IO Operations

Agenda

1. Taking user input
2. Formatting the output
3. Read from file
4. Write to file
5. Create a file/folder
6. Delete a file/folder

Taking user input

Taking user input

- **input( )** function is used to accept the user input through keyboard.
- When input( ) function is encountered, Python waits for the user to provide an input.
- By default input( ) function reads the user input as a string.
- Later we can use appropriate casting functions like int( ), float( ) to do type casting.

Program:

```
name = input() #waits for user input
print('Name is ',name)
```

Output:

```
Tanish #we need to enter
Name is Tanish
```

Taking user input continued..

```
Program:

city = input('Enter your city: ') #waits for user input
print('City is ',city)
```

```
Output:

Enter your city: Chennai
City is Chennai
```

Taking user input continued..

```
Program:

msg = input('Enter a msg: ') #waits for user input
print(msg)
```

```
Output:

Enter a msg: Hi I am a student
Hi I am a student
```

Taking user input continued..

```
Program:

s1 = input('Enter a number: ')
num = int(s1) #converting string to int
if(num%2 == 0):
    print('Even')
else:
    print('Odd')
```

```
Output:

Enter a number: 23
Odd
```

Formatting the output

Formatting the output

- From Python 3 version, print is a function.
- **print ( )** function is used to print the output to the console or any standard output device.
- The output to be displayed can be enclosed in either single or double quotes.
- By default **print ( )** function adds newline at the end after printing the output.

```
print('Hi..Welcome..')
print('I am a student')
```

```
Output:


Hi..Welcome..
I am a student
```

Formatting the output continued..

- If you want to print the outputs in a single line, use the **end** parameter.

```
print('Hello World', end='')
print('123')
```

```
Output:
Hello World123
```

- You can print more than one output in a single print ( ) function.

```
print('Wipro' , 'Technologies')
```

```
Output:
Wipro Technologies
```

- **Comma (,)** automatically puts a space between Wipro and Technologies.

Formatting the output using placeholders

- We can format the print statements using placeholders such as %d for number and %s for string.

```
a = 10
b = 20
print("The sum of %d and %d is %d" % (a,b,(a+b)))
#print("The sum of", a, "and", b, "is", (a+b))
```

```
Output:
The sum of 10 and 20 is 30
```

Formatting the output using placeholders

```
name = ‘Chandu’
age = 24
city = ‘Chennai’
print("Hi my name is %s, my age is %d, I am from %s"
      %(name, age, city))
```

```
Output:
Hi my name is Chandu, my age is 24, I am from Chennai
```

Read from file

Read from file

- To read contents from a file, first we need to open the file.
- **open ( )** is the built-in function used to open a file.
- It returns a file object if it is able to locate the file successfully.
- If the file we are trying to open is not present in the directory or the name/path of the file is incorrect, open( ) will raise **FileNotFoundError**.

```
Syntax:
open(filename, mode)
```

Mention the full path of the file if its not in the current working directory.

- We will discuss in detail on mode parameter shortly.

Read from file continued..

Opening a txt file in read mode:

```
f1 = open('check.txt', 'r') #file is in current directory

f1 = open('D:\\check.txt', 'r') #file is in different directory

f1 = open('check.txt')  #mentioning r is optional
```

- **r** denotes the read mode.
- By default the files are opened in read mode, so its optional to mention it explicitly.
- If the mentioned file is not available, we will end up with **FileNotFoundError**.

Read from file continued..

Reading the entire contents of the file and printing it in the console:

```
Execute the code:

f1 = open('check.txt', 'r')
print(f1.read())
f1.close()
```

Always close the file.

Keep this file ready in the current directory..!!

check.txt

Hi welcome to python session.

- **read ( )** built-in function reads the entire content of the file as a string.
- You can receive it in a variable also.

```
content = f1.read()
```

Read from file continued..

Reading a certain bytes of data from the file and printing it in the console:

Execute the code:

```
f1 = open('check.txt')
print(f1.read(2)) #2 characters
print(f1.read(8)) #next 8 characters
print(f1.read()) #remaining characters
f1.close()
```

Keep this file ready in the current directory..!!

check.txt

Hi welcome to python session.

Output:

```
Hi
 welcome
 to python session.
```

Read from file continued..

Reading one line at a time from the file and printing it in the console:

Execute the code:

```
f1 = open('check.txt')
line1 = f1.readline()
print(line1)

line2 = f1.readline()
print(line2)
f1.close()
```

Keep this file ready in the current directory..!!

check.txt

Hi welcome
This is python session
happy learning

Output:

```
Hi welcome
This is python session
```

Read from file continued..

Reading all the lines from the file and store it in a list :

```
Execute the code:

f1 = open('check.txt')
list_of_lines = f1.readlines()
print(list_of_lines)
f1.close()
```

Keep this file ready in the current directory..!!

check.txt

a
b
c
d

```
Output:

['a\n', 'b\n', 'c\n', 'd']
```

Write to file

Write to file

- **write ( )** is the built-in function used to write contents to a file.
- When opening a file for writing, we must mention the mode as either **a** or **w** mode.
- In both modes, if the file we are trying to open is not present in the given directory, a new file will be created.

```
Syntax:
f1=open('check.txt','w') #opens in write mode
f1=open('check.txt','a') #opens in append mode
```

Write to file continued..

Opening an existing file in write mode and writing contents to it :

```
Program:

f1=open('sample.txt','w')
f1.write('Adding new content')
f1.close()
```

Keep this file ready in the current directory..!!

sample.txt

Existing content..

- After executing the above code, open sample.txt and check the contents.
- Opening a file in write mode (**w**) overwrites all the existing contents with the new content.

Write to file continued..

Opening an existing file in append mode and writing contents to it :

```
Program:

f1=open('sample.txt','a')
f1.write('Adding new content')
f1.close()
```

Keep this file ready in the current directory..!!

sample.txt

Existing content..

- After executing the above code, open sample.txt and check the contents.
- Opening a file in append mode (**a**) appends the new content to the end of the existing contents.

Create a file/folder

Create a file

- **x** denotes the create mode.
- Opening a file in create mode will create that file in the given directory.
- If the mentioned file is already existing, we end up with **FileExistsError**.
- Keep the sample.txt file in the current directory and then run this code.

```
Program:

f1=open('sample.txt','x') #opens in create mode
f1.write('Hi..Welcome..')
f1.close()
```

Create a folder

- For creating a directory (folder), we need to load the **os** module.
- **mkdir ( )** method is used to create a directory.
- Below code creates a new directory called **python** in the desktop.

```
Program:

import os
os.mkdir('C:\\Users\\Desktop\\python')
print('Folder created')
```

- Trying to create a directory which already exists results in **FileExistsError**.

Create a folder continued..

How to avoid the FileExistsError?

- This error can be avoided by checking whether the given directory is already available or not. If it's not available then we can call **mkdir ( ).**

```
Program:

import os
result = os.path.exists('C:\\Users\\Desktop\\python')
if result==True:
    print('Folder already exists')
else:
    os.mkdir('C:\\Users\\Desktop\\python')
    print('Folder created')
```

Delete a file/folder

Delete a file

- For deleting a file, we need to load the **os** module.
- **remove ( )** method is used to delete a file.
- Below code deletes the **sample.txt** file from the given location.

```
Program:

import os
os.remove('C:\\Users\\Desktop\\sample.txt')
print('File deleted')
```

- Trying to delete a file which is not available results in **FileNotFoundError**.

Delete a file continued..

How to avoid the FileNotFoundError?

- This error can be avoided by checking whether the given file is already available or not. If it's available then we can call **remove ( ).**

```
Program:

import os
result = os.path.exists('C:\\Users\\Desktop\\sample.txt')
if result==True:
    os.remove('C:\\Users\\Desktop\\sample.txt')
    print('File deleted')
else:
    print('File doesn't exists')
```

Delete a folder

- For deleting a folder, we need to load the **os** module.
- **rmdir ( )** method is used to delete a folder. Only empty folder can be deleted.
- Below code deletes the **python** folder from the given location.

```
Program:

import os
os.rmdir('C:\\Users\\Desktop\\python')
print('Folder deleted')
```

- Trying to delete a folder which is not available results in **FileNotFoundError**.

Delete a folder continued..

How to avoid the FileNotFoundError?

- This error can be avoided by checking whether the given folder is already available or not. If it's available then we can call **rmdir ( ).**

```
Program:

import os
result = os.path.exists('C:\\Users\\Desktop\\python')
if result==True:
    os.rmdir('C:\\Users\\Desktop\\python')
    print('Folder deleted')
else:
    print('Folder doesn't exists')
```

Thank you