1. What are the key features of Python?

These are its qualities:

Interpreted

Dynamically-typed

Object-oriented

Concise and simple

Free(open source)

Has a large community

1. How is Python executed?

Python files are first compiled to bytecode and are then executed by the host. “OR” Type python.pv at the command line.

1. How are compile-time and run-time code checking done in Python?

Python has a unique way of performing compile-time and run-time code checking. A small portion checking is carried out during compile-time checking, but most of the checks such as type, name, etc are postponed until code execution. If the Python code references a user-defined function that does not exist, the code will compile successfully and the code will fail with an exception only when the code execution path references the function which does not exist.

1. State how arguments are passed by value or by reference.

[Python is an object-oriented language](https://www.springpeople.com/programming-languages/python) so all variables hold references to the objects. The references values are according to the functions and as a result, the value of references cannot be changed. However, the objects can be changed if it is mutable.

Eg:

1. What is a module and package in Python?

The module is the way to structure a program. Each Python program file is a module, which imports other modules like objects and attributes.

The folder of Python program is a package of modules.  A package can have modules or subfolders.

1. How does exception handling in Python differ from Java? Also, list the optional clauses for a <try-except> block In Python.

Python has its own method of implementing exception handling. <try-except> is the block which can be used by the programmer to see the error details without having to terminate the program. Also, in some cases, this <try-except> statement offers a solution to deal with the error.

1. Explain the use of try: except raise, and finally.

Python makes use of try, except and finally blocks for error handling. Try block is used to execute the code until an error occurs. We can make use of an except block to receive control which will receive control after all errors, or one can use specific exception handling blocks for various error types. Control is transferred to the appropriate except block. In all cases, the final block is executed. Raise may be used to raise your own exceptions.

1. How are instance variables different from class variables?

Instance variables are variables that are created locally within a class to refer to an object of the class. A class variable is one that is created globally in a class and is accessible within all instance of that class.

Class variables are declared with keyword static and Instance variables are declared without static keyword.

Class variables can be accessed anywhere within that class whereas an instance variable can only be accessed within the particular object of the class.

As class variables are common to all objects of a class, changes made to these variables through one object will reflect in another. As each object will have its own copy of instance variables, changes made to these variables through one object will not reflect in another object.

Class variables can be accessed using either class name or object reference. Instance variables can be accessed only through an object reference.

1. What is namespace in Python?

In Python, every name introduced has a place where resides and can be found. This space is known as a namespace. It is an address location where a variable name is mapped to the object placed. Whenever the variable is searched out, this address location will be searched, to get the corresponding object.

Global,

Local,

Module and

Class namespaces.

1. Explain the difference between local and global namespaces.

Local namespaces are created within a function when that function is called. Global namespaces are created when the program starts.

1. When would you use a continue statement in a for loop?

When processing a particular item is complete, to move on to the next, without executing further processing in the block continue statement is used. The continue statement states, the current item is done processing, move on to the next item.

1. When would you use a break statement in a for loop?

The break statement states that the function of the loop is over and to move on to the next block of code. For example, when the item being searched is found, there is no need to keep looping. The break statement comes into play here and the loop terminates and the execution moves on to the next section of the code.

1. What Are The Principal Differences Between The Lambda And Def?

Lambda Vs Def.

def can hold multiple expressions while lambda is a uni-expression function.

def generates a function and designates a name so as to call it later. lambda forms a function and returns the function itself.

def can have a return statement. lambda can’t have return statements

lambda supports to get used inside a list and dictionary.

1. Write A Reg Expression That Confirms An Email Id Using The Python Reg Expression Module <Re>?

Python has a regular expression module <re>.

Check out the <re> expression that can check the email id for .com and .co.in subdomain.

import re

print(re.search(r"[0-9a-zA-Z.]+@[a-zA-Z]+\.(com|co\.in)$","micheal.pages@mp.com"))

1. What Is A Built-In Function That Python Uses To Iterate Over A Number Sequence?

range() generates a list of numbers, which is used to iterate over for loops.

for i in range(5):

print(i)

1. What Are The Optional Statements That Can Be Used Inside A <Try-Except> Block In Python?

There are two optional clauses you can use in the **<try-except>** block.

The **<else>** clause

It is useful if you want to run a piece of code when the try block doesn’t create any exception.

The **<finally>** clause

It is useful when you want to execute some steps which run, irrespective of whether there occurs an exception or not.

1. How Does The Ternary Operator Work In Python?

The ternary operator is an alternative for the conditional statements. It combines of the true or false values with a statement that you need to test. The syntax would look like the one given below.

**[onTrue] if [Condition] else [onFalse]**

x, y = 35, 75

smaller = x if x < y else y

print(smaller)

1. What Does The <Self> Keyword Do?

The **<self>** keyword is a variable that holds the instance of an object. In almost, all the object-oriented languages, it is passed to the methods as hidden parameter.

1. What Is The Purpose Of Doc Strings In Python?

In Python, documentation string is popularly known as doc strings. It sets a process of recording Python functions, modules, and classes.

1. Why And When Do You Use Generators In Python?

A generator in Python is a function which returns an iterable object. We can iterate on the generator object using the <**yield**> keyword. But we can only do that once because their values don’t persist in memory, they get the values on the fly.

Generators give us the ability to hold the execution of a function or a step as long as we want to keep it. However, here are a few examples where it is beneficial to use generators.

We can replace loops with generators for efficiently calculating results involving large data sets.

Generators are useful when we don’t want all the results and wish to hold back for some time.

Instead of using a callback function, we can replace it with a generator. We can write a loop inside the function doing the same thing as the callback and turns it into a generator.

1. What Does The <Yield> Keyword Do In Python?

The <**yield**> keyword can turn any function into a generator. It works like a standard return keyword. But it’ll always return a generator object. Also, a function can have multiple calls to the <**yield**> keyword.

See the example below.

def testgen(index):

weekdays = ['sun','mon','tue','wed','thu','fri','sat']

yield weekdays[index]

yield weekdays[index+1]

day = testgen(0)

print next(day), next(day) #output: sun mon

1. How To Convert A List Into Other Data Types?

Sometimes, we don’t use lists as is. Instead, we have to convert them to other types.

Turn A List Into A String.

We can use the <”.join()> method which combines all elements into one and returns as a string.

weekdays = ['sun','mon','tue','wed','thu','fri','sat']

listAsString = ' '.join(weekdays)

print(listAsString)

#output: sun mon tue wed thu fri sat

Turn A List Into A Tuple.

Call Python’s <tuple()> function for converting a list into a tuple. This function takes the list as its argument. But remember, we can’t change the list after turning it into a tuple because it becomes immutable.

weekdays = ['sun','mon','tue','wed','thu','fri','sat']

listAsTuple = tuple(weekdays)

print(listAsTuple)

#output: ('sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat')

Turn A List Into A Set.

Converting a list to a set poses two side-effects.

Set doesn’t allow duplicate entries, so the conversion will remove any such item if found.

A set is an ordered collection, so the order of list items would also change.

However, we can use the <set()> function to convert a list to a set.

weekdays = ['sun','mon','tue','wed','thu','fri','sat','sun','tue']

listAsSet = set(weekdays)

print(listAsSet)

#output: set(['wed', 'sun', 'thu', 'tue', 'mon', 'fri', 'sat'])

Turn A List Into A Dictionary.

In a dictionary, each item represents a key-value pair. So converting a list isn’t as straight forward as it were for other data types.

However, we can achieve the conversion by breaking the list into a set of pairs and then call the <zip()> function to return them as tuples.

Passing the tuples into the <dict()> function would finally turn them into a dictionary.

weekdays = ['sun','mon','tue','wed','thu','fri']

listAsDict = dict(zip(weekdays[0::2], weekdays[1::2]))

print(listAsDict)

#output: {'sun': 'mon', 'thu': 'fri', 'tue': 'wed'}

1. **What is the difference between del() and remove() methods of list?**  
   To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.
2. **What is \_\_init\_\_.py used for?**  
   **Ans99**:It declares that the given directory is a  package.
3. What is [list comprehension](https://www.mytectra.com/python-training-in-bangalore.html)?

Creating a list by doing some operation over data that can be accessed using an iterator. For eg:

|  |  |
| --- | --- |
| 1  2  3 | >>>[ord(i) for i in string.ascii\_uppercase]  [65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90]  >>> |

1. Describe Python’s garbage collection mechanism in brief.

Python maintains a count of the number of references to each object in memory. If a reference count goes to zero then the associated object is no longer live and the memory allocated to that object can be freed up for something else

1. Whenever Python exits, why isn’t all the memory de-allocated?

Ans:

Whenever Python exits, especially those Python modules which are having circular references to other objects or the objects that are referenced from the global namespaces are not always de-allocated or freed.

It is impossible to de-allocate those portions of memory that are reserved by the C library.

On exit, because of having its own efficient clean up mechanism, Python would try to de-allocate/destroy every other object.

1. What is dictionary in Python?

Ans: The built-in datatypes in Python is called dictionary. It defines one-to-one relationship between keys and values. Dictionaries contain pair of keys and their corresponding values. Dictionaries are indexed by keys.

Let’s take an example:

The following example contains some keys. Country, Capital & PM. Their corresponding values are India, Delhi and Modi respectively.

|  |  |
| --- | --- |
| 1 | dict={'Country':'India','Capital':'Delhi','PM':'Modi'} |
| 1 | print dict[Country] |

India

|  |  |
| --- | --- |
| 1 | print dict[Capital] |

Delhi

|  |  |
| --- | --- |
| 1 | print dict[PM] |

Modi

A dictionary is mutable, and we can also use a comprehension to create it.

>>> roots={x\*\*2:x for x in range(5,0,-1)}

>>> roots

{25: 5, 16: 4, 9: 3, 4: 2, 1: 1}

1. What is monkey patching in Python?

Ans: In Python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

Consider the below example:

|  |  |
| --- | --- |
| 1  2  3  4 | # m.py  class MyClass:  def f(self):  print "f()" |

We can then run the monkey-patch testing like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import m  def monkey\_f(self):  print "monkey\_f()"    m.MyClass.f = monkey\_f  obj = m.MyClass()  obj.f() |

The output will be as below:

monkey\_f()

As we can see, we did make some changes in the behavior of f() in MyClass using the function we defined, monkey\_f(), outside of the module m.

1. What does this mean: \*args, \*\*kwargs? And why would we use it?

In cases when we don’t know how many arguments will be passed to a function, like when we want to pass a list or a tuple of values, we use \*args.

>>> def func(\*args):

for i in args:

print(i)

>>> func(3,2,1,4,7)

3

2

1

4

7

\*\*kwargs takes keyword arguments when we don’t know how many there will be.

>>> def func(\*\*kwargs):

for i in kwargs:

print(i,kwargs[i])

>>> func(a=1,b=2,c=7)

a.1

b.2

c.7

The words args and kwargs are convention, and we can use anything in their place.

1. Explain split(), sub(), subn() methods of “re” module in Python.

Ans: To modify the strings, Python’s “re” module is providing 3 methods. They are:

split() – split() lets us split a string around the character we specify.

>>> '1,2,3,4,5'.split(',')

[‘1’, ‘2’, ‘3’, ‘4’, ‘5’]

sub() – finds all substrings where the regex pattern matches and then replace them with a different string

subn() – it is similar to sub() and also returns the new string along with the no. of replacements.

join() lets us join characters from a string together by a character we specify.

>>> ','.join('12345')

‘1,2,3,4,5’

1. What is the difference between range & xrange?

**Ans:** For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please. The only difference is that range returns a Python list object and x range returns an xrange object.

This means that xrange doesn’t actually generate a static list at run-time like range does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators. That means that if you have a really gigantic range you’d like to generate a list for, say one billion, xrange is the function to use.

This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can to create your array of integers, which can result in a Memory Error and crash your program. It’s a memory hungry beast.

1. Explain the use of decorators.

**Ans:** Decorators in Python are used to modify or inject code in functions or classes. Using decorators, you can wrap a class or function method call so that a piece of code can be executed before or after the execution of the original code. Decorators can be used to check for permissions, modify or track the arguments passed to a method, logging the calls to a specific method, etc.

1. Why are local variable names beginning with an underscore discouraged?

a) they are used to indicate a private variables of a class

1. Differentiate between deep and shallow copy.

What Are Different Methods To Copy An Object In Python?

There are two ways to copy objects in Python.

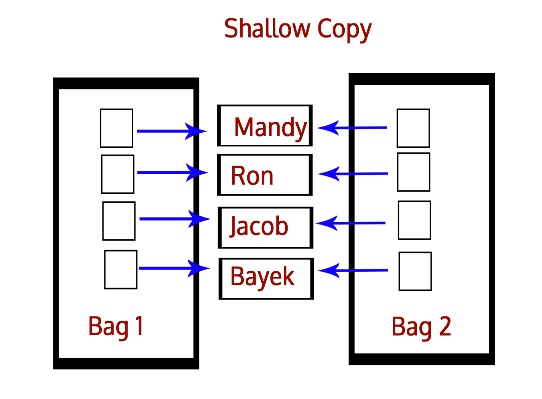
**copy.copy() function**

It makes a copy of the file from source to destination.

It’ll return a shallow copy of the parameter.

A shallow copy, however, copies one object’s reference to another. So, if we make a change in the copy, it will affect the original object. For this, we have the function copy(). We use it like:

>>> b=copy.copy(a)

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/Python-shallow-copy.jpg)

Shallow Copy – Python Interview Questions and Answers

**copy.deepcopy() function**

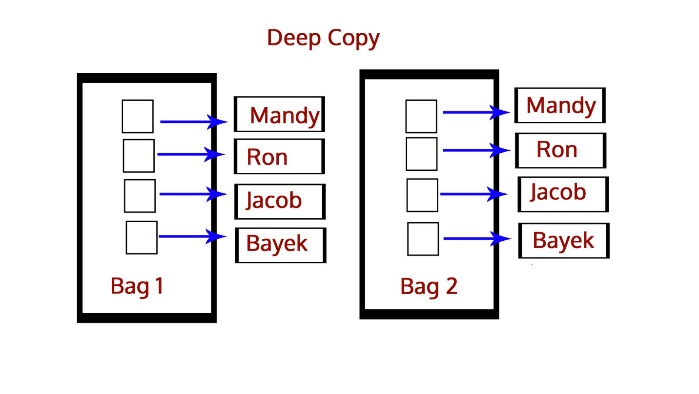
It also produces the copy of an object from the source to destination.

It’ll return a deep copy of the parameter that you can pass to the function.

A deep copy copies an object into another. This means that if you make a change to a copy of an object, it won’t affect the original object. In Python, we use the function deepcopy() for this, and we import the module copy. We use it like:

>>> import copy

>>> b=copy.deepcopy(a)



1. Differentiate between lists and tuples.

The major difference is that a list is mutable, but a tuple is immutable. Examples:

>>> mylist=[1,3,3]

>>> mylist[1]=2

>>> mytuple=(1,3,3)

>>> mytuple[1]=2

Traceback (most recent call last):

File "<pyshell#97>", line 1, in <module>

mytuple[1]=2

TypeError: ‘tuple’ object does not support item assignment

1. Explain inheritance.

When one class inherits from another, it is said to be the child/derived/sub class inheriting from the parent/base/super class. It inherits/gains all members (attributes and methods).

Inheritance lets us reuse our code, and also makes it easier to create and maintain applications. Python supports the following kinds of inheritance:

Single Inheritance- A class inherits from a single base class.

Multiple Inheritance- A class inherits from multiple base classes.

Multilevel Inheritance- A class inherits from a base class, which, in turn, inherits from another base class.

Hierarchical Inheritance- Multiple classes inherit from a single base class.

Hybrid Inheritance- Hybrid inheritance is a combination of two or more types of inheritance.

1. Explain help() and dir() functions in Python.

The help() function displays the documentation string and help for its argument.

>>> import copy

>>> help(copy.copy)

Help on function copy in module copy:

copy(x)

Shallow copy operation on arbitrary Python objects.

See the module’s \_\_doc\_\_ string for more info.

The dir() function displays all the members of an object(any kind).

>>> dir(copy.copy)

[‘\_\_annotations\_\_’, ‘\_\_call\_\_’, ‘\_\_class\_\_’, ‘\_\_closure\_\_’, ‘\_\_code\_\_’, ‘\_\_defaults\_\_’, ‘\_\_delattr\_\_’, ‘\_\_dict\_\_’, ‘\_\_dir\_\_’, ‘\_\_doc\_\_’, ‘\_\_eq\_\_’, ‘\_\_format\_\_’, ‘\_\_ge\_\_’, ‘\_\_get\_\_’, ‘\_\_getattribute\_\_’, ‘\_\_globals\_\_’, ‘\_\_gt\_\_’, ‘\_\_hash\_\_’, ‘\_\_init\_\_’, ‘\_\_init\_subclass\_\_’, ‘\_\_kwdefaults\_\_’, ‘\_\_le\_\_’, ‘\_\_lt\_\_’, ‘\_\_module\_\_’, ‘\_\_name\_\_’, ‘\_\_ne\_\_’, ‘\_\_new\_\_’, ‘\_\_qualname\_\_’, ‘\_\_reduce\_\_’, ‘\_\_reduce\_ex\_\_’, ‘\_\_repr\_\_’, ‘\_\_setattr\_\_’, ‘\_\_sizeof\_\_’, ‘\_\_str\_\_’, ‘\_\_subclasshook\_\_’]

1. Is Python case-sensitive?

A language is case-sensitive if it distinguishes between identifiers like myname and Myname. In other words, it cares about case- lowercase or uppercase. Let’s try this with Python.

>>> myname='Ayushi'

>>> Myname

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

Myname

NameError: name ‘Myname’ is not defined

As you can see, this raised a NameError. This means that Python is indeed case-sensitive.

1. What is the pass statement in Python?

There may be times in our code when we haven’t decided what to do yet, but we must type something for it to be syntactically correct. In such a case, we use the pass statement.

>>> def func(\*args):

pass

>>>

Similarly, the break statement breaks out of a loop.

>>> for i in range(7):

if i==3: break

print(i)

0

1

2

Finally, the continue statement skips to the next iteration.

>>> for i in range(7):

if i==3: continue

print(i)

0

1

2

4

5

6

1. What is a closure in Python?

A closure is said to occur when a nested function references a value in its enclosing scope. The whole point here is that it remembers the value.

>>> def A(x):

def B():

print(x)

return B

>>> A(7)()

1. Explain the //, %, and \*\* operators in Python.

The // operator performs floor division. It will return the integer part of the result on division.

>>> 7//2

3

Normal division would return 3.5 here.

Similarly, \*\* performs exponentiation. a\*\*b returns the value of a raised to the power b.

>>> 2\*\*10

1024

Finally, % is for modulus. This gives us the value left after the highest achievable division.

>>> 13%7

6

>>> 3.5%1.5

0.5

1. Why are identifier names with a leading underscore disparaged?

Since Python does not have a concept of private variables, it is a convention to use leading underscores to declare a variable private. This is why we mustn’t do that to variables we do not want to make private.

1. What is tuple unpacking?

First, let’s discuss tuple packing(boxing). It is a way to pack a set of values into a tuple.

>>> mytuple=3,4,5

>>> mytuple

(3, 4, 5)

This packs 3, 4, and 5 into mytuple.

Now, we will unpack the values from the tuple into variables x, y, and z.(Unboxing)

>>> x,y,z=mytuple

>>> x+y+z

12

1. What is Dict and List comprehensions are?

They are syntax constructions to ease the creation of a Dictionary or List based on existing iterable

13) What is lambda in Python?

It is a single expression anonymous function often used as inline function.

1. Why lambda forms in python does not have statements?

A lambda form in python does not have statements as it is used to make new function object and then return them at runtime.

1. Mention what are the rules for local and global variables in Python?

Local variables: If a variable is assigned a new value anywhere within the function’s body, it’s assumed to be local.

Global variables: Those variables that are only referenced inside a function are implicitly global

1. How can you share global variables across modules?

To share global variables across modules within a single program, create a special module. Import the config module in all modules of your application. The module will be available as a global variable across modules.

1. What happens if an error occurs that is not handled in the except block?

The program terminates. and an execution trace is sent to sys.stderr.

1. In a class definition, what does the \_\_ init\_O function do?

It overrides the any initialization from an inherited class, and is called when the class is instantiated.

1. What are negative indexes and why are they used?

Ans: The sequences in Python are indexed and it consists of the positive as well as negative numbers. The numbers that are positive uses ‘0’ that is uses as first index and ‘1’ as the second index and the process goes on like that.

The index for the negative number starts from ‘-1’ that represents the last index in the sequence and ‘-2’ as the penultimate index and the sequence carries forward like the positive number.

The negative index is used to remove any new-line spaces from the string and allow the string to except the last character that is given as S[:-1]. The negative index is also used to show the index to represent the string in correct order.

1. What is fixture in pytest.

[purpose of test fixtures](http://en.wikipedia.org/wiki/Test_fixture#Software) is to provide a fixed baseline upon which tests can reliably and repeatedly execute.

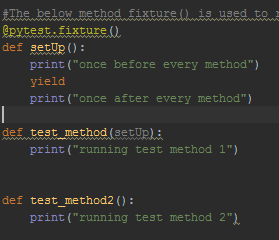
fixtures have explicit names and are activated by declaring their use from test functions, modules, classes or whole projects.

@pytest.fixture()

below method fixture() is used to run before the any function executes

yield\_fixture() is used to run before and after any function executes. Statements after yield are executed after the function  
@pytest.yield\_fixture()

currently yield\_fixture() is depreciated

eg: 

IBM:

1. what is list , tuple, set. operations in each

**List:** It is a datatype. Syntax [1,2,3]. List is mutable. Direct assignment of values are allowed in list. Eg: l[1] = 2.

.append(), .extend(), .del l[1], .remove, .pop()

**Tuple:** It is a datatype. Syntax (1,2,3). Tuple is immutable. Direct assignment of values are not allowed in tuple. Eg: l(1) = 2. Assignment not allowed error will be displayed. Tuple is faster than list as it does not allow direct assignment, hence it does not occupy large memory spaces

**Set:** It is a datatype. Syntax {1,2,3}. Set is mutable.

.union(), .intersection(), .add(), .clear(), .difference()

1. how to import excel file and read data from it

# import xlrd  
#  
# with open(r"C:\Users\surabhi.sahoo\Desktop\Automation\_Failure\_Skip\_Analysis.xlsx") as excel:  
# file = xlrd.open\_workbook(r"C:\Users\surabhi.sahoo\Desktop\Automation\_Failure\_Skip\_Analysis.xlsx")  
# print(file.nsheets)  
# print(file.sheet\_by\_index(0))  
# print(file.sheet\_names())  
# abc = file.sheet\_by\_name("Failure Analysis-<Module>")  
# print(abc.nrows)  
# print(abc.ncols)  
# for j in range(0, 3):  
# for i in range(1, 4):  
# print('{}'.format(abc.cell\_value(i, j)))

Cleo:

1. what is rest

REST: Representational State Transfer.

1. difference between URI and URL
2. various methods of http
3. difference between http and https
4. difference between get and post, post and put,
5. can we use put instead of post
6. program to remove duplicate values from a list

l1 = []  
for i in range(0, len(l)):  
 if l[i] not in l1:  
 l1.append(l[i])  
  
print(l1)

1. why we use "?" in api url provided
2. what is api key
3. what is crm. and examples

nutanix

1. program to read strings from a file and print the words that are not duplicate
2. roman numerals

file read

a =13

l = [1,2,5]

terralogic

1. All Data types questions from python
2. All selenium questions (mouse operation, drag and drop, framework structure etc)
3. Linked list,generators, decorators etc
4. deep copy/ shallow copy/list operations
5. difference between extend and append
6. read a file and find if the text is present in the file
7. write a program to find the missing number in a list of 1 to 100
8. convert two list into a dictionary(using Zip)
9. you have 10 rs, cost of each chocolate is 2. for every 2 wrapper you get 1 chocolate. find out how many chocolates you can eat at most
10. how to get entire items in a dropdown using selenium
11. how to remove duplicate letter from a string
12. how to navigate in the folder structure using program
13. what modules you have used
14. regular expressions
15. difference between partial link text and link text
16. what are the types of locaters
17. difference between assert and verify

techmahindra

1. private method--how to test it
2. interpretor name
3. python is interpreted or compiled
4. how to automate captcha
5. what is the virtual environment python uses

wipro

1. appium
2. how to set it up
3. adb commands
4. how to take screenshots
5. how many types of locaters are present
6. how can u locate an element
7. how many types of xpath
8. how do u set capablities
9. if device is in inactive state what do u do
10. scenario to automate launching of application,entering uname n password navigating to next screen and quiting the app
11. oops concept in python

pytest --collect-only---->gives the list of test cases that can be executed

generate automation report

py.test -v -s <filename.py> --html=htmlreport.html

1. Differences between Generator function and a Normal function

Here is how a generator function differs from a normal function.

Generator function contains one or more yield statement.

When called, it returns an object (iterator) but does not start execution immediately.

Methods like \_\_iter\_\_() and \_\_next\_\_() are implemented automatically. So we can iterate through the items using next().

Once the function yields, the function is paused and the control is transferred to the caller.

Local variables and their states are remembered between successive calls.

Finally, when the function terminates, StopIteration is raised automatically on further calls.

eg:

def my\_gen():

n = 1

print('This is printed first')

# Generator function contains yield statements

yield n

n += 1

print('This is printed second')

yield n

n += 1

print('This is printed at last')

yield n

a = my\_gen()

print(next(a))

print(next(a))

print(next(a))

print(next(a))

closure

Decorators

A decorator takes in a function, adds some functionality and returns it.

This is also called metaprogramming as a part of the program tries to modify another part of the program at compile time.

function that take other functions as arguments are also called higher order functions.

def inc(x):

return x + 1

def dec(x):

return x - 1

def operate(func, x):

result = func(x)

return result

operate(inc,3)

operate(dec,3)

6

def make\_pretty(func):

def inner():

print("I got decorated")

func()

print("I;m done")

return inner

def make\_prettier(func):

def inner():

print("I got really decorated")

func()

return inner

def ordinary():

print("I am ordinary")

def extraordinary():

print("I am extra-ordinary")

make\_pretty(make\_prettier(ordinary))

"I got decorated"

"I got really decorated"

"I am ordinary"

Copy

we usually use '=' operator to copy the values

for eg:

list1 = [1,2,3,4,5]

list2 = list1

here 2 reference variables are created for the same values

if we modify 1 value in list1, the values of list2 also gets modified automatically

if we dont want to do that the concept of deep and shallow copy comes

for making these copys work we use the Copy "module"

What is Assertion?

Assertions are statements that assert or state a fact confidently in your program.

For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.

Assertions are simply boolean expressions that checks if the conditions return true or not.

If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.

In Python we can use assert statement in two ways as mentioned above.

assert statement has a condition and if the condition is not satisfied the program will stop and give AssertionError.

assert statement can also have a condition and a optional error message.

If the condition is not satisfied assert stops the program and gives AssertionError along with the error message.

Key Points to Remember

Assertions are the condition or boolean expression which are always supposed to be true in the code.

assert statement takes an expression and optional message.

assert statement is used to check types, values of argument and the output of the function.

assert statement is used as debugging tool as it halts the program at the point where an error occurs.

1. lamda
2. boxing and unboxing
3. memory managment
4. format in which memory is stored
5. heirarchy inheritance
6. monkey patching
7. decorater in python
8. static in python
9. association and aggreagation
10. deep copy and shallow copy
11. pass by value and pass by reference in python

**Scope**  
Variables can only reach the area in which they are defined, which is called *scope*. Think of it as the area of code where variables can be used. Python supports global variables (usable in the entire program) and local variables.

By default, all variables declared in a function are local variables. To access a global variable inside a function, it’s required to explicitly define ‘global variable’.

If a variable can be reached anywhere in the code is called a ***global variable.*** If a variable is known only inside the scope, we call it a***local variable.***

# Global and Local variables

# 

A **global variable** (x) can be reached and modified anywhere in the code, **local variable** (z) exists only in block 3.

**class:** it is blue print of object or a template to create an object

for eg: Object : House is an object

,class : the blueprint or the architure design of house is called a class

Everything in Python is an object. Every object can contain methods and variables (with unique values). Objects are created (often called instantiated) from classes

class CoffeeMachine:

name = ""

beans = 0

water = 0

def \_\_init\_\_(self, name, beans, water):

self.name = name

self.beans = beans

self.water = water

def addBean(self):

self.beans = self.beans + 1

def removeBean(self):

self.beans = self.beans - 1

def addWater(self):

self.water = self.water + 1

def removeWater(self):

self.water = self.water - 1

def printState(self):

print "Name = " + self.name

print "Beans = " + str(self.beans)

print "Water = " + str(self.water)

pythonBean = CoffeeMachine("Python Bean", 83, 20)

pythonBean.printState()

print ""

pythonBean.addBean()

pythonBean.printState()

Without using the self keyword you are using local variables. Local variables cannot be accessed outside the functions, because of the scope of the variable.

The variable onlyInScope can only be accessed within \_\_init\_\_(). Adding the self keyword makes them accessibly outside by newly created objects. Adding self will tell Python these are class variables.

Encapsulation

In an object oriented python program, you can *restrict access* to methods and variables. This can prevent the data from being modified by accident and is known as *encapsulation*.

To summarize, in Python there are:

| **Type** | **Description** |
| --- | --- |
| public methods | Accessible from anywhere |
| private methods | Accessible only in their own class. starts with two underscores |
| public variables | Accessible from anywhere |
| private variables | Accesible only in their own class or by a method if defined. starts with two underscores |

Other programming languages have protected class methods too, but Python does not

Encapsulation gives you more control over the degree of coupling in your code, it allows a class to change its implementation without affecting other parts of the code.

We can still modify the values of private variables by using the set<variablename(value)>

The private method \_\_updateSoftware() can only be called within the class itself. It can never be called from outside the class.”

The “private” attributes and methods are not really hidden, they’re just automatically renamed adding “\_Car” in the beginning of their name… This truly prevents from accessing them \*accidentally\* but not intentionally

Method Overloading

Depending on the function definition, it can be called with zero, one, two or more parameters.

This is known as method overloading. Not all programming languages support method overloading, but Python does.

**class** Human:

 #since we have set the name value to none we can call the function without the value also

**def** sayHello(self, name=None):

**if** name **is** **not** None:

**print** 'Hello ' + name

**else**:

**print** 'Hello '

*# Create instance*

obj = Human()

*# Call the method*

obj.sayHello()

*# Call the method with a parameter*

obj.sayHello('Guido')

# Polymorphism

Sometimes an object comes in many types or forms. If we have a button, there are many different draw outputs  but they do share the same logic: onClick().  We access them using the same method . This idea is called Polymorphism.

INTERFACE

It is a template

# Recursion

In Python, **a function is recursive if it calls itself and has a termination condition**.

## **Limitations of recursions**

Everytime a function calls itself and stores some memory. Thus, a recursive function could hold much more memory than a traditional function. Python stops the function calls after a depth of 1000 calls by default.

When we try to exceed the depth it gives, RuntimeError: maximum recursion depth exceeded

We can still modify it by using: sys.setrecursionlimit(5000)

# Logging

# We can track events in a software application, this is known as **logging.** As opposed to just printing the errors, logging can be configured to disable output or save to a file. This is a big advantage to simple printing the errors.

## **Logging example**

**import** logging

*# print a log message to the console.*

logging.warning('This is a warning!')

# We can easily output to a file:

**import** logging

logging.basicConfig(filename='program.log',level=logging.DEBUG)

logging.warning('An example message.')

logging.warning('Another message')

# The importance of a log message depends on the severity.

# The logger module has several levels of severity. We set the level of severity using this line of code:

logging.basicConfig(level=logging.DEBUG)

These are the levels of severity:

| **Type** | **Description** |
| --- | --- |
| DEBUG | Information only for problem diagnostics |
| INFO | The program is running as expected |
| WARNING | Indicate something went wrong |
| ERROR | The software will no longer be able to function |
| CRITICAL | Very serious error |

## **Time in log**

You can enable time for logging using this line of code:

|  |
| --- |
| logging.basicConfig(format='%(asctime)s %(message)s') |

**import** logging

logging.basicConfig(format='%(asctime)s %(message)s', level=logging.DEBUG)

logging.info('Logging app started')

logging.warning('An example logging message.')

logging.warning('Another log message')

# Threading and threads

# 