SHARDA UNIVERSITY
Beyond Boundaries

Syllabus

UNIT-3: Exception and File Handling.

Exception Handling, Errors, Run Time Errors, Handling I/O Exception, Try-except statement, Raise, Assert.

Files and Directories: Introduction to File Handling, Reading and Writing files, Additional file methods, Working with Directories.

Errors and Exceptions (CO4)

- Certain mistakes may be made while writing a program that led to errors when we try to run it. A python program terminates as soon as it encounters an unhandled error. These errors can be broadly classified into two classes:
- Syntax errors
- Logical errors (Exceptions)

Syntax Error (CO4)

- Error caused by not following the proper structure (syntax) of the language is called **syntax error or parsing error.**
- Syntax errors are the most basic type of error. They arise when the Python parser is unable to understand a line of code. Examples:
- 2=x
  - SyntaxError: can't assign to literal
- x=2

  if x==2

  print('yes')
  - SyntaxError: invalid syntax
- print x
  - SyntaxError: Missing parentheses in call to 'print'.

Logical Errors (CO4)

- Errors that occur at runtime (after passing the syntax test) are called exceptions or logical errors.
- These are the most difficult type of error to find, because they will give unpredictable results and may crash your program.
- Whenever these types of runtime errors occur, Python creates an exception object. If not handled properly, it prints a trace back to that error along with some details about why that error occurred.

SHARDA UNIVERSITY
Beyond Boundaries

ferent Types of Python Exceptions

- **Value Error**
- **Zero division Error**
- **Import Error**
- **Indentation Error**
- **Index Error**
- **Memory Error**

Logical Errors (CO4)

- Logical Errors can be of two types:
  - First is caused by a mistake in the program's logic. You won't get an error message, because no syntax or runtime error has occurred. They occur when the program runs without crashing but produces an incorrect result.
  - Second are caused by Illegal operations that can raise exceptions.
- There are plenty of built-in exceptions in Python that are raised when corresponding errors occur
- We can view all the built-in exceptions using the built-in locals() function as follows:
- print(dir(locals()['__builtins__']))

Value Error (CO4)

- import math
- f=math.factorial(-5)
- print(f)
- ValueError: factorial() not defined for negative values

NameError (CO4)

- print(x)
- NameError: name 'x' is not defined

Type Error (CO4)

- x=5+'c‘
- TypeError: unsupported operand type(s) for +: 'int' and 'str'

Indentation Error(CO4)

- x,y=2,3
- if x==0:
- print(x)
- else:
- print(y)
- IndentationError: expected an indented block

Zero Division Error (CO4)

- a= 5/0
  - ZeroDivisionError: division by zero

import Error(CO4)

- import ab
  - ModuleNotFoundError: No module named 'ab'

Index Error (CO4)

- l=[1,2,3]
- l[3]=4
  - IndexError: list assignment index out of range

What is Python Exception?

An **exception** is a Python object that represents error that occurs during the execution of the program and this disturbs the flow of a program. In general, if Python does not encounter any scripts or lines of code then it raises a type of exception.

Exception Handling (CO4)

- Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them.
- **Exception Handling**
- **Assertions**

Exception Handling (CO4)

- When an error occurs, or exception as we call it, Python will normally stop and generate an error message.
- When a Python encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.
- When Python raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling Exception using try and except(CO4)

- We can handle exceptions in our program by using try block and except block.
- A critical operation which can raise exception is placed inside the try block and the code that handles exception is written in except block.

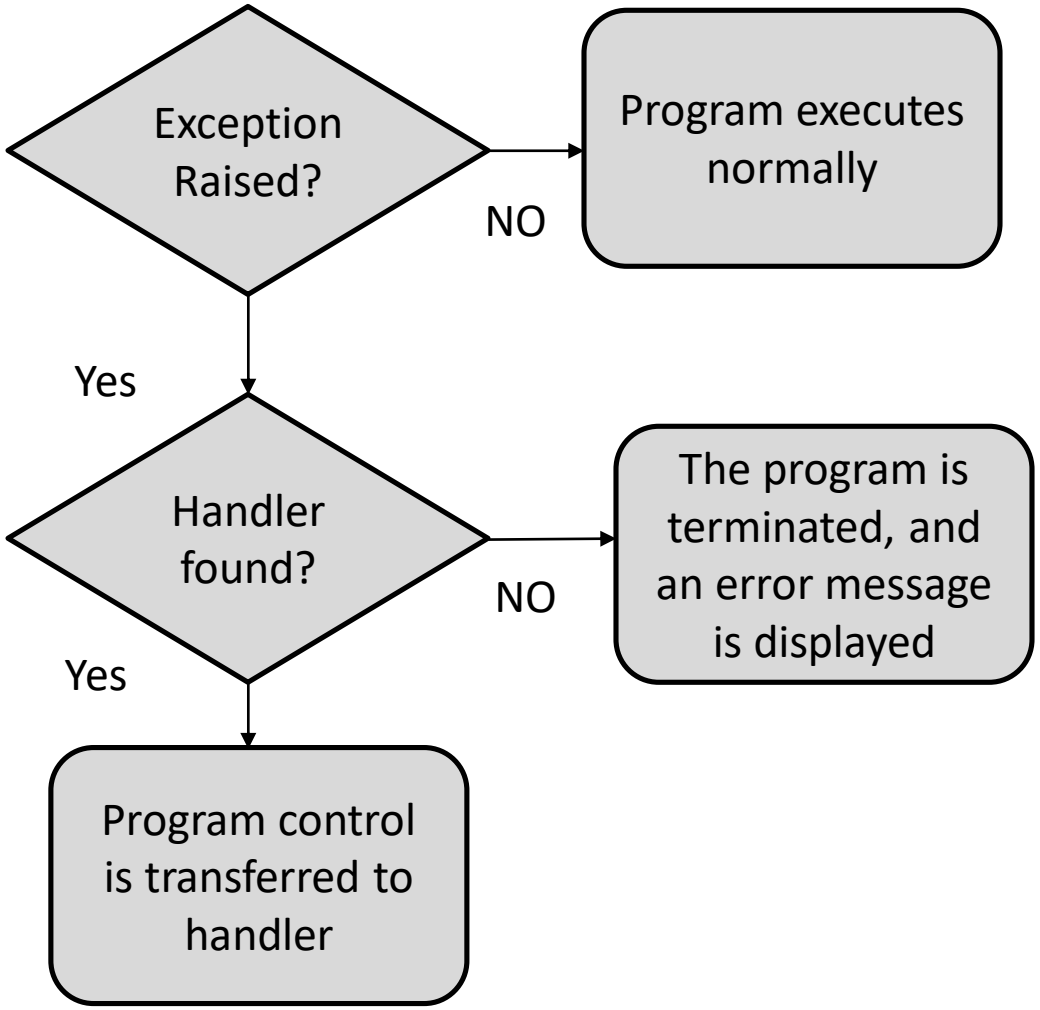
The basic syntax of `try` - `except` statement is,

```
try:
    #statement
except typeofError:
    #statement
```

Handling Exception using try and except(CO4)

- If the code or statement provided in the `try` block has no exception then only try blocks will be executed.
- If any exception occurs in the block of `try` then `try` block skipped and `except` block will be executed.

Handling Exception (CO4)




```
num=int(input("Enter the numerator : "))
num=int(input("Enter the denominator : "))
try:
   quo=num/deno
   print("Quotient :",quo)
Except ZeroDivisionError:
   print("Denominator cannot be zero")
```

Output:

Enter the numerator: 10
Enter the denominator: 0
Denominator cannot be zero

SHARDA UNIVERSITY
Beyond Boundaries

Handling Exception (CO4)

Example 1: Handling ZeroDivisionError

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division by zero is not allowed!")
```

Output:

```
Division by zero is not allowed!
```

Example 2: Handling IndexError

```
try:
    my_list = [10, 20, 30]
    print(my_list[3])
except IndexError:
    print("Index out of range!")
```

Output:

```
Index out of range!
```

Handling Exception (CO4)

Example 3: Handling ValueError

```
try:
    num = int("Hello")
except ValueError:
    print("Invalid value for conversion to int!")
```

Output:

```
Invalid value for conversion to int!
```

Example 4: Handling TypeError

```
try:
    sum = 15 + "20"
except TypeError:
    print("Unsupported operand type for +")
```

Output:

```
Unsupported operand type for +
```

Handling Exception (CO4)

Example 5: Handling FileNotFoundError

```
try:
    with open('anyfile.txt', 'r') as file:
        result = file.read()
except FileNotFoundError:
    print("File not found!")
```

Output:

```
File not found!
```

Example 6: Using else block in exception handling

```
try:
    result = 10 / 2
except ZeroDivisionError:
    print("Division by zero is not allowed!")
else:
    print("Result:", result)
```

Output:

```
Result: 5.0
```

Multiple Exception (CO4)

- Python allows you to have multiple except blocks for a single try block.
- The block which matches with the exception generated will get executed.
- A try block can be associated with more than one except block to specify handlers for different exceptions.
- Exception handlers only handle exceptions that occur in the corresponding try block.

Multiple Exception (CO4)

- **The syntax for specifying multiple except blocks for a single try block can be given as,**

```
try:
    operations are done in this block
     ……….
except Exception1:
    If there is Exception1, then execute this block,
except Exception2:
    If there is Exception2, then execute this block
    ……
else:
     If there is no exception then execute this block.
```

Multiple Exception (CO4)

```
try:
    num=int(input("Enter the number : "))
    print(num**2)
except(KeyboardInterrupt):
    print("You should have entered a number.....Program Terminating....")
except(valueError):
    print("Please check before you enter ..... Program Terminating ....")
print("Bye")
```

Output:

Enter the number : abc
Please check before you enter ..... Program Terminating ....
Bye

Multiple Exception (CO4)

```
try:
    num = int(input("Enter the number : "))
    print(num**2)
except(KeyboardInterrupt, ValueError, TypeError):
    print("Please check before you enter….Pogram terminating….")
Print("Bye")
```

Output:

Enter the number : abc
Please check before you enter….Pogram terminating….
Bye

Except Bock without Exception (CO4)

```
try:
   file = pen('File1.txt')
   str=f.readline()
   print(str)
except IOError:
   print("Error occurred during Input ……Program Terminating…..")
except valueError:
   print("Could not convert data to an integer.")
except:
    print("Unexcepted error……Program Terminating…..")
```

Output:

Unexcepted error……Program terminating….

else Clause (CO4)

- The try ... except block can optionally have an else clause, which, when present, must follow all except blocks. The statement(s) in the else block is executed only if the try clause does not raise an exception.

else Clause (CO4)

```
try:
    file = open('file.txt')
    str = file.readline()
    print(str)
Except IOError:
    print("Error occurred during Input…….Program Terminating……")
else:
     print("Program Terminating Successfully……")
```

Output:

Hello

Program Terminating Successfully….

List of some Standard Exceptions (CO4)

Exception Name	Description
StopIteration	Raised when the next() method of an iterator does not point to any object.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating-point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.

List of some Standard Exceptions (CO4)

Exception Name	Description
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing ctrl+c.
KeyError	Raised when the specified key is not found in the dictionary.
IndentationError	Raised when indentation is not specified properly

finally (CO4)

- The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Output-
Something went wrong
The 'try except' is finished

Raise an exception (CO4)

- As a Python developer you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the raise keyword.
- ***The raise keyword is used to raise an exception.***
- You can define what kind of error to raise, and the text to print to the user.

Example (CO4)

```
x = "hello“
if not type(x) is int:
  raise MyError("Only integers are allowed")
```

Output-

```
Traceback (most recent call last):
    File "demo_ref_keyword_raise2.py", line 4, in <module>
      raise MyError("Only integers are allowed")
  TypeError: Only integers are allowed
```

Python Assert (CO4)

- Python assert keyword is defined as a debugging tool that tests a condition.
- ***The Assertions are mainly the assumption that state a fact confidently in the program.***
- For example, while writing a division function, the divisor should not be zero, and you assert that divisor is not equal to zero.
- It is simply a boolean expression that has a condition or expression checks if the condition returns true or false. If it is true, the program does not do anything, and it moves to the next line of code. But if it is false, it raises an **AssertionError** exception with an optional error message.

Why Assertion is used (CO4)

- It is a debugging tool, and its primary task is to check the condition.
- If it finds that condition is true, it moves to the next line of code, and If not then stops all its operations and throws an error.
- It points out the error in the code.

Where Assertion in Python used (CO4)

- Checking the outputs of the functions.
- Used for testing the code.
- In checking the values of arguments.
- Checking the valid input.

Syntax-

assert condition, error_message(optional)

Example (CO4)

```
def avg(scores):
    assert len(scores) != 0,"The List is empty."
    return sum(scores)/len(scores)

scores2 = [67,59,86,75,92]
print("The Average of scores2:",avg(scores2))

scores1 = []
print("The Average of scores1:",avg(scores1))
```

Output:

The Average of scores2: 75.8

AssertionError: The List is empty.

Faculty Video Links, You tube & NPTEL Video Links and Online Courses Details

- Youtube/other Video Links
- Exception Handling
  - https://youtu.be/NMTEjQ8-AJM

MCQs

1. How many except statements can a try-except block have?
   a) zero
   b) one
   c) more than one
   d) more than zero

Answer: d

2. When is the finally block executed?
   a) when there is no exception
   b) when there is an exception
   c) only if some condition that has been specified is satisfied
   d) always

Answer: d

MCQs

3. What happens when ‘1’ = = 1 is executed?

a) we get a True
b) we get a False
c) an TypeError occurs
d) a ValueError occurs

Answer: b

4. What will be the output of the following Python code?

```
x=10
y=8
assert x>y, 'X too small'
```

a) Assertion Error
b) b) 10 8
c) No output
d) 108

Answer: c

MCQs

5 . What will be the output of the following Python code?

```
def a():
    try:
        f(x, 4)
    finally:
        print('after f')
    print('after f?')
a()
```

a) No output
b) after f?
c) Error
d) after f

Answer: c

References

1. Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd edition, Updated for Python 3, Shroff/O‘Reilly Publishers, 2016.
2. Robert Sedgewick, Kevin Wayne, Robert Dondero, “Introduction to Programming in Python: An Inter-disciplinary Approach” , Pearson India Education Services Pvt. Ltd., 2016.
3. Paul Barry, “Head First: A Brain Friendly Guide” O’Reilly publisher.
4. Reema Thareja, “Python Programming: Using Problem Solving Approach” 2nd Edition, Oxford University Press publisher.
5. Guido van Rossum and Fred L. Drake Jr, “An Introduction to Python”, Revised and updated for Python 3.2, Network Theory Ltd., 2011.

Files and directories (CO4)

FILE HANDLING IN PYTHON

Topic Objective

- To Introduce students about file Handling in Python
- To teach how to do Reading and Writing files
- To train about Working with Directories.

Recap

Let's Recap

In C programming we use files to store data in database permanently that can be accessed whenever required using functions such as:

- Open and close
- Read and write in files

Introduction to File Handling in Python (CO4)

- Files are named locations on disk to store related information. They are used to store data permanently in a non-volatile memory (e.g. hard disk).
- Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), files are used for future use of the data by permanently storing them.
- To read from or write to a file, it needs to be opened first. When operation is done, it needs to be closed so that the resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order:
  - Open a file
  - Read or write (perform operation)
  - Close the file

Relative path and absolute path (CO4)

- An absolute path is defined as specifying the location of a file or directory from the root directory(/).
- In other words, an absolute path is a complete path from start of actual file system from / directory.

Relative path

- Relative path is defined as the path related to the current working directly(cwd). It starts at your current directory and never starts with a / .

Types of files that can be handled in python (CO4)

- Python provides inbuilt functions for creating, writing and reading files.
  - There are two types of files that can be handled in python:
    - Normal text files
    - Binary files (written in binary language,0s and 1s).

Text files:

In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.

Binary files:

In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

Opening and closing file (CO4)

- The key function for working with files in Python is the open() function.

- The open() function takes two parameters
  - filename
  - mode.

Opening and closing file (CO4)

Syntax

- To open a file for reading it is enough to specify the name of the file:

```
f = open(“Myfile.txt")
```

- The code above is the same as:

```
f = open(“Myfile.txt", "rt")
```

- Because "r" for read, and "t" for text are the default values, you do not need to specify them.
- **Note:** Make sure the file exists, or else you will get an error.

Access Modes (CO4)

mode	Description
“r”	Read - Default value. Opens a file for reading, error if the file does not exist
“w”	Write - Opens a file for writing, creates the file if it does not exist
“a”	Append - Opens a file for appending, creates the specified file if it does not exist
“x”	Create - Creates the specified file, returns an error if the file exists

Access Modes (CO4)

mode	Description
“t”	Text - Default value. Text mode
“b”	Binary - Binary mode (e.g. images)

Closing file (CO4)

It is a good practice to always close the file when you are done with it.
Example:
Close the file when you are finish with it:

```
f = open("Myfile.txt", "r")
print(f.read())
f.close()
```

Myfile.txt

```
Hello! Welcome to Myfile.txt
This file is for testing
purposes.
Good Luck!
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

READING AND WRITING FILES

Topic Objective

- To teach how to perform Reading and Writing files and their use in developing projects.

Reading & writing file (CO4)

The open() function returns a file object, which has a read() method for reading the content of the file:

Example

```
f = open("Myfile.txt", "r")
print(f.read())
f.close()
```

Output:

```
Hello! Welcome to
Myfile.txt
This file is for testing
purposes.
Good Luck!
```

Myfile.txt

```
Hello! Welcome to
Myfile.txt
This file is for testing
purposes.
Good Luck!
```

Reading & writing file (CO4)

Read Only Parts of the File

By default, the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("Myfile.txt", "r")
print(f.read(5))
f.close()
```

Output: Hello

Myfile.txt

```
Hello! Welcome to
Myfile.txt
This file is for testing
purposes.
Good Luck!
```

Read Lines (CO4)

(1) You can return one line using readline() method:

Example:

```
f = open("Myfile.txt", "r")
print(f.readline())
f.close()
```

Output: Hello! Welcome to

(2) Calling readline() 2 times, makes reading of first 2 lines:

Example:

```
f = open("Myfile.txt", "r")
print(f.readline())
print(f.readline())
f.close()
```

Output:

Hello! Welcome to
Myfile.txt

Myfile.txt

Hello! Welcome to
Myfile.txt
This file is for testing
purposes.
Good Luck!

Read Lines (CO4)

By looping through the lines of the file, you can read the whole file, line by line:

Example
Loop through the file line by line:

```
f = open("Myfile.txt", "r")
for x in f:
        print(x)
```

Myfile.txt

```
Hello! Welcome to
Myfile.txt
This file is for testing
purposes.
Good Luck!
```

Write to an Existing File (CO4)

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

Write to an Existing File (CO4)

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

writing to an Existing File-append mode (CO4)

Example:

Open the file “Myfile2.txt" and append content to the file:

```
f = open(“Myfile2.txt", "a")
f.write("Now the file has more content!")
f.close()
#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Output:

```
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

Write to an Existing File (CO4)

Example:

Open the file "demofile3.txt" and overwrite the content:

```
f = open("Myfile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Output:

Woops! I have deleted the content!

Note: the "w" method will overwrite the entire file.

Write to an Existing File (CO4)

write() method is used to write a string to an already opened file.

Strings may include numbers, special characters or other symbols.

write() method does not add a newline(‘\n’) character to the end of string.

Syntax:

file.write(string)

Write to an Existing File (CO4)

writelines() method is used to write a list of strings.

Program to write a file using writelines() method

```
f=open("Myfile.txt","w");
lines=["Hello world", "Welcome to python", "Enjoy Python"]
f.writelines(lines)
f.close()
print("Data written to file")
```

Output:

Data written to file

split() using file handling (CO4)

We can also split lines using file handling in Python. This splits the variable when space is encountered. You can also split using any characters as we wish. Here is the code:

```
# Python code to illustrate split() function
file=open(“Myfile.text", "r")
data = file.readlines()
for line in data:
        word = line.split()
        print(word)
```

Additional file methods (CO4)

Method	Description	Example
fileno	returns the file number of the file	file=open(“File1.txt”, “w”); print(file.fileno); Output:3
flush()	flushes the write buffer of the file stream	file=open(“File1.txt”, “w”); file.flush()
isatty()	returns true if the file stream is interactive and false otherwise	file=open(“File1.txt”, “w”); file.write(“Hello”); print(file.isatty()) Output: false
readline(n)	reads and returns one line from file. N is optional. When specified at most n bytes are read	file = open(“MyFile.py”, “r”); print(file.readline(10)) Output file = ope

Additional file methods (CO4)

Method	Description	Example
truncate(n)	resizes the file to n bytes	file=open(“File1.txt”, “w”); file.write(“Welcome to python”); file.truncate(5) file = open(“File.txt”, “r”); print(file.read()) Output: Welco
rstrip()	strips off whitespaces including newline characters from the right side of the string read from the file.	File=open(“File.txt”); Line=file.readline(); print(line.rstrip()) OUTPUT: Greetings to all

Additional file methods (CO4)

- Renaming
- rename() method takes two arguments: Current filename and new filename
- Syntax:
- os.rename(old_filename,new_filename)

Program to rename file1.txt to student.txt (CO4)

```
import os
os.rename("file1.txt", "sudent.txt")
Print("File renamed")
```

Output:

File renamed

Delete files (CO4)

- remove() method:
- This method can be used to delete file. Filename needs to be passed as argument.

Syntax:

- os.remove (filename)

Delete files (CO4)

```
import os
os.remove (“file1.txt”)
print(“File deleted”)
```

Output:

File deleted

Daily Quiz (CO4)

1. To open a file c:\scores.txt for reading, we use _____

a) infile = open("c:\scores.txt", "r")

b) infile = open("c:\\scores.txt", "r")

c) infile = open(file = "c:\scores.txt", "r")

d) infile = open(file = "c:\\scores.txt", "r")

Answer: b

2. Which of the following statements are true?

a) When you open a file for reading, if the file does not exist, an error occurs

b) When you open a file for writing, if the file does not exist, a new file is created

c) When you open a file for writing, if the file exists, the existing file is overwritten with the new file

d) All of the mentioned

Answer: d

Daily Quiz (CO4)

3. To read two characters from a file object infile, we use _____
a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()
Answer: a

4. The readlines() method returns _____
a) str
b) a list of lines
c) a list of single characters
d) a list of integers
Answer: b

WORKING WITH DIRECTORIES

Topic Objective

- To train about Working with Directories and using directories in building real world applications.

Recap

Let's Recap

All folders in computer system are called directories.

We use two types of path in compute system:

- Absolute path
- relative path

Working with Directories (CO4)

- A directory or folder is a collection of files and subdirectories.
- Python has the os module that provides us with many useful methods to work with directories (and files as well).
- These methods allow users to create, remove and change directories.
- Files and folders whose path does not exists in the root folder are assumed to be present in the current working directory.

Working with Directories (CO4)

- mkdir() method
- getcwd() method
- chdir() method
- rmdir() method
- makedirs() method

Working with Directories (CO4)

1. mkdir() method:

Used to create directories in the current working directory.

Syntax:

os.mkdir("dir_name"):

2. getcwd() method:

- Method os.getcwd() displays the Current Working Directory(CWD) of the file used to execute the code.

Syntax:

- os.getcwd():

3. chdir() method:

- Method is used to change current Working Directory(CWD).

os.chdir("dir_name"):

Working with Directories (CO4)

4. rmdir() method:

•Method is used to remove or delete a directory whose name is passed s argument.

Syntax:

os.rmdir(“dir_name”):

•makedirs() method:

Used to create more than one directory.

example:

```
import os
os.makedirs(“C:\\Python12\\dir1\dir2\dir3”)
```

Working with Directories (CO4)

```
import os
print(os.getcwd())
# To print absolute path on your system
# os.path.abspath('.')
# To print files and directories in the current directory
# on your system
# os.listdir('.')
```

Working with Directories (CO4)

os.path.join() method:

This method concatenates various path components with exactly one directory separator (‘/’) following each non-empty part except the last path component. If the last path component to be joined is empty then a directory separator (‘/’) is put at the end.

If a path component represents an absolute path, then all previous components joined are discarded and joining continues from the absolute path component.

Syntax: os.path.join(path, *paths)

Working with Directories (CO4)

(1) os.path.abspath() method:

this method returns the pathname to the path passed as a parameter to this function.

Syntax: os.path.abspath(pathname)

(2) os.path.isabspath(path) method

This method accepts a file path as argument and returns true if path is an absolute path or false otherwise.

Working with Directories (CO4)

(3) os.path.relpath(path, start) method

os.path.relpath() method in Python is used to get a relative filepath to the given path either from the current working directory or from the given directory.

Note: This method only computes the relative path. The existence of the given path or directory is not checked.

Syntax: os.path.relpath(path, start = os.curdir)

Working with Directories (CO4)

(4) os.path.dirname(path) method

os.path.dirname() method in Python is used to get the directory name from the specified path.

Syntax: os.path.dirname(path)

(5) os.path.basename(path) method

os.path.basename() method in Python is used to get the base name in specified path. This method internally use os.path.split() method to split the specified path into a pair (head, tail).

os.path.basename() method returns the tail part after splitting the specified path into (head, tail) pair.

Syntax: os.path.basename(path)

Working with Directories (CO4)

(6) os.path.split(path) method

os.path.split() method in Python accepts a file path and returns its directory name as well as the basename.

Syntax: os.path.split(path)

it is equivalent to two seperate methods:

os.path.dirname() and os.path.basename()

(7) os.path.getsize() method in Python is used to check the size of specified path. It returns the size of specified path in bytes. The method raise OSError if the file does not exist or is somehow inaccessible.

Syntax: os.path.getsize(path)

Working with Directories (CO4)

(8) os.listdir(path) method:

os.listdir() method in python is used to get the list of all files and directories in the specified directory. If we don't specify any directory, then list of files and directories in the current working directory will be returned.

Syntax: os.listdir(path)

(9) os.path.exists(path) method

os.path.exists() method in Python is used to check whether the specified path exists or not. This method can be also used to check whether the given path refers to an open file descriptor or not.

Syntax: os.path.exists(path)

Working with Directories (CO4)

(10) os.path.isfile(path) method

os.path.isfile() method in Python is used to check whether the specified path is an existing regular file or not.

Syntax: os.path.isfile(path)

(11) os.path.isdir(path) method

os.path.isdir() method in Python is used to check whether the specified path is an existing directory or not.

if the specified path is an existing directory then the method will return True.

Syntax: os.path.isdir(path)

Weekly Assignment 5.1

1. Write one similarity and one difference between a+ and w+. CO4
2. What is the difference between read() and read(n) functions? CO4
3. What is the difference between readline() and readlines() ? CO4
4. Write a program to read first 10 characters from a file named “data.txt”. CO4
5. Write a program to read entire content from the file named “test.txt”. CO4

Weekly Assignment 5.1

6. Write a program in python to read first line from the file("data.txt"). CO4
7. Write a program in python to display number of lines in a file("data.txt"). CO4
8. Write a program in python to display first line from the file("data.txt") using readlines(). CO4
9. Accept five names from the user and write in a file "name.txt". CO4
10. Write a function bwrite() to write list of three names in a binary file? CO4

Weekly Assignment 5.1

11. What are files? Why we need them? CO5
12. Differentiate between text and binary files. CO5
13. write a program that reads a file and prints only those lines that has the word ‘print’. CO5
14. write a program that reads and writes the details of a student in a file. CO5
15. Write a program that copies the content of a file in another file. CO5

SUMMARY

- In python file can be either of type text or binary.
- File can be accessed in various modes such as read, write, append
- Open() and close() function are used to open a file and later close it.
- File can be read completely, line by line or a part of file.
- Many methods exists while working with directory such as getcwd(), mkdir(), rmdir() etc.