한다.

pip은 '핍'이라고 읽는다.

외부 라이브러리

EST

pip

pip은 파이썬 모듈이나 패키지를 쉽게 설치할 수 있도록 도와주는 도구이다. pip으로 파이썬 프로그램을 설치하면 의존성 있는 모듈이나 패키지를 함께 설치해 주기 때문에 매우 편리하다. 예를 들어 B라는 파이썬 패키지를 설치하려면 A라는 패키지가 먼저 설치되어야 하는 규칙이 있다고 가정할 때 pip을 이용하면 B 패키지를 설치할 때 A 패키지도 자동으로 함께 설치된다.

pip 사용법에 대해서 간략하게 알아보자.

pip install

PyPI(python package index)는 파이썬 소프트웨어가 모인 저장 공간이다. 현재 이곳에는 10만 건 이상의 파이썬 패키지가 등록되어 있으며 이곳에 등록된 파이썬 패키지는 누구나 내려받아 사용할 수 있다. 이곳에서 직접 내려받아 설치해도 되지만, pip을 이용하면 다음과 같이 간편하게 설치할 수 있다.

```
pip install SomePackage
```

여기서 SomePackage는 내려받을 수 있는 특정 패키지를 뜻한다.

pip uninstall

설치한 패키지를 삭제하고 싶다면 다음 명령어로 삭제할 수 있다.

```
pip uninstall SomePackage
```

외부 라이브러리

EST

특정 버전으로 설치하기

다음과 같이 버전을 지정하여 설치할 수도 있다. 다음 명령어를 실행하면 1.0.4 버전의 SomePackage를 설치한다.

```
pip install SomePackage==1.0.4
```

다음처럼 버전을 생략하면 최신 버전을 설치한다.

```
pip install SomePackage
```

최신 버전으로 업그레이드하기

패키지를 최신 버전으로 업그레이드하려면 `--upgrade` 옵션과 함께 사용한다.

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

설치된 패키지 확인하기

다음 명령은 pip을 이용하여 설치한 패키지 목록을 출력한다.

```
pip list
```

다음과 같이 설치된 패키지 목록을 출력할 것이다.

```
Package                  Version
------------------------ --------
amqp                     2.1.4
anyjson                  0.3.3
billiard                 3.3.0.23
celery                   3.1.0
defusedxml               0.4.1
diff-match-patch         20121119
(... 생략 ...)
```

외부 라이브러리

EST

Faker

이번에는 pip을 사용하여 유용한 외부 라이브러리중 하나인 Faker를 설치하고 사용해 보자. Faker는 테스트용 가짜 데이터를 생성할 때 사용하는 라이브러리이다.

Faker 라이브러리는 pip을 이용하여 설치해야 한다.

```
C:\> pip install Faker
```

Faker 사용해 보기

만약 다음과 같은 형식의 테스트 데이터 30건이 필요하다고 가정해 보자. 직접 데이터를 작성하지 말고 좀 더 편리한 방법으로 테스트 데이터를 만들려면 어떻게 해야 할까?

```
[(이름1, 주소1), (이름2, 주소2), ..., (이름30, 주소30)]
```

테스트 데이터는 Faker를 사용하면 매우 쉽게 만들 수 있다. 이름은 다음처럼 만들 수 있다.

```
>>> from faker import Faker
>>> fake = Faker()
>>> fake.name()
'Matthew Estrada'
```

한글 이름이 필요하다면 다음과 같이 한국을 의미하는 ko-KR을 전달하여 fake 객체를 생성하면 된다.

```
>>> fake = Faker('ko-KR')
>>> fake.name()
'김하은'
```

외부 라이브러리

주소는 다음과 같이 만들 수 있다.

```
>>> fake.address()
'충청북도 수원시 잠실6길 (경자주이읍)'
```

따라서 이름과 주소를 쌍으로 하는 30건의 테스트 데이터는 다음과 같이 만들 수 있다.

```
>>> test_data = [(fake.name(), fake.address()) for i in range(30)]
```

실행 결과는 다음과 같다.

```
>>> test_data
[('이예진', '인천광역시 동대문구 언주거리 (경자김면)'), ('윤도윤', '광주광역시 서초구 삼성로 (주원최박리)'), ('서동
현', '인천광역시 관악구 잠실가 (민석엄김마을)'), ('김광수', '울산광역시 양천구 서초대로'), (... 생략 ...), ('박성
현', '전라남도 서산시 가락27길 (준영박문읍)'), ('김성호', '경상남도 영월군 학동거리'), ('백지우', '경기도 계룡시 서초
대1로'), ('권유진', '경기도 양주시 서초중앙313가 (춘자나리)'), ('윤서준', '경상남도 청주시 서원 구 서초대64가')]
```

외부 라이브러리

EST

Faker 활용하기

Faker는 앞서 살펴본 name, address 이외에 다른 항목도 제공한다. 대표적인 몇 가지만 알아보자.

항목	설명
fake.name()	이름
fake.address()	주소
fake.postcode()	우편 번호
fake.country()	국가명
fake.company()	회사명
fake.job()	직업명
fake.phone_number()	휴대전화 번호
fake.email()	이메일 주소
fake.user_name()	사용자명
fake.pyint(min_value=0, max_value=100)	0부터 100 사이의 임의의 숫자
fake.ipv4_private()	IP 주소
fake.text()	임의의 문장 (한글 임의의 문장은 `fake.catch_phrase()` 사용)
fake.color_name()	색상명

외부 라이브러리

EST

sympy 사용해 보기

시윤이는 가진 돈의 $2/5$로 학용품을 샀다고 한다. 이때 학용품을 사는 데 쓴 돈이 1,760원이라면 남은 돈은 어떻게 구하면 될까?

이 문제는 연습장과 연필만 있으면 쉽게 구할 수 있는 일차방정식 문제이다. 파이썬으로는 다음처럼 sympy를 사용하면 방정식을 쉽게 풀 수 있다. 먼저 다음과 같이 fractions 모듈과 sympy 모듈이 필요하다.

```
>>> from fractions import Fraction
>>> import sympy
```

시윤이가 가진 돈을 x라고 하면 sympy 모듈을 사용하여 다음과 같이 표현할 수 있다.

```
>>> x = sympy.symbols("x")
```

sympy.symbols()는 x처럼 방정식에 사용하는 미지수를 나타내는 기호를 생성할 때 사용한다.

점프 투 파이썬

여러 개의 기호 사용하기

x, y 2개의 미지수가 필요하다면 다음처럼 표현할 수 있다.

```
x, y = sympy.symbols('x y')
```

외부 라이브러리

시윤이가 가진 돈의 $2/5$는 1,760원, 즉 일차방정식 $x * (2/5) = 1760$이므로 이를 코드로 표현하면 다음과 같다.

```
>>> f = sympy.Eq(x*Fraction('2/5'), 1760)
```

`sympy.Eq(a, b)` 는 a와 b가 같다는 방정식이다. 여기서 사용한 Fraction은 유리수를 표현할 때 사용하는 표준 라이브러리로, $2/5$를 정확하게 계산하고자 사용했다.

점프 투 파이썬

fractions.Fraction으로 유리수 연산하기

파이썬에서 유리수 연산을 정확하게 하려면 fractions.Fraction을 사용해야 한다.

```
>>> from fractions import Fraction
```

유리수는 다음처럼 Fraction(분자, 분모) 형태로 만들 수 있다.

```
>>> a = Fraction(1, 5)
>>> a
Fraction(1, 5)
```

또는 다음과 같이 Fraction('분자 / 분모')처럼 문자열로 만들 수도 있다.

```
>>> a = Fraction('1/5')
>>> a
Fraction(1, 5)
```

외부 라이브러리

f라는 방정식을 세웠으므로 `sympy.solve(f)` 로 x에 해당하는 값을 구할 수 있다.

```
>>> result = sympy.solve(f)
>>> result
[4400]
```

방정식의 해는 여러 개일 수 있으므로 solve() 함수는 결괏값으로 리스트를 리턴한다. 결과를 보면 시윤이가 원래 가진 돈이 4,400원이라는 것을 알 수 있다. 따라서 남은 돈은 다음처럼 가진 돈에서 1,760원을 빼면 된다.

```
>>> remains = result[0] - 1760
>>> remains
2640
```

지금까지 내용을 종합한 풀이는 다음과 같다.

```
# sympy_test.py
from fractions import Fraction
import sympy

# 가지고 있던 돈을 x라고 하자.
x = sympy.symbols("x")

# 가지고 있던 돈의 2/5가 1760원이므로 방정식은 x * (2/5) = 1760 이다.
f = sympy.Eq(x*Fraction('2/5'), 1760)

# 방정식을 만족하는 값(result)을 구한다.
result = sympy.solve(f)  # 결괏값은 리스트

# 남은 돈은 다음과 같이 가지고 있던 돈에서 1760원을 빼면 된다.
remains = result[0] - 1760

print('남은 돈은 {}원 입니다.'.format(remains))
```

프로그램을 실행한 결과는 다음과 같다.

```
남은 돈은 2640원입니다.
```

외부 라이브러리

EST

sympy 활용

$x^2 = 1$과 같은 2차 방정식의 해를 구해 보자.

```
>>> import sympy
>>> x = sympy.symbols("x")
>>> f = sympy.Eq(x**2, 1)
>>> sympy.solve(f)
[-1, 1]
```

또한 다음과 같은 연립방정식의 해도 구할 수 있다.

```
x + y = 10
x - y = 4
```

```
>>> import sympy
>>> x, y = sympy.symbols('x y')
>>> f1 = sympy.Eq(x+y, 10)
>>> f2 = sympy.Eq(x-y, 4)
>>> sympy.solve([f1, f2])
{x: 7, y: 3}
```

미지수가 2개 이상이라면 결괏값이 리스트가 아닌 딕셔너리라는 것에 주의하자.

百見而不如一作

- 점프투파이썬 예외 처리와 내장 함수, 라이브러리 단원, 연습 문제를 풀어보자
  - 위키독스 초보자를 위한 파이썬 300제, 291번~300번을 풀어보자

Q & A

맹광국 강사

ggmaeng@gmail.com