Imported Notes about python:

**What is attributes in python?**

 Attributes are the features of the objects or the variables used in a class whereas the methods are the operations or activities performed by that object defined as functions in the class.

For example, if Dog is an object of class animal,

Limbs=4

Eyes=2

Tail=1

are the attributes or features.

Move()

Bark()

Eat(food)

are the methods or functions

**Right way to assign at a time multiple variables:**

x, y, z = "Orange", "Banana", "Cherry"

x = y = z = "Orange"

**Declaration of private, protected and public variables in python:**

Private variable can defined with double underscore(\_\_variable), protected variable can be defined with single underscore(\_variable) and public variables can defined without underscore(variable)

Example:

class P:

def \_\_init\_\_(self, surname,lastname,alias):

self.surname = surname # public

self.\_lastname = lastname # protected

self.\_\_alias = alias # private

def who(self):

print('surname : ', self.surname)

print('lastname : ', self.\_lastname)

print('alias : ', self.\_\_alias)

call=P('peddinti','lakshman','lucky')

call.who()

print(call.surname)

print(call.\_lastname)

print(call.\_\_alias) 🡪 This one will give error because we defined alias as private variable

**Output:**

surname : peddinti

lastname : lakshman

alias : lucky

peddinti

lakshman

Traceback (most recent call last):

File "C:/Content/Python-World/Practice/oops.py", line 17, in <module>

print(call.\_\_alias)

AttributeError: 'P' object has no attribute '\_\_alias'

>>>

What is dir() in python?

**dir()**is a powerful inbuilt function in Python3, which returns list of the attributes and methods of any object (say functions , modules, strings, lists, dictionaries etc.)

The **dir()** function can also list out all the available attributes for a module/list/dictionary.

Python File I/O

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

>>> f = open("test.txt") # open file in current directory

>>> f = open("C:/Python33/README.txt") # specifying full path

**>>>> Even we can open file with context identifier below way**

**With open(“lakshman.txt”,”r+”) as rf:**

**Block of statements**

**-No need to close the file once this block came out,it will close automatically.**

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode. The default is reading in text mode.

Python File Modes.

|  |  |
| --- | --- |
| **Modes** | **Description** |
| r | Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. |
| rb | Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode. |
| r+ | Opens a file for both reading and writing. The file pointer placed at the beginning of the file. |
| rb+ | Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| w | Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| wb | Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| w+ | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| wb+ | Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| a | Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| ab | Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| a+ | Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| ab+ | Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |

JSON:

[JSON (JavaScript Object Notation)](http://json.org/), specified by [**RFC 7159**](https://tools.ietf.org/html/rfc7159.html) , is a lightweight data interchange format inspired by [JavaScript](https://en.wikipedia.org/wiki/JavaScript) object literal syntax

The [json](https://docs.python.org/2/library/json.html) library can parse JSON from strings or files. The library parses JSON into a Python dictionary or list. It can also convert Python dictionaries or lists into JSON strings.

Take the following string containing JSON data:

json\_string = '{"first\_name": "Guido", "last\_name":"Rossum"}'

It can be parsed like this:

**import** json

parsed\_json = json.loads**(**json\_string**)**

and can now be used as a normal dictionary:

**print(**parsed\_json**[**'first\_name'**])**

"Guido"

You can also convert the following to JSON:

d = **{**

'first\_name'**:** 'Guido'**,**

'second\_name'**:** 'Rossum'**,**

'titles'**:** **[**'BDFL'**,** 'Developer'**],**

**}**

**print(**json.dumps**(**d**))**

'{"first\_name": "Guido", "last\_name": "Rossum", "titles": ["BDFL", "Developer

|  |  |
| --- | --- |
| **Python** | **JSON** |
| dict | Object |
| list, tuple | Array |
| str | String |
| int, float, int- & float-derived Enums | Number |
| True | True |
| False | False |
| None | Null |

# SSH library in Python:

**Paramiko** – implements the SSH2 protocol for secure connections to remote machines.

**Pexpect** – spawn a child application and control it as if a human were typing commands.

## **Fabric**

Fabric is a high level Python (2.7, 3.4+) library designed to execute shell commands remotely over SSH, yielding useful Python objects in return:

**subprocess** - is a standard library module, so it'll be available with python installation. But it has a reputation of hard to use since it's api is non-intuitive.

# Pexpect:

Pexpect is a Python module for spawning child applications and controlling them automatically. Pexpect can be used for automating interactive applications  
such as ssh, ftp, passwd, telnet, etc. It can be used to a automate setup  
scripts for duplicating software package installations on different servers. It  
can be used for automated software testing.

There are two main interfaces to Pexpect -- the function, [run](http://pexpect.sourceforge.net/pexpect.html#-run)() and the class,  
[spawn](http://pexpect.sourceforge.net/pexpect.html#spawn). You can call the [run](http://pexpect.sourceforge.net/pexpect.html#-run)() function to execute a command and return the  
output. This is a handy replacement for os.system().

**Sub process Module:**

The [**subprocess**](https://docs.python.org/2/library/subprocess.html#module-subprocess) module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several older modules and functions:

os.system

os.spawn\*

os.popen\*

popen2.\*

commands.\*

Python subprocess module is a powerful tool and was introduced to replace various old modules/functions present in Python, like:

* [os.system](https://www.journaldev.com/17232/python-os-module)
* os.spawn and related functions
* os.popen and related functions
* popen2.\*
* commands.\*

Please note that replacements were not as is and some modifications in the program were needed to move to the subprocess module usage. Let’s start our journey with the functions of this module.

### subprocess call()

This function is used to run a command and get the return code of the command.

import subprocess

print(subprocess.call(["pwd", "-P"]))

output: will be 0 if executed successfully.

Let’s understand what happened in above program:

* When an argument list is passed, the first argument is interpreted as the executable.
* The parameters from second param onwards are treated as the command line arguments to the program.
* We could also have done:

import subprocess

print(subprocess.call('ls -l', shell=True))

With shell being True, call() function treats this as command completely and run it as is. The output would have shown all files and directories in current folder.

Note that in POSIX based systems, a 0 return code is for success and 1 to 255 are for anything else. These exit codes are interpreted by machine scripts to evaluate the events of success and failures.

### subprocess run()

This function works just like the call method and is used to run a command and get the return code of the command. Note that the run() function was added in Python 3.5. A clear difference between the run() and the call()function is that the call() function doesn’t supports the input and check parameters

### subprocess check\_call()

This function works like call() function but if there was an error in running the specified command, it raises a CalledProcessError exception.

### subprocess check\_output()

When we use the call() function to run a command, the output is bound to the parent process and is unretrievable for the calling program. We can use the check\_output() function to capture the output for later usage.

import subprocess

output = subprocess.check\_output(['ls', '-1'])

print(output)

### subprocess communicate()

We can use the communicate() function in this Python module to read input and the output from the process itself. stdout is the process output and stderr is populated in case of an error.

import subprocess

process = subprocess.Popen(

['cat', 'hello.py'], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

stdout, stderr = process.communicate()

print(stdout)

subprocess Popen:Here is example of changing the password of given user in linux using subprocess module

import subprocess

import os

import time

proc = subprocess.Popen(['passwd','lpeddint'],stdin=subprocess.PIPE,stdout=subprocess.PIPE,stderr=subprocess.PIPE)

stdin = proc.stdin.write('King$Kohli0peddint098\n')

stdin = proc.stdin.write('King$Kohli0peddint098\n')

time.sleep(1)

print("Password changed successfully")

**Module Turtle:**

Python has a library called turtle that is part of the standard python installation. To use it, you need only type:

from turtle import \*

or

import turtle

You can type this right in the python interpreter to experiment with turtle graphics or, better yet, include this line at the top of your program and then use turtle drawing commands in your program!

In the turtle package when you run a program with turtle commands, a special window will open where the drawing will take place.

# [sys](https://docs.python.org/3/library/sys.html#module-sys) Module — System-specific parameters and functions:

# This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

#!/usr/bin/python

import sys

# it's easy to print this list of course:

print sys.argv

# or it can be iterated via a for loop:

for i in range(len(sys.argv)):

if i == 0:

print "Function name: %s" % sys.argv[0]

else:

print "%d. argument: %s" % (i,sys.argv[i])

$ python arguments.py arg1 arg2

['arguments.py', 'arg1', 'arg2']

Function name: arguments.py

1. argument: arg1

2. argument: arg2

$

# [sysconfig](https://docs.python.org/3/library/sysconfig.html#module-sysconfig) — Provide access to Python’s configuration information[¶](https://docs.python.org/3/library/sysconfig.html#module-sysconfig)

# The [sysconfig](https://docs.python.org/3/library/sysconfig.html" \l "module-sysconfig" \o "sysconfig: Python's configuration information) module provides access to Python’s configuration information like the list of installation paths and the configuration variables relevant for the current platform.

# >>> import sysconfig

# >>> sysconfig.get\_path\_names()

# ('stdlib', 'platstdlib', 'purelib', 'platlib', 'include', 'scripts', 'data')

# >>> sysconfig.get\_config\_vars("userbase")

# ['C:\\Users\\lpeddint\\AppData\\Roaming\\Python']

**Data Structures in python:**

**Recursion:**

**What is Recursion?**

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are [Towers of Hanoi (TOH)](http://quiz.geeksforgeeks.org/c-program-for-tower-of-hanoi/), [Inorder/Preorder/Postorder Tree Traversals](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/), [DFS of Graph](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), etc.

**What is base condition in recursion?**

In recursive program, the solution to base case is provided and solution of bigger problem is expressed in terms of smaller problems.

def fact(n)

{

if (n < = 1) // base case

return 1;

else

return n\*fact(n-1);

}

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

If base case is not reached or not defined, then stack overflow problem may arise.

**What are the disadvantages of recursive programming over iterative programming?**  
Note that both recursive and iterative programs have same problem solving powers, i.e., every recursive program can be written iteratively and vice versa is also true. Recursive program has greater space requirements than iterative program as all functions will remain in stack until base case is reached. It also has greater time requirements because of function calls and return overhead.

**What are the advantages of recursive programming over iterative programming?**  
Recursion provides a clean and simple way to write code. Some problems are inherently recursive like tree traversals, Tower of Hanoi, etc. For such problems it is preferred to write recursive code. We can write such codes also iteratively with the help of stack data structure. For example refer [Inorder Tree Traversal without Recursion](https://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion/), [Iterative Tower of Hanoi](https://www.geeksforgeeks.org/iterative-tower-of-hanoi/).

## Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

The main benefit of a recursive approach to algorithm design is that it allows us to

succinctly take advantage of a repetitive structure present in many problems. By

making our algorithm description exploit the repetitive structure in a recursive way,

we can often avoid complex case analyses and nested loops.

What is tail recursion and non-tail recursion?

Argparse Module:

the recommended command-line parsing module in the Python standard library.

The [argparse](https://docs.python.org/3/library/argparse.html" \l "module-argparse" \o "argparse: Command-line option and argument parsing library.) module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and [argparse](https://docs.python.org/3/library/argparse.html" \l "module-argparse" \o "argparse: Command-line option and argument parsing library.) will figure out how to parse those out of [sys.argv](https://docs.python.org/3/library/sys.html" \l "sys.argv" \o "sys.argv). The [argparse](https://docs.python.org/3/library/argparse.html" \l "module-argparse" \o "argparse: Command-line option and argument parsing library.) module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

**Range function examples:**

>>> print(range(6))

[0, 1, 2, 3, 4, 5]

>>> print(range(1,6))

[1, 2, 3, 4, 5]

>>> print(range(0,6))

[0, 1, 2, 3, 4, 5]

>>> print(range(-2,2))

[-2, -1, 0, 1]

>>> print(range(-10,-5))

[-10, -9, -8, -7, -6]

>>> print(range(-5,-10))

[]

>>> print(range(5,3))

[]

>>> print(range(20,10,3))

[]

>>> print(range(10,20,3))

[10, 13, 16, 19]

>>> print(range(10,20,0))

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

print(range(10,20,0))

ValueError: range() step argument must not be zero

>>> print(range(20,10,-1))

[20, 19, 18, 17, 16, 15, 14, 13, 12, 11]

>>>>>> print(range(-10,-1,1))

[-10, -9, -8, -7, -6, -5, -4, -3, -2]

>>> print(range(-10,0,2))

[-10, -8, -6, -4, -2]

>>> print(range(-10,0,-2))

[]

>>> print(range(0,-10,-2))

[0, -2, -4, -6, -8]

>>> print(range(-10,-1))

[-10, -9, -8, -7, -6, -5, -4, -3, -2]

>>> print(range(-1,-10))

[]

>>> print(range(-1,-10,1))

[]

>>> print(range(-1,-10,-1))

[-1, -2, -3, -4, -5, -6, -7, -8, -9]

>>> tup = (1,3,4,5)

>>> print(tup[:-1])

(1, 3, 4)

>>> print(tup[-2:])

(4, 5)

>>> print(tup[:-3])

(1,)

>>>

**Requests module:**

Requests will allow you to send HTTP/1.1 requests using Python. With it, you can add content like headers, form data, multipart files, and parameters via simple Python libraries. It also allows you to access the response data of Python in the same way.

*pip install requests*

**Strings:**

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

**Ex:** quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

*Output: I want 3 pieces of item 567 for 49.95 dollars.*

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

a = 25

b = 30k

print("lakshman age is %s and salary is %s" %(a,b))

lakshman age is 25 and salary is 30k

**Operators:**

<https://www.w3schools.com/python/python_operators.asp>

|  |  |  |
| --- | --- | --- |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |