Chapter 8: Standard Libraries

The **sys module** in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter. Let's consider the below example.

```
import sys

print(sys.version)
```

Input and Output using sys

The sys modules provide variables for better control over input or output. We can even redirect the input and output to other devices. This can be done using three variables –

- stdin
- stdout
- stderr

stdin: It can be used to get input from the command line directly. It used is for standard input. It internally calls the input() method. It, also, automatically adds '\n' after each sentence.

```
import sys

for line in sys.stdin:
      if 'q' == line.rstrip():
            break
      print(f'Input : {line}')

print("Exit")
```

stdout: A built-in file object that is analogous to the interpreter's standard output stream in Python. stdout is used to display output directly to the screen console. Output can be of any form, it can be output from a print statement, an expression statement, and even a prompt direct for input. By default, streams are in text mode. In fact, wherever a print function is called within the code, it is first written to sys.stdout and then finally on to the screen.

```
import sys

sys.stdout.write('Output')
```

stderr: Whenever an exception occurs in Python it is written to sys.stderr.

```
import sys


def print_to_stderr(*a):

      # Here a is the array holding the objects
      # passed as the argument of the function
      print(*a, file = sys.stderr)

print_to_stderr("Hello World")
```

Command Line Arguments

are those which are passed during the calling of the program along with the calling statement. To achieve this using the sys module, the sys module provides a variable called sys.argv**.** It's main purpose is:

- It is a list of command-line arguments.
- len(sys.argv) provides the number of command-line arguments.
- sys.argv[0] is the name of the current Python script.

Example: Consider a program for adding numbers and the numbers are passed along with the calling statement.

```
# Python program to demonstrate
# command line arguments

import sys

# total arguments
n = len(sys.argv)
print("Total arguments passed:", n)

# Arguments passed
print("\nName of Python script:", sys.argv[0])

print("\nArguments passed:", end = " ")
for i in range(1, n):
      print(sys.argv[i], end = " ")

# Addition of numbers
sum = 0

for i in range(1, n):
      sum += int(sys.argv[i])

print("\n\nResult:", sum)
```

```
# Assigning all args to variables
filename,arg1,arg2=sys.argv
print(filename,arg1,arg2)
```

Exiting the Program

sys.exit([arg]) can be used to exit the program. The optional argument arg can be an integer giving the exit or another type of object. If it is an integer, **zero is considered "successful termination".**
Note: A string can also be passed to the sys.exit() method.

```
# Python program to demonstrate
# sys.exit()


import sys

age = 17

if age < 18:

      # exits the program
      sys.exit("Age less than 18")
else:
      print("Age is not less than 18")
```

sys.path is a built-in variable within the sys module that returns the list of directories that the interpreter will search for the required module.
When a module is imported within a Python file, the interpreter first searches for the specified module among its built-in modules. If not found it looks through the list of directories defined by **sys.path**.
Note: sys.path is an ordinary list and can be manipulated.

Example 1: Listing out all the paths

```
import sys

print(sys.path)
```

More properties/methods at https://docs.python.org/3/library/sys.html

Python math Module

Python has a built-in module that you can use for mathematical tasks.

The `math` module has a set of methods and constants.

More @ https://docs.python.org/3/library/math.html

Find the circumference and area of a circle with a given radius

```
import math
r = float(input("Input the radius of the circle: "))
c = 2 * math.pi * r
area = math.pi * r * r
print("The circumference of the circle is: ", c)
print("The area of the circle is: ", area)
```

Python filecmp module

What does filecmp do in Python?
The filecmp module **defines functions to compare files and directories, with various optional time/correctness trade-offs**. For comparing files, see also the difflib module. Compare the files named f1 and f2, returning True if they seem equal, False otherwise.
Usage:

`filecmp.cmp(f1, f2, shallow=True)`

`filecmp.cmp()` method in Python is used to compare two files. This method by default performs shallow comparison (as by default `shallow = True`) that means only the **os.stat()** signatures (like size, date modified etc.) of both files are compared and if they have identical signatures then files are considered to be equal irrespective of contents of the files.
If `shallow` is set to `False` then the comparison is done by comparing the contents of both files.
More info @ https://docs.python.org/3/library/filecmp.html

Python Random Module

Python **Random module** is an in-built module of Python which is used to generate random numbers. These are pseudo-random numbers means these are not truly random. This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc.

Example: Printing a random value from a list

```
# import random
import random

# prints a random value from the list
list1 = [1, 2, 3, 4, 5, 6]
print(random.choice(list1))

print(random.random()). # generates random no between 0 and 1

random.seed(5). # Always produce the same result

print(random.random())
random.seed(5)
print(random.random())

r1 = random.randint(5, 15) # Random number between 5 and 15

sample_list = [1, 2, 3, 4, 5]
random.shuffle(sample_list). # Shuffles the list
print(sample_list)
```

more @ https://docs.python.org/3/library/random.html

Python datetime module

In Python, date and time are not a data type of their own, but a module named **datetime** can be imported to work with the date as well as time. **Python Datetime module** comes built into Python, so there is no need to install it externally.
Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

The DateTime module is categorized into 6 main classes –

- date – An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month and day.
- time– An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.
- datetime – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.

- timedelta – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
- **tzinfo** – It provides time zone information objects.
- **timezone** – A class that implements the tzinfo abstract base class as a fixed offset from the UTC (New in version 3.2).

Example1

```
# Python program to
# demonstrate date class

# import the date class
from datetime import date

# initializing constructor
# and passing arguments in the
# format year, month, date
my_date = date(1996, 12, 11)

print("Date passed as argument is", my_date)

my_date = date(1996, 12, 39)
# will raise an ValueError as it is
# outside range

my_date = date('1996', 12, 11)
# will raise a TypeError as a string is
# passed instead of integer
```

Example 2: Get Current Date

To return the current local date today() function of date class is used. today() function comes with several attributes (year, month and day). These can be printed individually.

```
# Python program to
# print current date

from datetime import date

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

print("Today's date is", today)
```

Example 3: Get Today’s Year, Month, and Date

We can get the year, month, and date attributes from the date object using the year, month and date attribute of the date class.

```
from datetime import date

# date object of today's date
today = date.today()

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

Example 4: Get date from Timestamp

We can create date objects from timestamps y=using the fromtimestamp() method. The timestamp is the number of seconds from 1st January 1970 at UTC to a particular date.

```
from datetime import datetime

# Getting Datetime from timestamp
date_time = datetime.fromtimestamp(1887639468)
print("Datetime from timestamp:", date_time)
```

Example 5: Convert Date to String

We can convert date object to a string representation using two functions isoformat() and strftime().

```
from datetime import date

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

# Converting the date to the string
Str = date.isoformat(today)
print("String Representation", Str)
print(type(Str))
```

Time class

The time class creates the time object which represents local time, independent of any day. All the arguments are optional. tzinfo can be None otherwise all the attributes must be integer in the following range –

- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000
- fold in [0, 1]

Example 1: Time object representing time in Python

```
# Python program to
# demonstrate time class

from datetime import time

# calling the constructor
my_time = time(13, 24, 56)

print("Entered time", my_time)

# calling constructor with 1
# argument
my_time = time(minute=12)
print("\nTime with one argument", my_time)

# Calling constructor with
# 0 argument
my_time = time()
print("\nTime without argument", my_time)

# Uncommenting time(hour = 26)
# will rase an ValueError as
# it is out of range

# uncommenting time(hour ='23')
# will raise TypeError as
# string is passed instead of int
```

Example 2: Get hours, minutes, seconds, and microseconds

After creating a time object, its attributes can also be printed separately.

```
from datetime import time

Time = time(11, 34, 56)

print("hour =", Time.hour)
print("minute =", Time.minute)
print("second =", Time.second)
print("microsecond =", Time.microsecond)
```

Datetime class

The DateTime class contains information on both date and time. Like a date object, datetime assumes the current Gregorian calendar extended in both directions; like a time object, datetime assumes there are exactly 3600*24 seconds in every day.
The year, month and day arguments are mandatory. tzinfo can be None, rest all the attributes must be an integer in the following range –

- MINYEAR <= year <= MAXYEAR
- 1 <= month <= 12
- 1 <= day <= number of days in the given month and year
- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000
- fold in [0, 1]

Note – Passing an argument other than integer will raise a TypeError and passing arguments outside the range will raise ValueError.

Example 1: DateTime object representing DateTime in Python

```
# Python program to
# demonstrate datetime object

from datetime import datetime

# Initializing constructor
a = datetime(1999, 12, 12)
print(a)

# Initializing constructor
# with time parameters as well
a = datetime(1999, 12, 12, 12, 12, 12, 342380)
print(a)
```

More @ https://docs.python.org/3/library/datetime.html

Python Re(Regular expressions) Module

Regular expressions are a powerful language for matching text patterns. The Re module allows you to quickly check whether a given string *matches* a given pattern (using the `match` function), or *contains* such a pattern (using the `search` function). A regular expression is a string pattern written in a compact (and quite cryptic) syntax.
The match function attempts to match a pattern against the beginning of the given string. If the pattern matches anything at all (including an empty string, if the pattern allows that!), match returns a match object. The group method can be used to find out what matched.

# a single character, alphanumeric, spaces, special characters etc

```
import re
text=”This is an example of re module”
m = re.match(".", text)
if m: print(‘repr(".")’, "=>", repr(m.group(0)))
```

# any string of characters , alphanumeric, spaces, special characters etc

```
m = re.match(".*", text)
if m: print(‘repr(".")’, "=>", repr(m.group(0)))
```

# a string of letters (at least one) alphanumeric characters only

m = re.match("\w+", text)
if m: print(‘repr("\w")’, "=>", repr(m.group(0))) **# matches 1 alphanumeric character, Prints “T”**
if m: print(‘repr("\w+")’, "=>", repr(m.group(0))) **# matches until no alphanumeric characters are found, prints “This**

# a string of digits, terminates the search once it finds a non digit

m = re.match("\d", text) **#matches exactly 1 digit**
m = re.match("\d+", text) **# matches multiple digits**
if m: print(‘repr("\d+")’, "=>", repr(m.group(0)))

# search for a string of digits, anywhere in the string

m = re.search("\d", text) **#matches exactly 1 digit anywhere**
m = re.search("\d+", text) **# matches multiple digits anywhere**
if m: print(‘repr("\d+")’, "=>", repr(m.group(0)))

# find all the occurrences of a string of digits, anywhere in the string

m = re.findall("\d", text) **#find all occurrences of 1 digit anywhere print(m)**
m = re.findall("\d+", text) **# find all occurrences of multiple digits anywhere**
print(m)

re.compile()

Regular expressions are compiled into pattern objects, which have methods for various operations such as searching for pattern matches or performing string substitutions.

```
p=re.compile(“\d+”)
m = p.findall(text) # find all occurrences of multiple digits anywhere
print(m)

m = p.search(text) # match all occurrences of multiple digits
print(m)
```