PYTHON

Lecture - 09

Lecture Topics

- Reading and Writing Files
- Exception Handling

Files and File Paths

- A file has two key properties: a filename (usually written as one word) and a path (C:\Users\Al\Documents)
- Backslash on Windows and Forward Slash on macOS and Linux

```
>>> from pathlib import Path
>>> Path('spam', 'bacon', 'eggs')

WindowsPath('spam/bacon/eggs')
>>> str(Path('spam', 'bacon', 'eggs'))
'spam\\bacon\\eggs'
```

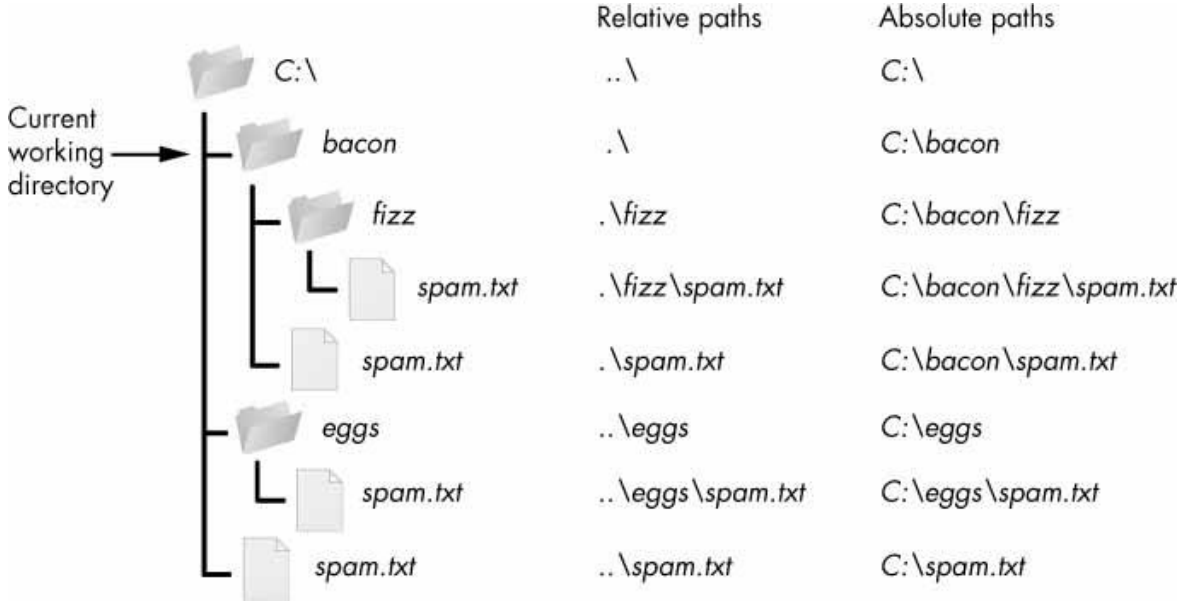
```
>>> from pathlib import Path
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
        print(Path(r'C:\Users\Al', filename))
C:\Users\Al\accounts.txt
C:\Users\Al\details.csv
C:\Users\Al\invite.docx
```

The Current Working Directory

- Current working directory: path.cwd()
- Change directory: path.chdir()

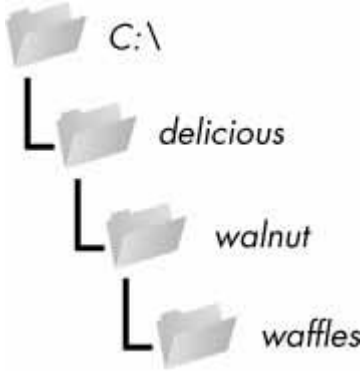
```
>>> from pathlib import Path
>>> import os
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')'
>>> os.chdir('C:\\Windows\\System32')
>>> Path.cwd()
WindowsPath('C:/Windows/System32')
```

Absolute vs. Relative Paths



	Relative paths	Absolute paths
C:\	..\	C:\
bacon	.\	C:\bacon
fizz	.\fizz	C:\bacon\fizz
spam.txt	.\fizz\spam.txt	C:\bacon\fizz\spam.txt
spam.txt	.\spam.txt	C:\bacon\spam.txt
eggs	..\eggs	C:\eggs
spam.txt	..\eggs\spam.txt	C:\eggs\spam.txt
spam.txt	..\spam.txt	C:\spam.txt

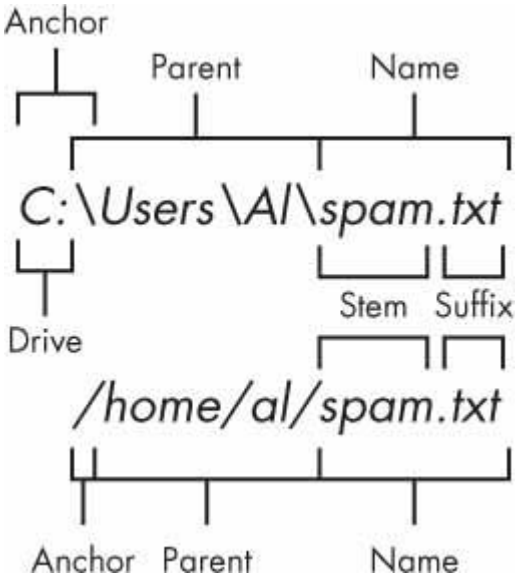
Creating New Folders Using the os.makedirs() Function




```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

Getting the Parts of a File Path (1)

- >>> p = Path('C:/Users/Al/spam.txt')
- >>> p.anchor
- 'C:\\'
- >>> p.parent # This is a Path object, not a string.
- WindowsPath('C:/Users/Al')
- >>> p.name
- 'spam.txt'
- >>> p.stem
- 'spam'
- >>> p.suffix
- '.txt'
- >>> p.drive
- 'C:'



Getting the Parts of a File Path (1)

- The anchor, which is the root folder of the filesystem
- On Windows, the drive, which is the single letter that often denotes a physical hard drive or other storage device
- The parent, which is the folder that contains the file
- The name of the file, made up of the stem (or base name) and the suffix (or extension)

Finding File Sizes and Folder Contents

- Calling os.path.getsize(path) will return the size in bytes of the file in the path argument.
- Calling os.listdir(path) will return a list of filename strings for each file in the path argument. (Note that this function is in the os module, not os.path)

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
27648
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--snip--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

Checking Path Validity

- Calling p.exists() returns True if the path exists or returns False if it doesn't exist.
- Calling p.is_file() returns True if the path exists and is a file, or returns False otherwise.
- Calling p.is_dir() returns True if the path exists and is a directory, or returns False otherwise.

```
>>> winDir = Path('C:/Windows')
>>> notExistsDir = Path('C:/This/Folder/Does/Not/Exist')
>>> calcFile = Path('C:/Windows
/System32/calc.exe')
>>> winDir.exists()
True
>>> winDir.is_dir()
True
>>> notExistsDir.exists()
False
>>> calcFile.is_file()
True
>>> calcFile.is_dir()
False
```

The File Reading/Writing Process

- Plaintext files
- Binary files

```
main.py
1
2  print("Hello World")
```

File Handling: Open

- open() function takes two parameters; filename, and mode.
- You can also specify the type of file
- f = open("demofile.txt")
- f = open("demofile.txt", "rt")

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

"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

Read and Close file

- ```
  f = open("demofile.txt", "r")
  print(f.read())
  ```
- ```
  f = open("D:\\myfiles\welcome.txt", "r")
  print(f.read())
  ```
- ```
  f = open("demofile.txt", "r")
  print(f.read(5)) # Return the 5 first characters of the file
  ```
- ```
  f = open("demofile.txt", "r")
  print(f.readline()) # Read one line of the file
  ```
- ```
  f = open("demofile.txt", "r")
  for x in f: # Loop through the file line by line
    print(x)
  ```
- ```
  f.close() # Close the file when you are finish with it
  ```

File Methods

Method	Description
close()	Closes the file
detach()	Returns the separated raw stream from the buffer
fileno()	Returns a number that represents the stream, from the operating system's perspective
flush()	Flushes the internal buffer
isatty()	Returns whether the file stream is interactive or not
read()	Returns the file content
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size
writable()	Returns whether the file can be written to or not
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

Create/Write Files

- Existing File
  - ```
    f = open("demofile2.txt", "a") # a = append, w = overwrite existing content
    f.write("Now the file has more content!")
    f.close()
    ```
- New File
  - ```
    f = open("myfile.txt", "x")
    ```

"x" - Create - will create a file, returns an error if the file exist
"a" - Append - will create a file if the specified file does not exist
"w" - Write - will create a file if the specified file does not exist

Delete Files

- Delete file
  - ```
    import os
    os.remove("demofile.txt") # if not exist produce an error
    ```
- Check
  - ```
    import os
    if os.path.exists("demofile.txt"):
      os.remove("demofile.txt")
    else:
      print("The file does not exist")
    ```
- Delete folder
  - ```
    import os
    os.rmdir("myfolder") # You can only remove empty folders
    ```
    ```

with statement in Python

- Automatic resource management
- Ensures that the file is automatically closed after operations are done, even if exceptions occur during file operations
- you can use multiple contexts in a single with statement

```python
# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()

# 2) without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()
```

```python
# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

```python
with open('file1.txt', 'r') as file1, open('file2.txt', 'r') as file2:
    content1 = file1.read()
    content2 = file2.read()
# Both files are automatically closed after the with block is exited.
```

Read line by line

- Split lines while reading files
- The split () returns words in list format

```
# Python code to illustrate split() function
with open("geeks.txt", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print (word)
```

File Name at Command Line Argument

```
import sys
import os
from os import path

if len(sys.argv) == 1:
    sys.exit("Format: <python-command> <argu1>...<arguN>")

#initialize the fileNames
fileList = []
i = 1
while i < len(sys.argv):
    if path.isfile(sys.argv[i]) is False:
        sys.exit("File " + sys.argv[i] +" not exit")
    else:
        fileList.append (sys.argv[i])
    i += 1

#count init
count = {}

# reading files and counting
for index in range(len(fileList)):
    try:
         file = open(fileList[index],"r")
....
```

Exercise Time

Problems

- Find all prime numbers from input file (input.txt) which contains values as follows: 2 13 10 12 15 17
- Find all vowel from a input.txt file.

Exception Handling/Managing Exception

- When an error occurs, or exception as we call it, Python will normally stop and generate an error message
-
```
try:
  print(x)
except:
  print("An exception occurred")
```

The `try` block lets you test a block of code for errors.
The `except` block lets you handle the error.
The `else` block lets you execute code when there is no error.
The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

Many Exceptions

- ```
  try:
    print(x)
  except NameError:
    print("Variable x is not defined")
  except:
    print("Something else went wrong")
  ```
  ```

Else

- The try block does not raise any errors, so the else block is executed:
- ```
  try:
    print("Hello")
  except:
    print("Something went wrong")
  else:
    print("Nothing went wrong")
  ```
  ```

Finally

- 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")
  ```
- can be useful to close objects and clean up resources.

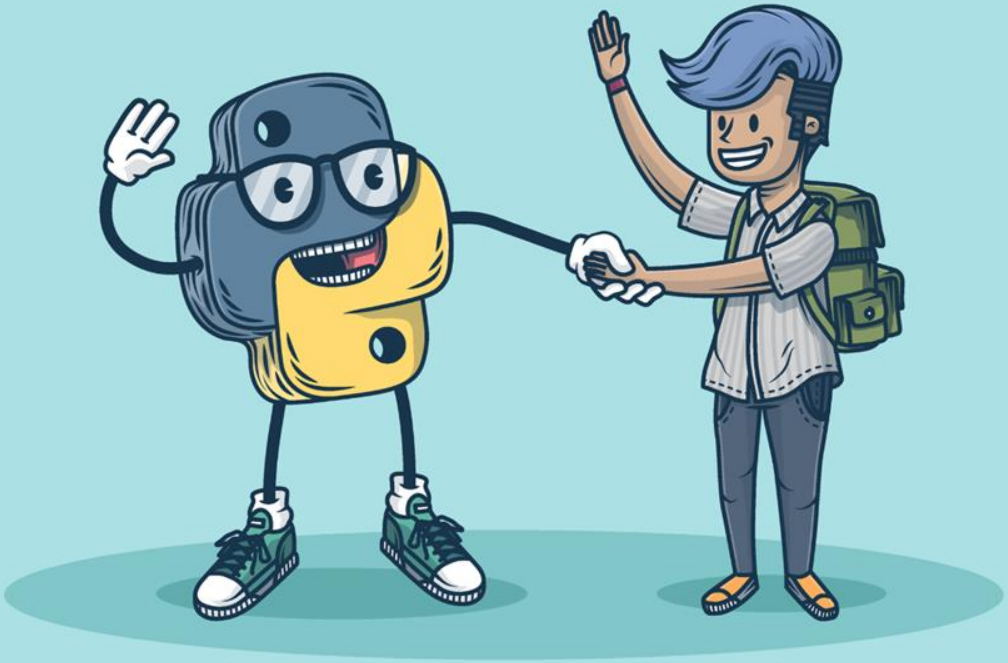
```
file = ('file_path', 'w’)
try:
    file.write('hello world’)
finally:
    file.close()
```
```

Raise an exception

- ```
  x = -1
  if x < 0:
    raise Exception("Sorry, no numbers below zero")
  ```
- ```
  x = "hello"
  if not type(x) is int:
    raise TypeError("Only integers are allowed")
  ```

Different Types of Exception

- **SyntaxError:** This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.
- **TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
- **NameError**: This exception is raised when a variable or function name is not found in the current scope.
- **IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence types.
- **KeyError**: This exception is raised when a key is not found in a dictionary.
- **ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.
- **AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.
- **IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.
- **ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.
- **ImportError**: This exception is raised when an import statement fails to find or load a module.



Thank You

Resources

- https://www.geeksforgeeks.org/file-handling-python/
- https://www.geeksforgeeks.org/with-statement-in-python/
- https://www.tutorialspoint.com/what-is-the-use-of-the-with-statement-in-python
- https://www.w3schools.com/python/python_try_except.asp
- Head First Python, 3rd Edition by Paul Barry
- Automate the Boring Stuff with Python By Al Sweigart.