21/4/23

ASSIGNMENT 3

PAGE NO. 37
DATE: / /

Unit-5 & 6 Exception Handling and Regular Expression & File Handling

Q1 What is Exception Handling? List out and explain Built-in Exceptions.

→ Exception Handling is a programming construct that deals with runtime errors or exceptional conditions that occur during the execution of a program. It allows the program to gracefully handle these exceptions and prevent the program from crashing or behaving unpredictably.

Some of the commonly used built-in exceptions are:

i) IndexError: When you try to access an Index that is outside the range of valid indices for a sequence.

ii) KeyError: When you access a dictionary key that does not exist.

iii) TypeError: Performing operation on a data type that is not supported, like adding a string and an integer.

iv) ValueError: When a function or method receives an argument that is of correct type but has an inappropriate value.

v) NameError: Accessing a variable which is not defined.

vi) ZeroDivisionError : When you try to divide a number by zero.

vii) ImportError : When a module or package cannot be imported.

viii) FileNotFound : When a file or directory cannot be found.

ix) KeyboardInterrupt : When user interrupts the execution of a program, (Ctrl + C)

x) AssertionError : When an assert statement fails.

Q.2 How to handle exceptions, explain syntax & give one examples.

→ To handle exceptions in Python, you use a try-except block. The syntax of the try-except block is as follows:

*
```
try:
    # Code that might raise an exception.
except ExceptionType:
    # Code to handle the exception.
```

* Example :

```
While True:
    try:
        x = int (input ("Please enter a number: "))
        break
    except ValueError:
        print ("Oops! That was not a valid number")
print ("You Entered:", x)
```

Q.3 Explain except clause with no exception with an example.

—↓ It is possible to use an 'except' clause with no exception specified, which is known as a catch-all or generic 'except' clause. The clause will catch any exception that is not caught by other 'except' clauses in the same 'Try' block.

eg:
```
Try:
    # Some code that might raise an exception.
except Exception as e:
    print("An exception occurred:", e)
```

Q.4 Explain except clause with multiple exceptions with an example.

—↓ You can use multiple 'except' clauses to catch different types of exceptions. This allows you to handle each type of exception differently.

eg:
```
Try:
    x = int(input("Please enter a number: "))
    y = 10/x.
except ValueError:
    print("Oops! That was not a valid number.")
except ZeroDivisionError:
    print("You cannot divide by zero.").
else:
    print("The result is:", y)
finally:
    print("This is the end of program.")
```

Q.5 Explain try and finally block with an example.

→ The "try" and "finally" blocks are used together to ensure that a piece of code is always executed, even if an exception is raised in the "try" block.

eg:
```
x=5
try:
    x > 3
except:
    print("Something went wrong.")
else:
    print("Nothing went wrong")
finally:
    print("Finally block will execute regardless of try and except block is raised or not").
```

Q.6 Explain Raising an exception with an example.

→ You can raise an exception using the 'raise' statement. This is useful when you want to indicate that an error has occured and provide information about the error.

eg:
```
def divide(x,y):
    if y == 0:
        raise ZeroDivisionError("Cannot divide by zero")
    return x/y
try:
```

```
    result = divide(10, 0)
except ZeroDivisionError as e:
    print("An error occurred:", e)
```

Q.7 List and explain any five exceptions in Python.

→ Python comes with many built-in exceptions that can be raised by the interpreter or by the code itself.

i) Syntax Error: When there is a syntax error in the code like to forget closing a bracket or a quote.
ii) Type Error
iii) Value Error
iv) IndexError
v) FileNotFoundError.

Q.8 Explain the match(), search(), findall() method function with syntax & an example.

→ In python, the 're' module provides several method for working with regular expressions.

i) match(): It checks if the regex pattern matches at the beginning of the string.
Syntax ⇒ re.match(pattern, string, flags=0)

eg:
```
import re
string = "The quick brown fox jumps over
          the lazy dog."
pattern = r"The quick".

result = re.match(pattern, string)
```

```
if result:
    print ("Match Found!")
else:
    print ("Match not found:")
```

Output: Match Found!

ii) search(): It checks if the regex pattern matches anywhere in the string
Syntax ⇒ re.search(pattern, string, flags=0)

eg:
```
import re
string = "The quick brown fox jumps over
          the lazy dog."
pattern = r"brown"

result = re.search(pattern, string)
if result:
    print ("Match Found!")
else:
    print ("Match not Found!")
```

Output: Match Found!

iii) findall(): It finds all the occurrences of the regex pattern in the string and returns them as a list of strings
Syntax: re.findall(pattern, string, flags=0)

eg:
```
import re
string = "The quick brown fox jumps over
          the lazy dog."
```

```
pattern = r"the"
result = re.findall(pattern, string)
if result:
        print("Match Found!")
        print(result)
else:
        print("Match not Found.")
```

Output : Match Found
['the', 'the']

Q59 Explain search and Replace ~~with~~ with example in re module

→ Here in re module, the 're.sub()' function can be used to search a string for a particular pattern and replace it with a specified string.

Syntax ⇒ re.sub(pattern, repl, string, count=0, flags=0)

eg:
```
import re
string = "The quick brown fox jumps over
          the lazy dog."
pattern = r"brown"
replacement = "red".
new_string = re.sub(pattern, replacement,
                    string)
print(new_string)
```

Output : The quick red fox jumps over
the lazy dog

Q.10 Write all regular expression pattern.

—> Regular Expressions (regex) are used to search for patterns in text or to manipulate text based on patterns. Some of the most commonly used regex patterns:
'^', '$', '.', '*', '+', '?', '\', '[]', '|', '()', '{}'

eg: import re.
p1 = re "^Hello"
p2 = re "world|$"
p3 = re "\ba\w{2,3}\b"
p4 = re "^(t|s)\w+"
p5 = re "\d+"
p6 = re "apple|orange"
p7 = re "\d{1,3}"
p8 = re "[a-zA-Z]\d"
p9 = re "ing$"
p10 = re "(?i)(cat|dog)"

Q.11 Write a program to retrieve all lines that contains "the" with lower or upper case letter.

—> import re
with open ("input.text", "r") as f:
    for line in f:
        if re.search (r"the", line, re.IGNORECASE):
            print (line.strip())

Q.12 Write a program to retrieve lines that starts with 'a' and ends with 'n'.

—

```
with open("input.text", "r") as f:
    for line in f:
        if line.startswith("a") and
        line.strip().endswith("n"):
            print(line.strip())
```

Q.13 Differentiate between Text file and Binary Files?

—

i) Text files contains human-readable characters, while binary files contain data in a non-human-readable format

ii) Text files are commonly used for storing plain text data, while binary files are used for storing data in a specific format.

iii) Text files are small in size compared to binary files.

iv) Text files can be easily read and edited by humans, while binary files require a specialized program to interpret and display their contents.

v) Text files are generally more portable than binary files.

Q.14 What are the different operations we can perform on file?

—

There are several operations that can be

performed on a file, four of them are:

i) Opening a file: Creating a file object and opening a file for reading and writing with the 'open()' function.

ii) Reading data from a file: Reading data from an open file object with the 'read()' 'readline()' or 'readlines()' methods.

iii) Writing data to a file: Writing data to an open file object with the 'write()' or 'writelines()' methods.

iv) Closing a file: Closing an open file object with the 'close()' function.

v) Checking the file status: Checking the status of file such as whether it exists, whether it is a directory or whether it is readable or writable with 'os.path.exists()', 'os.path.isdir()' and 'os.access()' functions.

vi) Checking the file position: Checking the current position of the file pointer within the file with the 'tell()' method.

Q.15 How to open and close files? Also, give the syntax for the same.

→ The 'open()' function is used to open a file and the 'close()' method is used to close it.

Syntax ⇒ file = open

Syntax ⇒ file_object = open (file_name, mode)

&

⇒ file_object.close()

eg: file = open ('example.txt', 'r')
data = file.read()
file.close()

Q.16 What are the different modes to open a file?

→ The different modes to open a file, are:

i) 'r' : Read mode (opens the file for reading and it is the default mode)

ii) 'w' : Write mode. (opens the file for writing. If file doesn't exists, creates a new file and if it is already there then it overwrites)

iii) 'a' : Append mode. (opens the file for writing. If file doesn't exists, creates a new file and if it is already there then it appends new data to the end of file).

iv) 't' : Text Mode (opens the file for reading or writing text data like strings)

v) 'b' : Binary Mode (opens the file in binary mode and is used for reading or writing binary data)

vi) 'x' : Exclusive creation mode. (creates a new file and opens it for writing and if the file already exist, it raises 'FileExistError' exception

Q.17 Differentiate between file modes r+ and w+ with respect to python.

→ The 'r+' mode opens the file for reading and writing. It positions the file pointer at the beginning of the file and if the file does not exist, it raises a 'FileNotFoundError' exception.

eg:
```
file = open('file.txt', 'r+')
data = file.read()
file.write('Hello, World!')
file.close()
```

On the other hand the 'w+' mode opens the file for reading and writing. If the file exists, the contents are deleted and if it does not exist, a new file is created.

eg:
```
file = open('file.txt', 'w+')
file.write('Hello, World!')
file.seek(0)
data = file.read()
file.close().
```

Q.18 Write a statement in python to perform the following operations:

i) To open a text file "MYPET.TXT" in write mode.

→
```
file = open('MYPET.TXT', 'w')
```

ii) To open a text file "MYPET.TXT" in read mod.

→
```
file = open('MYPET.TXT', 'r')
```

iii) To open a binary file 'LOG.DAT' in read mode.
—4 file = open ('LOG.DAT', 'rb')

iv) To open a binary file 'LOG.DAT' in write mode.
—4 file = open ('LOG.DAT', 'wb')

Q.[illegible]9 Differentiate between 'w' and 'a' modes.
—4 The 'w' mode opens the file for writing as well as and if the file already exists, its contents are deleted and if not, then a new file is created

eg: file = open ('file.txt', 'w')
file.write ('Hello!')
file.close()

— On the other hand, the 'a' mode opens the file for writing and if the file already exists, its ~~are~~ contents are preserved and it appends at the end and if not, then it creates a new file

eg: file = open ('file.txt', 'a')
file.write ('Hello!')
file.close()

Q.[illegible] Differentiate between read() and readlines().
—4 The 'read()' method reads the entire content of the file as a single string and returns it. If you specify a number, it will read that many characters from the file.

eg:
```
file = open ('file.txt', 'r')
content = file.read()
print(content)
file.close()
```

On the other hand the 'readlines()' method reads the entire content of the file as a list of ~~strings~~ String, which each ~~string~~ String represents a line of the file.

eg:
```
file = open ('file.txt', 'r')
lines = file.readlines()
print(lines)
file.close()
```

Q.21 Differentiate between Absolute Pathnames and relative pathnames.

→ Absolute pathnames specify the complete path to a file or directory, starting from the root directory. The root directory is the top-level directory of the file system and is represented by a forward slash in Unix-based system.

eg: /home/user/Documents/example.txt

Now, relative pathnames specify the path to a file relative to the current working directory.

eg: Documents/example.txt

Q.22 Write a program to display ~~to display~~ all the lines in a file "python.txt" which have the word "to" in it.

```
with open("python.txt", "r") as file:
    for line in file:
        if "to" in line:
            print(line.strip())
```

Q.23 A text file "PYTHON.TXT" contains alphanumeric text. Write a program that reads this text file and writes to another file "PYTHON1.TXT" entire file except the numbers or digits in the file.

```
with open("PYTHON.TXT", "r") as file_in,
    open("PYTHON1.TXT", "w") as file_out:
        for line in file_in:
            new_line = ""
            for char in line:
                if not char.isdigit():
                    new_line += char
            file_out.write(new_line)
```

Q.24 Explain any 3 ~~ntor~~ methods associated with files in Python.

i) 'open()': This method is used to open a file in a specified mode. It takes two arguments the file name and the mode and returns a file object which can be used to read from or write to the file.

ii) 'read()': This method is used to read the entire contents of a file as a string. If the

number of bytes are not specified then it reads the entire file

iii) 'write()': This method is used to write a string to a file. It takes a single argument which is the string to be written to the file and can be used with a file object in write or append mode.

Q.25 Explain the role of the regular expressions in the following snippets:

i) >>> p = re.compile("d+')
>>> p.findall('12 drummers drumming, 11 pipers piping, 10 lords a-leaping')

→ This code compiles a ~~reg~~ regex pattern '\d+' which matches one or more digits and then uses 'findall()' method to find all non-overlapping occurrences of this pattern.
#Output: '['12', '11', '10']'

ii) >>> ~~p = re.~~ compile ('ca+t')
>>> p.findall('ct cat caaat caaaaaaat caaat cttt ccattt')

→ This code compile a regex pattern 'ca+t' which matches a lowercase "c" followed by one or more lowercase "a"s and then a lowercase "t".
#Output: '['cat', 'caaat', 'caaaaaat', 'caaat']'.

iii) >>> p=re.compile('ca*t')
>>> p.findall('ct cat caaat caaaaaaat caaat cttt ccattt')

→ This code compile a regex pattern 'ca*t' which matches a lowercase "c" followed by zero or more "a"s and then "t".
#Output: '[Whole Pattern]'

iii) >>> p = re.compile ('a{1,3}b')
>>> p.findall ('ab abb aabb aaabb aaaaabb ab')

→ This code compile a regex pattern 'a{1,3}b' which matches a lowercase "a" followed by one to three forward slashes and then a 'b'.
Output: ['ab', 'abb', 'aabb', 'aaabb']

iv) >>> result = re.findall (r'\w*', 'AV is largest Analytics community of India')
>>> print (result)

→ This code uses the 'findall()' method with a regex pattern '\w*', which matches zero or more word characters. The whole string will split into words
Output: ['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India', '']

Q.26 Explain split() methods of regular expression with suitable examples

→ The 'split()' method of regex is used to split a string into list of substrings based on a specified pattern.
Syntax: re.split (pattern, string, maxsplit=0, flags=0)

eg: import re
String = "apple, banana, cherry, orange, pear"
fruits = re.split (",", string, maxsplit=2)
print (fruits)

Output: ['apple', 'banana', 'cherry, orange, pear']

Q.27 What is module? What are the advantages of using module?

-> A module is a file containing Python code that defines functions, classes and variables that can be used in other Python programs. Simply put, it is a way to organise python code in a file.

Advantages of using Module:

i) Code reusability
ii) Namespace management
iii) Improved performance
iv) Improved code organization
v) Easy collaboration

Q.28 Explain various functions of the math module.

-> The math module is a built-in module in Python that provides a range of mathematical functions, some of them are:

i) ceil(x): returns the smallest integer greater than or equal to x.
ii) pow(x, y): returns x raised to the power y.
iii) sqrt(x): returns the square root of x.
iv) floor(x): returns the largest integer less than or equal to x.
v) pi: constant representing value of π.
vi) sin(x): returns the sine of x
vii) cos(x): returns the cosine of x
viii) tan(x): returns the tangent of x
} x is in radian
ix) gcd(x, y): returns the greatest common divisor of x & y

x) factorial(x): returns the factorial of x.
xi) degrees(x): converts x from radians to degrees
xii) radians(x): converts x from degrees to radian
xiii) fabs(x): returns the absolute value of x.
xiv) log(x[, base]): returns the natural log of x to the given base (if no base, it defaults to e).

~~~ x ~~~ x ~~~
~~~