# The Python Libraries

NAVITA THAKKAR

HRDPD/SIPG

# Python Function

- USER DEFINED FUNCTION

```
# Defining main function
def sum(a,b):
        return a+b
a=1
b=1
c=sum(a+b)
print(c)
```

# Python Main Function

- Main function is like the entry point of a program.

```
# Defining main function
def sum(a,b):
        return a+b
def main():
        print("hey there")

        print(sum(1,2)
 # Using the special variable
 # __name__
__name__ is a built in variable which evaluates the name of the current module

if __name__=="__main__":
        main()
```

# Python Main Function

- It Allows You to Execute Code When the File Runs as a Script
- Not When It's Imported as a Module.
- For most practical purposes, you can think of the conditional block that you open with
- if __name__ == "__main__" as a way to store code that should only run when your file is executed as a script.

# Python Main Function

- Main function is like the entry point of a program.

```
# Defining main function
def sum(a,b):
        return a+b
def main():
        print("hey there")

        print(sum(1,2)
 # Using the special variable
 # __name__
__name__ is a built in variable which evaluates the name of the current module

if __name__=="__main__":
        main()
```

# What is Library

- Collection of related modules
- Python library contains built-in modules (written in C)
- Provide access to system functionality  such as file I/O
- Standardized solutions to many trivial problems

# Commonly used Python Libraries

- Python Standard library
  - Bundled with core Python

    os, glob, math ,datetime, cmath, statistics …
- Numpy Library
- Matplotlib Library
- SciPy Library
- ……
- U can build your own libraries

# Library

- **import** is used to include the modules in other program

  import  <filename>

  import numpy, string , os …

  from * statement  can be used all names from the module in the current calling namespace

  from <filename> import *

  from math import *

  math.sqrt(4)

  we can access any function by using dot notation

# Python
# Standard Libraries

- os
- glob
- math
- datetime
- cmath
- statistics …

# OS

- The **os** module in Python provides functions for interacting with the operating system.
- **os** comes under Python's standard utility modules.
- This module provides a portable way of using operating system-dependent functionality.
- The ***os*** and ***os.path*** modules include many functions to interact with the file system.
- Useful functions to aid in using this module are **dir(os)** which returns a list of all module functions

# OS

- To get the location of the current working directory  is used.

```
# Python program to explain os.getcwd() method

# importing os module
import os

# Get the current working
# directory (CWD)
cwd = os.getcwd()

# Print the current working
# directory (CWD)
print("Current working directory:", cwd)
os.chdir('../' )
```

# OS

- Useful functions to aid in using this module are dir(os) which returns a list of all module functions
- os.mkdir(path)  create a directory
- os.makedirs(path)  create all intermediate directories
- **os.listdir(path)** Files and directories in path
- os.rmdir(path)
- **os.name**

# Example

```
# Python code
import os
os.chdir("C:\\MyPythonProject")
os.getcwd()
os.chdir("..")
os.getcwd()
os.listdir(“C:\\MyPythonProject”)
```

# os.path

- os.path.exists(“file_name”)
- os.path.getsize(“file_name”) size in bytes
  - os.path.isfile(“file_name”)
  - os.path.isdir(“dir_name”)
- os.path.join(parent_dir, directory)

# os.walk

```
import os
for (root,dirs,files) in os.walk('.', topdown=True):
        print (root)
        print (dirs)
        print (files)
        print ('--------------------------------')
```

OS.walk() generate the file names in a directory tree by walking the tree either top-down or bottom-up. For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames).

- **root :** Prints out directories only from what you specified.
- **dirs :** Prints out sub-directories from root.
- **files :** Prints out all files from root and directories.

# os.system

os.system(cmd)

```
command = 'mkdir  output'
import os
os.system(command)
```

# File Handling in Python

- f = open(filename, mode)
  - **r:** open an existing file for a read operation.
  - **w:** open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
  - **a:** open an existing file for append operation. It won't override existing data.
  - **r+:** To read and write data into the file. The previous data in the file will be overridden.
  - **w+:** To write and read data. It will override existing data.
  - **a+:** To append and read data from the file. It won't override existing data.
- f.read()
- f.write()

# File Handling in Python

```
file = open("temp.txt", 'w')
file.write("This will write a line")
file.close()
file = open("temp.txt", 'a')
file.write("This will add a line")
file.close()

with open("temp.txt") as file:
    data = file.read()
print(data)
```

# Example

```
# Python code
import os


with open("file.text", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print (word)
```

# glob

- The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell

```
import glob
glob.glob('./[0-9].*')
        ['./1.gif', './2.txt']
glob.glob('*.gif')
        ['1.gif', 'card.gif']
glob.glob('?.gif')
         ['1.gif']
glob.glob('**/*.txt', recursive=True)
        ['2.txt', 'sub/3.txt']
glob.glob('./**/', recursive=True)
        ['./', './sub/']
```

# Example

- os.path.exists(“file_name”)
- os.path.getsize(“file_name”) size in bytes
  - os.path.isfile(“file_name”)
  - os.path.isdir(“dir_name”)
- os.path.join(parent_dir, directory)

# math

- math module
- Mathematical functions (and constants)

```
>>> import math
>>> print(math.pi)
    3.141592653589793
>>> print(math.e)
    2.718281828459045
>>> math.sqrt(100)
    10.0
>>> math.sqrt(40)
    6.324555320336759
>>> math.log(128)
    4.852030263919617
>>> math.log2(128)
    7.0
```

# math

>>math.exp(*x*)

>>math.log(x)

>> math.log10(x)

>>math.pow(x,y)

>>math. sqrt(x)

# math

```
import math

a = math.pi/6

# returning the value of sine of pi/6
print ("The value of sine of pi/6 is : ", end="")
print (math.sin(a))

# returning the value of cosine of pi/6
print ("The value of cosine of pi/6 is : ", end="")
print (math.cos(a))

# returning the value of tangent of pi/6
print ("The value of tangent of pi/6 is : ", end="")
print (math.tan(a))
```

# math

Let's create the following mathematical expression in Python:

$$f(x,y) = 3x^2 + \sqrt{x^2 + y^2} + e^{ln(x)}$$

$f(2,2) = ?$

Python Code:

```
import math as mt

x = 2
y = 2

f = 3*mt.pow(x,2) + mt.sqrt(mt.pow(x,2) + mt.pow(y,2)) + mt.exp(mt.log(x))

print(f)
```

The answer becomes $f(2,2) = 16.83$

# degrees() and radians() in Python

math.radians(degree)
math.degress(radians)

```
import math
print (math.radians( 180/math.pi))
print (math.radians(180))
print (math.radians(1))
print (math.degrees(math.pi / 180))
print (math.degrees(180))
print (math.degrees(1))
```

# cmath

- cmath module
- math with complex number support

```
>>> math.sqrt(-29)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: math domain error
>>> cmath.sqrt(-29)
5.385164807134504j
>>> type(cmath.sqrt(-29))
<class 'complex'>
>>> cmath.polar(3+7j)
(7.615773105863909, 1.1659045405098132)
```

# statistics

- statistics module

```
>>> import statistics as stat
>>> list1 =[8, -8, 7, 3, -5, 4, -1, 2, -7, -10, -6, -1, -3, 1, -2, 6,
10, 1, -9, -9]
>>> stat.mean(list1)
-0.95
>>> stat.median(list1)
-1.0
>>> stat.stdev(list1)
6.125657859407944
```

# datetime

- datetime module
  - Date and time calculations

```
import datetime as dt
t1 = dt.datetime(year=2015, month=2, day=28,
hour=1, minute=10, second=0 )
t2 = dt.datetime(year=2016, month=10, day=4,
hour=14, minute=00, second=0 )
print(t2-t1)
584 days, 12:50:00
```

# Elapsed Time

```
from datetime import datetime
starttime = datetime.now()

endtime = datetime.now()

elapsed_time=endtime-starttime
```

datetime.timedelta(seconds=3, microseconds=328517)

Time delta returns in days , seconds and microseconds

# Date

```
from datetime import date

# calling the today
# function of date class
today = date.today()

print("Today's date is", today)
# date object of today's date

print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

# Date Format

```
from datetime import datetime as dt

# Getting current date and time
now = dt.now()
print("Without formatting", now)

s = now.strftime("%A %m %Y")
print(s)


s = now.strftime("%H:%M:%S")
print(s)
```

# Date Format

| Directive | Meaning | Example |
|---|---|---|
| %a | Weekday as locale's abbreviated name. | Sun, Mon, ..., Sat (en_US); So, Mo, ..., Sa (de_DE) |
| %A | Weekday as locale's full name. | Sunday, Monday, ..., Saturday (en_US); Sonntag, Montag, ..., Samstag (de_DE) |
| %w | Weekday as a decimal number, where 0 is Sunday and 6 is Saturday. | 0, 1, ..., 6 |
| %d | Day of the month as a zero-padded decimal number. | 01, 02, ..., 31 |
| %b | Month as locale's abbreviated name. | Jan, Feb, ..., Dec (en_US); Jan, Feb, ..., Dez (de_DE) |
| %B | Month as locale's full name. | January, February, ..., December (en_US); Januar, Februar, ..., Dezember (de_DE) |
| %m | Month as a zero-padded decimal number. | 01, 02, ..., 12 |
| %y | Year without century as a zero-padded decimal number. | 00, 01, ..., 99 |

# Date Format

| Directive | Meaning | Example |
|---|---|---|
| %p | Locale's equivalent of either AM or PM. | AM, PM (en_US);<br>am, pm (de_DE) |
| %M | Minute as a zero-padded decimal number. | 00, 01, ..., 59 |
| %S | Second as a zero-padded decimal number. | 00, 01, ..., 59 |
| %f | Microsecond as a decimal number, zero-padded to 6 digits. | 000000, 000001, ..., 999999 |
| %z | UTC offset in the form ±HHMM[SS[.ffffff]] (empty string if the object is naive). | (empty), +0000, -0400, +1030, +063415, -030712.345216 |
| %Z | Time zone name (empty string if the object is naive). | (empty), UTC, GMT |
| %j | Day of the year as a zero-padded decimal number. | 001, 002, ..., 366 |
| %U | Week number of the year (Sunday as the first day of the week) as a zero-padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0. | 00, 01, ..., 53 |
| %W | Week number of the year (Monday as the first day of the week) as a zero-padded decimal number. All days in a new year preceding the first Monday are considered to be in week 0. | 00, 01, ..., 53 |

# string—Text Constants and Templates

**Strings in Python** can be created using single quotes or double quotes or even triple quotes.

```
# Creating a String
# with single Quotes
String1 = 'Welcome to the Python Tutorials'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = " Welcome to the Python Tutorials "
print("\nString with the use of Double Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Welcome
        to the
       Python Tutorials '''
print("\nCreating a multiline String: ")
print(String1)
```

# Accessing characters in Python String

- individual characters of a String can be accessed by using the method of Indexing
- indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last

```
String1 = "SPACEAPPLICATIONS"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
print(String1[::-1])
```

# String Slicing

- To access a range of characters in the String, the method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

```
# Creating a String
String1 = "SPACEAPPLICATIONS"
print("Initial String: ")
print(String1)

# Printing 3rd to 12th character
print("\nSlicing characters from 3-12: ")
print(String1[3:12])

# Printing characters between
# 3rd and 2nd last character
print("\nSlicing characters between " +
      "3rd and 2nd last character: ")
print(String1[3:-2])
```

| Built-In Function | Description |
| --- | --- |
| string.ascii_letters | Concatenation of the ascii_lowercase and ascii_uppercase constants. |
| string.ascii_lowercase | Concatenation of lowercase letters |
| string.ascii_uppercase | Concatenation of uppercase letters |
| string.digits | Digit in strings |
| string.hexdigits | Hexadigit in strings |
| string.letters | concatenation of the strings lowercase and uppercase |
| string.lowercase | A string must contain lowercase letters. |

# Strings

| | |
|---|---|
| string.endswith() | Returns True if a string ends with the given suffix otherwise returns False |
| string.startswith() | Returns True if a string starts with the given prefix otherwise returns False |
| string.isdigit() | Returns “True” if all characters in the string are digits, Otherwise, It returns “False”. |
| string.isalpha() | Returns “True” if all characters in the string are alphabets, Otherwise, It returns “False”. |
| string.isdecimal() | Returns true if all characters in a string are decimal. |

# Logging
# Levels of Logging

- **Debug :** These are used to give Detailed information, typically of interest only when diagnosing problems.
- **Info :** These are used to confirm that things are working as expected
- **Warning :** These are used an indication that something unexpected happened, or is indicative of some problem in the near future
- **Error :** This tells that due to a more serious problem, the software has not been able to perform some function
- **Critical :** This tells serious error, indicating that the program itself may be unable to continue running

# Logger

There are several logger objects offered by the module itself.
**Logger.info(msg) :** This will log a message with level INFO on this logger.
**Logger.warning(msg) :** This will log a message with a level WARNING on this logger.
**Logger.error(msg) :** This will log a message with level ERROR on this logger.
**Logger.critical(msg) :** This will log a message with level CRITICAL on this logger.
**Logger.log(lvl,msg) :** This will Logs a message with integer level lvl on this logger.
**Logger.exception(msg) :** This will log a message with level ERROR on this logger.

# Logging Example

```
# importing module
import logging

# Create and configure logger
logging.basicConfig(filename="newfile.log",
                    format='%(asctime)s %(message)s',
                    filemode='w')

# Creating an object
logger = logging.getLogger()

# Setting the threshold of logger to DEBUG
logger.setLevel(logging.DEBUG)

# Test messages
logger.debug("Harmless debug Message")
logger.info("Just an information")
logger.warning("Its a Warning")
logger.error("Did you try to divide by zero")
logger.critical("Internet is down")
```

Continued .....