Unit IV: Modules, Files, String and Dictionaries and Sets **Hours: 15**

Modular Design: Modules - Top-Down Design - Python Modules-Text Files: Opening, reading and writing text files - String Processing - Exception Handling- Dictionary type in Python - Set Data type.

Python Module :

It is a file that contains built-in functions, classes,**its** and variables. There are many **Python modules**, each with its specific work.

In this article, we will cover all about Python modules, such as How to create our own simple module, Import Python modules, From statements in Python, we can use the alias to rename the module, etc.

Create a Python Module

To create a Python module, write the desired code and save that in a file with **.py** extension. Let's understand it better with an example:

Example:

Let's create a simple calc.py in which we define two functions, one **add** and another **subtract**.

```
# A simple module, calc.py
def add(x, y):
   return (x+y)

def subtract(x, y):
   return (x-y)
```

Syntax to Import Module in Python

```
import module
```

```
# importing  module calc.py

import calc

 print(calc.add(10, 2))
```

Output:

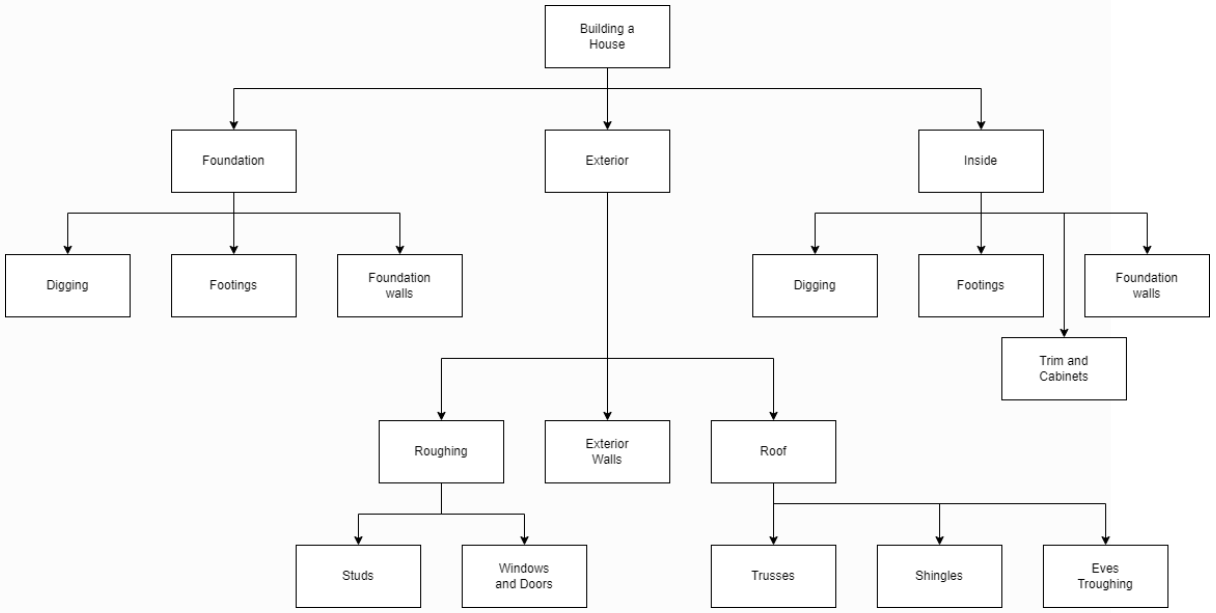
12

Top Down Design:

Top-down design is a method of breaking a problem down into smaller, less complex pieces from the initial overall problem.

Most "good" problems are too complex to solve in just one step, so we divide the problem up into smaller manageable pieces, solve each one of them and then bring everything back together again.

The process of making the steps more and more specific in top-down design is called "stepwise refinement".



Also known as:

• divide and conquer

• stepwise refinement

• structured programming

Complex problems can be solved using top-down design, where

• We break the problem into a sequence of smaller tasks

• Then break the sub-tasks into tasks

• Soon, each of the tasks will be easy to do

Reading and Writing to text files in Python:

Python provides built-in functions for creating, writing, and reading files. Two types of files can be handled in Python, normal text files and binary files (written in binary language, 0s, and 1s).

- **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.
- **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.

File Access Modes

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once its opened.

These modes also define the location of the **File Handle** in the file.

The file handle is like a cursor, which defines from where the data has to be read or written in the file and we can get Python output in text file.

- Read Only (‘r’)
- Read and Write (‘r+’)
- Write Only (‘w’)
- Write and Read (‘w+’)
- Append Only (‘a’)
- Append and Read (‘a+’)
    - **Read Only (‘r’) :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exist, raises the I/O error. This is also the default mode in which a file is opened.
    - **Read and Write (‘r+’):** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exist.
    - **Write Only (‘w’) :** Open the file for writing. For the existing files, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exist.
    - **Write and Read (‘w+’)** : Open the file for reading and writing. For an existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
    - **Append Only (‘a’)**: Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
    - **Append and Read (‘a+’) :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Opening a Text File in Python

It is done using the open() function. No module is required to be imported for this function.

```
File_object = open(r"File_Name","Access_Mode")
# Open function to open the file "MyFile1.txt"
# (same directory) in append mode and
file1 = open("MyFile1.txt","a")

# store its reference in the variable file1
# and "MyFile2.txt" in D:\Text in file2
file2 = open(r"D:\Text\MyFile2.txt","w+")
```

Closing a Text File in Python

close() function closes the file and frees the memory space acquired by that file. It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

```
File_object.close()
# Opening and Closing a file "MyFile.txt"
# for object name file1.
file1 = open("MyFile.txt","a")
file1.close()
```

Writing to a file in Python

There are two ways to write in a file:

- Using write()
- Using writelines()

Writing to a Python Text File Using write()

write() : Inserts the string str1 in a single line in the text file.

File_object.write(str1)

Writing to a Text File Using writelines()

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

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

Reading from a file in Python

There are three ways to read data from a text file:

- Using read()
- Using readline()
- Using readlines()
-

Reading From a File Using read()

read() : Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.

File_object.read([n])

Reading a Text File Using readline()

readline() : Reads a line of the file and returns in form of a string.For specified n, 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])

Reading a File Using readlines()

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

File_object.readlines()

```
# Program to show various ways to read and

# write data in a file.

file1 = open("myfile.txt", "w")

L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]

 # \n is placed to indicate EOL (End of Line)

file1.write("Hello \n")
```

```
file1.writelines(L)

file1.close()  # to change file access modes

 file1 = open("myfile.txt", "r+")

 print("Output of Read function is ")

print(file1.read())

print()

 # seek(n) takes the file handle to the nth

# byte from the beginning.

file1.seek(0)

 print("Output of Readline function is ")

print(file1.readline())

print()

 file1.seek(0)

 # To show difference between read and readline

print("Output of Read(9) function is ")

print(file1.read(9))

print()

 file1.seek(0)

 print("Output of Readline(9) function is ")

print(file1.readline(9))

 file1.seek(0)
```

```
# readlines function

print("Output of Readlines function is ")

print(file1.readlines())

print()

file1.close()
```

Output:
Output of Read function is
Hello
This is Delhi
This is Paris
This is London
Output of Readline function is
Hello
Output of Read(9) function is
Hello
Th
Output of Readline(9) function is
Hello
Output of Readlines function is
['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']

Create a String in Python”

To create a string with given characters, you can assign the characters to a variable after enclosing them in double quotes or single quotes as shown below.

```
word = "Hello World"
print(word)
output: Hello World
```

String Processing

Python string is a sequence of Unicode characters that is enclosed in quotation marks. In this article, we will discuss the in-built string functions i.e. the functions provided by Python to operate on strings.

Case Changing of Strings

The below Python functions are used to change the case of the strings. Let's look at some Python string methods with examples:

- **lower():** Converts all uppercase characters in a string into lowercase
- **upper():** Converts all lowercase characters in a string into uppercase
- **title():** Convert string to title case
- **swapcase():** Swap the cases of all characters in a string
- **capitalize():** Convert the first character of a string to uppercase

```
# Python3 program to show the
# working of upper() function
text = 'geeKs For geEkS'

# upper() function to convert
# string to upper case
print("\nConverted String:")
print(text.upper())

# lower() function to convert
# string to lower case
print("\nConverted String:")
print(text.lower())

# converts the first character to
# upper case and rest to lower case
print("\nConverted String:")
print(text.title())

#swaps the case of all characters in the string
# upper case character to lowercase and viceversa
print("\nConverted String:")
print(text.swapcase())

# convert the first character of a string to uppercase
print("\nConverted String:")
print(text.capitalize())

# original string never changes
print("\nOriginal String")
print(text)
```

Output

Converted String:

GEEKS FOR GEEKS

Converted String:

geeks for geeks

Converted String:

Geeks For Geeks

Converted String:

GEEkS fOR GEeKs

Original String

geeKs For geEkS

1. String Padding: Add Extra Character Elegantly

String padding is a term to adding characters to the beginning or end of a string until it reaches a certain length. It can be useful in formatting text to align with other data or to make it easier to read.

In Python, you can pad a string using the str.ljust(), str.rjust(), and str.center() methods.

```
1. text = "Python"
2. padded_text = text.ljust(10)
3. print(padded_text)
```

Output:

```
Python
```

String Splitting

String splitting refers to dividing a string into multiple substrings based on a specified delimiter or separator. In Python, you can split a string using the str.split() method.

Here's an example of splitting a string based on whitespace characters.

Example -

```
1. text = "Hello world, how are you today?"
2. words = text.split()
3. print(words)
```

Output:

```
['Hello', 'world,', 'how', 'are', 'you', 'today?']
```

3. Use F-Strings for String Formatting

The f-strings are a feature in Python 3.6 and above that allow to embed expressions inside string literals. They are a convenient way to create strings containing dynamic values or format strings with variable values. Let's understand the following example.

```
1. name = "Alice"
2. age = 30
3. greeting = f"Hello, my name is {name} and I'm {age} years old."
4. print(greeting)
```

Output:

```
Hello, my name is Alice and I'm 30 years old.
We can also format value in a specific way.
```

4. Eliminating Unnecessary Character of a String

Python's strip() method is useful for data cleaning, especially when removing unnecessary characters from the beginning or end of strings.

Data scientists often find data cleaning tedious, but Python's built-in string manipulation methods make it easier to remove unwanted characters from strings.

The strip() method, in particular, can remove leading or trailing characters from a string.

Example -

```
1. text = "!!!Hello, World!!!"
2. clean_text = text.strip("!")
3. print(clean_text)
```

Output:

Hello, World

5. Concatenate Strings

In Python, we can concatenate strings using the + operator. Below is an example:

Example -

```
1. string1 = "Hello"
2. string2 = "world"
3. result = string1 + " " + string2
4. print(result)
```

Output:

Hello world

6. Search for Substring Effective

Finding search string is a common requirement in daily programming. Python comes with the two methods. One is find() method -

Example -

```
1. title = 'How to search substrings of Python strings'
2. print(title.find('string'))
3. print(title.find('string'))
4. print(title.find('Yang'))
```

```
5. Output:
6. 17
7. 35
```

7.Easy way to Remove String

Generally, we use the loop to reverse the given string; it can be also reversed using the slicing. However, it is not a Pythonic way. Let's see the following example.

Example -

```
1. name = "Peter"
2. print(name[::-1])
```

Output:

```
reteP
```

Python Exception Handling

we will discuss how to handle exceptions in Python using try, except, and finally statements with the help of proper examples.

Error in Python can be of two types i.e. Syntax errors and Exceptions. Errors are problems in a program due to which the program will stop the execution.

On the other hand, exceptions are raised when some internal events occur which change the normal flow of the program.

Different types of exceptions in python:

In Python, there are several built-in Python exceptions that can be raised when an error occurs during the execution of a program. Here are some of the most common types of exceptions in Python:

- **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.

Python Try Except

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.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

Example

The try block will generate an exception, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

output:

An An exception occurred

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a NameError and another for other errors:

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

output:

Variable x is not defined cept

Dictionary type in Python:

Python allows the values in a dictionary to be any type – string, integer, a list, another dictionary, boolean, etc. However, keys must always be an immutable data type, such as strings, numbers, or tuples.

In the example code block, you can see that the keys are strings or numbers (int or float).

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

Example

String, int, boolean, and list data types:

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

output:

```
{
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]}
```

Set data types

Set. Sets are used to store multiple items in a single variable. Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage. A set is a collection which is unordered, unchangeable*, and unindexed.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

output:

{"apple", "banana", "cherry"}

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

output: {"apple", "banana", "cherry", "apple"}

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}

print(set1)

print(set2)

print(set3)
```

```
{"apple", "banana", "cherry"}
{1, 5, 7, 9, 3}
{True, False, False}
```