IO with Files

- File objects are Python code's main interface to external files on your computer.

Writing into a file

```
file=open('new.txt','w')
```

```
file.write("this is the first line ")
```

```
23
```

```
file.close()
```

Reading from a file

```
read=open('new.txt')
```

```
read.read()
```

```
'this is the first line '
```

- file.seek(0)…… to return the curser back to the first line

Files

- To get access to the files of different locations:

- Find the location of current file:
  - pwd

- File=open('C:\\Users\\Desktop\\New Folder\\read.txt', 'w')

More file operations

Reading, Writing, Appending Modes

- **mode='r'** is read only
- **mode='w'** is write only (will overwrite files or create new!)
- **mode='a'** is append only (will add on to files)
- **mode='r+'** is reading and writing
- **mode= 'w+'** is writing and reading (Overwrites existing files or creates a new file!)

A list of the different modes of opening a file:

Modes	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 will be at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be 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.
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.

A list of the different modes of opening a file:

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.
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.

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:
- 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.
- If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example:

```
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
position = fo.tell();
print "Current file position : ", position
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
fo.close()
```

- This would produce following result:

```
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 you need to import it first and then you can all any related functions.

The rename() Method:

The *rename()* method takes two arguments, the current filename and the new filename.

Syntax:

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

Example:

```
import os
os.rename( "test1.txt", "test2.txt" )
```

The *delete()* Method:

You can use the *delete()* method to delete files by supplying the name of the file to be deleted as the argument.

Syntax:

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

Example:

```
import os
os.remove("test2.txt")
```

Directories in Python:

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

The *mkdir()* Method:

You can use the *mkdir()* method of the os module to create directories in the current directory. You need to supply an argument to this method, which contains the name of the directory to be created.

Syntax:

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

Example:

```
import os # Create a directory "test"
os.mkdir("test")
```

The *chdir()* Method:

You can use the *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax:

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

Example:

```
import os
os.chdir("/home/newdir")
```

The *getcwd()* Method:

The *getcwd()* method displays the current working directory.

Syntax:

```
os.getcwd()
```

Example:

```
import os
os.getcwd()
```

The *rmdir()* Method:

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')
```

Example:

```
import os
os.rmdir( "/tmp/test" )
```

File & Directory Related Methods:

There are three important sources which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows:

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods.: This provides methods to process files as well as directories.

Thank You!

پوهنتون کاردان
KARDAN UNIVERSITY

MODERN PROGRAMMING LANGUAGE (Python)

Control Statements

Learning Outcome

- You should learn Comparison operators
- Should Know about If statement in python
- Should be able to use elif statements in python
- How to use For loops in python
- How to use While loops in python
- Some General programming practice

Comparison operator

We can use logical operators to combine comparisons:

- **and**
- **or**
- **not**

Python Comparison Operators		
Operator	**Name**	**Example**
==	Equal	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

- EX: X >3 and X<5
  - X== 6 or Y>4

- Python Arithmetic Operator
- Python Relational Operator
- Python Assignment Operator
- Python Logical Operator
- Python Membership Operator
- Python Identity Operator
- Python Bitwise Operator

Conditional statements

- There are basic three types of conditional statements:
  - **if** statements
  - **elif** statements
  - **else** statement

Indentation

- For the control flow in python we need to use indentation and colon
- Indentation makes the python to be different than other languages
- It is used to create blocks of code instead of bracket ({ })
- Example:

```
speed =160


if speed>=140:
    print("Be careful! speed is high")
```

```
Be careful! speed is high
```

elif, else

```
speed =120



if speed>=140:
    print("Be careful! speed is high")
elif speed >=120:
    print("The speed goes high, manage it!")
else:
    print("Normal Speed!")
```

```
The speed goes high, manage it!
```

Practice

- Using if elif statements, print the number for the corresponding grade.

- A=4, B=3, C=2, D=1, F=0

- Get the input from user, using:
- Grade=int(input(“Enter the grade”)) == 99 A+
- if grad => 91 and grad <=100:
  - Print (“A + ”)

```
if grad == 4:
        print (“A”)
```

For loops

- Most of Objects in Python are iterable
- We can iterate through the different parts or elements of a list, tuple, dictionary and string
- Example:

```
my_list=[23,442,'A','B',34.56]

for item in my_list:
    print(item)
```

```
23
442
A
B
34.56
```

Customized for loop

- To print only some portions of the elements of list we can define as the following:
- consider the following list example:

```
my_list=[1,2,3,4,5,6,7,8,9,10]
```

- To print from 2 to 5 :
- Start list[1:2]
- End
- Step

```
my_list[1:5:1]
```

```
[2, 3, 4, 5]
```

More examples

- Function range (num) can provide any number from 0 up to the range number-1.
- The range(starting, ending, step)
- for (int I =0; I < 5 ; I ++)
- Here we get number from zero up to 4

```
for number in range(5):
    print(number)
```

```
0
1
2
3
4
```

- To combine for with if :

```
for number in range(10):
    if number %2==0:
        print (f"The number is even: {number}")
```

```
The number is even: 0
The number is even: 2
The number is even: 4
The number is even: 6
The number is even: 8
```

Practice

- Question 1. Create a list of numbers
- sum
- Range(10)
  - sum/Len (list)
  - Sum all the numbers
  - Find the average and print the output

Iterate through String

- We can iterate through different letters or characters of a string :

```
my_string="WELCOME"

for letter in my_string:
    print(letter)
```

```
W
E
L
C
O
M
E
```

```
my_string="WELCOME"

for letter in my_string:
    print(letter, end=' ')
```

```
W E L C O M E
```

Having nested for

```
my_list=[[1,2,3],[4,5,6],[7,8,9]]

for item in my_list:
    for num in item:
        print(num)
```

```
1
2
3
4
5
6
7
8
9
```

Iterate through dictionary

- There are three function for the dictionary:
  - Dic.values()
  - Dic.keys()
  - Dic.items()
- Example:

```
dictionary={'apple': 78, 'orange': 100,'watermelon':10}
```

```
for goods in dictionary.keys():
    print (goods)
```

```
apple
orange
watermelon
```

Break , Continue, Pass

- Break:
  - Breaks out the current loop and goes to the rest of the code
- Continue:
  - Skips the rest of the current iteration and goes to the next one
- Pass:
  - Does nothing, lets the interpreter to continue with code

```
x=10


if x>10:
    pass
```

Thank You!