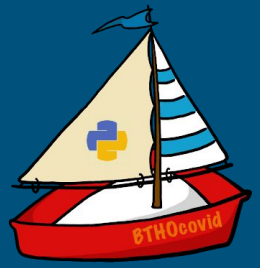


ShiP.py

Learn to Py while Shelter-in-Place

L6: File I/O

A volunteering educational initiative during COVID-19

ShiP Crew

JD

Teddy

Chinmay

Pratik

Siddharth

Umang

Waseem

A volunteering educational initiative during COVID-19

Topics

PHASE I: Foundations

All times are in CDT (GMT-5)

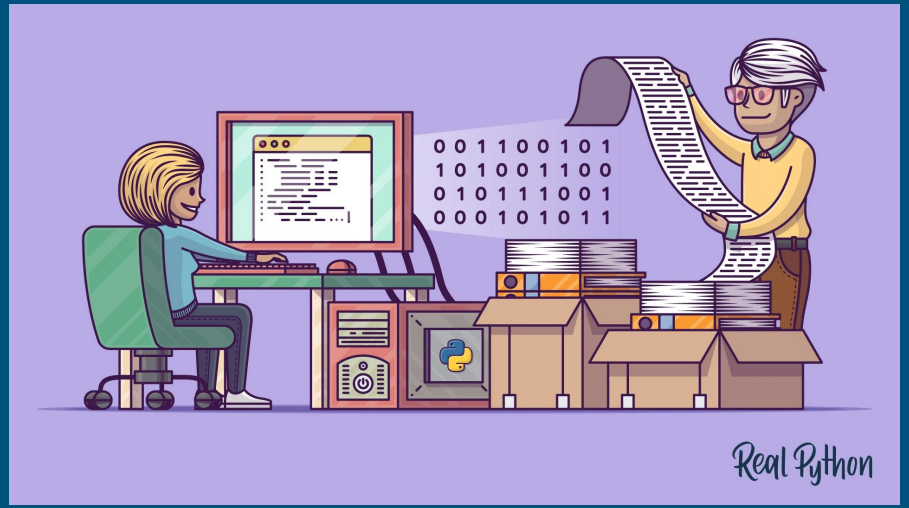
1.	Variables, Expressions, Simple I/O	Sat, April 18 (11 am-12 noon)
2.	Boolean Decisions (branching)	Wed, April 22 (9 pm-10 pm)
3.	Repetitions (loops)	Sat, April 25 (11 am-12 noon)
4A.	Collective Data Structures (Lists and Tuples)	Wed, April 29 (9 pm-10 pm)
4B.	CDS (Dictionaries and Sets)	Sat, May 02 (11 am-12 noon)
5.	Functions	Wed, May 06 (9 pm-10 pm)
6.	File I/O	Sat, May 09 (11 am-12 noon)

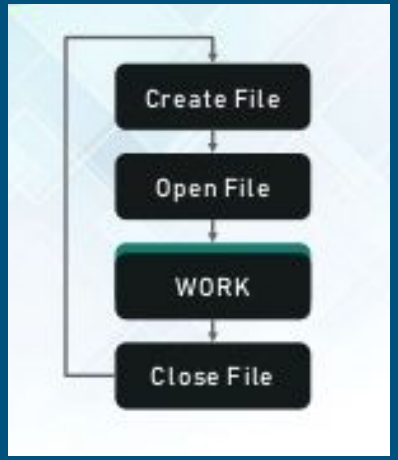
Lecture 6

AGENDA

- Files
- Opening File with access modes
- File reading and writing
- File close and delete
- Python Generators
- Running python scripts offline
- main ( ) function

Files





So far we have been using input ( ) and print ( ) to accept input and write output in the shell/ output window

File Handling in Python

Let's see how to work with common files in python. eg: text (.txt) files.

Opening a file in python

Name of file or file path if file not in the same folder

```
file_object = open(file_name , access_mode)
```

Variable to store the opened file

Mode to open file in eg: read, write, append etc

File access modes

```
file_object = open(file_name , access_mode)
```

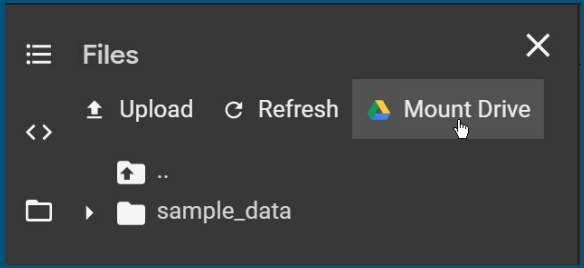
Mode	Description
r	Opens an existing file in reading only mode. Pointer at beginning of file
w	Creates a new file in write-only mode if it does not exist. Opens an existing file in write-only mode and Overwrites it
a	Opens an existing file in append mode. Pointer is at the end of the file
x	In Python3. Opens for exclusive creation, fails if the file already exists. Only writeable. x+ can write and read
<r/w/a>+	Adding + will enable a dual modes. Eg: r+ reading & writing mode
<r/w/a>b	Will open the file in binary format with the given modes
<r/w/a>b+	Similar to <r/w/a>+ but in binary format

Examples: Access modes

```
#mode r: use for reading
f = open('workfile', 'r')

#mode w: use for writing
f = open('workfile', 'w')

#mode x: use for creating and writing to a new file
f = open('workfile', 'x')

#mode a: use for appending to a file
f = open('workfile', 'a' )

#mode r+: use for reading and writing to the same file
f = open('workfile', 'r+')
```

Mounting your Google Drive on Colab

On left side click, click on folder icon, then click the ‘Mount Drive’ button



Run the generated cell code, click on the link , sign in using your google account

```
from google.colab import drive
drive.mount('/content/drive')
```

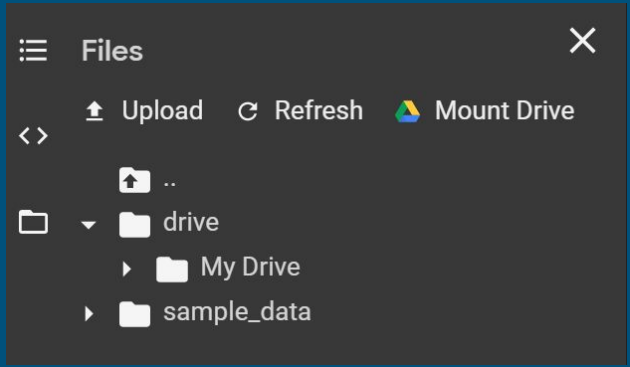
```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn

Enter your authorization code:
```

Enter the authorization code after signing in to your google account

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989

Enter your authorization code:
··********
Mounted at /content/drive
```

Your drive will now be mounted in Colab



File object attributes & closing the file

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

newFile.txt

```python
#Open a file
myFile = open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r')

#File object attributes
print('Name of the file: ', myFile.name)
print('Closed or not: ', myFile.closed)
print('Opening Mode: ', myFile.mode)
```

```
Name of the file:  /content/drive/My Drive/Colab Notebooks/newFile.txt
Closed or not:  False
Opening Mode:  r
```

It is good practice to close a file after working with it to free up space

```python
#Closing a file
myFile.close()
print('Closed or not: ', myFile.closed)
```

```
Closed or not:  True
```

Reading Files

Must be opened in `r` or `<r/w/a>+` or `x+`

Method	Description
`<file_object>.read()`	Returns all file content as a single string
`<file_object>.readline()`	Returns next line of the file as a string, returning text upto and including the next newline character `.strip()` methods removes the terminating newline character from a string
`<file_object>.readlines()`	Reads all remaining lines of a file and stores them in a list of strings

Examples: Reading

```python
myFile = open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r')

print(myFile.read())
```

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

`\n` from original file, not due to `print()`

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

newFile.txt

In Python, end of line character is newline `\n`

Writing to Files

Must be opened in `w` or `a` or `x` or `<r/w/a>+`

Method	Description
`<file_object>.write(<string>)`	Writes new content `<string>` to the file. Remember to add `\n` (newline character) manually after each line.
`<file_object>.writelines(<list>)`	Writes a list of strings as concatenated string at current file pointer

Example: Writing

Creating a file and writing to it, then reading it

```
newFile = open('/content/drive/My Drive/Colab Notebooks/newFile2.txt', 'w')

#Writing to the file and closing it
newFile.write('Howdy, We are almost done with the fundamentals course')
newFile.close()

#Opening the file up again and reading it
newFile = open('/content/drive/My Drive/Colab Notebooks/newFile2.txt', 'r')
print(newFile.read())
newFile.close()
```

```
Howdy, We are almost done with the fundamentals course
```

```
Howdy, We are almost done with the fundamental course
```

newFile2.txt

Example: Appending

Appending to existing file

Append and read mode

```
newFile = open('/content/drive/My Drive/Colab Notebooks/newFile2.txt', 'a+')

#appending to the file
newFile.write('\nI am adding a second line here')

#resetting pointer and reading
newFile.seek(0)
print(newFile.read())

newFile.close()
```

```
Howdy, We are almost done with the fundamentals course
I am adding a second line here
```

`\n` to ensure text is written in newline on file

```
Howdy, We are almost done with the fundamental course
I am adding a second line here
```

newFile2.txt

Writing to File with `print()`

```python
f = open("readme.md", "w")

print("First Line", file=f)
print("Second Line", file=f)
print("Third Line", file=f)

f.close()
```

This program produces the same output as before, the only difference is that in this case, the newline character (`\n`) is automatically added by the `print()` function.

source

seek() and tell()

```
fileObject.seek(offset, whence)
```

Sets the seek pointer at the given offset from whence
whence is optional and defaults to 0 meaning relative to beginning of the file.

```
fileObject.tell()
```

This returns the current position of the seek pointer within the file

Understanding the `seek()` pointer

```python
myFile = open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r')

#To read given number of characters
print(myFile.read(10))

#Reading from where reading ended above
print(myFile.read())
```

```
This is li
ne 1
This is line 2
This is line 3
This is line 4
This is line 5
```

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

newFile.txt

Once you have read through a file, the file pointer needs to be reset to start reading from beginning again in case you do not wish to close() and open() again

```python
newFile = open('/content/drive/My Drive/Colab Notebooks/newFile2.txt', 'r')
print(newFile.read())       #read all file
newFile.seek(0)             #reset the seek pointer

print(newFile.read(10))        #read 10 characters
print(newFile.tell())          #print out the seek pointer position

print(newFile.read(15))      #read 15 more characters
print(newFile.tell())
print(newFile.read())         #read to the end

newFile.close()
```

```
Howdy, We are almost done with the fundamentals course
I am adding a second line here
Howdy, We
10
are almost done
25
 with the fundamentals course
I am adding a second line here
```

```
Howdy, We are almost done with the fundamental course
I am adding a second line here
```

newFile2.txt

Deleting a file - requires os module

```
import os
os.remove('filename')
```

File name/ file path if not in same folder

```
import os
os.remove('/content/drive/My Drive/Colab Notebooks/newFile2.txt')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-13-f41d34bf6de7> in <module>()
      1 import os
----> 2 os.remove('/content/drive/My Drive/Colab Notebooks/newFile2.txt')

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/My D
```

Trying to delete a non-existing file will throw an error

```
import os

if os.path.exists('/content/drive/My Drive/Colab Notebooks/newFile2.txt'):
    os.remove('/content/drive/My Drive/Colab Notebooks/newFile2.txt')
else:
    print('File does not exist')
```

```
File does not exist
```

Using an path.exists to check if file exists and then delete it

Alternate way to open a file `with`

To state simply, with statement is better since it has exception handling built it.

It will also automatically close the file after the block

```
with open('filename', 'mode') as fileObject:
    statement1 ...
    statement2 ...
    statement3 ...
```

Example:

```
Howdy, We are almost done with the fundamental course
I am adding a second line here
```

newFile2.txt

```
with open('/content/drive/My Drive/Colab Notebooks/newFile2.txt', 'r') as myFile:
    print(myFile.readline())
    print(myFile.readline())
```

```
Howdy, We are almost done with the fundamentals course

I am adding a second line here
```

\n from original file

from first print()

Reading very large files

The code below loads all the data form the file into the RAM before the iterating begins

```
with open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r') as f:
  dataInFile = f.readlines()
  for line in dataInFile:
    print(line)
```

read the file in chunks. This will save us from memory crashing

```
with open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r') as f:
  while True:
    line = f.readline().strip()
    if not line:
      break
    print(line)
```

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

\n stripped from original file
So this newline is due to print()

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

newFile.txt

Python Generators

Special kind of functions that returns a lazy iterator. Instead of return we use yield.

Lazy operator - Something you can loop over like a list

Especially useful when you need to load large amount of data without loading all of the data in the memory

```
def myGenerator():
    newList = ['howdy', 3.5, 4, 'aggie']
    for i in newList:
        yield i

for item in myGenerator():
    print(item, end=' ')
```

Notice yield

```
howdy 3.5 4 aggie
```

Another Example

```python
#Reversing a string using a generator
def revString(myStr):
    length = len(myStr)
    for i in range(length-1, -1, -1):
        yield myStr[i]


# For loop to reverse the string
for char in revString("hello"):
    print(char, end='')
```

```
olleh
```

Working with very large files with Generators

The code below loads all the data form the file into the RAM before the iterating begins

```
with open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r') as f:
  dataInFile = f.readlines()
  for line in dataInFile:
    print(line)
```

We can use the generator function to read the file in chunks. This will save us from memory crashing

```
#generator function
def readLargeFile(fileObj):
  while True:
    content = fileObj.readline()
    if not content:
      break
    yield content
```

Example: File with generators

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

newFile.txt

```python
#generator function
def readLargeFile(fileObj):
    while True:
        content = fileObj.readline()
        if not content:
            break
        yield content

#Opening a file and calling generator
with open('/content/drive/My Drive/Colab Notebooks/newFile.txt', 'r') as f:
    print(readLargeFile(f))
    for line in readLargeFile(f):
        print(line)
```

```
<generator object readLargeFile at 0x7f90edc675c8>
This is line 1

This is line 2

This is line 3

This is line 4

This is line 5
```

\n from original file

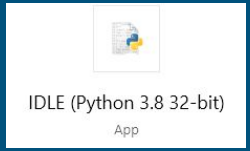
from print()

Working offline with python

Download and install python from the following link

https://www.python.org/downloads/

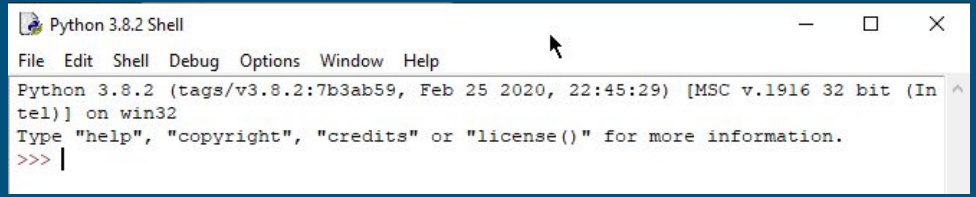
Python comes with an IDLE integrated development environment (IDE). You can use this to write python scripts - a file containing code written in python



You can also use a basic editor like notepad++ to write your python code

Writing scripts using IDLE

Once you open IDLE, you will get a python shell shown on the right. To start writing a script. File → new File. Saved files can be run from command prompt/terminal



Creating & running a python script

myScript.py ×

myScript.py > ...

```python
# A typical python script


def calculator(a, b, key):
    print("This is a calculator")

    if(key == '+'):   # perform addition
        print("Sum = ", a+b)
    elif(key == '-'):  # perform subtraction
        print("Difference = ", a-b)


def main():
    print('This is the main part of my script')
    calculator(20, 15, '+')


if __name__ == '__main__':
    main()
```

I am using VSCode (Editor) here

You can use terminal on mac & linux and command prompt or powershell on windows to run scripts

```
PS E:\ShipPy> dir


    Directory: E:\ShipPy


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----         5/8/2020  10:00 PM                __pycache__
-a----         5/8/2020  10:04 PM            235 myNewScript.py
-a----         5/8/2020  10:25 PM            442 myScript.py


PS E:\ShipPy> python myScript.py
This is the main part of my script
This is a calculator
Sum =  35
PS E:\ShipPy>
```

Windows powershell

`main()` function in python

- In many programming languages, `main()` function is automatically executed when the program runs
- In Python, the interpreter runs code from top to bottom
- `main()` provides control over the starting point
- Useful when running python programs as main scripts or importing them into other programs

```python
def main():
    print('Hello World!')

if __name__ == '__main__':
    main()
```

`__name__` variable

`__name__` is an implicit variable that gets assigned to "`__main__`"

```
print('Value in the name variable is:', __name__)
```

```
Value in the name variable is: __main__
```

By using `if()` statement, we can execute the `main()` function

```
def main():
    print('Hello World!')

if __name__ == '__main__':
    main()
```

Modules in python

- A module can be thought of as a code library
- It is a program containing a set of functions that you want to use in your python code
- When you import a python program `program1.py` as a module in another python program `program2.py`, and run the latter as main program,
  - `__name__` variable for `program1.py` (imported program) is set to `"program1"`
  - `__name__` variable for `program2.py` (main program) is set to `"__main__"`
  - This prevents running the `main()` function in the imported `file1.py` and rather runs the `main()` in the currently running `file2.py`

myScript.py × myNewScript.py

myScript.py > ...

```python
# A typical python script


def calculator(a, b, key):
    print("This is a calculator")

    if(key == '+'):   # perform addition
        print("Sum = ", a+b)
    elif(key == '-'):  # perform subtraction
        print("Difference = ", a-b)


def main():
    print('This is the main part of my script')
    calculator(20, 15, '+')


if __name__ == '__main__':
    main()

print('Value of built in variable:', __name__)
```

```
PS E:\ShipPy> & C:/Users/Waseem/AppData/Local/Programs
This is the main part of my script
This is a calculator
Sum =  35
Value of built in variable: __main__
PS E:\ShipPy>
```

myScript.py myNewScript.py ×

myNewScript.py

```python
import myScript

print('Done importing\n')
myScript.calculator(8, 10, '+')
```

```
PS E:\ShipPy> & C:/Users/Waseem/AppData/Local/Programs
Value of built in variable: myScript
Done importing

This is a calculator
Sum =  18
PS E:\ShipPy>
```

When you import a module `myScript.py` in `myNewScript.py` , python automatically runs all the code in `myScript.py`

`if()` statement in `myScript.py` (imported) does not allow running its `main()` function from within the currently running `myNewScript.py`

myScript.py | myNewScript.py ×

myNewScript.py > ...

```python
import myScript

print('Done importing\n')


def main():
    myScript.calculator(8, 10, '+')


if __name__ == '__main__':
    main()

print('Value of local __name__ variable:', __name__)
```

Executed first

```
PS E:\ShipPy> & C:/Users/Waseem/AppData/Local/Programs
Value of built in variable: myScript
Done importing

This is a calculator
Sum =  18
Value of local __name__ variable: __main__
```

The zen of python - guidelines for design in python

```python
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

https://inventwithpython.com/blog/2018/08/17/the-zen-of-python-explained/

Thank You