LECTURE 4

Python Basics Part 3

(원본 자료) http://www.cs.fsu.edu/~carnahan/cis4930/

INPUT

사용자 입력을 받는 2가지 방법

- raw_input()
  - 사용자로 부터 입력을 위한 string을 받음
  - argument로 사용자 입력 prompt 사용가능
- input()
  - 2.x 버전에서는 사용자 입력을 받아서 evaluation 한 후 그 결과를 retur한다.
  - Returns 값은 expression의 값
  - Dangerous – don't use it.

Note: In Python 3.x, input() 과 raw_input()이 동일하게 동작함

```
>>> print(raw_input('What is your name? '))
What is your name? Caitlin
Caitlin
>>>
```

```
>>> print(input(Do some math: ))
Do some math: 2+2*5
12
>>>
```

Note: reading an EOFwill raise an EOFError.

FILES

Python에서 파일을 다루기 위해 file object를 생성하는 두 가지 방법

- file() constructor 사용
  - The second argument accepts a few special characters: ‘r’ for reading (default), ‘w’ for writing, ‘a’ for appending, ‘r+’ for reading and writing, ‘b’ for binary mode.

```
>>> f = file("filename.txt", 'r')
```

- open() 함수 사용
  - 첫번째 파라미터는 파일 이름이고, 두번째는 mode.

```
>>> f = open("filename.txt", 'rb')
```

Use the open() method typically. The file() constructor is removed in Python 3.x.

Note: when a file operation fails, an `IOError` exception is raised.

FILE INPUT

파일로 부터 정보를입력을 획득하는 방법

f.read()
- 파일의 내용 전부를 하나의 sring으로 리턴함.
- 파라미터로 수집할 char 개수 제한 가능.

- f.readline()
  - 한라인씩 리턴함. (ends with a newline).
  - 파일 끝에 도달하면 리턴되는 스트링은 emtpy.
- Loop를 통해 file 오브젝트 접근
  - 대부분의 경우 사용. (just use a for loop!)

```
>>> f = file("somefile.txt",'r')
>>> f.read()
"Here's a line.\nHere's another line.\n"
>>> f.close()
>>> f = file("somefile.txt",'r')
>>> f.readline()
"Here's a line.\n"
>>> f.readline()
"Here's another line.\n"
>>> f.readline()
''
>>> f.close()
>>> f = file("somefile.txt",'r')
>>> for line in f:
...     print(line)
...
Here's a line.
Here's another line.
```

FILE INPUT

- f.close()
  - 파일을 닫고, 리소스를 반환함

```
>>> f = open("somefile.txt", 'r')
>>> f.readline()
"Here's line in the file! \n"
>>> f.close()
```

- 다른 방법의 open and read:
  - close 를 할 필요가 없음. scope를 벗어나면 자동적으로 닫힘

```
with open("text.txt", "r") as txt:
    for line in txt:
        print line
```

STANDARD FILE OBJECTS

- C++의 cin, cout, cerr 처럼 파이썬에서 표준 입력, 출력, 에러는 sys 모듈을 통해서 수행
  - 일반 파일처럼 취급.

```
import sys
for line in sys.stdin:
    print line
```

- 실행 명령에서의 파라미터는 sys.argv[] 를 통해서 획득 가능

```
for arg in sys.argv:
    print arg
```

```
$ python program.py here are some arguments
program.py
here
are
some
arguments
```

OUTPUT

- print or print()
  - print 구문이나 3.x버전의 print() 함수 사용
  - 콤마로 구분하거나(출력시 인자간 space 추가됨) 또는concatenate string 이용
  - 각 인자는 string type으로 변환되어 출력됨.
  - print()함수는 두 개의 추가적인 키워드 존재 (end와 sep)

```
>>> print 'Hello,', 'World', 2015
Hello, World 2015
>>> print "Hello, " + "World " + "2015"
Hello, World 2015
>>> for i in range(10):
...     print i, # Do not include trailing newline
...
0 1 2 3 4 5 6 7 8 9
```

PRINT FUNCTION

3.x버전의 print() 함수를 선호하는 사람들이 있음.

```
print(*objects, sep=' ', end='\n', file=sys.stdout)
```

- 2.0에서 다음과 같은 라인을 추가하면 print 구문이 아닌 print() 함수를 사용하도록 강제할 수 있음
- Import with `from __future__ import print_function`
- *sep* 는 파라미터 출력시 파라미터 출력 사이에 출력되는 구분자
- *end* 는 마지막 파라미터가 출력되고 난 이후에 출력.
- *file* 는 출력하고자 하는 파일 오브젝트 선택 가능

PRINT FUNCTION

```
>>> from __future__ import print_function
>>> print(555, 867, 5309, sep="-")
555-867-5309
>>> print("Winter", "is", "coming", end="!\n")
Winter is coming!
>>>
```

FILE OUTPUT

- f.write(*str*)
  - 파일 오브젝트에 str을 쓸때 사용. 함수 리턴 값은 없음(None).
  - print와 달리 명시적으로 string을 적어주어야 하며, 필요시 str() 활용.

```
>>> f = open("filename.txt", 'w')
>>> f.write("Heres a string that ends with " + str(2015))
```

- print >> f
  - 2.x 버전에서 사용되어짐

```
f = open("filename.txt","w")
for i in range(1, 10 + 1):
    print >> f, i # print(i, file=f)
f.close()
```

MORE ON FILES

File 오브젝트는 부가적인 built-in 함수가 정의:

- f.tell() 파일의 현재 위치를 리턴함.
- f.seek(*offset*[, *from*]) *from* 위치에서 *offset* 바이트(byte)만큼 이동.

  * from 위치가 명시되지 않으면 현재 위치
- f.flush() 내부 버퍼를 flush함 (내부 버퍼의 내용을 파일 출력으로 강제로 내보냄)

Python은 기본적으로 현재 디렉토리의 파일을 찾습니다. 다른 경로의 파일을 접근하고자 할 때에는 절대 경로를 지정해 주거나 os.chdir() 함수를 이용하여 디렉토리 변경이 가능합니다.

MODIFYING FILES AND DIRECTORIES

os 모듈을 이용하여 다음과 같이 파일 관련 수행

- os.rename(*current_name*, *new_name*) # 파일 이름 변경
- os.remove(*filename*) # 파일 삭제
- os.mkdir(*newdirname*) # 디렉토리 생성
- os.chdir(*newcwd*) # 현재 디렉토리 변경
- os.getcwd() # 현재 디렉토리를 알고자 할 때
- os.rmdir(*dirname*) # 디렉토리 삭제 (비어있는 디렉토리만 삭제 가능)

EXCEPTIONS

Errors가 발생했을 때 Python에서도 *exceptions 발생 및 처리 가능*

```
>>> print spam
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined

>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

There are a number of built-in exceptions, which are listed here.

HANDLING EXCEPTIONS

명지적인 exceptions 처리가 필요.
try/except 블록을 통해서 exception을 받아서 처리함

```
>>> while True:

...     try:
...         x = int(input("Enter a number: "))
...     except ValueError:
...         print("Ooops !! That was not a valid number. Try again.")
...
Enter a number: “two”
Ooops !! That was not a valid number. Try again.
Enter a number: 100
```

HANDLING EXCEPTIONS

- try 부분이 실행되고, 에러가 없으면 except 부분은 skip
- 만약 error가 발생하면 나머지 try 부분은 skip되고 except부분 수행
  - exception 타입이 일치하는 exception만 처리
- 이후 프로세스는 정상적으로 동작

```
>>> while True:
...     try:
...         x = int(input("Enter a number: "))
...     except ValueError:
...         print("Ooops !! That was not a valid number. Try again.") ...
Enter a number: “two”
Ooops !! That was not a valid number. Try again.
Enter a number: 100
```

HANDLING EXCEPTIONS

- 만약 exception 타입이 매칭되는 exception이 발생하면 프로그램이 중단됨

```
>>> while True:
...     try:
...         x = int(input("Enter a number: "))
...     except ValueError:
...         print("Ooops !! That was not a valid number. Try again.")
...
Enter a number: 3/0
Traceback (most recent call last):
File "<stdin>", line 3, in <module>
File "<string>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

HANDLING EXCEPTIONS

try/except 구문들:

Clause form	Interpretation
`except :`	모든 exception 타입을 잡음
`except name:`	특정 exception만 잡음
`except name as value:`	명명된 exception과 instance명명
`except (name1, name2):`	명명된 exception만 잡음
`except (name1, name2) as value:`	명명된 exception과 instance 명명
`else:`	exception이 없었을 경우에만 수행
`finally:`	항상 수행

HANDLING EXCEPTIONS

실 사용예)

```
>>> while True:
...     try:
...         x = int(input("Enter a number: "))
...     except ValueError:
...         print("Ooops !! That was not a valid number. Try again.")
...     except (RuntimeError, IOError) as e:
...         print(e)
...     else:
...         print("No errors encountered!")
...     finally:
...         print("We may or may not have encountered errors…")
...
```

RAISING AN EXCEPTION

강제적으로 exception을 발생시키고 싶을 경우 raise 구분을 사용

```
try:
    raise IndexError("Index out of range")
except IndexError as ie:
    print("Index Error occurred: ", (str(ie)))
```

CREATING AN EXCEPTION

자신만의 exception을 만들고자 할 시에는 *Exception* class 를 상속받아야 함.

```
>>> class MyError(Exception):
...     def __init__(self, value):
...         self.value = value
...     def __str__(self):
...         return repr(self.value)
...
>>> try:
...     raise MyError(2*2)
... except MyError as e:
...     print 'My exception occurred, value:', e
...
My exception occurred, value: 4
```

ASSERTIONS

assert 구문은 해당 condition이 만족하는지 보고, 만족하지 않을시 error 발생.

```
>>> assert a == 2
```

위와 아래가 동일함

```
>>> if not a == 2:
...     raise AssertionError()
```

EXAMPLE: PARSING CSV FILES

축구(Football):
football.csv 파일은 영국 프리미어 리그의 정보를 가지고 있음. columns은 'Goals'(득점)과 'Goals Allowed'(실점)이 있음. (예 아스날(Arsenal)은 79득점에 36실점을 함)

득점과 실점의 차이가 가장 적은 팀을 구하는 프로그램을 짜시오

Solutions available in the original post. (Also, there's some nice TDD info).
Credit: https://realpython.com/blog/python/python-interview-problem-parsing-csv-files/