사용자 입력과 출력

사용자 입력

- 다양한 방식으로 외부의 데이터가 프로그램으로 유입되며 그 중 하나는 사용자의 입력을 통해서 이루어짐
  - » 파이썬은 명령행 환경에서 사용자의 입력을 처리하는 input 함수 제공

```
>>> a = input()
Life is too short, you need python
>>> a
'Life is too short, you need python'
>>>
```

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요: 3
>>> print(number)
3
>>>
```

화면 출력

- 파이썬은 명령행 환경에서 메시지를 출력하기 위해 print 함수 제공

```
>>> a = 123
>>> print(a)
123
>>> a = "Python"
>>> print(a)
Python
>>> a = [1, 2, 3]
>>> print(a)
[1, 2, 3]
```

- 따옴표로 구분된 문자열을 여러 개 사용하면 + 연산을 자동으로 수행

```
>>> print("life" "is" "too short")
lifeistoo short
>>> print("life"+"is"+"too short")
lifeistoo short
```

화면 출력

- 파이썬은 명령행 환경에서 메시지를 출력하기 위해 print 함수 제공

- ,로 구분된 여러 개의 문자열은 공백을 삽입해서 문자열 결합 → 변경하려면 sep 전달인자 사용

```
>>> print("life", "is", "too short")
life is too short
```

- 전체 출력이 끝나면 개행 문자 출력 → end 전달인자를 사용해서 변경 가능

```
>>> for i in range(10):
...     print(i, end=' ')
...
0 1 2 3 4 5 6 7 8 9
```

화면 출력

- 기본 출력 장치는 표준 출력 장치(모니터) → 출력 대상을 변경하려면 file 전달인자에 사용

```
import sys

f = open('test.txt', 'w')
print('welcome', 'to', 'python', 'world', sep=' $ ', end='!', file=f)
f.close()
```

문자열 포맷팅

- format 함수 사용 사례

```
print("{0} is {1}".format('apple', 'red'))

print("{item} is {color}".format(item='apple', color='red'))

args = { "item": 'apple', 'color': 'red' }
print("{item} is {color}".format(item=args['item'], color=args['color']))

args = { "item": 'apple', 'color': 'red' }
print("{item} is {color}".format(**args))
```

```
apple is red

apple is red

apple is red

apple is red

apple is red

apple is red

apple is red
```

문자열 포맷팅

- format 함수 사용 사례

`print( '[{0:<15}]'.format(111) )`	`[111 ]`
`print( '[{0:>15}]'.format(111) )`	`[ 111]`
`print( '[{0:0>15}]'.format(111) )`	`[000000000000111]`
`print( '[{0:0>15d}]'.format(111) )`	`[000000000000111]`
`print( '[{0:0>15e}]'.format(111.11111) )`	`[0001.111111e+02]`
`print( '[{0:0>15.3e}]'.format(111.11111) )`	`[0000001.111e+02]`
`print( '[{0:0>15f}]'.format(111.11111) )`	`[00000111.111110]`
`print( '[{0:0>15.3f}]'.format(111.11111) )`	`[00000000111.111]`
`print( '[{0:0>15%}]'.format(2/5) )`	`[0000040.000000%]`

파일 입출력

- 메모리는 휘발성 저장 공간 → 메모리에 저장된 데이터는 프로그램이 종료되면 소멸

- 프로그램이 종료되거나 컴퓨터의 전원이 꺼진 후에도 데이터를 보존하기 위해 디스크와 같은 영구 저장 장치에 데이터 저장

- 파이썬의 파일 입출력 기능을 통해 디스크의 파일에 데이터를 기록하고 읽는 작업 수행

- 파일 사용

```
파일객체 = open('경로', 'mode')
…
파일객체 사용
…
파일객체.close()
```

```
with open('경로', 'mode') as 파일객체 :
    …
    파일객체 사용
    …
```

파일 입출력

- 파일에 데이터 쓰기

```
f = open('test.txt', 'wt', encoding="utf-8")
f.write('파일에 기록하는 데이터 1\n')
f.write('파일에 기록하는 데이터 2\n')
f.write('파일에 기록하는 데이터 3\n')
f.close()
```

```
with open('test.txt', 'wt', encoding="utf-8") as f:
    f.write('파일에 기록하는 데이터 4\n')
    f.write('파일에 기록하는 데이터 5\n')
    f.write('파일에 기록하는 데이터 6\n')
```

파일 입출력

- 파일 데이터 읽기

```
with open('test.txt', 'rt', encoding='utf-8') as f:
    print(f.read())
```

```
with open('test.txt', 'rt', encoding='utf-8') as f:
    lines = f.readlines()
    for line in lines:
        print(line, end='')
```

```
with open('test.txt', 'rt', encoding='utf-8') as f:
    while True:
        line = f.readline()
        if not line:
            break
        print(line, end='')
```

```
with open('test.txt', 'rt', encoding='utf-8') as f:
    print( f.readlines() )
    print( f.tell() )
    print( f.readlines() )
    f.seek(0)
    print(f.readlines())
```

파일 입출력

- 파이썬 객체 저장

```
import pickle

a = [1, 2, 3, 4, 5]
b = [10, 20, 30, 40, 50]
c = { 'a': 1, 'b': 2, 'c': 3}

with open('data.dat', 'wb') as f:
    pickle.dump(a, f)
    pickle.dump(b, f)
    pickle.dump(c, f)
```

- 파이썬 객체 읽기

```
with open('data.dat', 'rb') as f:
    a2 = pickle.load(f)
    print(a2)
    b2 = pickle.load(f)
    print(b2)
    c2 = pickle.load(f)
    print(c2)
```

정규 표현식

정규 표현식

- 특정한 규칙을 가진 문자열 표현에 사용되는 형식 언어 → 문자열의 패턴을 표현하기 위해 사용
- 패턴을 기반으로 문자열 검색, 치환 등의 작업에 주로 사용
- 파이썬은 re 모듈을 통해 정규 표현식 기반 검색, 치환, 분리 등의 기능 제공

정규 표현식 문법

- [\ ]
  - » 특수 문자가 아닌 문자(non-special character) 앞에서 사용된 백슬래시는 '해당 문자는 특별하고, 문자 그대로 해석되면 안된다'는 사실을 가리킵니다. 예를 들어, 앞에 \가 없는 'b'는 보통 소문자 b가 나오는 패턴과 대응됩니다. 그러나 '\b' 자체는 어떤 문자와도 대응되지 않습니다; 이 문자는 특별한 단어 경계 문자를 형성합니다.

  - » 특수 문자 앞에 위치한 백슬래시는 '다음에 나오는 문자는 특별하지 않고, 문자 그대로 해석되어야 한다'는 사실을 가리킵니다. 예를 들어, 패턴 /a*/ 에서의 특수문자 '*'는 0개 이상의 'a' 문자가 등장함을 나타냅니다. 이와는 다르게, 패턴 /a*/ 는 '*'이 특별하지 않다는 것을 나타내며, 'a*'와 같은 문자열과 대응될 수 있습니다.

정규 표현식 문법

문자	의미
.	개행 문자를 제외한 1개의 문자
^	문자열의 시작
$	문자열의 종료
[]	문자 집합 [abcd] → a, b, c, d / [a-d] → a, b, c, d / [^abcd] → NOT a, b, c, d
|	OR 연산 A|B → A 또는 B
()	괄호 안의 정규식을 그룹으로 지정
*	0회 이상 반복
+	1회 이상 반복
?	0회 또는 1회
{m}	m회 반복
{m,n}	m회 ~ n회 반복
{m,}	m회 이상 반복

정규 표현식 문법

문자	의미
\w	숫자, 밑줄을 포함하는 모든 언어의 표현 가능한 문자
\W	숫자, 밑줄을 포함하는 모든 언어의 표현 가능한 문자를 제외한 나머지
\d	[0-9]를 포함하는 모든 숫자
\D	숫자를 제외한 모든 문자
\s	[\t\n\r\f\v]를 포함하는 공백 문자
\S	공백문자를 제외한 문자
\b	단어의 시작과 끝의 빈 공백
\B	단어의 시작과 끝이 아닌 빈 공백
\\	'\' 문자
\숫자	숫자만큼 일치하는 문자열
\A	문자열의 시작
\Z	문자열의 끝

re 모듈 함수

- re.search(pattern, string)
  - » string 전체에 대해 pattern이 존재하는지 검사

- re.match(pattern, string)
  - » string이 시작하는 부분부터 pattern이 존재하는지 검사

- re.split(pattern, string)
  - » pattern을 구분자로 string을 분리해서 리스트로 반환

- re.findall(pattern, string)
  - » string에서 pattern과 매치되는 모든 경우를 찾아 리스트로 반환

- re.finditer(pattern, string)
  - » string에서 pattern과 매치되는 모든 경우를 찾아 이터레이터로 반환

re 모듈 함수

- re.sub(pattern, repl, string)
  - » string에서 pattern과 일치하는 부분에 대해 repl로 교체해서 결과 문자열 반환

- re.subn(pattern, repl, string)
  - » re.sub와 동일한 실행 → 반환 값이 (결과 문자열, 매칭 횟수)

- re.escape(string)
  - » 영문자, 숫자가 아닌 문자에 대해 백슬래시 문자 추가

- re.compile(pattern)
  - » pattern을 컴파일해서 정규 표현식 개체 반환