UNIT 3 FUNCTIONS AND TEXT FILES

Functions: Function declaration- Category of Functions- Parameter Passing -Keyword Arguments in Python - Default Arguments in Python - Variable Scope, Lambda function.
Files: Fundamentals – opening, reading and writing text files, .csv and .xlsx files.
https://www.tutorialgateway.org/python

3.1 Functions

3.1.1 Category of Functions:

- There are three types of functions in Python:
  - **Built-in functions**: such as help() to ask for help, min() to get the minimum value, print() to print an object to the terminal
  - **User-Defined Functions** (UDFs), which are functions that users create to help them out.
  - **Anonymous functions**, which are also called lambda functions because they are not declared with the standard def keyword.

Built-in functions

- The python interpreter has several functions that are always present for use.
- These functions are known as Built-in Functions.

1. id() - This is the address of the object in memory.

Example : a=5 print(id(a))	Output: 1823986800

2. print() – It is used to output data to the standard output device (screen). We can also output data to a file.

Example : print("Hello World")	Output: Hello World

3. input()-It is used to take input from the user.

Example : a=input("Enter name : ") print(a)	Output: Enter name : Cathy Cathy

4. int() - The int() function converts the specified **value** into an integer number.

Example : a=int(input("Enter the number : ")) print(a)	Output: Enter the number : 100 100

5. float() - The float() function converts the specified **value** into an float number.

Example : a=float(input("Enter the number : ")) print(a)	Output: Enter the number : 100 100.0

6. pow() - The pow() function returns the value of x to the power of y (x^y).

Example :	Output:
print(pow(5,3))	125

7. len() - len function finds the total number of items in a given object.

Example :	Output:
print(len("Hello"))	5

User Defined Functions

Function declaration

- Programs are divided into manageable pieces called *program routines* (or simply *routines*), program routines in Python, called *functions* .
- Python allows us to divide a large program into the basic building blocks known as a function..
- A function is a block of organized, reusable code that is used to perform a single, related action.
- a function is a group of related statements that performs a specific task
- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- Syntax:

```
def  function_name(parameters):
        Statements
```

- Example:

```
def   f1():
        print("Hello World")
```

- def keyword as the start of the function
- A function name to uniquely identify the function
- Parameters are optional. It pass values to a function.
- A colon (:) to mark the end of the function header.
- One or more valid python statements that make up the function body. Statements must have the same indentation level.

Advantage of Functions

- Avoid rewriting the same code.
- Call functions multiple times in a program.
- We can track a large Python program easily.
- Reusability.

Call a function:

- To call a function we simply type the function name with appropriate parameters
- Example:

```
def  f1():
     print("Hello World")
f1()
```

- Output : Hello World

The return statement

- The return statement is used to exit a function and go back to the place from where it was called.
- The return statement is used at the end of the function and returns the result of the function.
- A return statement with no arguments is the same as return **None**.

Syntax: return value

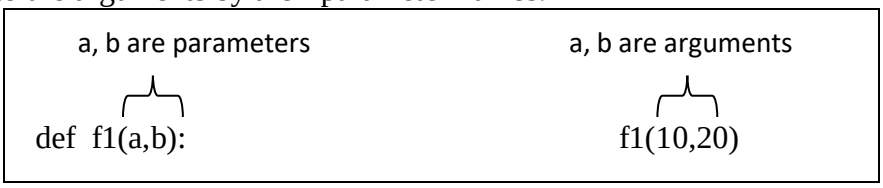
Example:

```
def   f1(a,b):
      c=a+b
      return c
sum=f1(10,20)
print(sum)
```

Output: 30

Parameters and arguments:

- Parameters are the names used when defining a function, and into which arguments will be mapped.
- Arguments are the things which are supplied to any function call, while the function code refers to the arguments by their parameter names.



Types of arguments:

1. Default arguments
2. Keyword arguments
3. Required arguments
4. Variable number of arguments

1.Default Arguments

- Default arguments are those that take a default value if no argument value is passed during the function call.
- Example

```
def   f1(a,b=20):
```

```
        print(a,b)
    f1(10,30)
    f1(10)
```

- Output: 10 30
  10 20

2. Keyword Arguments

- This kind of function call will enable us to pass the arguments in the random order.
- The name of the arguments is treated as the keywords and matched in the function calling and definition.
- If the same match is found, the values of the arguments are copied in the function definition.
- Example:

```
def f1(a,b):
    c=a+b
    print(c)
f1(a=30,b=40)
```

- Output : 70

3. Required Arguments

- Required arguments are the arguments passed to a function in correct positional order.
- Here, the number of arguments in the function call should match exactly with the function definition.
- Example:

```
def f1(a,b,c):
    print(a,b,c)
f1(10,20,30)
f1(5,10)
```

- Output : 10 20 30
  TypeError: f1() missing 1 required positional argument: 'r'

4. Variable number of arguments

- To process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments
- By using the variable-length arguments, we can pass any number of arguments.
- Define the variable-length argument using the *<variable - name >
- Example:

```
def f6(*name):
    print(name)

f6(['one','two','three'])
```

- Output : ['one', 'two', 'three']

Types of user defined functions in Python:

1. Function with no argument and no return value.
2. Function with no argument and with a Return value.
3. Function with argument and No Return value.
4. Function with argument and return value.

Function with no argument and no return value

- Defining, declaring, or calling the function, We won't pass any arguments to the function.
- This type of function won't return any value when we call the function.
- Example:

```
def   add():
      a=10
      b=20
      c=a+b
      print(c)
add()
```

- Output: 30

Function with no argument and with a Return value

- Not pass any arguments to the function while defining, declaring, or calling the function.
- When we call the function, this type of function returns some value.
- Example:

```
def   add():
      a=10
      b=20
      c=a+b
      return c
print(add())
```

- Output: 30

Function with argument and No Return value

- It allows to pass the arguments to the function while calling the function.
- But not return any value when we call the function.
- Example:

```
def   add(a,b):
      c=a+b
```

```
        print(c)
add(10,20)
```

- Output: 30

Function with argument and return value

- It is used to pass the arguments to the function while calling the function.
- It returns some value when we call the function.
- Example:

```
def  add(a,b):
     c=a+b
     return c
print(add(10,20))
```

- Output: 30

3.1.2 Anonymous functions

- **A lambda function** is a small anonymous function.
- Lambda functions can have any number of arguments but only one expression..
- It is not declared in the standard manner.
- Anonymous functions are defined using the lambda keyword.
- In Python, an anonymous function is a function that is defined without a name

Syntax: lambda *arguments* : *expression*	
Example: x=lambda a:a+10 print(x(5))	**Output : 15**

- Lambda functions can take any number of arguments.
- The power of lambda is better shown when you use them as an anonymous function inside another function.

Example: x=lambda a,b,c:a+b+c print(x(5,10,15))	**Output : 30**
Example: def f1(a): return lambda b:a*b c=f1(10) print(c(20))	**Output : 200**

3.1.3 Variable Scope

- The scope of the variable is simply lifetime of a variable.
- The period of time that a variable exists is called its **lifetime of the varaible**.
- Local variables are automatically created (allocated memory) when a function is called, and destroyed (deallocated) when the function terminates.
- Scope of a variable is the portion of a program where the variable is recognized.
- There are two basic scopes of variables in Python −
    - Global variables
    - Local variables

Global vs. Local variables

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
- **Local variables** can be accessed only inside the function in which they are declared
- **Global variables** can be accessed throughout the program body by all functions.
- A variable declared inside the function's body or in the local scope is known as a **local variable.**
- A variable declared outside of the function or in global scope is known as a **global variable.**

Example:	**Output :** Hello World

s1="Hello" #global variable
def f1():
  s2="World" #local variable
  print(s1,s2)
f1()	

- Although they have the same names, they are two different variables with different scopes.

Example:	**Output :** World Hello

s1="Hello" #global variable
def f1():
  s1="World" #local variable
  print(s1)
f1()
print(s1)	

- Global variables are available from within any scope, global and local.

- If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.
- The global keyword makes the variable global.

Example:	**Output :**

x = 300
def myfunc():
  global x
  x = 200
  print(x)
 	
myfunc()
print(x)
	200
	200

Difference between local and global variable

Local Varaible	Global Variable
Local variable is declared inside a function.	Global variable is declared outside the function.
It doesn't provide data sharing	It provides data sharing.
It can only be accessed within the function.	It can be accessed by any function present in the program.
It is lost when the function terminates.	It is lost when the program ends.
The scope is limited.	The scope remains throughout the program.
It is stored on the stack	It is stored on a fixed location decided by the compiler

3.2 Files in Python

Fundamentals

- **Definition:** A **file** is a collection of data that is stored in hard disk.
- Files are named locations on disk to store related information.
- They are used to permanently store data in a non-volatile memory(hard disk)
- There are two types of files that can be handled in python

    1. A **text file** is a file containing characters, structured as lines of text.
    2. A **binary file** is a file that is formatted in a way that only a computer program can read.

Operations of Files:

- Opening a File
- Reading from a File
- Writing to a File
- Closing a File

3.2.1 Opening, reading and writing text files

Opening a File:

- All files must first be opened before they can be used.
- In Python, when a file is opened, a file object is created that provides methods for accessing the file.
- The open() function takes two parameters; filename, and mode.

Syntax:

```
Fileobject = open("FileName", " Mode")
```

- open ( ) will return a file object
- File name is the name of the file.
- Mode is optional, but default mode is 'r'.
- There are four different methods (modes) for opening a file

Modes	Meaning
"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

- The file should be handled as binary or text mode

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

Example:	**Output:**
f=open("color.txt",'r') print(f.read())	RED BLUE GREEN YELLOW ORANGE

- The first argument is the file name to be opened, **color.txt** .
- The second argument, 'r', indicates that the file is to be opened for reading.
- If the file name does not exist, then the program will terminate with a "no such file or directory" error.
- Uppercase and lowercase letters are treated the same for file names
- In the above program, color.txt file is in the same directory otherwise full address of the file should be written

Example:	**Output:**
f=open("D:\CATHY\fruit.txt",'r')	Apple Mango

print(f.read())	Orange

- To open a file for writing in Python, the built-in function open is called with a second argument of 'w'.

Reading a file

- Open text file for reading. If the file does not exist, raises I/O error.
- Using **'r'** mode for reading a file.
- The open() function returns a file object.
- **read()** method for reading the content of the file:

Example:	**Output:**
f=open("fruit.txt",'r') print(f.read())	Apple Mango Orange

- The **readline()** returns as a string the next line of a text file, including the end-of-line character, \n.

Example:	**Output:**
f=open("color.txt",'r') print(f.readline()) print(f.readline()) f.close()	RED BLUE

- Using **looping** can read the whole file, line by line.

Example:	**Output:**
f=open("color.txt",'r') for x in f: print(x) f.close()	RED BLUE GREEN YELLOW ORANGE

Writing a file

- Open the file for writing. For existing file, the data is truncated and over-written.
- It Creates the file if the file does not exist.
- Using **'w'** mode to write data to a file
- **write()** method to output text to a file.
- write() method will overwrite any existing content.

Example:	**Output:**
f=open('color.txt','w') f.write("PINK") f=open('color.txt','r') print(f.read()) f.close()	PINK

- Using **'a'** mode to append data to a file.
- The file is created if it does not exist. The handle is positioned at the end of the file.

Example:	**Output:**
f=open('color.txt','a') f.write("GOLD") f=open('color.txt','r') print(f.read()) f.close()	PINK GOLD

Creating a File

- To create a new file in Python, use the open() method
- Using **'x'** mode to create a file.
- If the file name is already available then display the error message
- Using append mode "a" will create a file if the specified file does not exist
- Using Write mode "w" will create a file if the specified file does not exist

Example:	A new empty file is created
f = open("myfile.txt", "x")	

Closing a File

- All the operations are over finally closed the file
- calling the close() on the file object

Syntax:	Example:
Fileobject .close()	f=open("D:\CATHY\fruit.txt",'r') f.close()

3.2.2 Opening, reading and writing .csv files

CSV File

- A CSV file (Comma Separated Values) is a type of plain text file that uses specific structuring to arrange tabular data.
- It is one of the most simple and common ways to store tabular data.
- CSV files use a comma to separate each specific data value.

- The **csv library** contains objects and other code to read, write, and process data from and to CSV files.
- It is one of the most common methods for exchanging data between applications and popular data format used in Data Science.
- A CSV file stores tabular data in which each data field is separated by a delimiter(comma in most cases).
- To represent a CSV file, it must be saved with the **.csv** file extension.

Uses of CSV File

- CSV files are normally created by programs that handle large amounts of data
- They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs.
- CSV files are very easy to work with programmatically.

Writing csv file

- **csv.writer** class is used to insert data to the CSV file.
- **writer() function** returns a writer object that converts data into a delimited string and stores in a file object.
- The function needs a file object with write permission as a parameter..
- csv.writer class provides two methods for writing to CSV.
    1. writerow()
    2. writerows()
- **writerow():** This method writes a single row at a time. Field row can be written using this method.

  Syntax: writerow(fields)
- **writerows():** This method is used to write multiple rows at a time. This can be used to write rows
  list.

  Syntax: writerows(rows)

Example:	**Output:**
import csv head=['REGNO','NAME','BRANCH','YEAR'] data=[['111','AAA','MECH','I'],['222','BBB','CIVIL','I'],\ ['333','CCC','MECH','II'],['444','DDD','CIVIL','II']] filename="stu.csv" with open(filename,'w') as file: csvfile=csv.writer(file) csvfile.writerow(head) csvfile.writerows(data)	['REGNO', 'NAME', 'BRANCH', 'YEAR'] ['111', 'AAA', 'MECH', 'I'] ['222', 'BBB', 'CIVIL', 'I'] ['333', 'CCC', 'MECH', 'II'] ['444', 'DDD', 'CIVIL', 'II']

- Using **csv.DictWriter** class returns a writer object which maps dictionaries onto output rows.
- csv.DictWriter provides two methods for writing to CSV.

    1. writeheader()
    2. writerows()

- **writeheader():** This method simply writes the first row of your csv file using the pre-specified fieldnames.

    Syntax: writeheader()

- **writerows():** writerows method simply writes all the rows but in each row, it writes only the values(not keys).

    Syntax: writerows(mydict)

Example:	**Output:**

import csv
dict=[{'name':'AAA','branch':'MECH'},{'name':'BBB','branch':'CIVIL'},{'name':'CCC','branch':'ECE'},
name':'DDD','branch':'EEE'}]
field=['name','branch']
filename="student1.csv"
with open(filename,'w') as file:
  csvfile=csv.DictWriter(file,field)
  csvfile.writeheader()
  csvfile.writerows(dict)
	['name', 'branch']
	['AAA', 'MECH']
	['BBB', 'CIVIL']
	['CCC', 'ECE']
	['DDD', 'EEE']

Reading csv file

- **csv.reader** class is used to read data from a CSV file.

Example:	**Output:**

import csv
with open(student1.csv,'r') as file:
  csvfile=csv.reader(file)
  for a in csvfile:
  print(a)
	['name', 'branch']
	['AAA', 'MECH']
	['BBB', 'CIVIL']
	['CCC', 'ECE']
	['DDD', 'EEE']

- csv.DictReader class can be used to read a CSV file as a dictionary.

Example:	**Output:**
import csv with open('student1.csv','r') as file: csvfile=csv.DictReader(file)	{'name': 'AAA', 'branch': 'MECH'} {'name': 'BBB', 'branch': 'CIVIL'} {'name': 'CCC', 'branch': 'ECE'}

for a in csvfile: print(a)	{'name': 'DDD', 'branch': 'EEE'}

3.2.3 Opening, reading and writing .xlsx files

xlsx File

- The XLSX document file is the Microsoft Excel standard format file type..
- XLSX files are the standard extension for the modern Microsoft Excel spreadsheet files.
- It is used to analyze and organize data.
- They contain numerical data separated by rows and columns within a cell.
- It is a spreadsheet program.
- It is used to create grids of text, numbers and formulas specifying calculations.

Uses of xlsx File

- Data entry
- Data management
- Accounting
- Financial analysis
- Charting and graphing.
- Time management
- Task management
- Financial modeling

Writing xlsx file

- importing the pandas module.
- Pandas has excellent methods for reading all kinds of data from Excel files. .
- Pandas defaults to storing data in DataFrames.
- By default, pandas will automatically assign a numeric index or row label starting with zero.
- Save DataFrame in an Excel file with **.to_excel()**

Example:

```
import pandas as pd
df = pd.DataFrame({'States':['TN', 'AP', 'UP', 'MP'],
   'Capitals':['CHENNAI', 'AMARAVATHI',
'LUCKNOW', 'BHOPAL']})
df.to_excel('data1.xlsx')
```

Output:

	States	Capitals
0	TN	CHENNAI
1	AP	AMARAVATHI
2	UP	LUCKNOW
3	MP	BHOPAL

- Each row of the table is written as an dictionary whose keys are the column names and values are the corresponding data.

Reading xlsx file

- **read_excel** method to read in data from the Excel file.
- The easiest way to call this method is to pass the file name.
- If no sheet name is specified then it will read the first sheet in the index.
- Pandas **read_excel** method read the data from the Excel file into a Pandas dataframe object.
-

Syntax:	df = pd.read_excel(Excel file name)

Example:	**Output:**
import pandas as pd df = pd.read_excel('data1.xlsx') print(df)	States Capitals 0 TN CHENNAI 1 AP AMARAVATHI 2 UP LUCKNOW 3 MP BHOPAL

- read_excel() returns a new DataFrame that contains the values from data1.xlsx.
- The pandas read_excel method takes an argument called sheetname that tells pandas which sheet to read in the data from.
- index_col specifies the title column
- **Syntax:** df = pd.read_excel('excelfilename', sheet_names='name of the excelsheet', index_col=column number)
- **Example:** df = pd.read_excel(file, sheet_name = 1, index_col = 0)