Files I/O

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Python

Printing to the Screen

- The simplest way to produce output is using the **`print`** statement where can pass zero or more expressions separated by ***commas***.
- This function converts the expressions pass into a string and writes the result to standard output as follows :

```
print ("Python is really a great language,", "isn't it?")
```

Reading Keyboard Input

- Python 2 has two built-in functions to read data from standard input, which by default comes from the keyboard.
- These functions are `input()` and `raw_input()`.
- In Python 3, `raw_input()` function is deprecated.
- Moreover, `input()` functions read data from keyboard as string, irrespective of whether it is enclosed with quotes (`''` or `""`) or not.

Opening and Closing Files

- Until now, you have been reading and writing to the standard input and output.
- Now, we will see how to use actual data files.
- Python provides basic functions and methods necessary to manipulate files by default.
- Can do most of the file manipulation using a `file` object.

The open Function

- Before read or write a file, have to open it using Python's built-in **`open()`** function.
- This function creates a **`file`** object, which would be utilized to call other support methods associated with it.
- Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

The open Function (Cont.)

- **`file_name`**
  - Is a string value that contains the name of the file that want to access.
- **`access_mode`**
  - Determines the mode in which the file has to be opened, i.e., ***read***, ***write***, ***append***, etc.
  - A complete list of possible values is given next slide in the table.
  - This is an optional parameter and the default file access mode is read (**`r`**).

The open Function (Cont.)

- **`buffering`**
  - If the buffering value is set to 0, no buffering takes place.
  - If the buffering value is 1, line buffering is performed while accessing a file.
  - If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size.
  - If negative, the buffer size is the system default(default behavior).

File Access Mode

Mode	Description
`r`	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
`rb`	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
`r+`	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
`rb+`	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
`w`	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
`wb`	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

File Access Mode (Cont.)

Mode	Description
`w+`	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
`wb+`	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
`a`	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

File Access Mode (Cont.)

Mode	Description
`ab`	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
`a+`	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
`ab+`	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file Object Attributes

- Once a file is opened and you have one `file` object, can get various information related to that file.
- `file.closed`
  - Returns true if file is closed, false otherwise.
- `file.mode`
  - Returns access mode with which file was opened.
- `file.name`
  - Returns name of the file.

The file Object Attributes (Cont.)

demo

```
1 # Open a file
2
3 fo = open("demo.py", "wb")
4 print ("Name of the file: ", fo.name)
5 print ("Closed or not : ", fo.closed)
6 print ("Opening mode : ", fo.mode)
7 fo.close()
8
```

Console | Problems

<terminated> demo.py [/usr/bin/python3.6]

```
Name of the file:  demo.py
Closed or not :  False
Opening mode :  wb
```

The close() Method

- Flushes any unwritten information and closes the file object, after which no more writing can be done.
- Python automatically closes a file when the reference object of a file is reassigned to another file.
- It is a good practice to use the `close()` method to close a file.
- Syntax

```
fileObject.close()
```

The write() Method

- Writes any string to an open file.
- It is important to note that Python strings can have binary data and not just text.
- The `write()` method does not add a newline character (`\n`) to the end of the string.
- Syntax

```
fileObject.write(string)
```

The write() Method (Cont.)

```
1 # Open a file
2
3 fo = open("foo.txt", "w")
4 fo.write( "Python is a great language.\nYeah its great!!\n")
5
6 # Close opened file
7 fo.close()
8
```

foo.txt

```
1 Python is a great language.
2 Yeah its great!!
3
```

The read() Method

- Reads a string from an open file.
- It is important to note that Python strings can have binary data apart from text data.
- Syntax

```
fileObject.read([count])
```

The read() Method (Cont.)

- Here, passed parameter is the number of bytes to be read from the opened file.
- This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

The read() Method (Cont.)

```
1 # Open a file
2
3 fo = open("foo.txt", "r+")
4 str = fo.read(10)
5 print ("Read String is : ", str)
6
7 # Close opened file
8 fo.close()
9
```

Console Problems

<terminated> demo.py [/usr/bin/python3.6]

```
Read String is :  Python is
```

File Positions

- The `tell()` method tells you the current position within the file.
- In other words, the next read or write will occur at that many bytes from the beginning of the file.

File Positions (Cont.)

- The `seek(offset[, from])` method changes the current file position.
- The `offset` argument indicates the number of bytes to be moved.
- The `from` argument specifies the reference position from where the bytes are to be moved.

File Positions (Cont.)

- If `from` is set to 0, the beginning of the file is used as the reference position.
- If it is set to 1, the current position is used as the reference position.
- If it is set to 2 then the end of the file would be taken as the reference position.

File Positions (Cont.)

```
1  # Open a file
2
3  fo = open("foo.txt", "r+")
4  str = fo.read(10)
5  print ("Read String is : ", str)
6
7  # Check current position
8  position = fo.tell()
9  print ("Current file position : ", position)
10
11 # Reposition pointer at the beginning once again
12 position = fo.seek(0, 0)
13 str = fo.read(10)
14 print ("Again read String is : ", str)
15
16 # Close opened file
17 fo.close()
```

Console Problems

<terminated> demo.py [/usr/bin/python3.6]

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

Renaming and Deleting Files

- Python `os` module provides methods that help you perform file-processing operations, such as renaming and deleting files.
- To use this module, need to import it first and then can call any related functions.

The rename() Method

- Takes two arguments, the current filename and the new filename.
- Syntax

```
os.rename(current_file_name, new_file_name)
```

```
1 import os
2
3 # Rename a file from foo.txt to bar.txt
4 os.rename( "foo.txt", "bar.txt" )
5
```

The remove() Method

- Can use the **`remove()`** method to delete files by supplying the name of the file to be deleted as the argument.
- Syntax

```
os.remove(file_name)
```

```
1 import os
2
3 # Delete file bar.txt
4 os.remove("bar.txt")
5
```

Directories in Python

- All files are contained within various directories.
- Python has no problem handling these too.
- The `os` module has several methods that help you create, remove, and change directories.

The mkdir() Method

- Can use the **`mkdir()`** method of the **`os`** module to create directories in the current directory.
- Need to supply an argument to this method, which contains the name of the directory to be created.
- Syntax

```
os.mkdir("newdir")
```

```
1 import os
2
3 # Create a directory "test"
4 os.mkdir("test")
5
```

The chdir() Method

- Can use the **`chdir()`** method to change the current directory.
- Takes an argument, which is the name of the directory that you want to make the current directory.
- Syntax

```
os.chdir("newdir")
```

```
1 import os
2
3 # Changing a directory to "/usr/local/lib/python3.6"
4 os.chdir("/usr/local/lib/python3.6")
5
```

The getcwd() Method

- Displays the current working directory.
- Syntax

```
os.getcwd()
```

```
1 import os
2
3 # This would give location of the current directory
4 print (os.getcwd())
5
```

The rmdir() Method

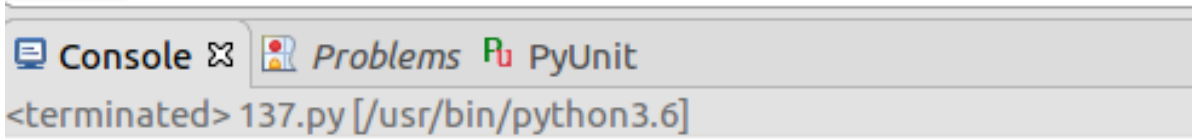
- Deletes the directory, which is passed as an argument in the method.
- Before removing a directory, all the contents in it should be removed.
- Syntax

```
os.rmdir('dirname')
```

```
1 import os
2
3 # This would  remove "./test"  directory.
4 os.rmdir( "./test"  )
5
```

Lab

```
1 f = open('stockcode.txt', 'r')
2 data = f.read()
3 print(data)
4 f.close()
5
```




```
000020 동화약품
000040 S&T모터스
000050 경방
000060 메리츠화재
000070 삼양사
000071 삼양사우
000100 유한양행
```

Lab (Cont.)

```
1 f = open('stockcode.txt', 'r')
2 line_num = 1
3 line = f.readline()
4 while line:
5     print('%d %s' %(line_num, line), end='')
6     line = f.readline()
7     line_num += 1
8 f.close()
9
```

Console Problems PyUnit

<terminated> 138.py [/usr/bin/python3.6]

```
1 000020 동화약품
2 000040 S&T모터스
3 000050 경방
4 000060 메리츠화재
5 000070 삼양사
6 000071 삼양사우
7 000100 유한양행
8 000101 유한양행우
```

Lab (Cont.)

```
1 f = open('stockcode.txt', 'r')
2 lines = f.readlines()
3 #print(lines)
4 for line_num, line in enumerate(lines):
5     print('%d %s' %(line_num+1, line), end='')
6 f.close()
7
```

Console Problems PyUnit

<terminated> 139.py [/usr/bin/python3.6]

```
392 006210 휴리프
393 006220 제주은행
394 006260 LS
395 006280 녹십자
396 006340 대원전선
397 006341 대원전선우
398 006350 전북은행
399 006360 GS건설
```

Lab (Cont.)

```
1 text = input('파일에 저장할 내용을 입력하세요: ')
2 f = open('mydata.txt', 'w')
3 f.write(text)
4 f.close()
```

mydata.txt

```
1 파이썬은 배워볼만한 언어네요...
```

Console | Problems | PyUnit

<terminated> 140.py [/usr/bin/python3.6]

파일에 저장할 내용을 입력하세요: 파이썬은 배워볼만한 언어네요...

Lab (Cont.)

```python
count = 1
data = []
print('파일에 내용을 저장하려면 내용을 입력하지 말고 [Enter]를 누르세요')
while True:
    text = input('[%d] 파일에 저장할 내용을 입력하세요: ' %count)
    if text == '':
        break
    data.append(text+'\n')
    count += 1

f = open('mydata.txt', 'w')
f.writelines(data)
f.close()
```

Console Problems PyUnit

<terminated> 141.py [/usr/bin/python3.6]

```
파일에 내용을 저장하려면 내용을 입력하지 말고 [Enter]를 누르세요
[1] 파일에 저장할 내용을 입력하세요: Hello, World
[2] 파일에 저장할 내용을 입력하세요: Good Morning
[3] 파일에 저장할 내용을 입력하세요: Best Language is the Python.
[4] 파일에 저장할 내용을 입력하세요:
```

Lab (Cont.)

```
1 f = open('stockcode.txt', 'r')
2 h = open('stockcode_copy.txt', 'w')
3
4 data = f.read()
5 h.write(data)
6
7 f.close()
8 h.close()
9
```

Lab (Cont.)

```
1  bufsize = 1024
2  f = open('img_sample.jpg', 'rb')
3  h = open('img_sample_copy.jpg', 'wb')
4
5  data = f.read(bufsize)
6  while data:
7      h.write(data)
8      data = f.read(bufsize)
9
10 f.close()
11 h.close()
12
```

Lab (Cont.)

```
1 with open('stockcode.txt', 'r') as f:
2     for line_num, line in enumerate(f.readlines()):
3         print('%d %s' %(line_num+1, line), end='')
4
```

Console ⊠ Problems PyUnit

<terminated> 144.py [/usr/bin/python3.6]

```
909 101060 SBS홀딩스
910 101140 아티스
911 101380 거북선2호
912 101790 케이알제2호
913 101990 파브코
914 102000 거북선3호
915 102260 동성홀딩스
916 102280 트라이
```

Lab (Cont.)

```
1 spos = 105     # 파일을 읽는 위치 지정
2 size = 500      # 읽을 크기를 지정
3
4 f = open('stockcode.txt', 'r')
5 h = open('stockcode_part.txt', 'w')
6
7 f.seek(spos)
8 data = f.read(size)
9 h.write(data)
10
11 h.close()
12 f.close()
13
```

Console | Problems | PyUnit

<terminated> 144.py [/usr/bin/python3.6]

```
12 000150 두산
13 000151 두산우
14 000152 두산2우B
15 000180 성창기업지주
```

Lab (Cont.)

```
1 from os.path import getsize
2
3 file1 = 'stockcode.txt'
4 file2 = './img_sample.jpg'
5 file_size1 = getsize(file1)
6 file_size2 = getsize(file2)
7
8 print('File Name: %s \tFile Size: %d' %(file1, file_size1))
9 print('File Name: %s \tFile Size: %d' %(file2, file_size2))
10
```

Console ⊠ Problems PyUnit

<terminated> 146.py [/usr/bin/python3.6]

```
File Name: stockcode.txt        File Size: 20083
File Name: ./img_sample.jpg     File Size: 170005
```

Lab (Cont.)

```
1 from os import remove
2
3 target_file = 'stockcode_copy.txt'
4 k = input('[%s] 파일을 삭제하겠습니까? (y/n)' %target_file)
5 if k == 'y':
6     remove(target_file)
7     print('[%s]를 삭제했습니다.' %target_file)
8
```

Console Problems PyUnit

<terminated> 147.py [/usr/bin/python3.6]

```
[stockcode_copy.txt] 파일을 삭제하겠습니까? (y/n)y
[stockcode_copy.txt]를 삭제했습니다.
```

Lab (Cont.)

```
1 from os import rename
2
3 target_file = 'stockcode.txt'
4 newname = input('[%s]에 대한 새로운 파일 이름을 입력하세요: ' %target_file)
5 rename(target_file, newname)
6 print('[%s] -> [%s] 로 파일이름이 변경되었습니다.' %(target_file, newname))
7
8
```

Console ⊠ Problems PyUnit

<terminated> 148.py [/usr/bin/python3.6]

```
[stockcode.txt]에 대한 새로운 파일 이름을 입력하세요: mystockcode.txt
[stockcode.txt] -> [mystockcode.txt] 로 파일이름이 변경되었습니다.
```

Lab (Cont.)

```
1  from os import rename
2
3  target_file = 'mystockcode.txt'
4  newpath = input('[%s]를 이동할 디렉터리의 절대경로를 입력하세요: ' %target_file)
5
6  if newpath[-1] == '/':
7      newname = newpath + target_file
8  else:
9      newname = newpath + '/' + target_file
10
11 try:
12     rename(target_file, newname)
13     print('[%s] -> [%s]로 이동되었습니다.' %(target_file, newname))
14 except FileNotFoundError as e:
15     print(e)
16
```

Console | Problems | PyUnit

<terminated> 149.py [/usr/bin/python3.6]

```
[mystockcode.txt]를 이동할 디렉터리의 절대경로를 입력하세요: /home/instructor/Downloads
[mystockcode.txt] -> [/home/instructor/Downloads/mystockcode.txt]로 이동되었습니다.
```

Lab (Cont.)

```
1  import os, glob
2
3  folder = '/home/instructor/PythonHome/0830'
4  file_list = os.listdir(folder)
5  print(file_list)
6
7  files = '*.py'
8  file_list = glob.glob(files)
9  print(file_list)
10
11
```

Console Problems PyUnit

<terminated> 150.py [/usr/bin/python3.6]

```
['__pycache__', 'test.py', '.project', 'demo.py', '.pydevproject']
['157.py', '197.py', 'myturtle_color.py', '188.py', '151.py', '183.py', '175.py', 'test.py',
```

Lab (Cont.)

```
1 import os
2
3 pdir = os.getcwd(); print(pdir)
4 os.chdir('..'); print(os.getcwd())
5 os.chdir(pdir); print(os.getcwd())
6
7
```

Console Problems PyUnit

<terminated> 151.py [/usr/bin/python3.6]

```
/home/instructor/PythonHome/0823
/home/instructor/PythonHome
/home/instructor/PythonHome/0823
```

Lab (Cont.)

```python
import os

newfolder = input('새로 생성할 디렉터리 이름을 입력하세요: ')
try:
    os.mkdir(newfolder)
    print('[%s] 디렉터리를 새로 생성했습니다.' %newfolder)
except Exception as e:
    print(e)

```

Console | Problems | PyUnit

<terminated> 152.py [/usr/bin/python3.6]

```
새로 생성할 디렉터리 이름을 입력하세요: Util
[Util] 디렉터리를 새로 생성했습니다.
```

Lab (Cont.)

```python
import os

target_folder = 'Util'
k = input('[%s] 디렉터리를 삭제하겠습니까? (y/n)' %target_folder)
if k == 'y':
    try:
        os.rmdir(target_folder)
        print('[%s] 디렉터리를 삭제했습니다.' %target_folder)
    except Exception as e:
        print(e)

```

Console Problems PyUnit

<terminated> 153.py [/usr/bin/python3.6]

```
[Util] 디렉터리를 삭제하겠습니까? (y/n)y
[Util] 디렉터리를 삭제했습니다.
```

Lab (Cont.)

```python
import shutil
import os

target_folder = '/home/instructor/PythonHome/0829'
print('[%s] 하위 모든 디렉터리 및 파일들을 삭제합니다.' %target_folder)
for file in os.listdir(target_folder):
    print(file)
k = input('[%s]를 삭제하겠습니까? (y/n) ' %target_folder)
if k == 'y':
    try:
        shutil.rmtree(target_folder)
        print('[%s]의 모든 하위 디렉터리와 파일들을 삭제했습니다.' %target_folder)
    except Exception as e:
        print(e)

```

Console | Problems | PyUnit

<terminated> 154.py [/usr/bin/python3.6]

```
.project
demo.py
.pydevproject
[/home/instructor/PythonHome/0829]를 삭제하겠습니까? (y/n) y
[/home/instructor/PythonHome/0829]의 모든 하위 디렉터리와 파일들을 삭제했습니다.
```

Lab (Cont.)

```python
import os
from os.path import exists

dir_name = input('새로 생성할 디렉터리 이름을 입력하세요: ')
if not exists(dir_name):
    os.mkdir(dir_name)
    print('[%s] 디렉터리를 생성했습니다.' %dir_name)
else:
    print('[%s]은(는) 이미 존재합니다.' %dir_name)

```

Console | Problems | PyUnit

<terminated> 155.py [/usr/bin/python3.6]

```
새로 생성할 디렉터리 이름을 입력하세요: Util
[Util]은(는) 이미 존재합니다.
```

Lab (Cont.)

```python
import os
from os.path import exists, isdir, isfile

files = os.listdir('/home/instructor/PythonHome/0830')
for file in files:
    if isdir(file):
        print('DIR: %s' %file)

for file in files:
    if isfile(file):
        print('FILE: %s' %file)


```

Console | Problems | PyUnit

<terminated> 156.py [/usr/bin/python3.6]

```
DIR: __pycache__
FILE: test.py
FILE: .project
FILE: .pydevproject
```