Exception Handling and File Handling in Python

STUDY NOTES

- **Syntax Error:** Syntax errors are produced by Python when it is translating the source code into byte code. They usually indicate that there is something wrong with the syntax of the program. **Example:** Omitting the colon at the end of a def statement yields the somewhat redundant message **SyntaxError: invalid syntax**.
- **Exception:** An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error. When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- **User-defined exception:** In Python, users can define custom exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from the built-in Exception class. Most of the built-in exceptions are also derived from this class.
- **Raising exception:** The raise keyword is used to raise an exception. The raise keyword raises an error and stops the control flow of the program.
- **The try-except statement:** If the Python program contains suspicious code that may throw the exception, we must place that code in the try block. The try block must be followed with the except statement, which contains a block of code that will be executed if there is some exception in the try block.

  Syntax

```
try:
   #block of code
except Exception1:
   #block of code
except Exception2:
   #block of code
#other code
```

- **Finally clause:** It defines a block of code to run when the try... except...else block is final. The finally block will be executed no matter if the try block raises an error or not. This can be useful to close objects and clean up resources.
- **Built in exceptions:** The table below shows built-in exceptions that are usually raised in Python:

Exceptions	Description
ArithmeticError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
FloatingPointError	Raised when a floating point calculation fails

GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indendation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+C, Ctrl+Z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
NotImplementedError	Raised when an abstract method requires an inherited class to override the method
OSError	Raised when a system related operation causes an error
OverflowError	Raised when the result of a numeric calculation is too large
ReferenceError	Raised when a weak reference object does not exist
RuntimeError	Raised when an error occurs that do not belong to any specific exceptions
StopIteration	Raised when the next() method of an iterator has no further values
SyntaxError	Raised when a syntax error occurs
TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
UnboundLocalError	Raised when a local variable is referenced before assignment
UnicodeError	Raised when a unicode problem occurs
UnicodeEncodeError	Raised when a unicode encoding problem occurs
UnicodeDecodeError	Raised when a unicode decoding problem occurs
UnicodeTranslateError	Raised when a unicode translation problem occurs
ValueError	Raised when there is a wrong value in a specified data type
ZeroDivisionError	Raised when the second operator in a division is zero

- **File Handling:** File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

  There are three types of files that can be handled in Python, normal text files, binary files (written in binary language,0s and 1s) and CSV files.

Access 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. Gives error if file does not exist.
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 placed at the beginning of the file.

rb+	Opens a file for both reading and writing in binary format. The file pointer is placed 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.
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.

- **Text File:** Text files are structured as a sequence of lines, where each line includes a sequence of characters. Each line is terminated with a special character, called the EOL or End of Line character. There are several types, but the most common is the comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.
- In Python, a file operation takes place in the following order.
  1. Open a file
  2. Read or write (perform operation)
  3. Close the file
- **Open a file:** To open the file, use the built-in open() function. The open() function returns a file object, it is used to read or modify the file accordingly

  Syntax

  ```
  file_object = open("filename", "mode")
  ```

  where

  file_object is the variable to add the file object.

  Filename is the name of file

  Mode(optional) tells which way the file will be used.
- Mode specifies whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode. The default is reading in text mode. In this mode, we get strings when reading from the file.

  For example:

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

  where

```
f is the object of file
"abc.txt" is the name of file
"w" is the mode
```

- **Closing a file:** The close() function closes the file and frees the memory space acquired by that file.

  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.

```
f=open("abc.txt","w")
   … . . .. . . .
f.close()
```

- **With statement:** The with statement will automatically close the file after the nested block of code.

```
with open("filename") as file:
do something with data
with open('abc.txt', 'w') as f:
    f.write('Hi there!')
```

- **Writing in file:** There are two ways to write in a file.
  1. **write():** Inserts the string str1 in a single line in the text file.

     `File_object.write(str1)`
  2. **writelines():** For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

```
L = [str1, str2, str3]
File_object.writelines(L)
```

- **Reading from text file:** Using read() function: Read `n` no of bytes and returns it in form of a string. If `n` is not specified, reads the entire file.

  File_object.read([n])

- **Using readline():** Reads a line of the file and returns in form of a string.

  If `n` is given, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.

  File_object.readline([n])

- **Using readlines():** Reads all the lines and return them as each line a string element in a list.

  File_object.readlines()

- **Binary Files:** A binary file is a file stored in binary format. A binary file is computer-readable but not human-readable. All executable programs are stored in binary files, as are most numeric data files. In contrast, text files are stored in a form (usually ASCII) that is human-readable.
- **Differentiate between Binary Files and Text Files: 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. A text file stores data in the form of alphabets, digits and other special symbols by storing their ASCII values and are in a human readable format.
- **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. A binary file stores the data in the same way as stored in the memory. The .exe files, mp3 file, image files, word documents are some of the examples of binary files. We can't read a binary file using a text editor.
- **Pickle:** Python Pickle is used to serialize and deserialize a Python object structure. It's the process of converting a Python object into a byte stream to store it in a file/database. Any object on Python can be pickled so that it can be saved on disk.

- **Writing in Binary File:** For writing in binary file, open the file in 'wb'(or 'w+b') mode.
  The 'w' means that you'll be writing to the file, and 'b' refers to binary mode.
  `f=open(filename,"wb")`
- **Using dump():** dump() is used for writing in file. It takes two parameters, the object you want to pickle and the file to which the object has to be saved.
  `dump('object to write', 'file object')`
- **Reading from file:** For reading a binary file, open the file in 'rb' mode, 'r' means you'll be reading from file.
  `f=open(filename,"rb")`
- **using load():** load() is used for reading from file. It takes one 'file object' parameter, reading data from and then assigned the contents of file to new variable.