1).What is Python really?
Python is a programming language with objects, modules, threads, exceptions and automatic memory management. The benefits of pythons are that it is simple and easy, portable, extensible, build-in data structure and it is an open source.

2)What is the Python interpreter?
The Python interpreter is a program that runs the Python programs you write

3). What is source code?
Source code is the statements you write for your program—it consists of text in text files that normally end with a .py extension.

4). What is byte code?
Byte code is the lower-level form of your program after Python compiles it. Python automatically stores byte code in files with a .pyc extension.

5). What is the PVM?
The PVM is the Python Virtual Machine—the runtime engine of Python that interprets your compiled byte code.

6). Name two or more variations on Python's standard execution model.
Psyco, Shed Skin, and frozen binaries are all variations on the execution model. In addition, the alternative implementations of Python named in the next two answers modify the model in some fashion as well—by replacing byte code and VMs, or by adding tools and JITs.

7). How are CPython, Jython, and IronPython different?
CPython is the standard implementation of the language. Jython and IronPython implement Python programs for use in Java and .NET environments, respectively; they are alternative compilers for Python

8). How can you start an interactive interpreter session?
Stackless is an enhanced version of Python aimed at concurrency, and PyPy is a reimplementation of Python targeted at speed. PyPy is also the successor to Psyco, and incorporates the JIT concepts that Psyco pioneered.

9). Where do you type a system command line to launch a script file?
You type system command lines in whatever your platform provides as a system console: a Command Prompt window on Windows; an xterm or terminal window on

Unix, Linux, and Mac OS X; and so on. You type this at the system's prompt, not at the Python interactive interpreter's ">>>" prompt—be careful not to confuse these prompts.

10). Name four or more ways to run the code saved in a script file.
Code in a script (really, module) file can be run with system command lines, file icon clicks, imports and reloads, the exec built-in function, and IDE GUI selections such as IDLE's Run→Run Module menu option. On Unix, they can also be run as executables with the #! trick, and some platforms support more specialized launching techniques (e.g., drag and drop). In addition, some text editors have unique ways to run Python code, some Python programs are provided as standalone "frozen binary" executables, and some systems use Python code in embedded mode, where it is run automatically by an enclosing program written in a language like C, C++, or Java. The latter technique is usually done to provide a user customization layer.

11). Name two pitfalls related to clicking file icons on Windows.
Scripts that print and then exit cause the output file to disappear immediately, before you can view the output (which is why the input trick comes in handy); error messages generated by your script also appear in an output window that closes before you can examine its contents (which is one reason that system command lines and IDEs such as IDLE are better for most development).

12). Why might you need to reload a module?
Python imports (loads) a module only once per process, by default, so if you've changed its source code and want to run the new version without stopping and restarting Python, you'll have to reload it. You must import a module at least once before you can reload it. Running files of code from a system shell command line, via an icon click, or via an IDE such as IDLE generally makes this a nonissue, as those launch schemes usually run the current version of the source code file each time.

13). How do you run a script from within IDLE?
Within the text edit window of the file you wish to run, select the window's Run→Run Module menu option. This runs the window's source code as a top-level script file and displays its output back in the interactive Python shell window
14). Name two pitfalls related to using IDLE.
IDLE can still be hung by some types of programs—especially GUI programs that perform multithreading (an advanced technique beyond this book's scope). Also, IDLE has some usability features that can burn you once you leave the IDLE GUI: a script's variables are automatically imported to the interactive scope in IDLE and working

directories are changed when you run a file, for instance, but Python itself does not take such steps in general.

15). What is a namespace, and how does it relate to module files?
A namespace is just a package of variables (i.e., names). It takes the form of an object with attributes in Python. Each module file is automatically a namespace— that is, a package of variables reflecting the assignments made at the top level of the file. Namespaces help avoid name collisions in Python programs: because each module file is a self-contained namespace, files must explicitly import other files in order to use their names.
16). Name four of Python's core data types.

17). Why are they called "core" data types?
They are known as "core" types because they are part of the Python language itself and are always available; to create other objects, you generally must call functions in imported modules. Most of the core types have specific syntax for generating the objects: 'spam', for example, is an expression that makes a string and determines the set of operations that can be applied to it. Because of this, core types are hardwired into Python's syntax. In contrast, you must call the built-in open function to create a file object (even though this is usually considered a core type too).

18). What does "immutable" mean, and which three of Python's core types are considered immutable?
An "immutable" object is an object that cannot be changed after it is created. Numbers, strings, and tuples in Python fall into this category. While you cannot
change an immutable object in place, you can always make a new one by running an expression. Bytearrays in recent Pythons offer mutability for text, but they are not normal strings, and only apply directly to text if it's a simple 8-bit kind (e.g., ASCII).

19). What does "sequence" mean, and which three types fall into that category?
. A "sequence" is a positionally ordered collection of objects. Strings, lists, and tuples are all sequences in Python. They share common sequence operations, such as indexing, concatenation, and slicing, but also have type-specific method calls. A related term, "iterable," means either a physical sequence, or a virtual one that produces its items on request.
20). What does "mapping" mean, and which core type is a mapping?
The term "mapping" denotes an object that maps keys to associated values. Python's dictionary is the only mapping type in the core type set. Mappings do not maintain any

left-to-right positional ordering; they support access to data stored by key, plus type-specific method calls.

21). What is "polymorphism," and why should you care?

"Polymorphism" means that the meaning of an operation (like a +) depends on the objects being operated on. This turns out to be a key idea (perhaps the key idea) behind using Python well—not constraining code to specific types makes that code automatically applicable to many types

22). What is the value of the expression 2 * (3 + 4) in Python?

The value will be 14, the result of 2 * 7, because the parentheses force the addition to happen before the multiplication.

23). What is the value of the expression 2 * 3 + 4 in Python?

The value will be 10, the result of 6 + 4. Python's operator precedence rules are applied in the absence of parentheses, and multiplication has higher precedence than (i.e., happens before) addition

24). What is the value of the expression 2 + 3 * 4 in Python?

This expression yields 14, the result of 2 + 12, for the same precedence reasons as in the prior question

25). What tools can you use to find a number's square root, as well as its square?

Functions for obtaining the square root, as well as pi, tangents, and more, are available in the imported math module. To find a number's square root, import math and call math.sqrt(N). To get a number's square, use either the exponent expression X ** 2 or the built-in function pow(X, 2). Either of these last two can also compute the square root when given a power of 0.5 (e.g., X ** .5).

26). What is the type of the result of the expression 1 + 2.0 + 3?

The result will be a floating-point number: the integers are converted up to floating point, the most complex type in the expression, and floating-point math is used to evaluate it.

27). How can you truncate and round a floating-point number?

The int(N) and math.trunc(N) functions truncate, and the round(N, digits) function rounds. We can also compute the floor with math.floor(N) and round for display with string formatting operations

28). How can you convert an integer to a floating-point number?
The float(I) function converts an integer to a floating point; mixing an integer with a floating point within an expression will result in a conversion as well. In some sense, Python 3.X / division converts too—it always returns a floating-point result that includes the remainder, even if both operands are integers.

29). How would you display an integer in octal, hexadecimal, or binary notation?
The oct(I) and hex(I) built-in functions return the octal and hexadecimal string forms for an integer. The bin(I) call also returns a number's binary digits string in Pythons 2.6, 3.0, and later. The % string formatting expression and format string method also provide targets for some such conversions.

30). How might you convert an octal, hexadecimal, or binary string to a plain integer?
The int(S, base) function can be used to convert from octal and hexadecimal strings to normal integers (pass in 8, 16, or 2 for the base). The eval(S) function can be used for this purpose too, but it's more expensive to run and can have security issues. Note that integers are always stored in binary form in computer memory; these are just display string format conversions.

31). Can the string find method be used to search a list?
No, because methods are always type-specific; that is, they only work on a single data type. Expressions like X+Y and built-in functions like len(X) are generic, though, and may work on a variety of types. In this case, for instance, the in membership expression has a similar effect as the string find, but it can be used to search both strings and lists. In Python 3.X, there is some attempt to group methods by categories (for example, the mutable sequence types list and bytearray have similar method sets), but methods are still more type-specific than other operation sets.

32). Can a string slice expression be used on a list?
Yes. Unlike methods, expressions are generic and apply to many types. In this case, the slice expression is really a sequence operation—it works on any type of sequence object, including strings, lists, and tuples. The only difference is that when you slice a list, you get back a new list

33). How would you convert a character to its ASCII integer code? How would you convert the other way, from an integer to a character?
The built-in ord(S) function converts from a one-character string to an integer character code; chr(I) converts from the integer code back to a string. Keep in mind, though, that these integers are only ASCII codes for text whose characters are drawn only from

ASCII character set. In the Unicode model, text strings are really sequences of Unicode code point identifying integers, which may fall outside the 7-bit range of numbers reserved by ASCII

34). How might you go about changing a string in Python?
Strings cannot be changed; they are immutable. However, you can achieve a similar effect by creating a new string—by concatenating, slicing, running formatting expressions, or using a method call like replace—and then assigning the result back to the original variable name.

35). Given a string S with the value "s,pa,m", name two ways to extract the two characters in the middle.
You can slice the string using S[2:4], or split on the comma and index the string using S.split(',')[1]. Try these interactively to see for yourself.

36). How many characters are there in the string "a\nb\x1f\000d"?
Six. The string "a\nb\x1f\000d" contains the characters a, newline (\n), b, binary 31 (a hex escape \x1f), binary 0 (an octal escape \000), and d. Pass the string to the built-in len function to verify this, and print each of its character's ord results to see the actual code point (identifying number) values.

37). Why might you use the string module instead of string method calls?
You should never use the string module instead of string object method calls today —it's deprecated, and its calls are removed completely in Python 3.X. The only valid reason for using the string module at all today is for its other tools, such as predefined constants. You might also see it appear in what is now very old and dusty Python code .

38). Name two ways to build a list containing five integer zeros.
A literal expression like [0, 0, 0, 0, 0] and a repetition expression like [0] * 5 will each create a list of five zeros. In practice, you might also build one up with a loop that starts with an empty list and appends 0 to it in each iteration, with L.append(0). A list comprehension ([0 for i in range(5)]) could work here, too, but this is more work than you need to do for this answer.

39). Name two ways to build a dictionary with two keys, 'a' and 'b', each having an associated value of 0.
A literal expression such as {'a': 0, 'b': 0} or a series of assignments like D = {}, D['a'] = 0, and D['b'] = 0 would create the desired dictionary. You can also use the newer and simpler-to-code dict(a=0, b=0) keyword form, or the more flexible dict([('a', 0), ('b', 0)])

key/value sequences form. Or, because all the values are the same, you can use the special form dict.fromkeys('ab', 0). In 3.X and 2.7, you can also use a dictionary comprehension: {k:0 for k in 'ab'}, though again, this may be overkill here.

40). Name four operations that change a list object in place.
The append and extend methods grow a list in place, the sort and reverse methods order and reverse lists, the insert method inserts an item at an offset, the remove and pop methods delete from a list by value and by position, the del statement deletes an item or slice, and index and slice assignment statements replace an item or entire section.

41). Name four operations that change a dictionary object in place.
Dictionaries are primarily changed by assignment to a new or existing key, which creates or changes the key's entry in the table. Also, the del statement deletes akey's entry, the dictionary update method merges one dictionary into another in place, and D.pop(key) removes a key and returns the value it had. Dictionaries also have other, more exotic in-place change methods not presented in this chapter, such as setdefault; see reference sources for more details.

42). Why might you use a dictionary instead of a list?
Dictionaries are generally better when the data is labeled (a record with field names, for example); lists are best suited to collections of unlabeled items (such as all the files in a directory). Dictionary lookup is also usually quicker than searching a list, though this might vary per program.

43). How can you determine how large a tuple is? Why is this tool located where it is?
The built-in len function returns the length (number of contained items) for any container object in Python, including tuples. It is a built-in function instead of a type method because it applies to many different types of objects. In general, builtin functions and expressions may span many object types; methods are specific to a single object type, though some may be available on more than one type (index, for example, works on lists and tuples).

44). Write an expression that changes the first item in a tuple. (4, 5, 6) should become (1, 5, 6) in the process.
Because they are immutable, you can't really change tuples in place, but you can generate a new tuple with the desired value. Given T = (4, 5, 6), you can change the first item by making a new tuple from its parts by slicing and concatenating: T = (1,) + T[1:]. (Recall that single-item tuples require a trailing comma.) You could also convert the

tuple to a list, change it in place, and convert it back to a tuple, but this is more expensive and is rarely required in practice—simply use a list if you know that the object will require in-place changes.

45). What is the default for the processing mode argument in a file open call?
The default for the processing mode argument in a file open call is 'r', for reading text input. For input text files, simply pass in the external file’s name.

46). What module might you use to store Python objects in a file without converting them to strings yourself?
The pickle module can be used to store Python objects in a file without explicitly converting them to strings. The struct module is related, but it assumes the data is to be in packed binary format in the file; json similarly converts a limited set of Python objects to and from strings per the JSON format.

47). How might you go about copying all parts of a nested structure at once?
Import the copy module, and call copy.deepcopy(X) if you need to copy all parts of a nested structure X. This is also rarely seen in practice; references are usually the desired behavior, and shallow copies (e.g., aList[:], aDict.copy(), set(aSet)) usually suffice for most copies.

48). When does Python consider an object true?
An object is considered true if it is either a nonzero number or a nonempty collection object. The built-in words True and False are essentially predefined to have the same meanings as integer 1 and 0, respectively.

49). What three things are required in a C-like language but omitted in Python?
C-like languages require parentheses around the tests in some statements, semicolons at the end of each statement, and braces around a nested block of code.
50). How is a statement normally terminated in Python?
The end of a line terminates the statement that appears on that line. Alternatively, if more than one statement appears on the same line, they can be terminated with semicolons; similarly, if a statement spans many lines, you must terminate it by closing a bracketed syntactic pair.

51). How are the statements in a nested block of code normally associated in Python?
The statements in a nested block are all indented the same number of tabs or spaces

52). How can you make a single statement span multiple lines?

You can make a statement span many lines by enclosing part of it in parentheses, square brackets, or curly braces; the statement ends when Python sees a line that contains the closing part of the pair.
53). How can you code a compound statement on a single line?
The body of a compound statement can be moved to the header line after the colon, but only if the body consists of only non compound statements.

54). Is there any valid reason to type a semicolon at the end of a statement in Python?
Only when you need to squeeze more than one statement onto a single line of code. Even then, this only works if all the statements are non compound, and it's discouraged because it can lead to code that is difficult to read.

55). What is a try statement for?
The try statement is used to catch and recover from exceptions (errors) in a Python script. It's usually an alternative to manually checking for errors in your code.
56). What is the most common coding mistake among Python beginners?
Forgetting to type the colon character at the end of the header line in a compound statement is the most common beginner's mistake. If you're new to Python and haven't made it yet, you probably will soon!

57). Name three ways that you can assign three variables to the same value.
You can use multiple-target assignments (A = B = C = 0), sequence assignment (A, B, C = 0, 0, 0), or multiple assignment statements on three separate lines (A = 0, B = 0, and C = 0). With the latter technique, as introduced in Chapter 10, you can also string the three separate statements together on the same line by separating them with semicolons (A = 0; B = 0; C = 0).
58). Why might you need to care when assigning three variables to a mutable object?
If you assign them this way: A = B = C = [] all three names reference the same object, so changing it in place from one (e.g., A.append(99)) will affect the others. This is true only for in-place changes to mutable objects like lists and dictionaries; for immutable objects such as numbers and strings, this issue is irrelevant

59). What's wrong with saying L = L.sort()?
The list sort method is like append in that it makes an in-place change to the subject list—it returns None, not the list it changes. The assignment back to L sets L to None, not to the sorted list. As discussed both earlier and later in this book , a newer built-in function, sorted, sorts any sequence and returns a new list with the sorting result;

because this is not an in-place change, its result can be meaningfully assigned to a name.

60). How might you use the print operation to send text to an external file?
To print to a file for a single print operation, you can use 3.X's print(X, file=F) call form, use 2.X's extended print >> file, X statement form, or assign sys.stdout to a manually opened file before the print and restore the original after. You can also redirect all of a program's printed text to a file with special syntax in the system shell, but this is outside Python's scope

61). How might you code a multiway branch in Python?
An if statement with multiple elif clauses is often the most straightforward way to code a multiway branch, though not necessarily the most concise or flexible. Dictionary indexing can often achieve the same result, especially if the dictionary contains callable functions coded with def statements or lambda expressions.
62). How can you code an if/else statement as an expression in Python?
In Python 2.5 and later, the expression form Y if X else Z returns Y if X is true, or Z otherwise; it's the same as a four-line if statement. The and/or combination (((X and Y) or Z)) can work the same way, but it's more obscure and requires that the Y part be true.

63). How can you make a single statement span many lines?
Wrap up the statement in an open syntactic pair ((), [], or {}), and it can span as many lines as you like; the statement ends when Python sees the closing (right) half of the pair, and lines 2 and beyond of the statement can begin at any indentation level. Backslash continuations work too, but are broadly discouraged in the Python world.
64). What do the words True and False mean?
True and False are just custom versions of the integers 1 and 0, respectively: they always stand for Boolean true and false values in Python. They're available for use in truth tests and variable initialization, and are printed for expression results at the interactive prompt. In all these roles, they serve as a more mnemonic and hence readable alternative to 1 and 0

65). What are the main functional differences between a while and a for?
The while loop is a general looping statement, but the for is designed to iterate across items in a sequence or other iterable. Although the while can imitate the for with counter loops, it takes more code and might run slower.
66). What's the difference between break and continue?

The break statement exits a loop immediately (you wind up below the entire while or for loop statement), and continue jumps back to the top of the loop (you wind up positioned just before the test in while or the next item fetch in for).
67). When is a loop’s else clause executed?
The else clause in a while or for loop will be run once as the loop is exiting, if the loop exits normally (without running into a break statement). A break exits the loop immediately, skipping the else part on the way out (if there is one).

68). How can you code a counter-based loop in Python?
Counter loops can be coded with a while statement that keeps track of the index manually, or with a for loop that uses the range built-in function to generate successive integer offsets. Neither is the preferred way to work in Python, if you need to simply step across all the items in a sequence. Instead, use a simple for loop instead, without range or counters, whenever possible; it will be easier to code and usually quicker to run

69). What can a range be used for in a for loop?
The range built-in can be used in a for to implement a fixed number of repetitions, to scan by offsets instead of items at offsets, to skip successive items as you go, and to change a list while stepping across it. None of these roles requires range, and most have alternatives—scanning actual items, three-limit slices, and list comprehensions are often better solutions today (despite the natural inclinations of ex–C programmers to want to count things!).

70). How are for loops and iterable objects related?
The for loop uses the iteration protocol to step through items in the iterable object across which it is iterating. It first fetches an iterator from the iterable by passing the object to iter, and then calls this iterator object’s __next__ method in 3.X on each iteration and catches the StopIteration exception to determine when to stop looping. The method is named next in 2.X, and is run by the next built-in function in both 3.x and 2.X. Any object that supports this model works in a for loop and in all other iteration contexts. For some objects that are their own iterator, the initial iter call is extraneous but harmless.

71). How are for loops and list comprehensions related?
Both are iteration tools and contexts. List comprehensions are a concise and often efficient way to perform a common for loop task: collecting the results of applying an expression to all items in an iterable object. It’s always possible to translate a list comprehension to a for loop, and part of the list comprehension expression looks like the header of a for loop syntactically.

72). Name four iteration contexts in the Python language.
. Iteration contexts in Python include the for loop; list comprehensions; the map built-in function; the in membership test expression; and the built-in functions sorted, sum, any, and all. This category also includes the list and tuple built-ins, string join methods, and sequence assignments, all of which use the iteration protocol (see answer #1) to step across iterable objects one item at a time.

73). What is the best way to read line by line from a text file today?
The best way to read lines from a text file today is to not read it explicitly at all: instead, open the file within an iteration context tool such as a for loop or list comprehension, and let the iteration tool automatically scan one line at a time by running the file's next handler method on each iteration. This approach is generally best in terms of coding simplicity, memory space, and possibly execution speed requirements.

74). When should you use documentation strings instead of hash-mark comments?
Documentation strings (docstrings) are considered best for larger, functional documentation, describing the use of modules, functions, classes, and methods in your code. Hash-mark comments are today best limited to smaller-scale documentation about arcane expressions or statements at strategic points on your code. This is partly because docstrings are easier to find in a source file, but also because they can be extracted and displayed by the PyDoc system.

75). Name three ways you can view documentation strings.
You can see docstrings by printing an object's __doc__ attribute, by passing it to PyDoc's help function, and by selecting modules in PyDoc's HTML-based user interfaces—either the -g GUI client mode in Python 3.2 and earlier, or the -b allbrowser mode in Python 3.2 and later (and required as of 3.3). Both run a client/ server system that displays documentation in a popped-up web browser. PyDoc can also be run to save a module's documentation in an HTML file for later viewing or printing.

76). How can you obtain a list of the available attributes in an object?
The built-in dir(X) function returns a list of all the attributes attached to any object. A list comprehension of the form [a for a in dir(X) if not a.starts with('__')] can be used to filter out internals names with underscores (we'll learn how to wrap this in a function in the next part of the book to make it easier to use).
77). How can you get a list of all available modules on your computer?

In Python 3.2 and earlier, you can run the PyDoc GUI interface, and select “open browser”; this opens a web page containing a link to every module available to your programs. This GUI mode no longer works as of Python 3.3. In Python 3.2 and later, you get the same functionality by running PyDoc’s newer all-browser mode with a -b command-line switch; the top-level start page displayed in a web browser in this newer mode has the same index page listing all available modules.

78). Explain how Python does Compile-time and Run-time code checking?
Python performs some amount of compile-time checking, but most of the checks such as type, name, etc are postponed until code execution. Consequently, if the Python code references a user -defined function that does not exist, the code will compile successfully. In fact, the code will fail with an exception only when the code execution path references the function which does not exists.

79) At what time does Python create a function?
Ans: A function is created when Python reaches and runs the def statement; this statement creates a function object and assigns it the function’s name. This normally happens when the enclosing module file is imported by another module.
80) When does the code nested inside the function definition statement run?
Ans: The function body (the code nested inside the function definition statement) is run when the function is later called with a call expression. The body runs anew each time the function is called.
81) How are lambda expressions and def statements related?
Ans: Both lambda and def create function objects to be called later. Because lambda is an expression, though, it returns a function object instead of assigning it to a name, and it can be used to nest a function definition in places where a def will not work syntactically
82) Compare and contrast map, filter, and reduce.
Ans: These three built-in functions all apply another function to items in a sequence (or other iterable) object and collect results. map passes each item to the function and collects all results, filter collects items for which the function returns a True value, and reduce computes a single value by applying the function to an accumulator and successive items.
83) What are recursive functions, and how are they used?
Ans: Recursive functions call themselves either directly or indirectly in order to loop. They may be used to traverse arbitrarily shaped structures, but they can also be used for iteration in general
84) What are some general design guidelines for coding functions?

Ans: Functions should generally be small and as self-contained as possible, have a single unified purpose, and communicate with other components through input arguments and return values. They may use mutable arguments to communicate results too if changes are expected, and some types of programs imply other communication mechanisms.
85) What is the difference between enclosing a list comprehension in square brackets and parentheses?
Ans: . List comprehensions in square brackets produce the result list all at once in memory. When they are enclosed in parentheses instead, they are actually generator expressions—they have a similar meaning but do not produce the result list all at once. Instead, generator expressions return a generator object, which yields one item in the result at a time when used in an iteration context.
86) How can you tell if a function is a generator function?
Ans: A generator function has a yield statement somewhere in its code. Generator functions are otherwise identical to normal functions syntactically, but they are compiled specially by Python so as to return an iterable generator object when called. That object retains state and code location between values
87) How are map calls and list comprehensions related? Compare and contrast the two.
Ans: The map call is similar to a list comprehension—both produce a series of values, by collecting the results of applying an operation to each item in a sequence or other iterable, one item at a time. The primary difference is that map applies a function call to each item, and list comprehensions apply arbitrary expressions.
88) Why might you have to set your PYTHONPATH environment variable?
Ans: You only need to set PYTHONPATH to import from directories other than the one in which you are working.
89) Name four file types that Python might load in response to an import operation.
Ans: Python might load a source code (.py) file, a byte code (.pyc or .pyo) file, a C extension module (e.g., a .so file on Linux or a .dll or .pyd file on Windows), or a directory of the same name for package imports. Imports may also load more exotic things such as ZIP file components.
90) How do you make a module?
Ans: To create a module, you simply write a text file containing Python statements; every source code file is automatically a module, and there is no syntax for declaring one. Import operations load module files into module objects in memory.
91) How is the reload function related to imports?
Ans: By default, a module is imported only once per process. The reload function forces a module to be imported again. It is mostly used to pick up new versions of a module’s source code during development, and in dynamic customization scenarios.
92) Name three potential pitfalls of the from statement.

Ans: The from statement can obscure the meaning of a variable ,can have problems with the reload call, and can corrupt namespaces.
93) How can you avoid repeating the full package path every time you reference a package’s content?
Ans: Use the from statement with a package to copy names out of the package directly, or use the as extension with the import statement to rename the path to a shorter synonym. In both cases, the path is listed in only one place, in the from or import statement.
94) When must you use import instead of from with packages?
Ans: You must use import instead of from with packages only if you need to access the same name defined in more than one path. With import, the path makes the references unique, but from allows only one version of any given name .

95) What is a namespace package?
Ans: A namespace package is an extension to the import model, available in Python 3.3 and later, that corresponds to one or more directories that do not have __init__.py files.
96) What does it mean when a module’s __name__ variable is the string "__main__"?
Ans: If a module’s __name__ variable is the string "__main__", it means that the file is being executed as a top-level script instead of being imported from another file in the program. That is, the file is being used as a program, not a library.
97) How is changing sys.path different from setting PYTHONPATH to modify the module search path?
Ans: Changing sys.path only affects one running program (process), and is temporary —the change goes away when the program ends. PYTHONPATH settings live in the operating system—they are picked up globally by all your programs on a machine, and changes to these settings endure after programs exit
98) What is the main point of OOP in Python?
Ans: OOP is about code reuse—you factor code to minimize redundancy and program by customizing what already exists instead of changing code in place or starting from scratch.
99) What is the difference between a class object and an instance object?
Ans: Both class and instance objects are namespaces. The main difference between them is that classes are a kind of factory for creating multiple instances. Classes also support operator overloading methods, which instances inherit
100) Why is the __init__ method used for?
Ans: If the __init__ method is coded or inherited in a class, Python calls it automatically each time an instance of that class is created. It’s known as the constructor method; it is passed the new instance implicitly, as well as any arguments passed explicitly to the class name.

101) How do you create a class?
Ans: You create a class by running a class statement; like function definitions, these statements normally run when the enclosing module file is imported.
102) How are classes related to modules?
Ans: . Classes are always nested inside a module; they are attributes of a module object. Classes and modules are both namespaces, but classes correspond to statements and support the OOP notions of multiple instances, inheritance, and operator overloading. In a sense, a module is like a singleinstance class, without inheritance, which corresponds to an entire file of code.

103) Where and how are class attributes created?
Ans: Class attributes are created by assigning attributes to a class object. They are normally generated by top-level assignments nested in a class statement—each name assigned in the class statement block becomes an attribute of the class object. Class attributes can also be created, though, by assigning attributes to the class anywhere a reference to the class object exists—even outside the class statement.
104) What does self mean in a Python class?
Ans: self is the name commonly given to the first argument in a class's method function Python automatically fills it in with the instance object that is the implied subject of the method call.
105) When might you want to support operator overloading in your classes?
Ans: Operator overloading is useful to implement objects that resemble built-in types (e.g., sequences or numeric objects such as matrixes), and to mimic the built-in type interface expected by a piece of code.
106) What are two key concepts required to understand Python OOP code?
Ans: The special self argument in method functions and the __init__ constructor method are the two cornerstones of OOP code in Python.
107) Why is it better to call back to a superclass method to run default actions, instead of copying and modifying its code in a subclass?
Ans: Copying and modifying code doubles your potential work effort in the future, regardless of the context. If a subclass needs to perform default actions coded in a superclass method, it's much better to call back to the original through the superclass's name than to copy its code.
108) What is an abstract superclass?
Ans: An abstract superclass is a class that calls a method, but does not inherit or define it—it expects the method to be filled in by a subclass. This is often used as a way to generalize classes when behavior cannot be predicted until a more specific subclass is coded.

109) Why might a class need to manually call the __init__ method in a superclass?
Ans: A class must manually call the __init__ method in a superclass if it defines an __init__ constructor of its own and still wants the superclass's construction code to run. Python itself automatically runs just one constructor—the lowest one in the tree. Superclass constructors are usually called through the class name, passing in the self instance manually: Superclass.__init__(self,).
110) How does a class's local scope differ from that of a function?
Ans: A class is a local scope and has access to enclosing local scopes, but it does not serve as an enclosing local scope to further nested code. Like modules, the class local scope morphs into an attribute namespace after the class statement is run.
111) What two operator overloading methods handle printing, and in what contexts?
Ans: Classes can support iteration by defining __getitem__ or __iter__. In all iteration contexts, Python tries to use __iter__ first, which returns an object that supports the iteration protocol with a __next__ method: if no __iter__ is found by inheritance search, Python falls back on the __getitem__ indexing method, which is called repeatedly, with successively higher indexes. If used, the yield statement can create the __next__ method automatically.
112) How can you catch in-place addition in a class?
Ans: In-place addition tries __iadd__ first, and __add__ with an assignment second. The same pattern holds true for all binary operators. The __radd__ method is also available for right-side addition.
113) What is multiple inheritance?
Ans: Multiple inheritance occurs when a class inherits from more than one superclass; it's useful for mixing together multiple packages of class-based code. The left-toright order in class statement headers determines the general order of attribute searches.
114) What is composition?
Ans: Composition is a technique whereby a controller class embeds and directs a number of objects, and provides an interface all its own; it's a way to build up larger structures with classes.
115) What are pseudoprivate attributes?
Ans: Attributes whose names begin but do not end with two leading underscores: __X.
116) Name two ways to extend a built-in object type.
Ans: You can embed a built-in object in a wrapper class, or subclass the built-in type directly. The latter approach tends to be simpler, as most original behavior is automatically inherited.
117) How do you code a new-style class?
Ans: New-style classes are coded by inheriting from the object built-in class. In Python 3.X, all classes are new-style automatically, so this derivation is not required (but

doesn’t hurt); in 2.X, classes with this explicit derivation are new-style and those without it are “classic
118) How are normal and static methods different?
Ans: Normal (instance) methods receive a self argument (the implied instance), but static methods do not. Static methods are simple functions nested in class objects. To make a method static, it must either be run through a special built-in function or be decorated with decorator syntax.
119) What happens to an exception if you don’t do anything special to handle it?
Ans: Any uncaught exception eventually filters up to the default exception handler Python provides at the top of your program. This handler prints the familiar error message and shuts down your program.
120) Name two ways to trigger exceptions in your script.
Ans: The raise and assert statements can be used to trigger an exception, exactly as if it had been raised by Python itself.
121) What is the try statement for?
Ans: The try statement catches and recovers from exceptions—it specifies a block of code to run, and one or more handlers for exceptions that may be raised during the block’s execution.
122) What is the raise statement for?
Ans: The raise statement raises (triggers) an exception. Python raises built-in exceptions on errors internally, but your scripts can trigger built-in or user-defined exceptions with raise, too.
123) What is the with/as statement designed to do, and what other statement is it like?
Ans: The with/as statement is designed to automate startup and termination activities that must occur around a block of code.
124) Name two ways that you can attach context information to exception objects.
Ans: You can attach context information to class-based exceptions by filling out instance attributes in the instance object raised, usually in a custom class constructor. For simpler needs, built-in exception superclasses provide a constructor that stores its arguments on the instance automatically.
125) Why should you not use string-based exceptions anymore today?
Ans:It have been removed from both Python 2.6 and 3.0. There are arguably good reasons for this: string-based exceptions did not support categories, state information, or behavior inheritance in the way class-based exceptions do. In practice, this made string-based exceptions easier to use at first when programs were small, but more complex to use as programs grew larger
126) How are properties and decorators related?
Ans: Properties can be coded with decorator syntax. Because the property built-in accepts a single function argument, it can be used directly as a function decorator to

define a fetch access property. Due to the name rebinding behavior of decorators, the name of the decorated function is assigned to a property whose get accessor is set to the original function decorated (name = property(name)).
127) What is multithreading?
Ans: Running several threads is similar to running several different programs concurrently.

1) Write a Python program to get the Python version you are using.

```
import sys
print("Python version")
print (sys.version)
print("Version info.")
print (sys.version_info)
```

2) Write a Python program to display the current date and time.
Sample Output :
Current date and time :
2014-07-05 14:34:14

```
import datetime
now = datetime.datetime.now()
print ("Current date and time : ")
print (now.strftime("%Y-%m-%d %H:%M:%S"))
```

3) Write a Python program to display the first and last colors from the following list.
color_list = ["Red","Green","White" ,"Black"]

```
color_list = ["Red","Green","White" ,"Black"]
print( "%s %s"%(color_list[0],color_list[-1]))
```

4)Write a Python program to calculate the length of a string.

```
def string_length(str1):
  count = 0
  for char in str1:
    count += 1
  return count
print(string_length('hello alll'))
```

5)Write a Python program to convert a string in a list

```
str1 = "The quick brown fox jumps over the lazy dog."
print(str1.split(' '))
str1 = "The-quick-brown-fox-jumps-over-the-lazy-dog."
print(str1.split('-'))
```

6) Write a Python program to sum all the items in a list

```
def sum_list(items):
    sum_numbers = 0
    for x in items:
        sum_numbers += x
    return sum_numbers
print(sum_list([1,2,-8]))
```

7)Write a Python program to remove duplicates from a list.

```
a = [10,20,30,20,10,50,60,40,80,50,40]

dup_items = set()
uniq_items = []
for x in a:
    if x not in dup_items:
        uniq_items.append(x)
        dup_items.add(x)

print(dup_items)
```

8) Write a Python program to replace the last element in a list with another list.
Sample data : [1, 3, 5, 7, 9, 10], [2, 4, 6, 8]
Expected Output: [1, 3, 5, 7, 9, 2, 4, 6, 8]

```
num1 = [1, 3, 5, 7, 9, 10]
num2 = [2, 4, 6, 8]
num1[-1:] = num2
print(num1)
```

9) Write a Python script to add a key to a dictionary.
Sample Dictionary : {0: 10, 1: 20}Expected Result : {0: 10, 1: 20, 2: 30}

```
d = {0:10, 1:20}

print(d)
```

```
d.update({2:30})
print(d)
```

10) Write a Python program to create a tuple with different data types

```
tuplex = ("tuple", False, 3.2, 1)
print(tuplex)
```

11)Write a Python program to convert a list to a tuple

```
listx = [5, 10, 7, 4, 15, 3]
print(listx)
tuplex = tuple(listx)
print(tuplex)
```

12)Write a Python program to create an intersection of sets

```
setx = set(["green", "blue"])
sety = set(["blue", "yellow"])
setz = setx & sety
print(setz)
```

13)Write a Python program to create a union of sets

```
setx = set(["green", "blue"])
sety = set(["blue", "yellow"])
seta = setx | sety
print(seta)
```

14)Write a Python program to create set difference

```
setx = set(["apple", "mango"])
sety = set(["mango", "orange"])
setz = setx & sety
print(setz)
#Set difference
setb = setx - setz
print(setb)
```

15) Write a Python program to create a symmetric difference

```
setx = set(["apple", "mango"])
sety = set(["mango", "orange"])
#Symmetric difference
setc = setx ^ sety
print(setc)
```

16)Write a Python program to issubset and issuperset.

```
setx = set(["apple", "mango"])
sety = set(["mango", "orange"])
setz = set(["mango"])
issubset = setx <= sety
print(issubset)
issuperset = setx >= sety
print(issuperset)
issubset = setz <= sety
print(issubset)
issuperset = sety >= setz
print(issuperset)
```

17)Write a Python program to construct the following pattern, using a nested for loop.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

```
n=5;
for i in range(n):
    for j in range(i):
        print ('* ', end="")
    print('')

for i in range(n,0,-1):
    for j in range(i):
        print('* ', end="")
    print('')
```

18)Write a Python program that prints all the numbers from 0 to 6 except 3 and 6.
Note : Use 'continue' statement. Expected Output : 0 1 2 4 5

```
for x in range(6):
    if (x == 3 or x==6):
        continue
    print(x,end=' ')
print("\n")
```

19)Write a Python program to check the validity of password input by users.
Validation :

- At least 1 letter between [a-z] and 1 letter between [A-Z].
- At least 1 number between [0-9].
- At least 1 character from [$#@].
- Minimum length 6 characters.
- Maximum length 16 characters.

```
import re
p= input("Input your password")
x = True
while x:
    if (len(p)<6 or len(p)>12):
        break
    elif not re.search("[a-z]",p):
        break
    elif not re.search("[0-9]",p):
        break
    elif not re.search("[A-Z]",p):
        break
    elif not re.search("[$#@]",p):
        break
    elif re.search("\s",p):
        break
    else:
        print("Valid Password")
        x=False
        break
```

```
if x:
    print("Not a Valid Password")
```

20)Write a Python program to reverse a string.

Sample String : "1234abcd".*Expected Output* : "dcba4321"

```
def string_reverse(str1):

    rstr1 = ''
    index = len(str1)
    while index > 0:
        rstr1 += str1[ index - 1 ]
        index = index - 1
    return rstr1
print(string_reverse('1234abcd'))
```

21)Write a Python function that takes a list and returns a new list with unique elements of the first list.*Sample List :* [1,2,3,3,3,3,4,5].*Unique List :* [1, 2, 3, 4, 5]

```
def unique_list(l):
  x = []
  for a in l:
    if a not in x:
      x.append(a)
  return x

print(unique_list([1,2,3,3,3,3,4,5]))
```

22)Write a Python class to reverse a string word by word.

Input string : 'hello .py'

Expected Output : '.py hello'

```
class py_solution:
    def reverse_words(self, s):
        return ' '.join(reversed(s.split()))


print(py_solution().reverse_words('hello .py'))
```

23)Write a Python program to check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9).

```
import re
def is_allowed_specific_char(string):
    charRe = re.compile(r'[^a-zA-Z0-9.]')
    string = charRe.search(string)
    return not bool(string)

print(is_allowed_specific_char("ABCDEFabcdef123450"))
print(is_allowed_specific_char("*&%@#!}{"))
```

24)Write a Python program that matches a string that has an *a* followed by zero or one 'b'.

```
import re
def text_match(text):
        patterns = 'ab*?'
```

```
    if re.search(patterns,  text):
        return 'Found a match!'
    else:
        return('Not matched!')

print(text_match("ac"))
print(text_match("abc"))
print(text_match("abbc"))
```

25)Write a Python program to read an entire text file

```
def file_read(fname):
    txt = open(fname)
    print(txt.read())

file_read('test.txt')
```

26)Write a Python program to append text to a file and display the text.

```
def file_read(fname):
    from itertools import islice
    with open(fname, "w") as myfile:
        myfile.write("Python Exercises\n")
        myfile.write("Java Exercises")
    txt = open(fname)
    print(txt.read())
file_read('abc.txt')
```

27)Write a Python program of recursion list sum.

Test Data: [1, 2, [3,4], [5,6]].Expected Result: 21

```
def recursive_list_sum(data_list):
        total = 0
        for element in data_list:
                if type(element) == type([]):
                        total = total + recursive_list_sum(element)
                else:
                        total = total + element

        return total
print( recursive_list_sum([1, 2, [3,4],[5,6]]))
```

28)Write a Python program to determine whether a given year is a leap year.

```
def leap_year(y):
    if y % 400 == 0:
        return True
    if y % 100 == 0:
        return False
    if y % 4 == 0:
        return True
    else:
        return False
print(leap_year(1900))
print(leap_year(2004))
```

29)Write a Python program to convert a string to datetime. Sample String : Jan 1 2014 2:43PM .Expected Output : 2014-07-01 14:43:00

```
from datetime import datetime
date_object = datetime.strptime('Jul 1 2014 2:43PM', '%b %d %Y %I:%M%p')
print(date_object)
```

30) Write a Python program that accepts a string and calculate the number of digits and letters.

Sample Data : Python 3.2

Expected Output :

Letters 6

Digits 2

```
s = input("Input a string")
d=l=0
for c in s:
    if c.isdigit():
        d=d+1
    elif c.isalpha():
        l=l+1
    else:
        pass
print("Letters", l)
print("Digits", d)
```