Python (8)

사용자 입출력, 파일처리

04-2 사용자 입출력

○ 사용자 입력 활용하기

사용자가 입력한 값을 어떤 변수에 대입하고 싶을 때는 어떻게 해야 할까?

◆ input 사용하기

```
a = input("아무 글자나 넣어보세요:")
print("입력 결과:", a)
```

위 코드를 실행하고, 터미널에서 글자를 입력해 본다.
그러면, 입력한 글자가 터미널에 출력될 것이다.

위와 같이 input은 사용자가 키보드로 입력한 모든 것을 문자열로 저장한다.

◆ 입력한 값을 정수로 변경하여 출력하기

필요에 따라서 문자열이 아닌 숫자를 입력 받아야 할 때가 있다.
이런 경우에는 아래와 같이 **int()**함수를 사용하면 문자열이 정수로 변환된다.

```
a = int(input("숫자를 입력하세요:"))
print("입력 결과:", a)
```

또 다른 예를 하나 보자.

```
a = int(input("첫번째 숫자를 입력하세요:"))
b = int(input("두번째 숫자를 입력하세요:"))

print("입력 결과:", a + b)
```

◆ 입력한 값을 실수로 변경하여 출력하기

아래와 같이 **float()**함수를 사용하면 문자열이 실수로 변환된다.

```
a = float(input("숫자를 입력하세요:"))
print("입력 결과:", a)
```

참고)
input에서 **int()**함수를 사용하여 정수로 변환하거나, **float()**함수를 사용하여 실수로 변경할 경우, 숫자가 아닌 문자를 입력하게 되면 오류가 발생한다.

○ print 자세히 알기

지금까지 우리가 사용한 **print** 문의 용도는 데이터를 출력하는 것이었다. 데이터를 출력하는 **print** 문의 사용 예는 다음과 같다.

```
a = 123
b = "Python"
c = [1, 2, 3]
--------------------
123
Python
[1, 2, 3]
```

이제 **print** 문으로 할 수 있는 일에 대해서 좀 더 자세하게 알아보자.

◆ 따옴표로 둘러싸인 문자열이 연속하여 이어지면 + 연산과 동일하다

```
print("life" "is" "too short")  # 1번
print("life"+"is"+"too short")  # 2번
--------------------
lifeistoo short
lifeistoo short
```

위 예에서 1번과 2번은 완전히 동일한 결괏값을 출력한다. 즉, 따옴표로 둘러싸인 문자열을 연속해서 쓰면 + 연산을 한 것과 같다.

◆ 문자열 띄어쓰기는 쉼표로 한다

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

쉼표(,)를 사용하면 출력되는 문자열 사이에 공백이 하나씩 들어간다.

◆ 한 줄에 결과값 출력하기

앞서 **for** 문을 공부할 때 만들었던 구구단 프로그램에서 보았듯이, 한 줄에 결괏값을 계속 이어서 출력하려면 매개변수 **end**를 사용해 끝 문자를 지정해야 한다.

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

end 매개변수의 기본값은 줄바꿈(**\n**) 문자이다.

◆ 여러 매개변수를 출력할 때 구분자 넣기

sep="" 매개변수를 사용하면, 여러 매개변수가 출력될 때 구분자를 삽입할 수 있다.

```
print("하나", "둘", "셋", "넷", sep="*")
--------------------
하나*둘*셋*넷
```

04-3 파일 읽고 쓰기

지금까지 값을 '입력'받을 때는 사용자가 직접 입력하는 방식을 사용했고,
'출력'할 때는 모니터 화면에 결괏값을 출력하는 방식을 사용했다.
하지만 입출력 방법이 꼭 이것만 있는 것은 아니다.
이번에는 파일을 통한 입출력 방법에 대해 알아보자.

○ 파일 생성하기

다음 코드를 I작성하여 실행해 보자.

```
f = open("newfile.txt", 'w', encoding='utf8')
f.close()
```

프로그램을 실행한 디렉터리에 새로운 파일이 하나 생성된 것을 확인할 수 있을 것이다.
파일을 생성하기 위해 파이썬 내장 함수 open을 사용했다.
open 함수는 다음과 같이 '파일 이름'과 '파일 열기 모드'를 입력값으로 받고 결과값으로 파일 객체를 리턴한다.

```
파일_객체 = open(파일_이름, 파일_열기_모드, encoding='utf8')
```

파일 열기 모드에는 다음과 같은 것들이 있다.

파일 열기모드	설명
r	읽기 모드: 파일을 읽기만 할 때 사용한다.
w	쓰기 모드: 파일에 내용을 쓸 때 사용한다.
a	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용한다.

파일을 쓰기 모드로 열면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고 해당 파일이 존재하지 않으면 새로운 파일이 생성된다.
위 예에서는 디렉터리에 파일이 없는 상태에서 ‘newfile.txt’ 파일을 쓰기 모드인 'w'로 열었기 때문에 ‘newfile.txt’라는 이름의 새로운 파일이 현재 디렉터리에 생성되었다.

만약 ‘newfile.txt’ 파일을 D:/python 디렉터리에 생성하고 싶다면 다음과 같이 작성해야 한다.

```
f = open("D:/python/newfile.txt", 'w', encoding='utf8')
f.close()
```

위 예에서 **f.close()**는 열려 있는 파일 객체를 닫아 주는 역할을 한다. 사실 이 문장은 생략해도 된다.
프로그램을 종료할 때 파이썬 프로그램이 열려 있는 파일의 객체를 자동으로 닫아 주기 때문이다.
하지만 **close()**를 사용해서 열려 있는 파일을 직접 닫아 주는 것이 좋다.
쓰기모드로 열었던 파일을 닫지 않고 다시 사용하려고 하면 오류가 발생하기 때문이다.

참고) 파일 경로와 슬래시(/)

파이썬 코드에서 파일 경로를 표시할 때 "D:/python/newfile.txt"처럼 슬래시(/)를 사용할 수 있다.
만약 역슬래시(\)를 사용한다면 "D:\\python\\newfile.txt"처럼 역슬래시를 2개 사용하거나 r"D:\python\newfile.txt"와 같이 문자열 앞에 r 문자(raw string)를 덧붙여 사용해야 한다. 왜냐하면 "D:\note\test.txt"처럼 파일 경로에 \n과 같은 이스케이프 문자가 있을 경우, 줄바꿈 문자로 해석되어 의도했던 파일 경로와 달라지기 때문이다.

○ 파일을 쓰기 모드로 열어 내용 쓰기

위 예에서는 파일을 쓰기 모드로 열기만 했을 뿐, 정작 아무것도 쓰지는 않았다. 이번에는 문자열 데이터를 파일에 직접 써 보자.

```
f = open("newfile.txt", 'w', encoding='utf8')
for i in range(1, 11):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
```

```
f.close()
```

이제 명령 프롬프트 창에서 첫 번째 예제를 실행해 보자.
파일 안에는 어떤 내용이 담겨 있는지 확인해 보자.

○ 파일을 읽는 여러 가지 방법

파이썬에는 파일을 읽는 방법이 여러 가지 있다. 이번에는 그 방법을 자세히 알아보자.

◆ readline 함수 이용하기

첫 번째는 readline 함수를 사용하는 것이다. 다음 예를 살펴보자.

```
f = open("newfile.txt", 'r', encoding='utf8')
line = f.readline()
print(line)
f.close()
--------------------
1번째 줄입니다.
```

위는 ‘newfile.txt’ 파일을 읽기 모드로 연 후 readline()을 사용해서 파일의 첫 번째 줄을 읽어 출력하는 예제이다.
앞에서 만든 newfile.txt 파일을 수정하거나 지우지 않았다면 위 프로그램을 실행했을 때 newfile.txt 파일의 가장 첫 번째 줄이 화면에 출력되었을 것이다.

만약 모든 줄을 읽어 화면에 출력하고 싶다면 다음과 같이 작성하면 된다.

```
f = open("newfile.txt", 'r', encoding='utf8')
while True:
    line = f.readline()
    if not line: break
    print(line)
f.close()
```

while True: 무한 루프 안에서 f.readline()을 사용해 파일을 계속 한 줄씩 읽어 들인다.
만약 더 이상 읽을 줄이 없으면 break를 수행한다(readline()은 더 이상 읽을 줄이 없을 경우, 빈 문자열('')을 리턴한다).

한 줄씩 읽어 출력할 때 줄 끝에 \n 문자가 있으므로 빈 줄도 같이 출력된다.

◆ readlines 함수 사용하기

두 번째 방법은 **readlines** 함수를 사용하는 것이다. 다음 예를 살펴보자.

```
f = open("newfile.txt", 'r', encoding='utf8')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 가지는 리스트를 리턴한다.
따라서 위 예에서 **lines**는 리스트 **["1번째 줄입니다.\n", "2번째 줄입니다.\n", ..., "10번째 줄입니다.\n"]**가 된다.
f.readlines()는 **f.readline()**와 달리 **s**가 하나 더 붙어 있다는 것에 유의하자.

◆ 줄 바꿈(\n) 문자 제거하기

파일을 읽을 때 줄 끝의 줄 바꿈(**\n**) 문자를 제거하고 사용해야 할 경우가 많다.
다음처럼 **strip** 함수를 사용하면 줄 바꿈 문자를 제거할 수 있다.

```
f = open("newfile.txt", 'r', encoding='utf8')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

◆ read 함수 사용하기

세 번째는 read 함수를 사용하는 방법이다. 다음 예를 살펴보자.

```
f = open("newfile.txt", 'r', encoding='utf8')
data = f.read()
print(data)
f.close()
```

f.read()는 파일의 내용 전체를 문자열로 리턴한다. 따라서 위 예의 data는 파일의 전체 내용이다.

◆ 파일 객체를 for 문과 함께 사용하기

네 번째는 파일 객체를 for 문과 함께 사용하는 방법이다.

```
f = open("newfile.txt", 'r', encoding='utf8')
for line in f:
    print(line)
f.close()
```

파일 객체(f)는 기본적으로 위와 같이 for 문과 함께 사용하여 파일을 줄 단위로 읽을 수 있다.

○ 파일에 새로운 내용 추가하기

쓰기 모드('w')로 파일을 열 때 이미 존재하는 파일을 열면 그 파일의 내용이 모두 사라지게 된다.
하지만 원래 있던 값을 유지하면서 단지 새로운 값만 추가해야 할 경우도 있다.
이런 경우에는 파일을 추가 모드('a')로 열면 된다.

```
f = open("newfile.txt", 'a', encoding='utf8')
for i in range(11, 20):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

위는 newfile.txt 파일을 추가 모드('a')로 열고 write를 사용해서 결과값을 기존 파일에 추가해 적는 예이다.
여기에서 추가 모드로 파일을 열었기 때문에 newfile.txt 파일이 원래 가지고 있던 내용 바로 다음부터 결과값을 적기 시작한다.

newfile.txt 파일을 열어 보면 원래 있던 내용 뒤에 새로운 내용이 추가된 것을 확인할 수 있다.

❍ with 문과 함께 사용하기

지금까지 살펴본 예제를 보면 항상 다음과 같은 방식으로 파일을 열고 닫은 것을 알 수 있다.

```
f = open("foo.txt", 'w')
f.write("Life is too short, you need python")
f.close()
```

파일을 열면(open) 항상 닫아(close) 주어야 한다.
이렇게 파일을 열고 닫는 것을 자동으로 처리할 수 있다면 편리하지 않을까?
파이썬의 with 문이 바로 이런 역할을 해 준다.
다음 예는 with 문을 사용해서 위 예제를 다시 작성한 모습이다.

```
with open("foo.txt", "w") as f:
    f.write("Life is too short, you need python")
```

위와 같이 with 문을 사용하면 with 블록(with 문에 속해 있는 문장)을 벗어나는 순간, 열린 파일 객체 f가 자동으로 닫힌다.

04-4 프로그램의 입출력

명령 프롬프트에서 다음과 같은 명령어를 실행해 보자.

```
D:\> type a.txt
```

type은 바로 뒤에 적힌 파일 이름을 인수로 받아 해당 파일의 내용을 출력해 주는 명령어이다.
대부분의 명령 프롬프트에서 사용하는 명령어는 다음과 같이 인수를 전달하여 프로그램을 실행하는 방식을 따른다.

```
명령어 [인수1 인수2 ...]
```

이러한 기능을 파이썬 프로그램에도 적용할 수 있다.

❍ sys 모듈 사용하기

파이썬에서는 sys 모듈을 사용하여 프로그램에 인수를 전달할 수 있다. sys 모듈을 사용하려면 다음 예의 import sys처럼 import 명령어를 사용해야 한다.

참고) 모듈을 사용하고 만드는 방법에 대해서는 뒤에서 자세히 알아본다.

sys1.py 파일을 새로 생성하고, 아래와 같이 코딩하자.

```
import sys

args = sys.argv[1:]
for i in args:
    print(i)
```

위는 프로그램 실행 시 전달받은 인수를 **for** 문을 사용해 차례대로 하나씩 출력하는 예이다.
sys 모듈의 **argv**는 프로그램 실행 시 전달된 인수를 의미한다.
즉, 다음과 같이 입력했다면 **argv[0]**은 파일 이름이 되고, **argv[1]**부터는 뒤에 따라오는 인수가 차례대로 **argv**의 요소가 된다.

이 프로그램을 명령 프롬프트에서 아래와 같이 실행하여, 학습 디렉터리로 이동 후, **sys1.py** 파일을 실행한다.

```
cd D:/python/python-study
D:/python/python-study> sys1.py aaa bbb ccc
--------------------
aaa
bbb
```

```
ccc
```

위 예를 응용하여 다음과 같은 간단한 프로그램을 만들어 보자.
sys2.py 파일을 새로 생성하고, 아래와 같이 코딩한다.

```
import sys
args = sys.argv[1:]
for i in args:
    print(i.upper(), end=' ')
```

문자열 관련 함수인 **upper()**를 사용하여 프로그램 실행 시 전달된 인수를 모두 대문자로 바꾸어 주는 간단한 프로그램이다. 명령 프롬프트 창에서 다음과 같이 실행해 보자.

```
D:/python/python-study> sys2.py life is too short, you need python
--------------------
LIFE IS TOO SHORT, YOU NEED PYTHON
```

출력이 제대로 되었는지 확인한다.