14.what is builtin functions?

*Functions that come built into the Python language itself are called built-in functions and are readily available to us. Functions like print(), input(), eval() etc. that we have been using, are some examples of the built-in function. There are 68 built-in functions (may change with version) in Python*

120.what is repr()?

*The [repr](https://docs.python.org/2/library/functions.html" \l "repr" \o "repr) module or alternatively repr() implementation provides a means for producing object representations with limits on the size of the resulting strings. This is used in the Python debugger and may be useful in other contexts as well.*

118.what is continue?

*The continue statement in Python returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.*

*The continue statement can be used in both while and for loops.*

117. what is break?

*The****break****statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.*

*The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The****break****statement can be used in both while and for loops.*

88.what is deque?

*Deque is a part of the collections library. It provides you with a double ended queue which means that you can append and delete elements from either side of the list. First we have to import the deque module from the collections library. This can be done by typing:*

from collections import deque

*Now we can instantiate a deque object.*

d = deque()

*That was simple. It works like python lists and provides you with somewhat similar methods as well. For example you can do:*

d = deque()

d.append('1')

d.append('2')

d.append('3')

len(d)

d[0]

d[-1]

*output:*

3

'1'

'3'

*You can pop values from both sides of the deque. This means that you can do this:*

d = deque('12345')

len(d)

d.popleft()

d.pop()

d

*Output:*

5

'1'

'5'

deque(['2', '3', '4'])

*We can also limit the amount of items a deque can hold. By doing this when we achieve the maximum limit of out deque it will simply pop out the items from the opposite end. It is better to explain it using an example so here you go:*

d = deque(maxlen=30)

*Now whenever you insert values after 30, the leftmost value will be popped from the list. You can also expand the list in any direction with new values:*

d = deque([1,2,3,4,5])

d.extendleft([0])

d.extend([6,7,8])

d

*Output:*

deque([0, 1, 2, 3, 4, 5, 6, 7, 8])

94.what is glue language?

*Glue language refers to a programming language that is designed specifically to write and manage program and code, which connects together different software components. It enables interconnecting, support and the integration of software programs and components created using different programming languages and platforms.*

107.what is local and global variable?

*Local Variable: A local variable is a variable which is either a variable declared within the function or is an argument passed to a function.*

*Global Variable: In computer programming, a global variable is a variable with globalscope, meaning that it is visible (hence accessible) throughout the program, unless shadowed.*

106.what is private variable?

*Private variables, are variables that are visible only to the class to which they belong.*

111.what is function?example?

*A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.*

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

112.what is calling function?

*A****function****which is called by other****function****is called* ***calling function***

71.what Is keyword argurments?

*If you have some functions with many parameters and you want to specify only some of them, then you can give values for such parameters by naming them - this is called keyword arguments - we use the name (keyword) instead of the position (which we have been using all along) to specify the arguments to the function.*

*There are two advantages - one, using the function is easier since we do not need to worry about the order of the arguments. Two, we can give values to only those parameters which we want, provided that the other parameters have default argument values.*

#!/usr/bin/python

# Filename: func\_key.py

def func(a, b=5, c=10):

print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)

func(25, c=24)

func(c=50, a=100)

#### Output

$ python func\_key.py

a is 3 and b is 7 and c is 10

a is 25 and b is 5 and c is 24

a is 100 and b is 5 and c is 50

72.what is arbitry argurments lists?

*The special syntax, \*args and \*\*kwargs in function definitions is used to pass a variable number of arguments to a function. The single asterisk form (\*args) is used to pass a*non-keyworded*, variable-length argument list, and the double asterisk form is used to pass a*keyworded*, variable-length argument list. Here is an example of how to use the non-keyworded form. This example passes one formal (positional) argument, and two more variable length arguments*.

def test\_var\_args(farg, \*args):

print "formal arg:", farg

for arg in args:

print "another arg:", arg

test\_var\_args(1, "two", 3)

*Results:*

formal arg: 1

another arg: two

another arg: 3

*Here is an example of how to use the keyworded form. Again, one formal argument and two keyworded variable arguments are passed.*

def test\_var\_kwargs(farg, \*\*kwargs):

print "formal arg:", farg

for key in kwargs:

print "another keyword arg: %s: %s" % (key, kwargs[key])

test\_var\_kwargs(farg=1, myarg2="two", myarg3=3)

*Results:*

formal arg: 1

another keyword arg: myarg2: two

another keyword arg: myarg3: 3

#### Using \*args and \*\*kwargs when calling a function

*This special syntax can be used, not only in function definitions, but also when*calling*a function.*

def test\_var\_args\_call(arg1, arg2, arg3):

print "arg1:", arg1

print "arg2:", arg2

print "arg3:", arg3

args = ("two", 3)

test\_var\_args\_call(1, \*args)

*Results:*

arg1: 1

arg2: two

arg3: 3

*Here is an example using the keyworded form when calling a function:*

def test\_var\_args\_call(arg1, arg2, arg3):

print "arg1:", arg1

print "arg2:", arg2

print "arg3:", arg3

kwargs = {"arg3": 3, "arg2": "two"}

test\_var\_args\_call(1, \*\*kwargs)

*Results:*

arg1: 1

arg2: two

arg3: 3

77.what is default arguments?

*Every now and then you need to define a function that may accept an argument and shall behave differently whether or not the argument is present. If a language does not provide support for such cases you only have two choices: the first one is to define two different functions and to decide which is the most suitable one to call every time, the second one is to force the presence of the argument, selecting a "null" value that signals that the argument must not be used (such as 0 or None, for example). Both solutions are viable but suboptimal.*

*Python, like other languages, provides support for default argument values, that is function arguments that can either be specified by the caller or left blank to automatically receive a predefined value.*

*A very simple (and rather useless) example of default value is the following:*

def log(message=None):

if message:

print("LOG: {0}".format(message))

*This function may be run with an argument (which can be None)*

>>> log("File closed")

LOG: File closed

>>> log()

LOG: None

>>>

*but can also be called without arguments, in which case it will receive the default value set in its prototype (None, in this case)*

24.what is file wild card?

*File in a directory can be searched using wild cards. This can be done by importing glob module and then searching the file names using wild cards.*

>>> import glob

>>> glob.glob('./[0-9].\*')

['./1.gif', './2.txt']

>>> glob.glob('\*.gif')

['1.gif', 'card.gif']

>>> glob.glob('?.gif')

['1.gif']

10.what is suite means?

*Compound statements consist of one or more ‘clauses.’ A clause consists of a header and a ‘suite.’ The clause headers of a particular compound statement are all at the same indentation level. Each clause header begins with a uniquely identifying keyword and ends with a colon. A suite is a group of statements controlled by a clause. A suite can be one or more semicolon-separated simple statements on the same line as the header, following the header’s colon, or it can be one or more indented statements on subsequent lines. Only the latter form of suite can contain nested compound statements; the following is illegal, mostly because it wouldn’t be clear to which*[*if*](https://docs.python.org/2/reference/compound_stmts.html#if)*clause a following*[*else*](https://docs.python.org/2/reference/compound_stmts.html#else)*clause would belong:*

if test1: if test2: print x

37.how to find elements in set?

>>> a\_set = set(['a', 'b', 'c'])

>>> 'a' in a\_set

True

>>>'d' in a\_set

False

64.what is return stmt?

*It returns the flow of control to the calling function. It also returns output/results to the calling function.*

*Consider the function below:*

def am\_i\_wrong(answer):

if answer == 'yes':

return True

else:

return False

*You have multiple returns. So return doesn't simply end the function definition. It instead is the point at which the function returns the result to the caller.*

*If answer is equal to 'yes' then anything after the if statement (after if and else) is never run because the function has already returned.*