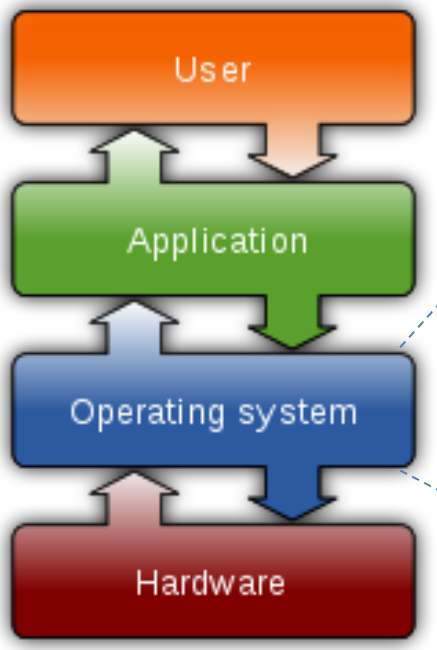
SIT32004
ICT Application Development

File System and Class Overview

Prof. Changbeom Choi

File System [1]

- Definition
  - In computing, a file system or filesystem controls how data is stored and retrieved.



- Operating system controls data storage system (Hardware)
- Operating system provides file system to the application (software)
  - **Filenames**
    - used to identify a storage location in the file system
  - **Directories**
    - allow the user to group files into separate collections

Operating system placement from Wikipedia

File Manipulation in Python [2]

- Requirements
  - You don’t have to import any modules to handle files
- File object
  - You should instantiate file object before processing read, append, write operations to files
  - Use built-in open() function
- Open()
  - You pass the filename and mode as arguments
  - Modes
    - “r”: reading
    - “w”: writing
    - “a”: appending

Open() Example (1/3)

- Write operations

```
00 """file_open.py"""
01 import random
02
03 f = open("test.txt", "w")
04
05 for i in range(100):
06         f.write(str(random.randint(0, 100)))
07         f.write("\n")
08
09 f.close()
```

```
1   59
2   72
3   100
4   1
5   6
6   67
7   49
8   1
9   29
10  86
11  48
12  30
13  92
14  43
15  91
16  20
17  5
18  61
19  34
20  12
21  39
22  54
23  85
```

Output file

Open() Example (2/3)

- Read and Append Operations

```
00 """file_open.py"""
01 import random
02
03 f = open("test.txt", "w")
04
05 for i in range(100):
06         f.write(str(random.randint(0, 100)))
07         f.write("\n")
08
09 f.close()
10
11 f1 = open("append.txt", "a")
12 f2 = open("test.txt", "r")
13
14 for line in f2:
15         f1.write(str(int(line) + 10))
16         f1.write("\n")
17
18 f1.close()
19 f2.close()
```

```
1  97
2  20
3  77
4  63
5  51
6  90
7  64
8  80
9  54
10 54
11 96
12 71
13 62
14 30
15 18
16 24
17 65
18 71
19 110
20 107
21 16
22 105
23 34
24 76
```

Output file

File Read, Write

- Read Operations
  - read() : return as a single string
  - readline() : return one line at a time
  - readlines(): return a list of lines

- Write Operations
  - write() : write a fixed sequence of characters to a file
  - writelines(): write a list of strings

Read, Write Example

```
00 """file_readlines_writelines.py"""
01 import random
02
03 f = open("test.txt", "r")
04
05 lst = f.readlines()
06 f.close()
07 print(lst)
08
09 f = open("dup_test.txt", "a")
10 f.writelines(lst)
11 f.close()
12
13 f1 = open('test.txt', "r")
14 f2 = open("dup_test.txt", "r")
15
16 lst_f1 = f1.readlines()
17 lst_f2 = f2.readlines()
18
19 if len(lst_f1) != len(lst_f2):
20         print("Two files are not same")
21 else:
22         for i in range(len(lst_f1)):
23                 if lst_f1[0] != lst_f2[0]:
24                         print("Two files are not same")
25         else:
26                 print("Two files are same")
```

```
S D:₩Google₩03. Lectures₩2019-1₩SIT32004₩Lecture08₩02Example> .₩file_readlines_writelines.py
'70₩n', '82₩n', '99₩n', '85₩n', '75₩n', '1₩n', '64₩n', '65₩n', '88₩n', '22₩n', '32₩n', '92₩n', '38₩n', '99₩n', '77₩n',
32₩n', '75₩n', '82₩n', '35₩n', '30₩n', '84₩n', '69₩n', '45₩n', '83₩n', '61₩n', '22₩n', '90₩n', '63₩n', '28₩n', '88₩n',
40₩n', '93₩n', '73₩n', '99₩n', '53₩n', '99₩n', '8₩n', '50₩n', '17₩n', '40₩n', '23₩n', '69₩n', '3₩n', '42₩n', '4₩n', '45
n', '98₩n', '14₩n', '96₩n', '24₩n', '35₩n', '25₩n', '76₩n', '65₩n', '20₩n', '74₩n', '22₩n', '0₩n', '94₩n', '13₩n', '3₩n
, '64₩n', '67₩n', '84₩n', '67₩n', '48₩n', '69₩n', '77₩n', '54₩n', '13₩n', '5₩n', '65₩n', '1₩n', '41₩n', '42₩n', '42₩n',
'18₩n', '65₩n', '6₩n', '39₩n', '94₩n', '55₩n', '5₩n', '93₩n', '45₩n', '67₩n', '56₩n', '22₩n', '7₩n', '66₩n', '57₩n', '1
₩n', '58₩n', '59₩n', '61₩n', '54₩n', '29₩n', '65₩n', '87₩n', '24₩n']
wo files are same
S D:₩Google₩03. Lectures₩2019-1₩SIT32004₩Lecture08₩02Example> .₩file_readlines_writelines.py
'70₩n', '82₩n', '99₩n', '85₩n', '75₩n', '1₩n', '64₩n', '65₩n', '88₩n', '22₩n', '32₩n', '92₩n', '38₩n', '99₩n', '77₩n',
32₩n', '75₩n', '82₩n', '35₩n', '30₩n', '84₩n', '69₩n', '45₩n', '83₩n', '61₩n', '22₩n', '90₩n', '63₩n', '28₩n', '88₩n',
40₩n', '93₩n', '73₩n', '99₩n', '53₩n', '99₩n', '8₩n', '50₩n', '17₩n', '40₩n', '23₩n', '69₩n', '3₩n', '42₩n', '4₩n', '45
n', '98₩n', '14₩n', '96₩n', '24₩n', '35₩n', '25₩n', '76₩n', '65₩n', '20₩n', '74₩n', '22₩n', '0₩n', '94₩n', '13₩n', '3₩n
, '64₩n', '67₩n', '84₩n', '67₩n', '48₩n', '69₩n', '77₩n', '54₩n', '13₩n', '5₩n', '65₩n', '1₩n', '41₩n', '42₩n', '42₩n',
'18₩n', '65₩n', '6₩n', '39₩n', '94₩n', '55₩n', '5₩n', '93₩n', '45₩n', '67₩n', '56₩n', '22₩n', '7₩n', '66₩n', '57₩n', '1
₩n', '58₩n', '59₩n', '61₩n', '54₩n', '29₩n', '65₩n', '87₩n', '24₩n']
wo files are not same
```

Output file

Deleting, renaming, and moving files

- Required Python Modules
  - import os
- Deleting files
  - os.remove(path)
- Check file exists
  - os.path.exists(path)
- Rename file
  - os.rename(before, after)
- Move
  - os.rename(before, after)

Directory Management [3]

- Object-oriented file system paths: pathlib
  - This module offers classes representing filesystem paths with semantics appropriate for different operating systems.
  - Basic use
    - » Importing
      - from pathlib import Path
    - » Listing subdirectories
      - p = Path('.')
        [x for x in p.iterdir() if x.is_dir()]
    - » Listing Python source files in this directory tree:
      - list(p.glob('**/*.py'))
    - » Navigating inside a directory tree:
      - p = Path('/etc')
        q = p / 'init.d' / 'reboot'
    - » Retrieve absolute path
      - q.resolve()

Manipulating Directories [4]

- Required Python Modules
  - import os
- Get current working directory
  - os.getcwd()
- Creating a directory
  - os.mkdir(path)
- Creating a Directory with subdirectories
  - os.makedirs(path)
- Deleting a Directory
  - os.rmdir(path)

Python Classes [5]

- Object Oriented Programming
  - Python is an object oriented programming language.
  - Almost everything in Python is an object, with its properties and methods.

- Create a Class
  - Use keyword **class**

```
class MyClass:
  x = 5
```

- Create Object

```
p1 = MyClass()
print(p1.x)
```

Python Classes

- __init__() Function
  - All classes have a function called __init__(), which is always executed when the class is being initiated.
  - Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Python Classes

- Object Methods
  - Objects can also contain methods. Methods in objects are functions that belongs to the object.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Python Classes

- The self Parameter
  - The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Manipulating Object

- Modify Object Properties
  - Assign values to the properties

    ```
    p1.age = 40
    ```

- Delete Object Properties
  - You can delete properties on objects by using the del keyword:

    del p1.age

- Adding Properties to Objects

    p1.abc = 10

- Delete Objects
  - You can delete objects by using the del keyword:

    del p1

Python Inheritance [6]

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
  - **Parent class** is the class being inherited from, also called base class.
  - **Child class** is the class that inherits from another class, also called derived class.

- Create a Parent Class

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

Python Inheritance

- Create a Child Class

```
class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

- Customizing the Child Class
  - Add __init__() function
  - Add properties

Note: The child's __init__() function overrides the inheritance of the parent's __init__() function.

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
    self.graduationyear = 2019
```

Reference

[1] File System, https://en.wikipedia.org/wiki/File_system

[2] File Handling Cheat Sheet in Python, https://www.pythonforbeginners.com/cheatsheet/python-file-handling

[3] File and Directory Access, https://docs.python.org/ko/3.6/library/filesys.html

[4] Creating and Deleting Directories with Python, https://stackabuse.com/creating-and-deleting-directories-with-python/

[5] Python Classes and Objects, https://www.w3schools.com/python/python_classes.asp

[6] Python Inheriance, https://www.w3schools.com/python/python_inheritance.asp

Practice: File Telegram Bot

- Review the telegram_bot.py

Homework10-01

- Attaching Line Number to Source Code
  - You task is to build a program that read source code and generate a text file that contains line number and the source code.
  - You may use format function of the string object

```
"""file_open.py"""
import random

f = open("test.txt", "w")

for i in range(100):
    f.write(str(random.randint(0, 100)))
    f.write("\n")

f.close()
```

Input

```
00 """file_open.py"""
01 import random
02
03 f = open("test.txt", "w")
04
05 for i in range(100):
06  f.write(str(random.randint(0, 100)))
07  f.write("\n")
08
09 f.close()
```

Output

Homework10-02

- Complete the file_telegram_bot.py