UNIT 1

PYTHON BASICS, NUMBERS AND SEQUENCES

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Write in brief about identifiers in python.

Answer :

Model Paper-II, Q1(a)

Identifiers refers to a set of valid strings that can be used as names in a computer language.

Python considers an identifier to be valid only when

1. The first character of identifier is either underscore (_) or a character.
2. The second character can be any alphanumeric character or underscore.
3. Identifiers cannot be initiated with a number or symbol except underscore(_).
4. Identifiers are case sensitive i.e., man, Man and MAN are not identical.

Q2. What are Python objects?

Answer :

Model Paper-III, Q1(a)

In Python, object model abstraction is used for the purpose of data storage. It treats any construct that holds a value as an object. All objects possess three common features.

1. **Identifier**

An identifier uniquely identifies an object. It is determined using id( ) built-in function.

2. **Type**

Type shows the kind of value an object can store, kind of operations that can be performed on the object and behaviour of object. It returns an object.

3. **Value**

It denotes data item which is represented by an object.

These three attributes are assigned to objects during the creation of objects and remain with the objects until they are deallocated.

Q3. Write in short about numbers in python.

Answer :

Numbers are of immutable type i.e., when the value of a number is changed a new object is allocated they also allows scalar storage and direct access.

Creating Number

A number is created by assigning a value to a variable.

Syntax

```
var = value
```

Example

```
aFloat = 2.3497
```

Updating a Number

Existing number can be updated by reassigning a variable to another number. The new value can be a different number or related to previous value.

Example

```
aComplex = aComplex + 2
```

Deleting a Number

Generally a number can not be deleted but, its usage can be stopped. However, it can be explicitly deleted by using del statement once the variable name is removed, it is no longer used. In case, if it is used then it results an "NameError" exception.

Example

```
del aComplex
```

Q4. Define sequences.

Answer : Model Paper-I, Q4(a)

A 'sequence' is a data structure, that stores elements, which can be accessed using index offsets. For example, the strings are sequence of characters i.e., a string 'Hello world!' containing the sequence of characters, H, e, l, l, o, W, o, r, l, d, !. These sequence of elements are stored, starting at index 0 till the index N – 1, where N is the length of the sequence.

i.e.,

0	1	2	3	4	5	6	7	8	9	10	11
H	E	L	L	O		W	O	R	L	D	!

There are three sequence types that are supported by python. They are strings, lists and tuples.

Q5. Define lists.

Answer :

List is similar to a string which provides sequential storage mechanism. Through slices, any number of consecutive elements of list can be accessed. The unique and flexible characteristics of lists that, differentiate them from strings and arrays are,

(i) List can contain different types of Python objects-standard types and user-defined objects.

(ii) At any given time, single or multiple objects can be inserted, removed and updated to the list.

(iii) It can grow and shrink.

(iv) It can even be sorted, reversed, emptied and populated.

(v) It can be added/removed to/from other lists.

(vi) It can be created easily.

(vii) Individual or multiple elements of a list can be easily added, updated or removed.

Q6. List out the built in functions of tuples.

Answer :

The following are built-in functions of tuples.

1. len( )

It returns the total length of a tuple.

```
syntax : len(tuple)
```

2. tuple( )

It converts a given list into tuple.

```
syntax : tuple(seq)
```

cmp( )

It compares the items of two given tuples.

syntax : cmp(tuple1, tuple2)

max( )

It returns the element with maximum value from a given tuple.

syntax : max(tuple)

min( )

It returns the element with minimum value from a given tuple.

syntax : min(tuple)

Q7. Discuss in brief about dictionaries.

Answer :

Model Paper-I, Q1(b)

Dictionaries refer to python's mapping type. They are similar to associative arrays, or hashes in perl. The syntax of a dictionary entry is key : value. Keys correspond to any python type and it is usually numbers or strings. Whereas, *values* correspond to any arbitrary python object. Dictionaries are enclosed by curly braces ({ }).

Creating and Assigning Dictionaries

Dictionaries are created by simply assigning a dictionary to a variable, without considering whether dictionary contains elements or not. The syntax is as follows.

```
>>>dicta = { }
>>> dictb = {'name' : 'Ramana', 'port' : 90}
>>>dicta, dictb
({ }, {'port' : 90, 'name' : 'Ramana'})
```

As shown in the example, dictionary consists of keys and their corresponding values. Each key is separated from its value by a colon (:), the items are separated by commas and the whole thing is enclosed in curly braces. In version 2.2, factory function dict( ) is used to create dictionaries.

```
>>>fdict = dict((['x', 1], ['y', 2]))
>>>fdict
{'y' : 2, 'x' : 1}
```

Q8. What are dictionary keys?

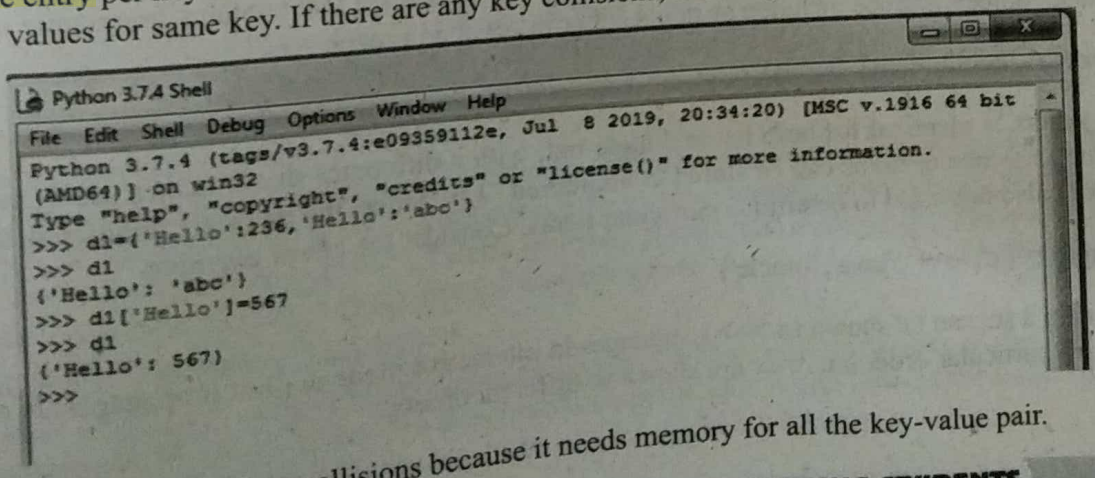
Model Paper-II, Q1(b)

Answer :

Dictionary Keys

The dictionary values does not have restrictions imposed on them. Any python object ranging from standard objects to user defined objects can be dictionary value. But its not same for dictionary keys. The properties of dictionary keys are as follows,

1. More than one entry per key not allowed one major constraint is for each dictionary key only one value is allowed rather than multiple values for same key. If there are any key collisions, the assignment that is done at last will be fixed.



Python will not check for any key collisions because it needs memory for all the key-value pair.

2. **Keys must be hashable**

The python objects can be used as keys and they must be hashable objects. The mutable types like lists and dictionaries are supported by python because they cannot be hashed. All the immutable types can be used as keys because they are hashable. Certain mutable objects are hashable and they can be used as keys.

The key need to be hashable. The hash function that is used by interpreter for computing the location to store the data depends upon key value. The value is modifiable when key is mutable object. The hash function can map to a different location if the key is modified. Therefore, the hash function can never reliably store or retrieve the associated value. The hashable keys can be selected because their values cannot be modified.

Q9. How tuples are created.

Answer :

This operation is similar to that of lists. The tuple elements are enclosed in parentheses and element is separated by a comma (,).

Example

```
>>> mytuple = ('SIA', 'GROUP', 'of', 'Companies')
>>> mytuple1 = ('Hyderabad', ['old city', 500008], 28)
>>> print my tuple
('SIA', 'GROUP', 'of', 'Companies')
>>> print mytuple1
('Hyderabad', ['old city', 500008], 28)
```

Note that in an empty tuple, we need to add a trailing comma (,) inside the tuple parentheses. (This is done to avoid confusion with a natural grouping operation of parentheses). The following example shows the creation of an empty tuple:-

```
>>> emptytuple = (None,)
>>> print emptytuple
(None,)
```

Moreover tuples can also be created by using the factory function tuple( ), as shown below:

```
>>> tuple ('sure')
('s', 'u', 'r', 'e')
```

Q10. Define sets.

Answer :

Model Paper-III, Q1(b)

Sets

A set can be defined as an unordered group of zero or more hashable objects like ints, floats, strings or tuples. Here, the word 'hashable' means immutable. It does not defines lists or other sets. A set that contains Immutable object is referred to as Frozen set. Immutable objects can be included as items either in ser or in Frozen sets.

Syntax: name = {item1, item2, item3, item4}

The above syntax is identical for both list and tuple but, with a difference that it makes use of curly braces instead of square brackets. The sequence of items can be stored in unordered way. Since, the items are unordered, sets cannot be shared, copied or the index is also not used to determine particular items. Consider the below example,

```
>>>Languages = {'c', 'c++', 'java', 'oracle'}
```

The elements in a set can be shown as >>> Languages in interactive mode or print (Languages) in a program. The items in a set are stored in a particular order but, they are shown in different orders.

PART-B
ESSAY QUESTIONS WITH SOLUTIONS
1.1 PYTHON BASICS

Q11. What is Python? Explain the features of Python language.

Answer :

Python

It is a simple interactive, interpreted object oriented programming language, when compared to other programming languages, it is a very robust, elegant and a powerful language.

Features of Python

1. **Follows Object-oriented Programming Paradigm**

   Python is an object-oriented programming language, as it separates data from logic. Moreover, Python is not only object-oriented, but also a proper combination of multiple programming paradigms.

2. **Easy to Learn**

   Very few keywords, simple structure and well defined syntaxes possessed by Python made it easy to learn.

3. **Easily Readable**

   Unlike other programming languages, it does not make use of symbols like ($), semicolon (;), tildes (~) for accessing variables, defining code blocks and pattern matching. Rather, it defines a simple and clear code, which is easier for the programmer to read. Python language has been designed in a way, that the code written by one programmer can be easily understood by the other.

4. **Easily Maintainable**

   Since, Python is simple unlike other programming languages, it does not require much difficulty for maintenance. For example, for the existence of its code, it does not depend on the presence of the programmer at a company. The most important advantage of Python is that, at the time of reviewing a script written very long time ago, the programmer does not require much guidance from a reference book.

5. **Uses High-level Data Structure**

   Python has been designed in a way, that the programmer does not need to look after the low level details such as, memory management. It supports high level data structures such as, realizable arrays (lists in Python), hash tables (dictionary in Python). As these building blocks are defined in the core language, they are extensively used. This reduces the development time, as well as the size of the code. Thereby, making the code more simpler to read.

6. **Supports Extensibility**

   Python allows the programmer to code some part of the program in different languages like, C, C++, Java and utilize them through Python code.

7. **Support Portability**

   Since Python is written in C, it can work on different platforms i.e., the Python applications built on one system, can run on any other system, with little or no modification. Any platform that contains ANSI C compiler, supports Python. Python is also independent of the type of architecture.

8. **Robust**

   Python is more powerful in determining as well as handling error conditions, through software handlers. It guides the programmer, by providing full details of errors, in case of program crashing due to errors. For example, the location where the error has occurred in the code, by specifying the filename, line number and function. It also allows the programmer to take an action for the elimination of the error.

   The exception handlers in Python are capable of reducing debugging process, by simplifying the problem, redirecting the program flow, performing cleanup or maintenance measure, closing the application elegantly or just ignoring it. In case, if the errors could not be handled properly, Python generates a stack that displays the type and location of the error, in addition to the location of module that contains the erroneous code.

9. **Possesses the Rights of a Memory Manager**

   One of the main advantages of Python programming language is that, it itself handles all the functionalities related to memory management. Thereby, reducing the programmers burden. This in turn reduces the overall development time and includes fewer bugs. It handles memory management through the use of interpreter.

10. Interpreted and (Byte) Compiled Language

Just like Java, Python is also an interpreted and a byte-compiled language. That is, it allows the programmer to run the program directly from the source code without performing the compilation process.

The process followed by Python is,

(i) It just transforms the source code into an intermediate form known as, 'byte code'.

(ii) Then translates the byte code into a machine code (0's and 1's)

(iii) And then run it.

The programmer need not have to worry about the program compilation, proper linkage and loading of libraries etc.

11. Supports Scalability

The term 'scalability' here, refers to the capability of the programming language, to accept and handle the additional performance required, upon the addition of new hardware to the system. Unlike other programming languages, Python can expand the code size, by adding new or existing Python elements and reusing the code when required. This is possible because, Python has a pluggable and modular architecture.

12. Works as an Effective Rapid Prototyping Tool

Python provides several interfaces to other systems, which makes it powerful and robust enough to prototype the entire systems within it, in a very less time.

Q12. Explain the syntax of Python language.

Answer :

Python language defines a very simple syntax. Its simplicity lies in design choice. There are several ways to represent a single program but this leads to difficulty in learning and using the language. The philosophy of Pythons states that "There is one and one way to do it".

Since, Python is a non-functional language we mostly use assignment statement. The syntax is as follows,

Syntax

name = expression

Here, the name is binded to the value of the expression.

In other programming language this line will assign the expression value to name.

Thus, python uses assignment operator to bind a name to data value generated by the expression. If a reference value is taken in place of expression then the name will be binded to the existing value.

Unlike, other programming language name is not bound to particular data type and no type checking is needed to bind the name to the value.

Identifiers in Python

Python define its identifier using letters, digits and a underscore. But, the first character of a name should never be a digit.

If several names have to be binded different to the same value, it can be done in single statement as follows,

>>>$S_1 = S_2 = S_3$ = "Hello world"

Here, S_1, S_2 and S_3 are binded to hello world string print.

>>>print S_1

>>>print S_2

>>>print S_3

The output of these three statements is same i.e., Hello world as all three names binded to same string.

Q13. What are the rules and symbols used with regard to statements in Python?

Answer :

Python statements need to follow rules and symbols such as,

1. Comments
2. Module
3. Newline
4. Colon
5. Semicolon
6. Indentation.

1. Comments

Python allows programmers to comment anywhere in the program. Comments are provided using the hash (#) symbol. They are placed in the beginning of the line. The text followed by '#' symbol is ignored by the interpreter. Comments increases the readability of programs and makes the programmer to understand the logic and flow.

2. Module

A module is a python script, it exists physically on the disk in the form of files. A portion of module that becomes very lengthy can be shifted to another module.

3. Newline

Newline breaks a single python statement into multiple lines. The backslash symbol ('\') is used to denote a newline.

Example

```
if(student_age > 17) and \
  (student_GPA > 3.5) :
  admission = 1
```

However, lines can be continued on next line without using '\' in two exceptions.

- One when enclosing operators are used, such as square brackets or parenthesis.
- Another when triple quotes are used for strings.

Example

(i) '''SIA edu group pioneer in ..... today'''

(ii) SIA need (240, more....)

4. Colon

- In python, a set of individual statements that forms a single code block are known as "suites".
- Complex or compound statements such as (if, while, def and class) that need a suite and a header line.
- A header line starts with a keyword and ends with colon (:) and then followed by one or more lines.

Semicolon

5. Multiple statements are allowed in a single line using semicolon (;). However, they can be used only when none of the statements initiate a new code block.

Example

import abc; p = 'Today';

Indentation

6. In python, indentation is used to delimit blocks of code. All lines of a program must be indented the same number of spaces i.e., it needs exact indentation. Indentation is applied with spaces or tab. However, spaces are preferred.

Q14. How variable assignment is done in Python?

Answer :

Python assigns variables through assignment operator and augmented assignment operator.

Model Paper-I, Q2(a)

Assignment Operator

In python, the main assignment operator is equal (=) sign. Unlike other programming languages this operator does not explicitly assign a value to a variable. In this language, objects are referenced.

Augmented Assignment Operator

In python, augmented assignment refers to the use of combination of arithmetic operation and equal (=) sign.

The following example illustrates augmented assignment.

$$i = i + 1$$

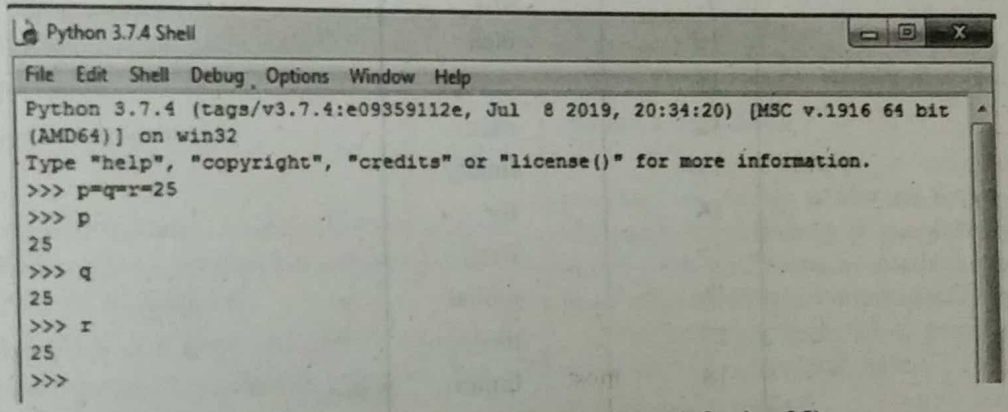
The above statement can also be written as follows,

$$i + = 1$$

Multiple Assignment

In python, multiple assignment refers to the assignment of single object to multiple variables.

Example

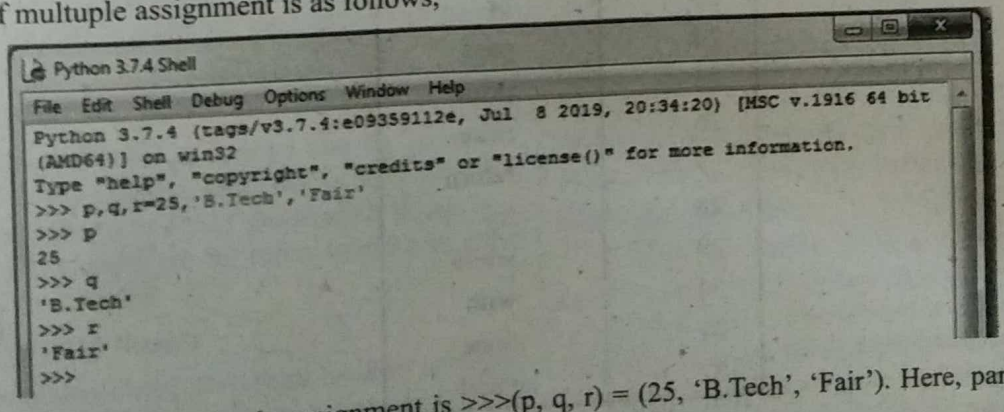


In the above example, p, q and r are assigned to single integer object (of value 25).

Multuple Assignment

In python, multuple assignment refers to the assignment of multuple objects to multiple variables wherein objects on both sides of the assignment operator are tuples.

An example of multuple assignment is as follows,



Another representation of multuple assignment is >>>(p, q, r) = (25, 'B.Tech', 'Fair'). Here, parenthesis are used for representing tuples and they are optional (which makes user easier to read).

Q15. List and explain different identifiers used in Python.

Model Paper-II, Q2(a)

Answer :

Identifiers refers to a set of valid strings that can be used as names in a computer language.

Python considers an identifier to be valid only when

1. The first character of identifier is either underscore (_) or a character.
2. The second character can be any alphanumeric character or underscore.
3. Identifiers cannot be initiated with a number or symbol except underscore(_).
4. Identifiers are case sensitive i.e., man, Man and MAN are not identical.

Keywords

Keywords are reserved words which forms a construct of the language. Use of these keywords for any other purpose results syntax error.

Table depicts keywords used in Python.

S.No.	Keyword
1	and
2	as
3	assert
4	break
5	class
6	continue
7	def
8	del
9	elif
10	else
11	except
12	exec
13	finally
14	for
15	from
16	global
17	if
18	import
19	in
20	is
21	lambda
22	not
23	or
24	pass
25	print
26	raise
27	return
28	try
29	while
30	with
31	none
32	yield

Table: Keywords

Built-ins

Built-ins are special names identifiers used by interpreter. These are not keywords but considered as reserved for system. Therefore, programmers should not use them. Built-ins are members of built-in module. Interpreter automatically imports the built-in module before the program beings or user enter the >>> prompt in the interpreter. They can also be used as global variables that can be accessed anywhere in programs.

Special Underscore Identifiers

In Python, special underscore variables are also used. These variables are both prefixed and suffixed. Some of these underscore identifiers are as follows,

_xxx

It requests private name that integrates within classes.

_xxx

It states that do not import with 'from module import *'.

xxx

It is a system defined name.

Q16. Write the basic style guidelines.

Answer :

Basic style guidelines for python are as follows,

1. Comments
2. Indentation
3. Documentation
4. Selecting identifier name.

1. Comments

Comments should be used in programs to provide information about the function a program is going to perform, the logic used and the process adopted.

Comments should be clear, short and convey meaningful messages. They increase program readability and save time of others who use the programs.

2. Indentation

The spaces left while starting a new paragraph is called indentation. Four to six spaces are ideal for indentation while less than three are bad and more than seven spaces are wastage of resources. Tabs can also be used for indentation but spaces should be preferred.

3. Documentation

In python, documentation strings can be retrieved dynamically using a variable named _doc_. The attribute obj_obj._doc_ is used to access first unassigned string in a class or function declaration. Here obj refers to module, class or function name.

4. Selecting Identifier Names

Selection of names of identifiers must be short in length and meaningful. Modern computing storage does not hinder length of the name however, short names are preferred.

Q17. Discuss how memory management is done in python.

Answer :

Variable Declaration

The compiled languages need the variables to be declared before they are used. Among others C is a restrictive language that in which the variables must be declared at the beginning of the code. Other languages such as C++, Java etc., allow "on the fly" declarations. But they still need name and type declarations.

In python, no explicit variable declarations are needed. They get declared upon variable assignments until then they cannot be accessed or used.

```
>>> x
    Traceback (innermost last):
      File <stdin> line1, in?
    Name error: x
```

The variable can be used by its name once it is assigned

```
>>> x = 2
>>> x
2
```

Dynamic Typing

Unlike other languages, the type and memory space for an object in Python are determined and allocated at run time. Python is byte compiled as well as interpreted language. The type of the object is dictated by the value that is at right side in assignment statement, Once the object is created a reference of it is assigned to the variable that is at the left side of the assignment statement.

Memory Allocation

Memory allocation of variables involves borrowing of system resources which are returned back to the system. The application writing is simplified by python because the complexities of memory management is pushed down to interpreter. Therefore, Python can be used to solve the problems without worrying about lower level problems which are not much related to the solution.

Reference Counting

Reference counting in Python is used to keep track of objects in the memory. In internal variable reference counter will keep track of all the references made to every object. This is also called as refcount.

A reference will be made to an object upon its creation and the reference count will be 1. When this object is not needed, the refcount becomes zero and it is garbage collected. The refcount increments when aliases are created for this object. The refcount decrements when aliases are deleted or when the variable is reassigned with some other value.

Garbage Collection

Garbage collection is a mechanism that is used for reclaiming the memory that is not used any more. It checks for objects that have reference count as zero and then deallocates the memory of it. The objects that have reference count more than zero and that need to be deallocated. Some of the objects refer to each other. Such situations lead to cycles.

Q18. Write down the steps to create and run python scripts.

Answer : Model Paper-III, Q2(a)

Writing and executing the python scripts is an easy process. Initially type "Python" or "Python 3" at the command prompt to access python. With this even python REPL (Read-Eval-Print-Loop) gets opened. User can enter the commands in it just like at command prompt. To exit from REPL type Ctrl-D. Follow the below steps to write and run a python script.

1. Type sudo nano MyScript.py at command prompt to open nano text editor.
2. Create a new file with name MyScript.py.
3. Type the below code in it

```
#!/user/bin/python
x = 2
y = 6
sum = x + y
print sum
```

4. Save the file with .py extension. Even text editors such as notepad or notepad ++ to write the code and press ctrl + x to exit.
5. Without making the file executable it can be run by navigating the location of the script and then by typing MyScript.py.
6. MyScript.py can be made executable by typing chmod +x MyScript.py at command prompt.
7. To run the script type

   ./MyScript.py

   at command prompt.

   The below result will be displayed finally.
8.

1.2 OBJECTS

1.2.1 Python Objects, Standard Types, Other Built-in-types, Internal Types

Q19. Discuss about Python objects.

Answer : Model Paper-II, Q2(b)

In python, object model abstraction is used for the purpose of data storage. It treats any construct that holds a value as an object. All objects possess three common features.

1. Identifier

An identifier uniquely identifies an object. It is determined using id( ) built-in function.

2. Type

Type shows the kind of value an object can store, kind of operations that can be performed on the object and behaviour of object. It returns an object.

3. Value

It denotes data item which is represented by an object.

These three attributes are assigned to objects during the creation of objects and remain with the objects until they are deallocated.

Among these objects identifier and type attributes are read only while value is write only. An object supporting update operation is called mutability and such objects value can be modified otherwise it is treated as read only.

Q20. List and explain the standard types and built-in types.

Answer :

Standard Types

The standard types are also known as primitive data types. Different types of standard types includes.

1. Number

For answer refer Unit-I, Q25, Topics: Numbers, Creating Number.

2. Plain Integer

For answer refer Unit-I, Q25, Topic: Plain Integers.

3. Long Integer

For answer refer Unit-I, Q25, Topic: Long Integer.

4. Floating Point Real Number

For answer refer Unit-I, Q25, Topic: Floating Point Real Numbers.

5. Complex Number

For answer refer Unit-I, Q25, Topic: Complex Number.

6. String

Strings are the most popular type in python. They are created by enclosing characters in single or double quotes. These characters are assigned to a string variable.

Example

var = 'Extreme program'

7. List

List is like a string, which provides sequential storage mechanism. Through slices, we can access any number of consecutive elements of list.

8. Tuple

Tuples are containers, that are used to store data items. They are similar to lists because they allow a list of elements to store. However, there are few differences between lists and tuples. Tuples use parentheses whereas, lists use square brackets. Tuples are immutable (i.e., they cannot be modified), whereas, lists are not. Due to the property of immutability, tuples can be used as a dictionary-key and also while dealing with object groups.

9. **Dictionary**

Dictionaries refer to python's mapping type. They are similar to associative arrays, or hashes in perl. The syntax of a dictionary entry is key : value. Keys correspond to any python type and it is usually numbers or strings. Whereas, *values* correspond to any arbitrary python object. Dictionaries are enclosed by curly braces ({ }).

Built-in Types

The built-in types include,

1. **Type**

For answer refer Unit-I, Q19, Topic: Type.

2. **None**

It is a special type in python known as Null object or null type. It has only a single value that is none and the type of it would be nonetype.

3. **Function**

Functions in python are similar, to that of other programming languages. They organize program logic in a structural or procedural programming fashion. They store the frequently separated code, in order to be called whenever required and save memory space, from storing multiple copies of the same data.

4. **Module**

A Module stores pieces of python code, which can be shared by other python programs. It is a mechanism, that is used to organize the code logically. Lengthy codes can be divided into several independent modules, that can interact with each other.

The code present in a module, may be a single class with its methods and data variables or a group of related classes and functions. A module can also share or use the code present in some other module. This is done, by importing a module. Hence, modules provide code reusability.

5. **Class**

A class contains all the related data members and logic as class instances. It is defined using the keyword "class", and a user defined name for it.

Example

```
class manclass
    static member declarations.
    method declarations.
```

6. **Class Instance**

An instance of a class is an individual object of a class. They are created through init( ) method.

Example

```
fool = FooClass( )
```

Q21. What are the various internal types in Python.

Answer :

In Python there are six different internal types, they are as follows,

1. **Code Objects**

The code objects are executable Python source code pieces that can be byte-compiled as return values that call compile () BIF. Such objects can be executed by exec () or eval () BIF. Although, does not hold any data related to execution environment, they are considered as heart of all the user defined functions. Not only code objects, the function attributes also contain administrative support needed by a function such as name, default arguments, global namespace and documentation string.

2. **Frame Objects**

In Python, the frame objects represent the execution stack frames. They hold the data required by the interpreter about runtime execution environment. The attributes of it are link to previous stack frame, code object that is to be executed, dictionaries for local and global namespaces and current instruction. Every function call results in new frame object for each of which a C stack frame is created. There is possibility in trace back object to access the frame object.

3. **Traceback Objects**

An exception is raised whenever an error occurs in Python. If they are not caught or handled the interpreter exits showing some information such as the following,

```
Traceback(innermost last):
    File "<stdin>", line N?, in ???
    ErrorName: error reason
```

The traceback object is created when an exception occurs and it contains the information about stack trace for an exception. The handler of an exception can only access the traceback object.

4. **Slice Objects**

Slice objects are created with the Python extended slice syntax. This enables for different types of indexing such as multi dimensional, stride indexing and indexing through ellipsis. The slice objects can also be created using slice() BIF.

5. **Ellipsis Objects**

The ellipsis objects are used in extended slice notations for depicting the actual ellipses in slice syntax. The ellipsis objects contain a single name ellipsis and a Boolean True Value.

6. **X Range Objects**

The XRange objects are created by using the BIF range() which is sibling of range() BIF. They are used upon creation of large data sets and limited memory.

1.2.2 Standard Type Operators, Standard Type Built-in Functions

Q22. List and explain various standard type operators of objects in Python.

Answer :

Various standard type operators in Python are as follows,

1. Object Value Comparison

The comparison operators check the few data values among members of same type for equality. All the built in types support these operators. These operators generate either true or false as result. Various types of comparison operators are as follows,

(i) less than

It checks whether expr1 is less than expr2.

expr1 < expr2

```
>>> 'pqr'<'abc'
False
>>> 8<20
True
>>>
```

(ii) greater than

It checks whether expr1 is greater than expr2.

expr1 > expr2

```
>>> 'pqr'>'abc'
True
>>> 8>20
False
>>>
```

(iii) less than or equal to

It checks whether expr1 is less than or equal to expr2.

```
>>> 24<=5
False
```

(iv) greater than or equal to

It checks whether expr1 is greater than or equal to expr2.

```
>>> 90>=16
True
>>>
```

It checks whether expr1 is equal to expr2. It returns true if they are equal, other wise false is returned.

```
>>> 38==15
False
>>> 'abc'=='xyz'
False
>>>
```

(vi) **not equal to (C-style)**

It checks whether expr1 is not equal to expr2. It returns true if they are not equal, otherwise a false is returned.

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 38!=15
True
```

(vii) **not equal to (pascal style)**

It checks whether expr1 is not equal to expr2 in pascal style.

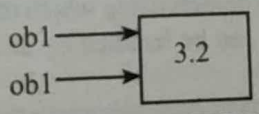
```
>>> 'abc' <> 'xyz'
```

2. **Object Identity Comparison**

Python provides support to directly compare the objects by themselves. They are allowed to be assigned to other variables. Modifications done to any object will effect all of its references because all these variables point to same data object. For example, the variables ob1 and ob2 reference to the same object.

ob1 = ob2 = 3.2

In the above example, the value 3.2 is assigned to both ob1 and ob2. For this initially a numeric object is created internally and the reference of it is assigned to ob1 and ob2. The fact is that both ob1 and ob2 are aliased to same object.



Q23. Discuss about standard type built in functions.

Answer :

Python provides various types of standard type built in functions such as the following,

1. **type()**

type() is a BIF in Python and it became a "factory function" from then onwards. The syntax of type() is as follows,

type(object)

It accepts object as parameter and returns the type of it. This value is a type object.

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> type(6)
<class 'int'>
>>> type('Good morning!')
<class 'str'>
>>> type(type(2))
<class 'type'>
>>>
```

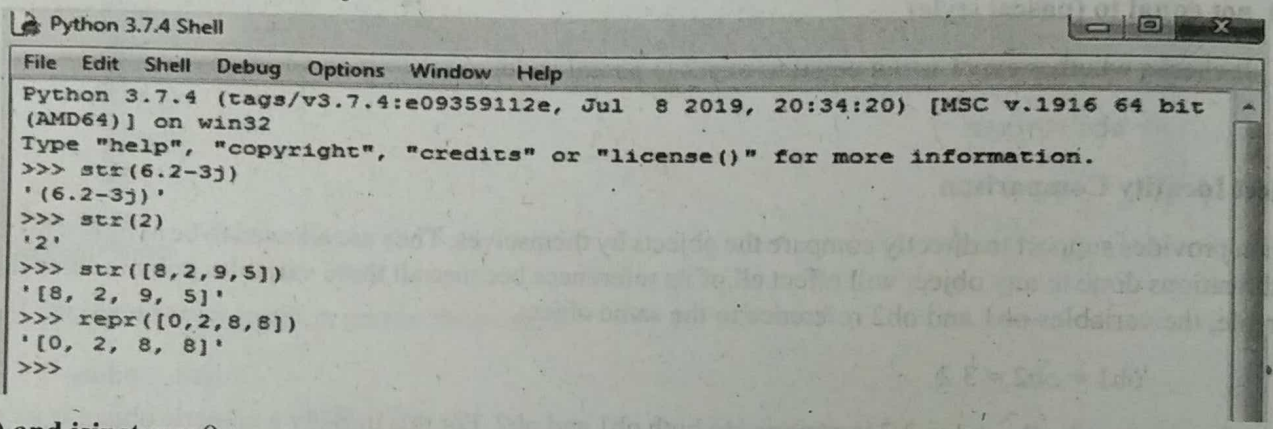
2. **cmp ()**

For example the cmp() function is used to compare two objects called **ob1 and ob2.** It returns negative value when ob1 is less than ob2, or it returns a positive value when ob1 is greater than **ob2. When** both the values are equal, a zero is returned. Syntax of cmp() is as follows,

```
cmp(ob1, ob2)
>>> x,y = –2, 18
>>> cmp(x,y)
>>> y = –5
>>> cmp(x,y)
>>> x,y = 'pqr' , 'ijk'
>>> cmp(x,y)
>>> (y,x)
```

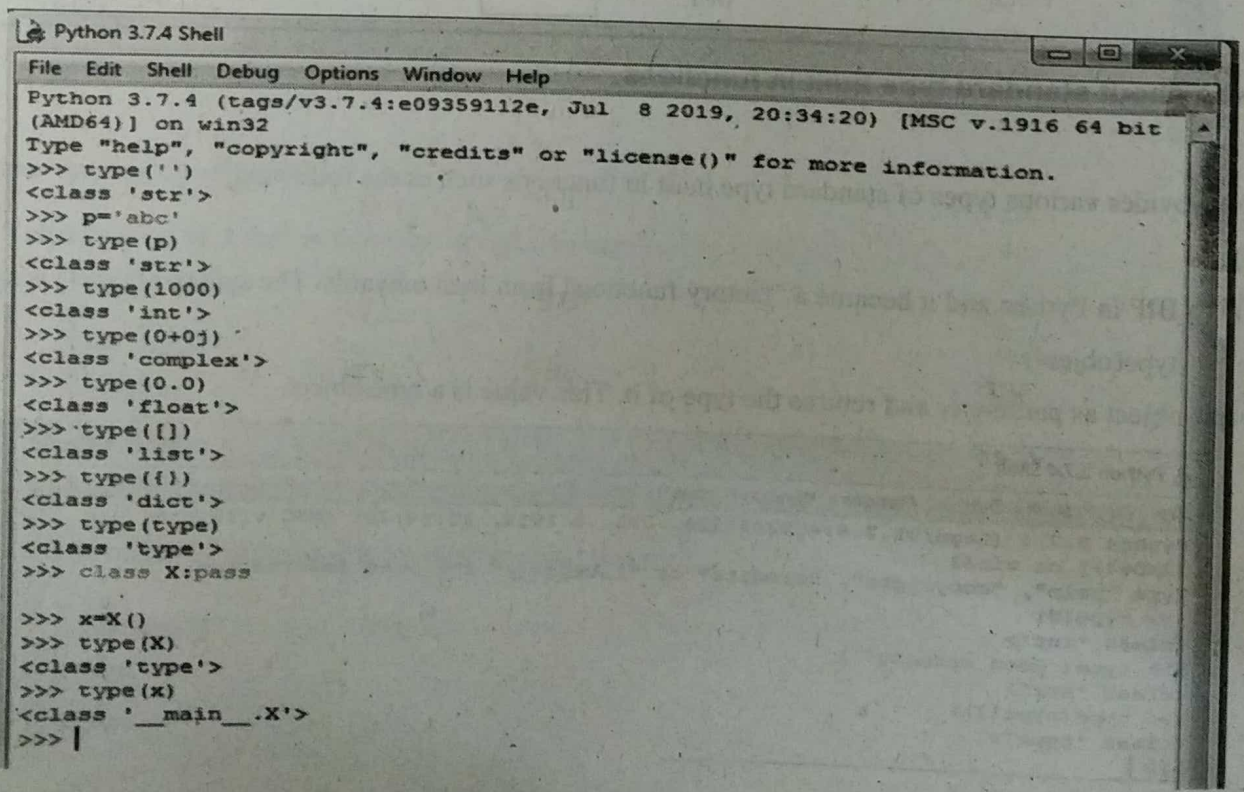
3. **str() and repr() (and "operator)**

The function str() represent STRing and function repr() REPResentation BIFs. Including these, the single back and reverse quote operator(") are very much useful when there is need to recreate an object through evaluation or to get human readable view of objects contents, data values, object types etc. These operations need a python object as argument and result would be string representation of this object.



4. **type() and isinstance()**

Users are responsible for the introspection of objects using which the functions are called because python does not provide support for function overloading. But this situation can be handled by using type() function. The purpose of this function is to return the type of object in python.



1.2.3 Categorizing the Standard Types, Unsupported Types

Q24. List out various standard types and unsupported types.

Answer :

Categorizing the Standard Types

Standard types are categorized based on the data types,

Datatype	Storage model	Update model	Access model
Numbers	**Scalar** The type holds single literal object	**Immutable** The values of objects cannot be changed	**Direct** They are non container type with single element
Strings	**Scalar** The type holds single literal object	**Immutable** The values of objects cannot be changed	**Sequences** They can be sequentially accessed using index values.
Lists	**Container** The type holds multiple objects.	**Mutable** The values of objects can be changed.	**Sequences** They can be sequentially accessed using index values.
Tuples	**Container** The type holds multiple objects.	**Immutable** The values of objects cannot be changed.	**Sequences** They can be sequentially accessed using index values.
Dictionaries	**Container** The type holds multiple objects.	**Mutable** The values of objects can be changed.	**Mapping** They are hashed key-value pairs. The elements are unordered and accessed with keys.

Unsupported Types

Python does not support standard types like char or byte, pointer, int versus short versus long, and float versus double.

Char or Byte

Python does not supports char or byte type for holding the single character or 8 bit integers.

Pointer

Python can directly manage the memory for the users. Therefore pointer addresses need not be accessed. This is some what possible by referring objects identity through id().

int versus short versus long

The plain integers in Python are 'standard' integer type that fulfill the need of int, short and long.

Float Versus Double

Float in Python is similar to double in C. It provides decimal floating point type available in decimal module. Floats are used for estimates such as lengths, weights and measurements.

1.3 NUMBERS

1.3.1 Introduction to Numbers, Integers, Floating Point Real Numbers, Complex Numbers

Q25. Discuss different types of numbers in python.

Model Paper-I, Q2(b)

Answer :

Numbers

Numbers are of immutable type i.e., when the value of a number is changed a new object is allocated they also allows scalar storage and direct access.

Creating Number

A number is created by assigning a value to a variable.

Syntax

var = value

Example

aFloat = 2.3497

Updating a Number

Existing number can be updated by reassigning a variable to another number. The new value can be a different number or related to previous value.

Example

aComplex = aComplex + 2

Deleting a Number

Generally a number can not be deleted but, its usage can be stopped. However, it can be explicitly deleted by using del statement once the variable name is removed, it is no longer used. In case, if it is used then it results an "NameError" exception.

Example

del aComplex

Number Types

Python supports the following number types,

1. Plain integer
2. Long integer
3. Floating point real numbers
4. Complex numbers.

1. Plain Integers

In Python, plain integers correspond to universal numeric type. The range of integers is –2, 147, 483, 698 to 2, 147, 483, 647 i.e., -2^{31} to $2^{31} - 1$.

Example

89, – 169, – 0 × 43, 017, 0101

2. Long Integer

Long integers are represented as (L) or (l). They are superset of integers and are used when the range of number exceed -2^{31} range.

Those can be expressed in octal, decimal or hexadecimal notations.

Example

312L, – 249L

3. Floating Point Real Numbers

Floating point real numbers are represented in straight forward decimal or scientific notations. In decimal notation, decimal point is used while in scientific notation, 'E' or 'e' is used.

Positive or negative signs between the "e" and the exponent represents the sign of the exponent. If sign is not present, then it refers positive exponent.

Example

1.28, $-2.204e^{-19}$

4. Complex Number

A complex number is an ordered pair of floating po real number and imaginary number.

Syntax

real + imag j

Here j = imaginary number

Example

24.37j, 22 + 3.4j

1.3.2 Operators, Built-in Functions

Q26. Describe different types of operators supporte by Python with an example for each.

Answer : Model Paper-III, Q2(

There are three different types of operators supporte by python. They are listed below,

1. Arithmetic or numeric operators
2. Comparison operators
3. Boolean operators.

1. Arithmetic or Numeric Operators

The numeric type arithmetic operators that are supporte by python are listed below,

(i) Division (/, //)
(ii) Modulus (%)
(iii) Exponentiation (**)
(iv) Addition (binary (+) operator)
(v) Subtraction (binary (–) operator)
(vi) No change (Unary(+) operator)
(vii) Negation (Unary (–) operator).

(i) Division(/, //)

There are three different types of division operations,

(a) Classic division
(b) True division (/)
(c) Floor division (//).

(a) Classic Division

In the Classic Division Operation, the quotient or result obtained will have the same type as that of the operand involved. That is, if integer type operands are involved, then the classic division operation returns an integer type by truncating the fraction. Similarly, for floating point operands, it returns the actual floating point quotient.

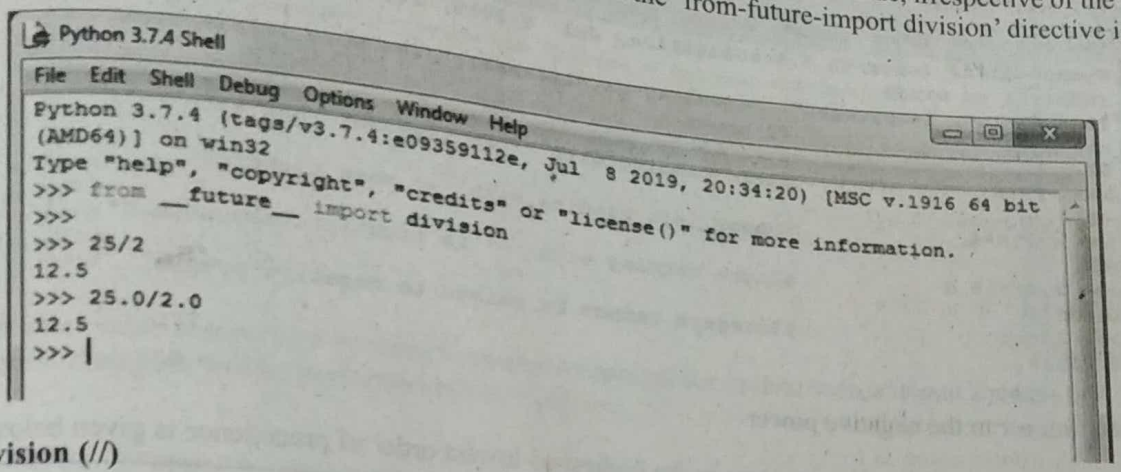
Example

```
>>> 25/2
12
>>> 25.0/2.0
12.5
```

True Division (/)

In the True Division Operation, the quotient or result obtained will be the actual value, irrespective of the operand types. This type of division operation can be performed, only when the 'from-future-import division' directive is defined.

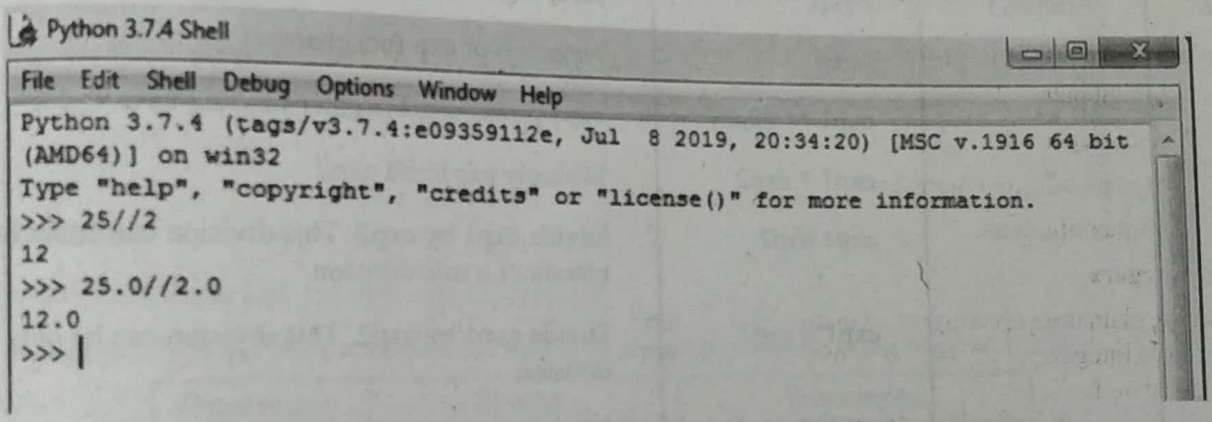
Example



(c) **Floor Division (//)**

In the Floor Division Operation, the fractions obtained in the quotient are always truncated and rounded to the smallest whole number which is on its immediate left on the number line.

Example



(ii) **Modulus(%)**

The computation of modulus is different for different numeric types.

- The modulo of two integer numbers is the remainder of their integer division.

Example

$17 \% 3 = 2$

- The modulo of two floating point numbers is obtained using the following formula.

$$\left[\text{dividend} - \left(\text{math.floor}\left(\frac{\text{dividend}}{\text{divisor}}\right) * \text{divisor}\right)\right]$$

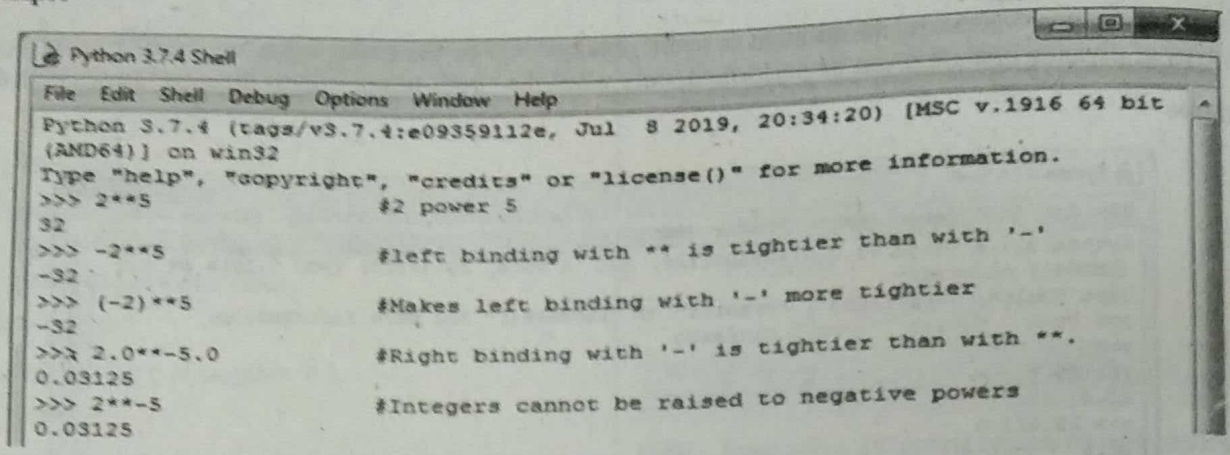
- The modulo of two complex numbers is obtained using the following formula.

$$\left[\text{dividend} - \left(\text{math.floor}\left(\left(\frac{\text{dividend}}{\text{divisor}}\right)\text{real} * \text{divisor}\right)\right)\right]$$

(iii) **Exponentiation (**)**

The Exponentiation Operation, determines the power of a numeric type. The binding between the exponentiation operator and its left operand is more tighter than that with unary operators (+, –). Whereas, the binding between the exponentiation operator and its right operand is less tighter than that with unary operators (+, –).

Example



ValueError:integer to the negative power

The list of all numeric arithmetic operators in the highest-to-lowest order of precedence is given below.

S.No.	Operator	Syntax	Function
1.	**	exp1 ** exp2	exp1 power exp2 (tighter binding to its left)
2	+ (unary)	+exp	Keep sign of exp, unchanged
3.	– (unary)	–exp	Negation of exp (or) change sign of exp to '–'
4.	**	exp1 ** exp2	exp1 power exp2 (tighter binding to its right)
5.	*	exp1 * exp2	Multiply exp1 and exp2
6.	/	exp1/exp2	Divide exp1 by exp2. This division can either be a classic or a true division
7.	//	exp1 // exp2	Divide exp1 by exp2. This division can be only a floor division
8.	%	exp1 % exp2	Module of exp1 divided by exp2
9.	+(binary)	exp1 + exp2	Add exp1 and exp2
10.	–(binary)	exp1 – exp2	Subtract exp1 and exp2

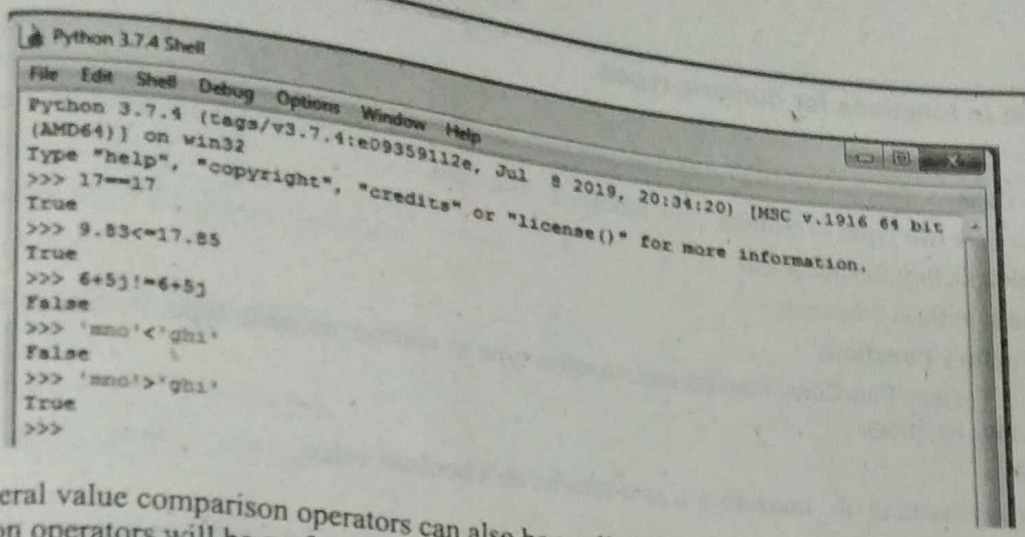
2. Comparison Operators

These operators compare only the data values of the participating objects. The comparisons performed are based on the sign and magnitude for numeric values, whereas a lexicography (i.e.,) alphabetical order is followed, in writing dictionaries for strings. The comparisons result in either Boolean True or False values.

The list of standard type value comparison operators in highest-to-lowest precedence order is given below,

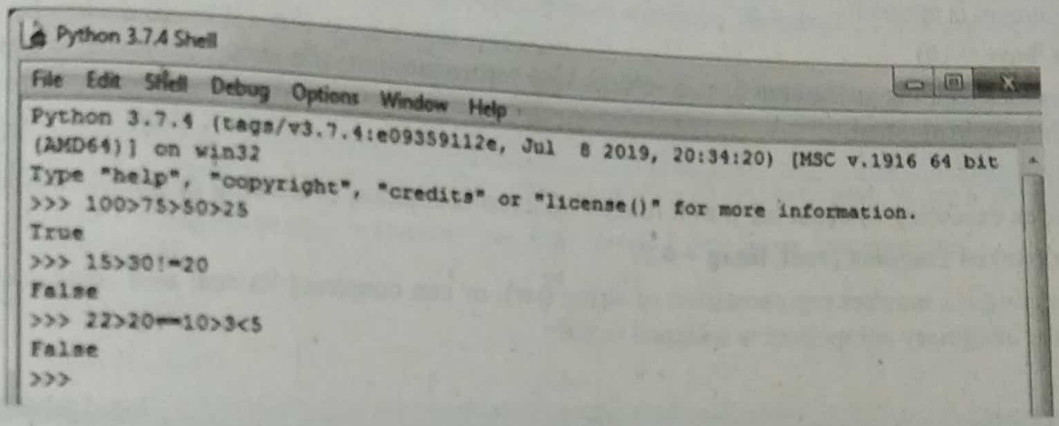
Operator	Syntax	Determines whether
<	exp1 < exp2	exp1 is less than exp2
>	exp1 > exp2	exp1 is greater than exp2
<=	exp1< = exp2	exp1 is less than or equal to exp2
>=	exp1 > = exp2	exp1 is greater than or equal to exp2
= =	exp1 = = exp2	exp1 is equal to exp2
! =	exp1! = exp2	exp1 is not equal to exp2

Examples



In Pascal, several value comparison operators can also be applied for several values within a single line. The evaluation of all such comparison operators will be performed left-to-right.

Example



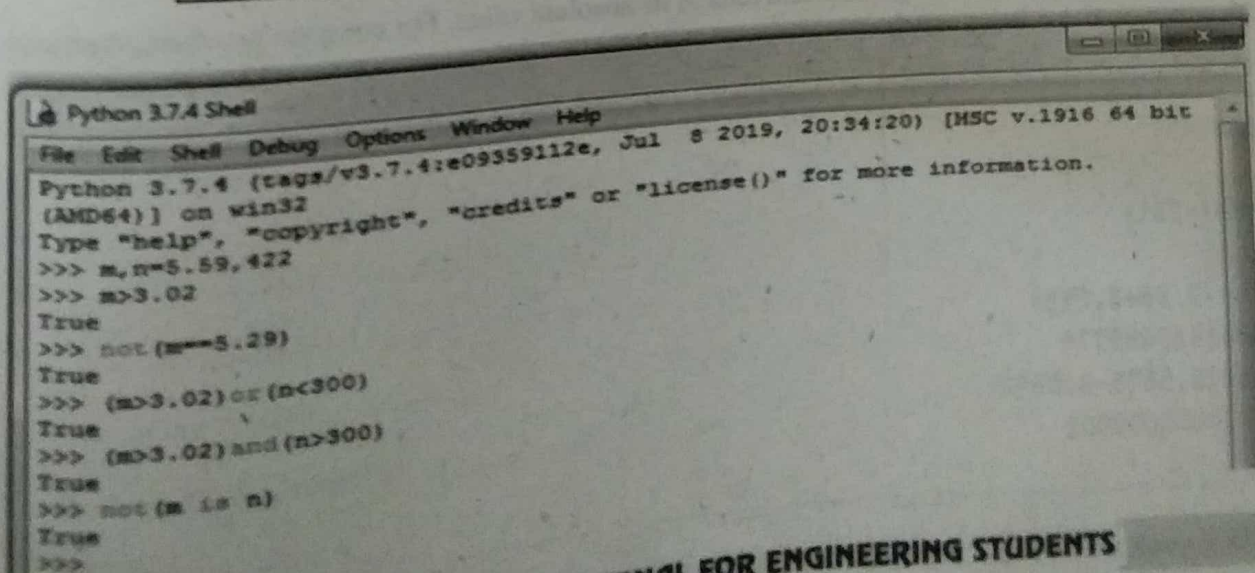
3. Boolean Operator

These operators negate or link two or more expressions.

The list of standard type boolean operators in highest to lowest precedence order is given below:

Operator	Syntax	Determines
not	not exp	negation or logical NOT of exp
and	exp1 and exp2	conjunction or logical AND of exp1 and exp2
or	exp1 or exp2	disjunction or logical OR of exp1 and exp2

Examples



Q27. Explain built-in functions for numeric types. Model Paper-I, Q3(a)

Answer :

Numeric Type Functions

In Python, there are two types of numeric type functions. They are,

(i) Conversion factory functions and

(ii) Operational built-in functions.

(i) Conversion Factory Functions

The Conversion Factory Functions, convert any numeric type to another numeric type. Python provides the following conversion factory functions:

1. **bool (ob)**

   It is same as executing ob._nonzero_( ), as it returns ob's boolean value.

2. **int(ob, base = 10)**

   It is same as executing string.atoi( ), as it returns integer representation of a string or numeric type object, '*ob*'. The **base** argument is optional.

3. **long(ob, base = 10)**

   It is same as executing string.atol( ), as it returns long representation of a string or numeric type object, '*ob*'. The **base** argument is optional.

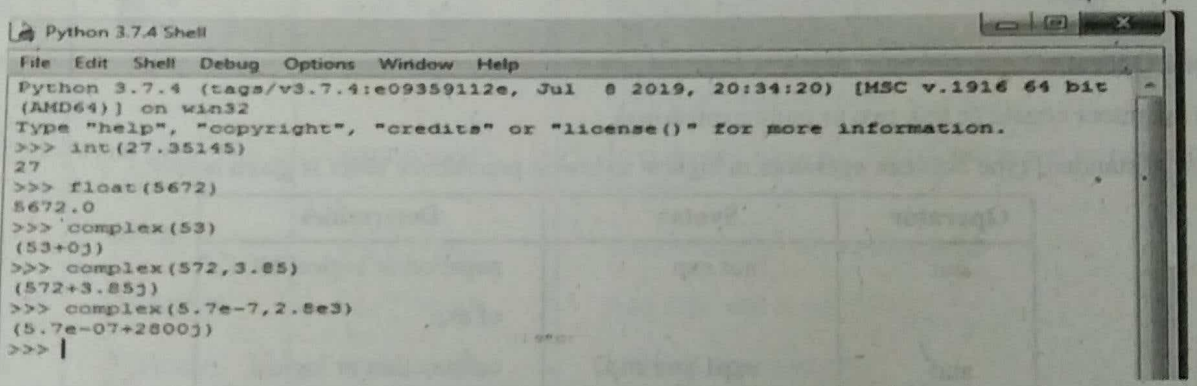
4. **float(ob)**

   It is same as executing string.at of( ), as it returns float representation of a string or numeric type object, '*ob*'.

5. **complex (str) or complex (real, imag = 0.0)**

   It returns complex number representation of string (str), or can construct its real and imaginary components. By default, the imaginary component is assigned to 0.0.

Examples



(ii) Operational Built-in Functions

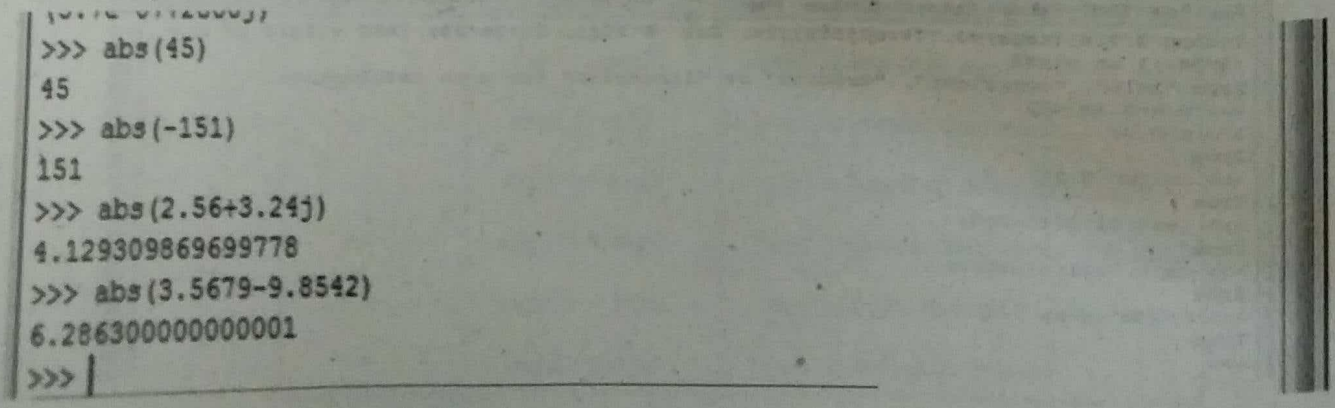
The operational built-in functions perform certain operations on the numeric types.

Python provides the following operational built-in functions:

1. **abs(n)**

   It takes a numeric type as an argument and returns its absolute value. For complex numbers, it returns the value of math.sqrt(n.real^2 + n.imaginary^2)

   Example



2. **coerce(n1, n2)**

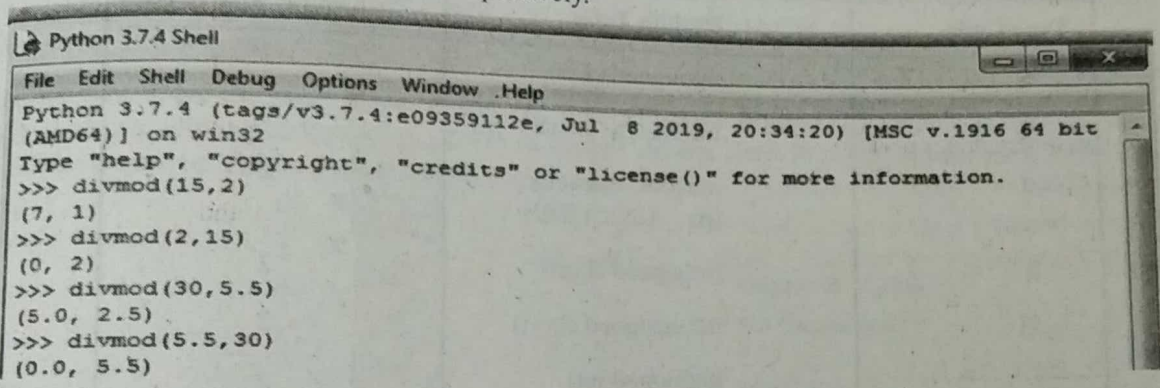
It takes two numeric type arguments and checks, whether both have the same type. If yes, it returns back the same tuple without performing any conversion. Otherwise, it converts one of the types to match the other type and returns the tuple containing the converted pair.

Example

```
>>>coerce(3.45, 5.62) #same type-float
    (3.45, 5.62)
>>>coerce(5,3962L) #convert int to long
    (5L, 3962L)
>>>coerce(3.5692L, 3.67 – 314J)
    (3.5692 + 0J, 3.67 – 314J)
```

3. **divmod(n1, n2)**

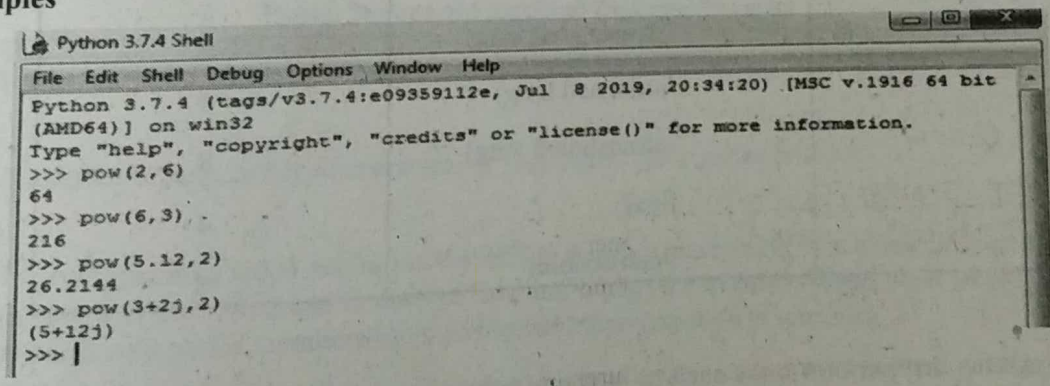
It is a division-modulo function, that takes two arguments $n1$ and $n2$, and returns a tuple containing their quotient and remainder as ($n1/n2$, $n1\%n2$). For floats and complex numbers, the quotients returned are rounded as math.floor ($n1/n2$) and math.floor(($n1/n2$).real), respectively.



4. **pow(n1, n2, mod = 1)**

It performs exponentiation, by raising $n1$ to the power of $n2$. It takes three arguments, $n1$, $n2$ and a mode (optional).

Examples



5. **round(f, todig = 0)**

This function is applicable only to floats. It rounds the given float 'f', to 'todig' digits. The default value of 'todig' argument is '0'. If 'todig' argument is not specified, the float 'f' is rounded to the nearest integral number and is returned as a float.

Example

```
>>> round(5)
5
>>> round(5.856)
6
>>> round(5.1956)
5
>>> round(5.1965893,1)
5.2
>>>
```

1.3.3 Related Modules

Q28. Write about related modules in python.

Answer :

Python provides a number of related modules among which the most commonly used ones are as follows,

These modules are used with numeric types,

1. Decimal

This module consists of decimal floating point class decimal. It supplies a class decimal that contains immutable instances such as decimal numbers, exception classes, and functions and classes that deal with arithmetic context. This arithmetic context determines the above things as precision, rounding and computational anomalies that lead to exceptions.

2. Array Module

The array module provides an object type that can be used to represent an array. It is a set of type of values same as Python list. But type is specified at the time of creation. Various type codes are as follows,

Type Code	Python Type	Minimum Size (bytes)
b	int(signed char)	1
B	int(unsigned char)	1
µ	unicode character; (py _ UNICODE)	2
h	int(signed short)	2
H	int(unsigned short)	2
i	int(signed int)	2
I	int(signed long)	4
l	int(signed long)	4
L	int(unsigned long)	4
q	int(signed long)	8
Q	int(unsigned long)	8
f	float	4
d	float(double)	8

3. Math/Cmath

The Python programs that perform tasks such as studying periodic motion or simulating electric circuits need certain trigonometric functions and complex numbers to be included. These functions cannot be added directly, instead they can be accessed by using mathematical modules called math and Cmath. The math module allows to access hyperbolic, trigonometric and logarithmic functions for real numbers. And the Cmath module allows to work with complex numbers.

4. Operator

It provides the numeric operators that are available as function calls i.e., operator. sub (m,n) is said to be equal to difference (m - n) for m and n.

5. Random

The random module is meant for random numbers. It has pseudo random number generators that is seeded with current timestamp. Some of the most commonly used functions of random module are randiant(), randrange(), uniform(), random(), and choice().

1.4 SEQUENCES

1.4.1 Strings

Q29. List the sequence types that python supports. Explain how sequences are stored and accessed.

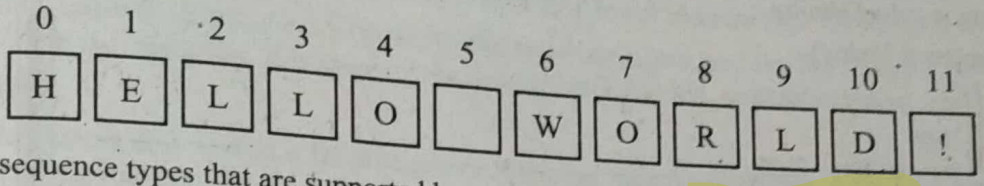
Answer :

Sequence

Model Paper-II, Q3(a)

A 'sequence' is a data structure, that stores elements, which can be accessed using index offsets. For example, the strings are sequences of characters i.e., a string 'Hello world!' contains the sequence of characters, H, e, l, l, o, W, o, r, l, d, !. These sequence elements are stored, starting at index 0 till the index N – 1, where N is the length of the sequence.

i.e.,



There are three sequence types that are supported by python. They are strings, lists and tuples.

Storing and Accessing Sequences

All sequence type share the same access model i.e., ordered set are indexed sequentially to retrieve each element. To retrieve multiple elements, slice operators are used. The sequence number starts from zero(0) and ends with one number less than the length of the sequence. This is depicted in figure.

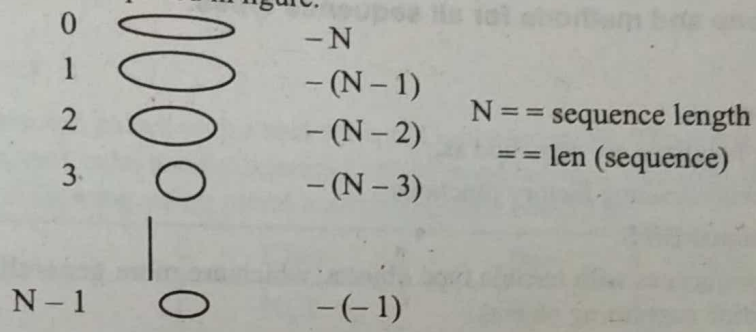


Figure: Storage and Access of Sequence Elements

Q30. List and explain various standard type operators of sequences.

Answer :

Python provides different standard type operators for sequences. They are as follows,

1. **Membership**

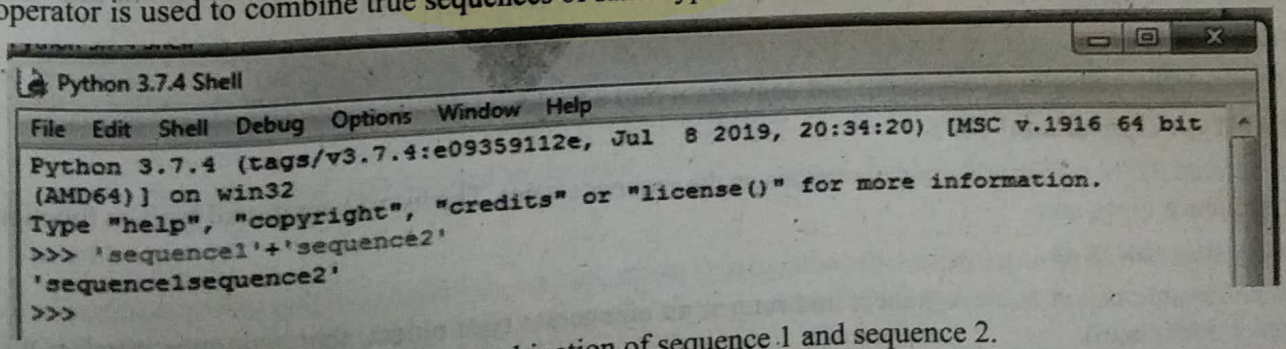
The membership test operators are in and not in that define whether an element is in or not in a member of the sequence. In case of strings the characters are tested and in case of lists and tuples, the object is tested. The nature of these operators is boolean and therefore true is returned if membership is confirmed otherwise false is returned.

object in sequence

object not in sequence

2. **Concatenation (+)**

This operator is used to combine true sequences of same type.



The result of this operator would be the combination of sequence 1 and sequence 2.

3. Repetition (*)

This operator is used when consecutive copies of sequence elements are needed.

```
>>> 'sequence'*3
'sequencesequencesequence'
>>>
```

4. Slices ([ ], [:], [: :])

Sequences are the data structures that contains hold objects in structured format. The elements can be accessed individually through an index and pair of brackets or consecutive group of elements with brackets and colons providing the element indices. This method of accessing is called slicing.

sequence[index]

```
>>> 'sequence'[0:3]
'seq'
>>> 'sequence'[2:8]
'quence'
>>> 'sequence'[5]
'n'
>>>
```

Q31. Explain built-in functions and methods for all sequence types.

Answer :

Sequence Type Built-in Functions

The sequence type built-in functions are classified as,

1. Sequence type conversion/casting factory functions
2. Sequence type operational BIFS.

These functions, combine sequences with terrible type objects, which are more generalized and include datatypes such as sequences, iterators or any iteration supporting objects.

1. Sequence Type Conversion/Casting Factory Functions

The Conversion/casting factory functions, convert or cast any sequence type to another sequence type. These functions or 'converters', are actually factory functions, that do not perform any conversion or casting. Instead, it generates an object of the desired type and copies those contents into it that are to be converted.

Python provides the following sequence type conversion/casting factory functions,

(i) list(iterableObj)

It takes an iterable object as an argument and converts it into a list.

(ii) str(ob)

It takes an object (ob) of any type as argument and converts it to a string.

(iii) unicode (ob)

It takes an object (ob) of any type as argument and converts it to a unicode string through default encoding.

(iv) basestring( )

It is an abstract factory function, that behaves as the parent class of str and unicode factory functions and hence, it cannot be instantiated or called.

(v) tuple(iterableObj)

It takes an iterable object as an argument and converts it into a tuple.

2. Sequence Type Operational BIFS

The operational BIFS, performs certain operations on the sequence types. The list of operational built-in functions supported by python for sequence types are,

(i) enumerate(iterableObj)

It takes an iterable object as an argument and returns an enumerate type object, that generates iterableObj's two tuple elements, (index, item).

It takes the name of a sequence as an argument and returns its length, i.e., the number of elements within the sequence.

max(iterableObj, key = None) or max(arg0, arg1,..., key = None)

It takes an iterable object or a set of arguments and returns the largest element among them. The 'key' argument is used for testing purpose. If it is not 'None', then its value should be a callback that passes the list of arguments to the sort( ) method.

reversed(sequenceName)

It takes a sequence name as an argument and returns an iterator that first points to the last element in the sequence and traverses along, until it reaches the first element. That is, it returns the reversed sequence.

sorted(iterableObj, func = None, key = None, reverse = False)

It takes four arguments, iterable object, function, key and reverse, out of which the later three are all optional and are equivalent as for list.sort( ) built-in method. It returns the sorted list of elements for the iterable object.

sum(sequenceName, init = 0). It takes two arguments the sequence name, and the initial value (optional) and returns the sum of the numbers in the sequence. It is same as the reduce (operator.add, seq,init) method.

zip([it0, it1,...itN])

It takes N iterable objects and returns a list of tuples containing members of each of the N iterable objects i.e., it0[0], it1[0], ...itN[0]), (it0[1], it1[1],...itN[1]),...(it0[N], it1[N]...itN[N]. Where, *n* represents minimum cardinality of all of the iterable objects.

32. Explain in detail about strings.

nswer :

ecial Features of Strings

Special/Control Characters

A special character is a character, paired with a backslash and is non-printable. The special characters can be used through eir ASCII values. The ASCII decimal, octal and hexadecimal values range between (0 to 255), (0000 – 0177) and (0x00 – 0xFF), spectively. Python supports the following string literal backslash escape characters:

\x	Name	Char	Dec	Oct	Hex.
\0	Null character	NUL	0	000	0x00
\a	Bell (BEL)	7	007	0x07	
\b	Backspace (BS)	8	010	0x08	
\t	Horizontal tab	HT	9	011	0x09
\n	Newline/line feed	LF	10	012	0x0A
\v	Vertical Tab	VT	11	013	0x0B
\f	Form Feed (FF)	12	014	0x0C	
\r	Carriage Return	CR	13	015	0x0D
\e	Escape (ESC)	27	033	0x1B	
\"	Double Quote	"	34	042	0x22
\'	Apostrophe/single quote	'	39	047	0x27
\\	Backslash (\)	92	134	0x5C	

The Python strings can have two or more Null characters (ASCII value : 0) within them and they may not necessarily nd, with a NULL character. In Python, a NULL character is considered as any other control character. The control characters re used as delimiters in strings. Because, the use of colon(:) as a delimiter, to identify whether the character is a delimiter or a ata item, may limit the characters that can be used in the data.

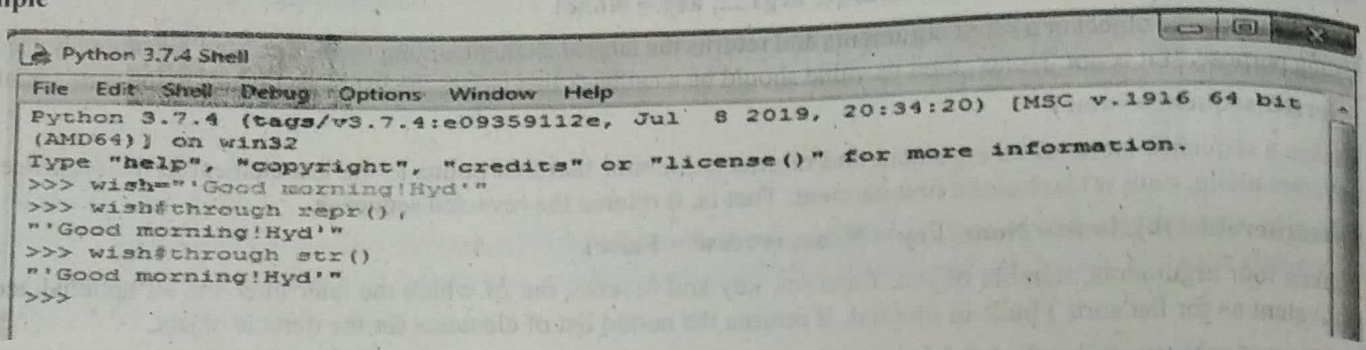
Triple Quotes

The use of single or double quote delimitation, restricts string manipulation that contains special or non-printable characters r example, \n, \t etc. Python overcomes this problem, through 'triple quotes' that allows strings to be provided in multiple line, include special characters such as \n, \t, etc.

Syntax

variable = "'any-string-either-in-one-or-multiple-lines"'

Example



Python's triple quotes are generally helpful while writing a large block of HTML or SQL.

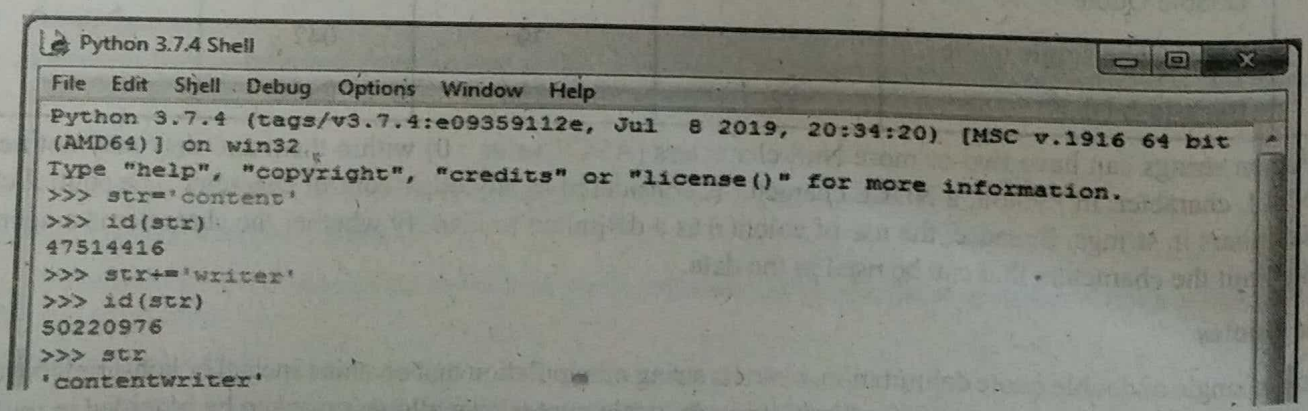
Example

```
#HTML
back = '''
,<HTML> <HEAD> <TITLE>
      Home Page </TITLE> </HEAD>
<BODY> <H3> HI ! USER </H3>
<B> %S </B> <P>
<FORM> <INPUT TYPE = button VALUE = signout
      ONCLICK = "login( )"> </Form>
</BODY> </HTML>
      '''
#SQL
cursor.execute('''
      CREATE TABLE students(
            stud_name VARCHAR(10),
            stud_id INTEGER,
            marks INTEGER)
      ''')
```

3. String Immutability

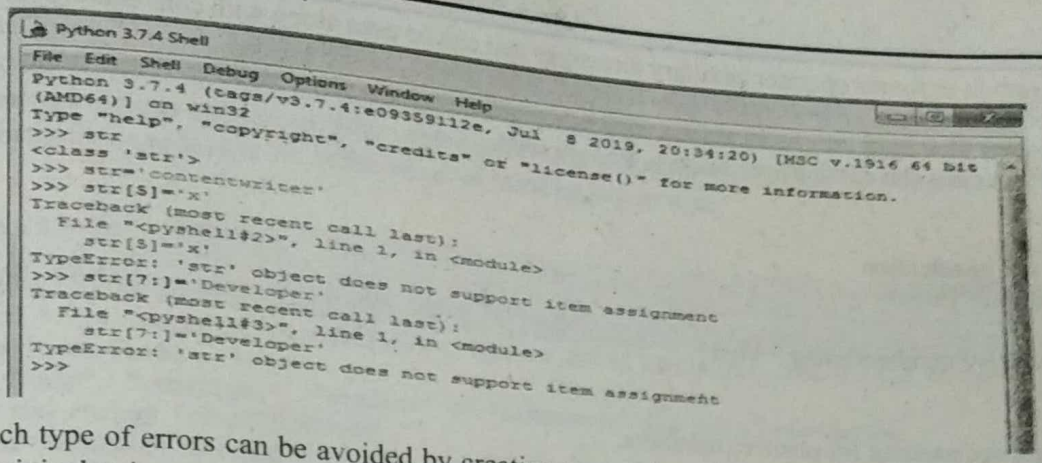
Strings are immutable data types, restricting their values to be changed. That is a new string object will be created, each time a string is modified. This can be checked using the id( ) built-in function that returns 'identity' similar to 'memory address' of an object.

Example



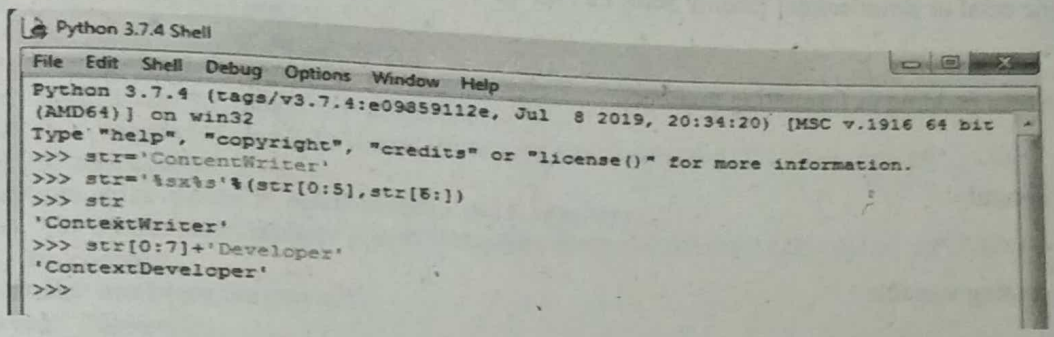
Moreover, individual characters or substrings of a string cannot be modified, doing so will result in an error.

Example



However, such type of errors can be avoided by creating new strings from the existing string and then assigning these strings back to the original string.

Example



As strings are immutable objects, the expressions on the left hand side of the assignment operator (=) cannot have single characters (ex., str[5]) or substrings (ex., str[7:]), as a variable. Instead, it can only have an entire string object (ex., str). However, there are no restrictions for the expressions on the right hand side of the assignment operator (=).

String Operators

The string-only operators are operators that can be applied only to string objects. They are,

1. Format operator (%)
2. String templates
3. Raw string operator (r/R)
4. Unicode string operator (u/U)

1. Format Operator (%)

Python supports the string format operator (%) and all the formatting codes defined for C's printf( ) method.

Syntax

FormatString %(list_of_arguments_to_be_ converted)

Where,

Format string contains % codes, which are the format operator conversion symbols.

The list of format operator conversion symbols supported by Python is given below.

%C : Converts to a character

%r : Converts to a string through repr( )

%s : Converts to a string through str( )

%d or %i : Converts to a signed decimal integer

%u : Converts to an unsigned decimal integer

%o : Converts to an unsigned octal integer

%x or %x : Converts to an unsigned hexadecimal integer represented either in lower case or upper_case letters.

%e or %E: Converts to an exponential notation either in lower case 'e' or upper case 'E'

%f or %F: Converts to a floating point real numbers.

%g or %G: Converts to a shorter form of %e (or %E%) and %f(or %F%)

%%: Converts to an unescaped % character.

The other symbols or format operator auxiliary directives that can be used along with conversion symbols are listed below

(i) *

Its argument specifies the width or precision to be followed.

(ii) –

To perform left justification

(iii) +

To specify positive numbers using + sign

(iv) <sp>

To introduce space padding for positive numbers.

(v) #

To include the octal or hexadecimal leading zero, i.e., 'O' or Ox or OX, respectively.

(vi) 0

To introduce zero padding in formatting numbers.

(vii) %

To use the % literal

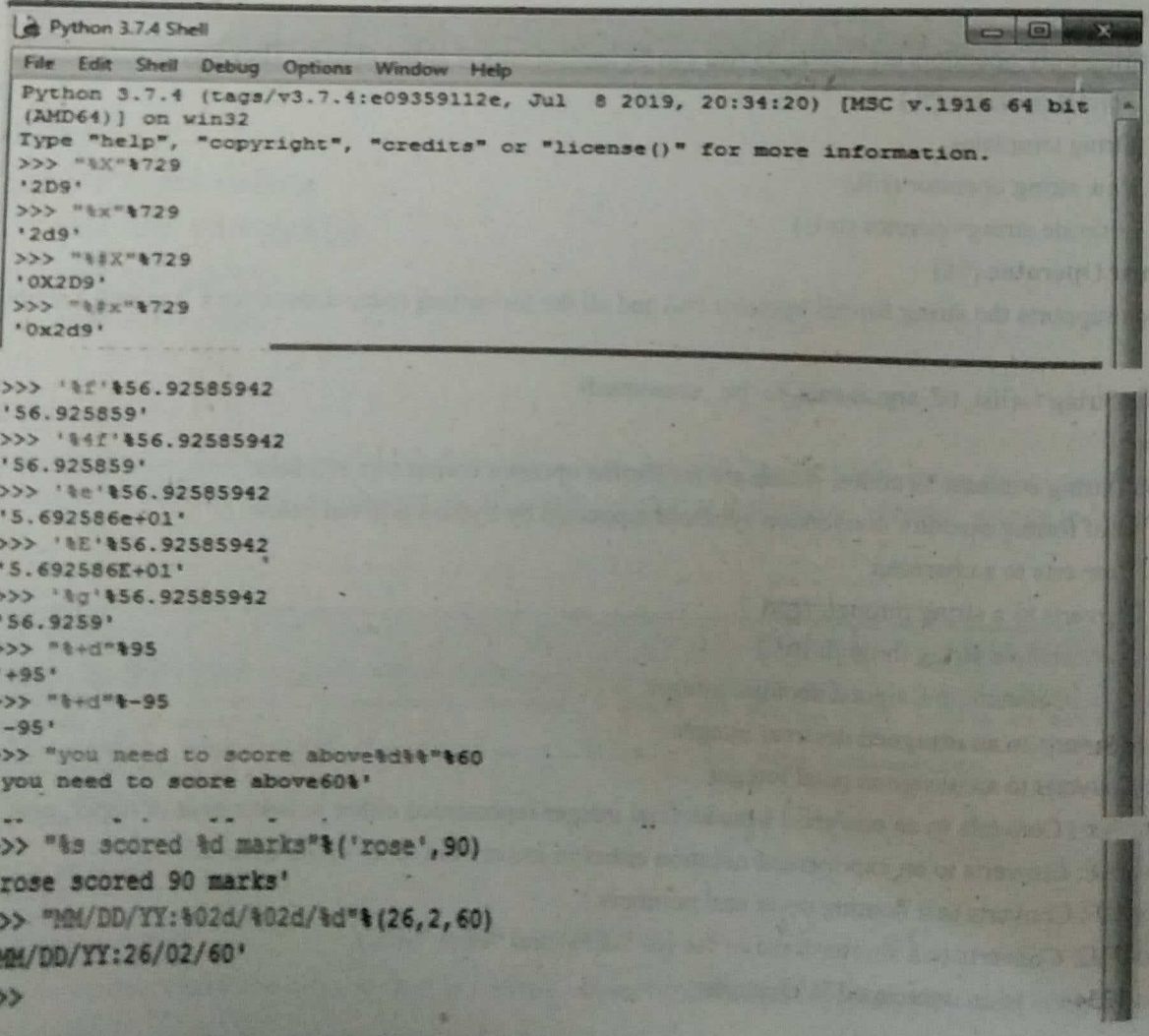
(viii) (var)

To specify mapping variable

(ix) m.n

To specify minimal total width (m) and number of digits (n) following the decimal point.

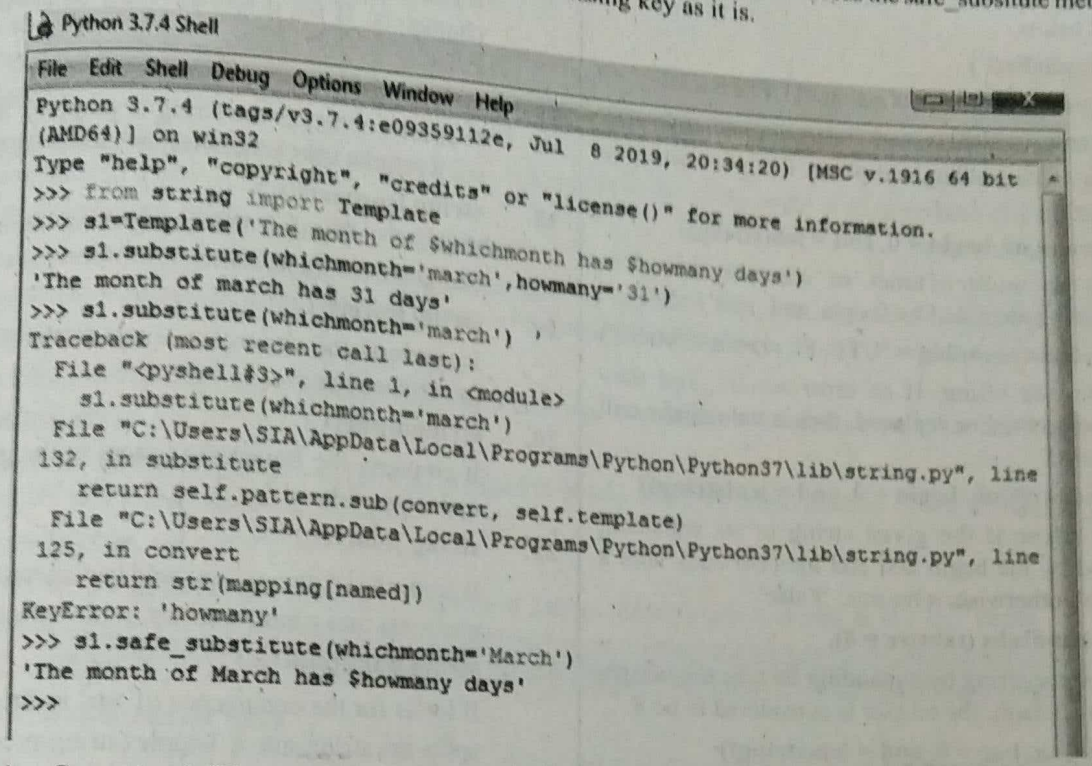
Example



String Templates

Python provides String Templates using which, string substituting is very much simplified. When the **Template** module is imported into the **string** module, the two methods substitute( ) and safe_substitute can be used with template objects. The substitute( ) method, throws keyError exceptions, when missing keys are encountered, whereas the safe_subsitute method performs only the part of the substitution and leaves the substitution of missing key as it is.

Example



3. Raw String Operator (r/R)

The raw string operator 'r' or 'R' is used, to display the raw strings without performing any translation to special or non-printed characters such as \n, \r, \t etc.

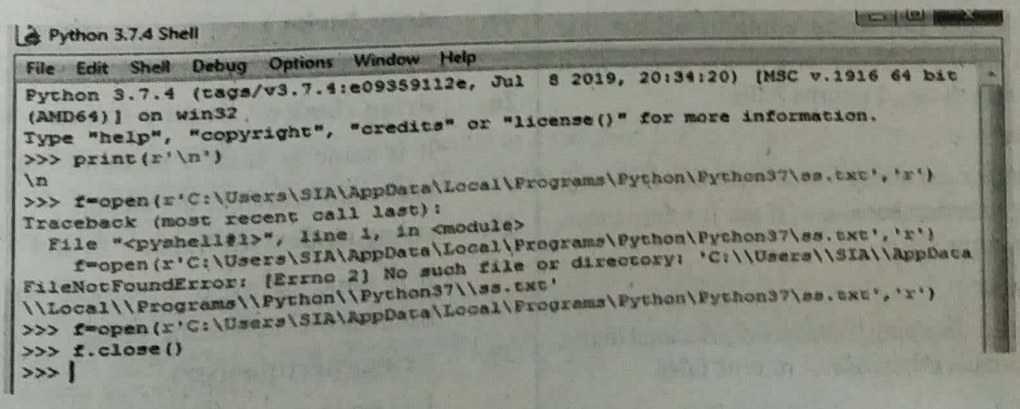
Syntax

r'any_string'

(or)

R'any_string'

Example



4. Unicode String Operator (u/U)

The Unicode String Operator 'u' or 'U' is used, to convert the standard strings or unicode character strings to a complete unicode string object.

Example

u 'abc' U + 0061 U + 0062 U + 0063

u '\u1234' U + 1234

u 'abc\u1234\n' U + 0061 U + 0062 U + 0063 U + 1234 U + 0012.

Q33. Explain built-in functions and methods for strings.

Answer :

Python supports several string type built-in methods. They are listed below,

1. **string.capitalize( )**
   It changes the first letter of string to UPPER case.
2. **string.center(width)**
   It adds a space padding and places the given string in the center.
3. **string.count(str, begin = 0, end = len(string))**
   It returns the number of times 'str' occurs in 'string', or in a substring identified by 'begin' and 'end' indices.
4. **string.decode(encoding = 'UTF–8', errors = 'strict')**
   It decodes the string. If an error occurs, and they cannot be ignored or replaced, then a valueError will be generated.
6. **string.endswith(ob, begin = 0, end = len(string))**
   It returns True if the given string or its substring (identified by the begin and end indices) ends with a string 'ob', otherwise, it returns, 'False'.
7. **string.expandtabs (tabsize = 8)**
   It modifies the string by expanding its tabs to multiple spaces. By default, the tabsize is considered to be 8.
8. **string.find(str, beg = 0, end = len(string))**
   It checks whether the string 'str' occurs in the given string or substring (identified by the begin and end indices). If successful, it returns the index, otherwise, it returns –1.
9. **string.index(str, beg = 0, end = len(string))**
   It is same as find( ) method, the only difference is that, if not successful, it raises an exception instead of returning –1.
10. **string.isalnum( )**
    It checks, whether the string contains atleast one character and are all alphanumeric characters. If so, it returns True; otherwise, it returns False.
11. **string.isalpha( )**
    It checks, whether the string has at least one character and are all alphabetic characters. If so, it returns true, otherwise, it returns false.
12. **string.isdecimal( )**
    It checks, whether the string contains only decimal digits. If so, it returns true, otherwise, it returns false.
13. **string.isdigit( )**
    It checks, whether the string contains only digits. If so, it returns true, otherwise, it returns false.
14. **string.islower( )**
    It checks, whether the string contains atleast one cased character and are all in lower case. If so, it returns true, otherwise, it returns false.
15. **string.isupper( )**
    It checks, whether the string contains atleast one cased character and are all in upper case. If so, it returns true, otherwise, it returns false.
16. **string.isspace( )**
    It checks, whether the string contains only whitespace characters. If so, it returns True, otherwise, it returns False.
17. **string.istitle( )**
    It checks, whether the string is properly "titlecased". If so, it returns true; otherwise, it returns false.
18. **string.isnumeric( )**
    It checks, whether the string contains only numeric characters. If so, it returns true, otherwise it returns false.
19. **string.lower( )**
    It converts the uppercase letters in the given string to lower case letters.
20. **string.upper( )**
    It converts the lowercase letters in the given string to uppercase letters.
21. **string.join(seq)**
    It reads the sequence 'seq' and concatenates all its string elements into a single string, with separator string.
22. **string.partition(str)**
    It looks for the occurrence of 'str' in string. If found, it splits the string into a 3-tuple (string-pre-str, str, string-post-str); otherwise, it checks whether string-pre-string = = string.
23. **string.replace(str1, str2, n = string.count(str1))**
    It looks for str1 in string and replaces each of its occurrences, or 'n' number of its occurrences with str2.
24. **string.ljust(width)**
    It left justifies the string for about width number of columns by including the space padding.
25. **string.lstrip( )**
    It truncates the leading whitespace in string.
26. **string.rfind(str, begin = 0, end = len(string))**
    It is same as find( ) method, but it also performs a backward search in the string.
27. **string.rindex(str, begin = 0, end = len(string))**
    It is same as index( ) method, but it also performs a backward search in the string.
28. **string.rpartition(str)**
    It is same as partition( )method, but it also performs a backward search in the string.
29. **string.rjust(width)**
    It right justifies the string for about width number of columns by including the space padding.
30. **string.rstrip( )**
    It truncates all the trailing whitespaces of string.

string.strip([obj])

It is a combination of both lstrip( ) and rstrip( ) methods on string.

string.split(str = " ", n =string.count(str))

It considers 'str' as, the delimiter and splits the string into atmost 'n' number of substrings (if given). It returns the resulting list of substrings.

string.splitlines(n = string.count('\n'))

It considers the NEWLINES as, the delimiter and splits the string accordingly. It returns the resulting list of each line, by removing the NEWLINES.

string.title( )

It determines the 'titlecased' version of string (i.e., the words in the string begin with uppercase and are followed by all lower case letters.

string.startswith(obj, begin = 0, end = len(string))

It returns 'True', if the string or its substring (identified by begin and end indices) starts with string 'obj'. If 'obj' is a tuple, it checks, whether the string starts with any of the strings in that tuple.

string.swapcase( )

It inverts the case of all the letters in string, i.e., it converts uppercase letters to lower case and lower case letters to upper case.

string.zfill(width)

It adds zeros as the left padding to the string, in order to obtain a total of width number of characters. It is specially used with numbers and it also retains their sign.

string.translate(str, del = " ")

It translates the string as per the translation table and deletes those strings included in the 'del' string.

1.4.2 Lists and Tuples

Q34. Explain in detail about lists.

Answer : Lists **Model Paper-III, Q3(a)**

List is like a string, which provides sequential storage mechanism. Through slices, we can access any number of consecutive elements of list. The unique and flexible characteristics of lists that, differentiate them from strings and arrays are,

(i) List can contain different types of Python objects-standard types, objects and user-defined objects.

(ii) At any given time, single or multiple objects can be inserted, removed and updated to the list.

(iii) List can grow and shrink.

(iv) List can even be sorted, reversed, empty and populated.

(v) A list can be added/removed to/from other lists.

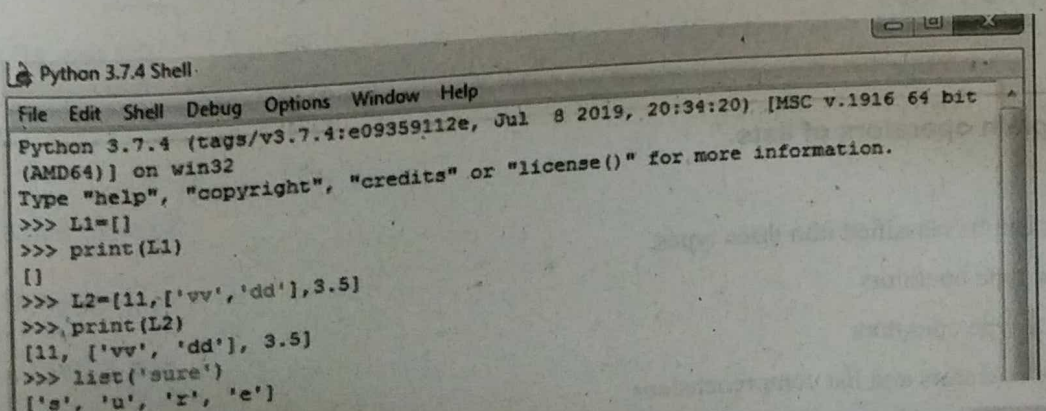
(vi) Creating a list is also very simple.

(vii) Individual or multiple elements of a list can be easily added, updated or removed.

Creating and Assigning Lists

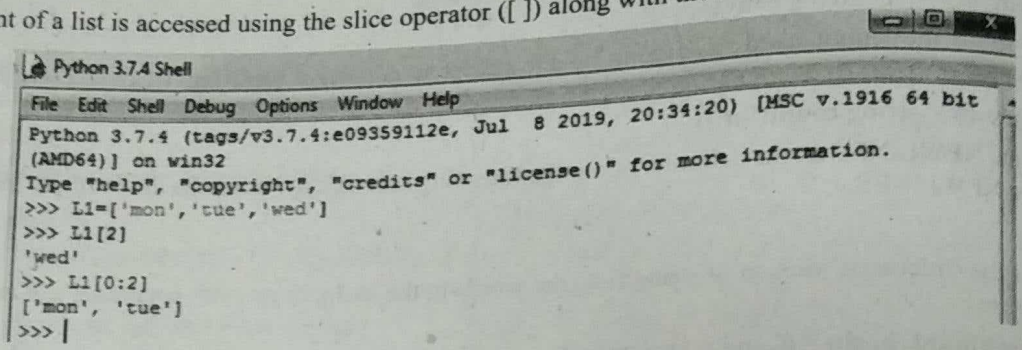
Creating list is simple and similar as assigning a value to a variable. List is created by an assignment operator. A list can be created, as 'empty' or with elements. The elements of a list are enclosed in a square bracket ([ ]). A list can also be created using the factory function namely list( ).

Example



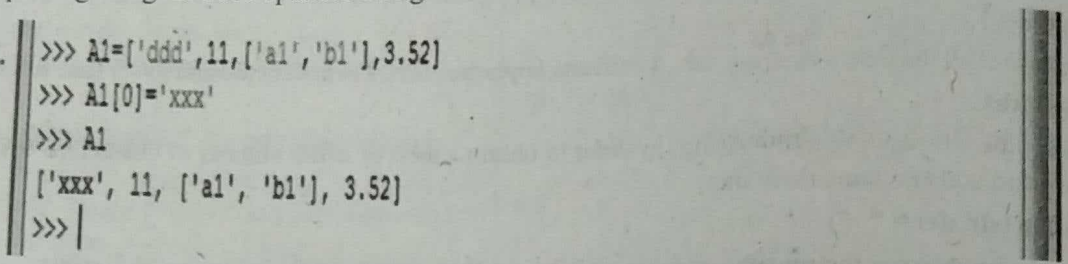
Accessing Values in a List

An element of a list is accessed using the slice operator ([ ]) along with the index or indexes.

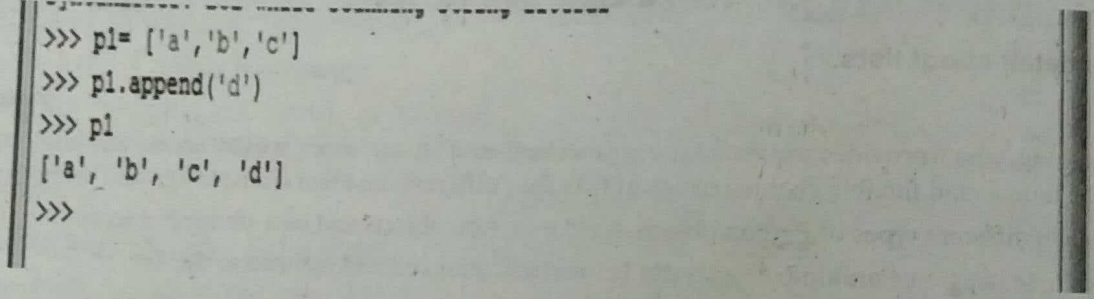


Updating Elements to a List

A list is updating using the slice operator along with the index on the left-hand side of the assignment operator.

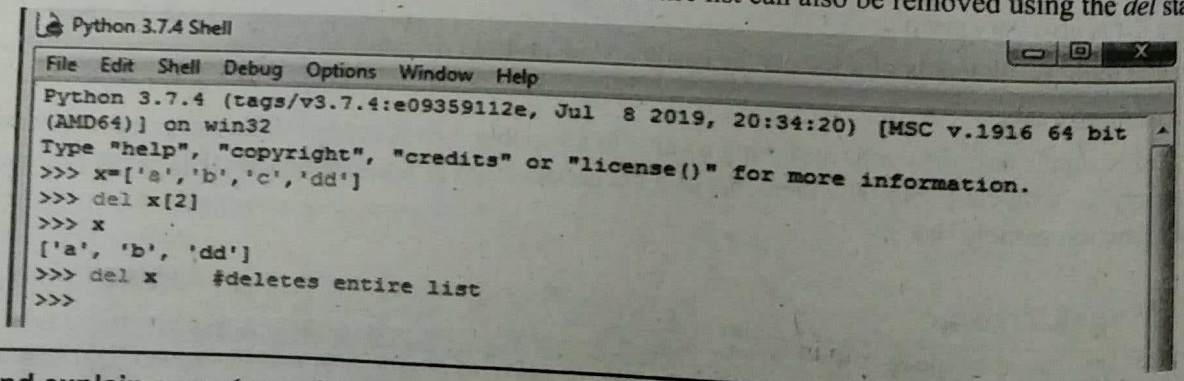


New elements can be added to a list using the append( ) method.



Removing Elements of a List

An element of a list can be deleted, using either the *del* statement if the index of the element is known or the remove method, which takes the value of the element as an argument. The entire list can also be removed using the *del* statement.



Q35. List and explain operators of lists.

Answer :

The operators can be classified into three types,

1. Standard type operators
2. Sequence type operators
3. List type operators and list comprehensions.

One of the standard type operators that are used on lists are comparison operators. They compare numbers and strings in straight forward manner excluding the lists.

```
'n'
>>> l1=['pqr',346]
>>> l2=['abc',123]
>>> l3=['xyz',789]
>>> l1<l2
False
>>> l1>l3
False
>>> l3>l2 and l1==l3
False
>>>
```

Sequence Type Operators

Several sequence type operators are as follows,

slices ([ ] and [:])

The slicing operations on lists is similar to that of performed on strings. Slicing extracts an object or group of objects i.e., the elements of list.

```
>>> n=[22,-4.6,-5,3.19e3]
>>> str=['follow','you','over','streets']
>>> mix=[3.5,[1,'p'],'Good',-1.9+8j]
```

The slicing operators follow the same rules with respect to the positive and negative indexes, beginning and ending indexes including the missing indexes.

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> n=[22,-4.6,-5,3.19e3]
>>> str=['follow','you','over','streets']
>>> mix=[3.5,[1,'p'],'Good',-1.9+8j]
>>> n[2]
-5
>>> n[2:]
[-5, 3190.0]
>>> n[2:-5]
[]
>>> n[:3]
[22, -4.6, -5]
>>> mix
[3.5, [1, 'p'], 'Good', (-1.9+8j)]
>>> mix[2]
'Good'
>>> mix[1][1]
'p'
```

Membership (in, not in)

An object is checked for membership in list by using lists.

```
>>> mix
[3.5, [1, 'p'], 'Good', (-1.9+8j)]
>>> 'Good'in mix
True
>>> 26 in n
False
>>> 5 not in mix
True
>>>
```

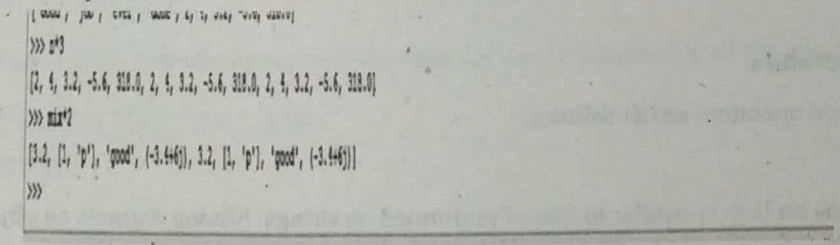
Concatenation

This operator enables various lists to be joined together. This is possible only on the objects of same type.

```
>>> n=[2,4,3.2,-5.6,3.18e2]
>>> str=['Good','job','ever','done']
>>> mix=[3.2,[1,'p'],'good',-3.4+6j]
>>> n+mix
[2, 4, 3.2, -5.6, 318.0, 3.2, [1, 'p'], 'good', (-3.4+6j)]
>>> str+n
['Good', 'job', 'ever', 'done', 2, 4, 3.2, -5.6, 318.0]
>>>
```

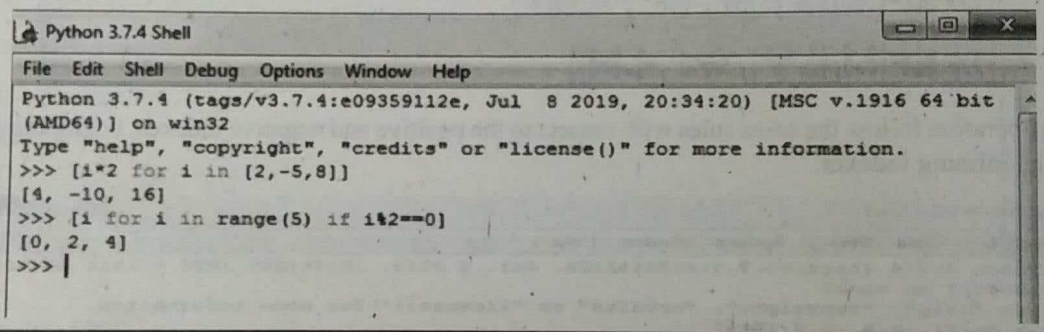
Repetition

This operator is used to repeat this lists.



3. List Type Operators and List Comprehensions

Lists are allowed to be used with object and sequence operators. The objects contain their own methods. These lists also have list comprehensions for them. They contain list square brackets and for-loop in a piece of logic depicting the list object contents that is to be created.



Q36. Write about different built-in functions of lists and tuples.

Answer :

Built-in Functions of Lists

The following are built in functions of lists.

1. **len( )**

   It returns the total length of a list.

   syntax : len(list)

2. **list( )**

   It transforms a given tuple into list.

   syntax : list(seq)

3. **cmp( )**

   It compares the elements of two given lists.

   syntax : cmp(list 1, list 2)

4. **max( )**

   It returns the element with maximum value from the given list.

   syntax : max(list)

5. **min( )**

   It returns the element with minimum values from the given list.

   syntax : min(list)

Define tuples. Explain in detail about tuples.

Model Paper-II, Q3(b)

Tuples are containers, that are used to store data items. They are similar to lists because they allow a list of elements to ... However, there are few differences between lists and tuples. Tuples use parentheses whereas, lists use square brackets. ...les are immutable (i.e., they cannot be modified), whereas, lists are not. Due to the property of immutability, tuples can be ...d as a dictionary-key and also while dealing with object groups.

...ating and Assigning Tuples

This operation is similar to that of lists. The tuple elements are enclosed in parentheses and element is separated by a ...ma (,).

...mple

```
>>> mytuple=('SIA','Group','of','Companies')
>>> mytuple1=('Hyd',['oldcity',500008],28)
>>> mytuple
('SIA', 'Group', 'of', 'Companies')
>>> mytuple1
('Hyd', ['oldcity', 500008], 28)
>>>
```

Note that in an empty tuple, we need to add a trailing comma (,) inside the tuple parentheses. (This is done to avoid ...fusion with a natural grouping operation of parentheses). The following example shows the creation of an empty tuple:-

```
>>> emptytuple=(None,)
>>> emptytuple
(None,)
```

Moreover tuples can also be created by using the factory function tuple( ), as shown below:

```
(None,)
>>> tuple('SIA')
('S', 'I', 'A')
>>>
```

...cessing Tuple Values

To access tuple values, we need to use the square bracket slice operator ([ ]) with the index or indices of element(s) within it.

tuple_name[index]

...ample

```
>>> mytuple=('SIA','Group','of','Companies')
>>> mytuple[0]      #accessing the zeroth element
'SIA'
>>> mytuple[0:2]    #accessing first two elements
('SIA', 'Group')
>>> mytuple[:2]
('SIA', 'Group')
>>> mytuple[2][1]
'f'
>>> mytuple[2][1]   #accessing 2nd element's 1st element
'f'
>>>
```

Updating Tuples

Tuples are immutable entities, meaning that update operation is not applicable to them. In other words, we cannot modify the contents of tuples. However, we can extract portions of tuples to create new strings or tuples.

Example

Consider that we have the following tuple:

```
>>> old=('a','b','c','d',1,2,3)
>>> new=(old[0],old[3],old[2]) #Creating new tuple by using old tuple
>>> new
('a', 'd', 'c')
>>>
```

We can even create new tuple as a combination of different tuples:

```
>>> T1=(1,2,3)
>>> T2=('aa','bb','cc')
>>> T3=T1+T2  # combination of different tuples
>>> T3
(1, 2, 3, 'aa', 'bb', 'cc')
>>>
```

Deleting Tuples and Tuple Elements

Since tuples are immutable entities, it is not possible to delete individual tuple elements. However, we can indirectly remove the undesired elements, by creating a new tuple, containing only desired elements and all undesired elements removed.

To delete the entire tuple, we need to use the 'del' statement as follows, >>> del mytuple.

The 'del' statement reduces the object's reference count and if it reaches 0, it will be deallocated.

Q38. List out the built-in functions of tuples.

Answer :

Built-in Functions of Tuples

The following are built-in functions of tuples.

1. **len( )**

   It returns the total length of a tuple.

   syntax : len(tuple)

2. **tuple( )**

   It converts a given list into tuple.

   syntax : tuple(seq)

3. **cmp( )**

   It compares the items of two given tuples.

   syntax : cmp(tuple1, tuple2)

4. **max( )**

   It returns the element with maximum value from a given tuple.

   syntax : max(tuple)

5. **min( )**

   It returns the element with minimum value from a given tuple.

   syntax : min(tuple)

What are dictionaries? Explain with examples.

Model Paper-I, Q3(b)

swer 1

ctionaries

Dictionaries refer to python's mapping type. They are similar to associative arrays, or hashes in perl. The syntax of a ctionary entry is key : value. Keys correspond to any python type and it is usually numbers or strings. Whereas, *values* correspond y arbitrary python object. Dictionaries are enclosed by curly braces ({ }).

eating and Assigning Dictionaries

Dictionaries are created by simply assigning a dictionary to a variable, without considering whether dictionary contains ments or not. The syntax is as follows.

```
>>> dicta={}
>>> dictb={'name':'Jack','port':100}
>>> dicta,dictb
({}, {'name': 'Jack', 'port': 100})
>>>
```

As shown in the example, dictionary consists of keys and their corresponding values. Each key is separated from its value a colon (:), the items are separated by commas and the whole thing is enclosed in curly braces. In version 2.2, factory function ct( ) is used to create dictionaries.

```
>>> fdict=dict([['a',1],['b',2]])
>>> fdict
{'a': 1, 'b': 2}
>>>
```

The fromkeys( ) method, creates a new dictionary with the given keys, each with a default corresponding value of none.

```
>>> ddict={}.fromkeys(('x','y'),-1)
>>> ddict
{'x': -1, 'y': -1}
>>> edict={}.fromkeys(('foo','bar'))
>>> edict
{'foo': None, 'bar': None}
>>>
```

cessing Values from Dictionaries

User can access the values of the dictionary by using unique keys.

```
>>>dict b = {'name' : 'Ramana', 'Port' : 90}
>>>
>>>for key in dictb.keys( ):
    print 'key = %s, value = %s'% (key, dictb[key])
    key = name, value = Ramana
    key = port, value = 90.
```

In version 2.2, the function keys( ), is not used to retrieve a list of keys. Sequence objects like, dictionaries and files are mply accessed by creating iterators.

The dictionary name itself creates an iterator over the dictionary, which is to be used in for loop.

```
>>>dictb = {'name' : 'Ramana', 'port' : 90}
>>>
>>>for key in dictb :
    print 'key = %s' %(key, dictb[key])
    key = name, value = Ramana
    key = port, value = 90.
```

The square brackets along with the key, can be used to access individual dictionary elements.

```
>>>dictb['name']
'Ramana'
>>>
>>>print'Host %s is running on port %d' %\
(dictb['name'], dictb['port'])
Host Ramana is running on port 90.
```

The dictionary 'dictb' contains, two data items. The keys in dictb are 'name' and 'port' and its corresponding values are 'Ramana' and '90' respectively.

If user accesses a value through the key that is not present in the dictionary, then error occurs, which is shown in the following example.

```
>>>dictb['Sai']
Traceback(innermost last):
File"<stdin>", line1, in?
KeyError : Sai
```

In the above example, the key 'Sai' is not present in the dictionary 'dictb'. Hence error occurs.

User can use has_key( ) method, which checks whether a dictionary has specific key or not. The has_key( ) method and the in and not in operators are boolean, that is, they return true if key exists in the dictionary otherwise returns false.

```
>>>'Sai' in dictb # or dictb.has_key('Sai')
False
>>>'name' in dict # or dictb.has_key('name')
True
>>>dictb['name']
'Ramana'
```

The following example shows the key as combination of numbers and strings.

```
>>>dictc = { }
>>>dictc[2] = 'Shiva'
>>>dictc['2'] = 4.567
>>>dictc[4.9] = 'abc'
>>>dictc
{4.9 : 'abc', 2 : 'Shiva', '2' : 4.567}
```

If user knows all key_value pairs in advance, then user can add all the data items simultaneously, instead of adding each data item individually. The syntax of adding multiple items at the same time is as follows,

```
dictc = {4.9 : 'abc', 2 : 'Shiva', '2' : 4.567}
```

Updating Dictionaries

User can update a dictionary, by adding a new key_value pair, modifying an already existing key_value pair or removing an existing key_value pair.

```
>>dictb[ ] = 'windows2000' # adding new key_value
>>>dictb['name'] = 'Shiva' # updating existing key_value
>>>dictb['port'] = 99# updating existing key_value
>>>
>>>print 'host %(name)s is running on port %(port)d' %dict2 host shiva is running on port 99.
```

If the key added already exists then the old value will be replaced by its new value.

Removing Dictionaries and Elements of Dictionaries

Users can delete the elements of the dictionary, or an entire dictionary, using *del* statement. The entire contents of a dictionary can be removed using clear statement as follows,

```
del dictb['name']# Delete entry with key 'name'
dictb.clear( )      # Delete the entire contents of dict b
del dictb           # Delete entire dictionary
dictb.pop('name') # Delete and return entry w/key
```

swer :

rators of Dictionaries

The operators of dictionaries are classified into two types namely standard type operators and mapping type operators.

Standard Type Operators

The standard type operators of dictionaries are comparison operators including less than (<), greater than (>), less than ual to (<=), greater than equal to (>=) etc.

```
>>> d1=('xyz',456)
>>> d2=('abc',123)
>>> d3=('pqr',789)
>>> d1>d3
True
>>> d2>d1
False
>>> d3<d2
False
>>> (d1>d3) and (d2<d3)
True
>>>
```

Mapping Type Operators

There are two different types of mapping type operators, they are,

(i) **Dictionary Key-Lookup Operator ([ ])**

The key-lookup operator is specific to dictionaries. It similar to that of single element slice operator for sequence types. The index offset is passed as an argument to sequence types for accessing single element of it. And in case of dictionary the lookups are by key. Therefore it is not an index rather it is considered as argument. The lookup operator can assign as well as retrieve the values from dictionary.

(ii) **(key) Membership (in, not in)**

The membership can be checked by using the in and not in operators.

```
True
>>> 'for' in d3
False
>>> 5 in d1
False
>>> 'Bye'not in d2
True
>>>
```

ictionary Keys

The dictionary values does not have restrictions imposed on them. Any python object ranging from standard objects to user efined objects can be dictionary value. But its not same for dictionary keys. The properties of dictionary keys are as follows,

1. More than one entry per key not allowed one major constraint is for each dictionary key only one value is allowed rather than multiple values for same key. If there are any key collisions, the assignment that is done at last will be fixed.

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> d1={'Hello':236,'Hello':'abc'}
>>> d1
{'Hello': 'abc'}
>>> d1['Hello']=567
>>> d1
{'Hello': 567}
>>>
```

Python will not check for any key collisions because it needs memory for all the key-value pair.

2. **Keys must be hashable**

The python objects can be used as keys and they must be hashable objects. The mutable types like lists and dictionaries are supported by Python because they cannot be hashed. All the immutable types can be used as keys because they are hashable. Certain mutable objects are hashable and they can be used as keys.

The key need to be hashable. The hash function that is used by interpreter for computing the location to store the data depends upon key value. The value is modifiable when key is mutable object. The hash function can map to a different location if the key is modified. Therefore, the hash function can never reliably store or retrieve the associated value. The hashable keys can be selected because their values cannot be modified.

Q41. List the built-in functions and methods of dictionary objects.

Answer :

Built-in Functions of Dictionary Objects

The following are the built-in functions of dictionary objects.

1. **type( )**

   It returns the type of variable that is passed to it as parameter. If the variable is a dictionary then it returns a dictionary type.

   syntax : type(variable)

2. **str( )**

   It produces a printable string representation of a dictionary. A dictionary is passed to it as parameter.

   syntax : str(dict)

3. **cmp( )**

   It compares the elements of two dictionaries. The comparison is done through an algorithm which compares two dictionaries using their size, key and values.

   syntax : comp(dict1, dict2)

Built-in Methods of Dictionary Objects

The following are the built-in methods of dictionary objects.

1. **items( )**

   It returns a list of dict's tuple pairs of (key, value).

   syntax : dict.items( )

2. **copy( )**

   It returns a shallow copy of dictionary dict.

   syntax : dict.copy( )

3. **update( )**

   It appends the dictionary dict2's key-value pairs to dict.

   syntax : dict.update(dict 2)

4. **clear( )**

   It deletes all elements of dictionary dict.

   syntax : dict.clear( )

5. **values( )**

   It returns a list of dictionary dict's values.

   syntax : dict.values( )

6. **fromkeys( )**

   It produces a new dictionary with keys from seq and value set to the value.

   syntax : dict.fromkeys( )

7. **keys( )**

   It produces a list of dictionary dict's keys.

   syntax : dict.keys( )

8. **has_key( )**

   It returns true if the specified key is found in dictionary and false if it is not found in dictionary.

   syntax : dict.has_key(key)

9. **get( )**

   It returns value of the specified key if it is found in the dictionary or returns default if key is not found in dictionary.

   syntax : dict.get(key, default = None)

10. **setdefault( )**

    It sets dict[key] as default if key does not exist in dict.

    syntax : setdefault(key, default = None).

Q42. Explain about sets.

Answer :

Model Paper-III, Q3(b)

Sets

A set can be defined as an unordered group of zero or more hashable objects like ints, floats, strings or tuples. Here, the word 'hashable' means immutable. It does not defines lists or other sets. A set that contains Immutable object is referred to as Frozen set. Immutable objects can be included as items either in ser or in Frozen sets.

Syntax: name = {item1, item2, item3, item4}

The above syntax is identical for both list and tuple but, with a difference that it makes use of curly braces instead of square brackets. The sequence of items can be stored in unordered way. Since, the items are unordered, sets cannot be shared, copied or the index is also not used to determine particular items. Consider the below example,

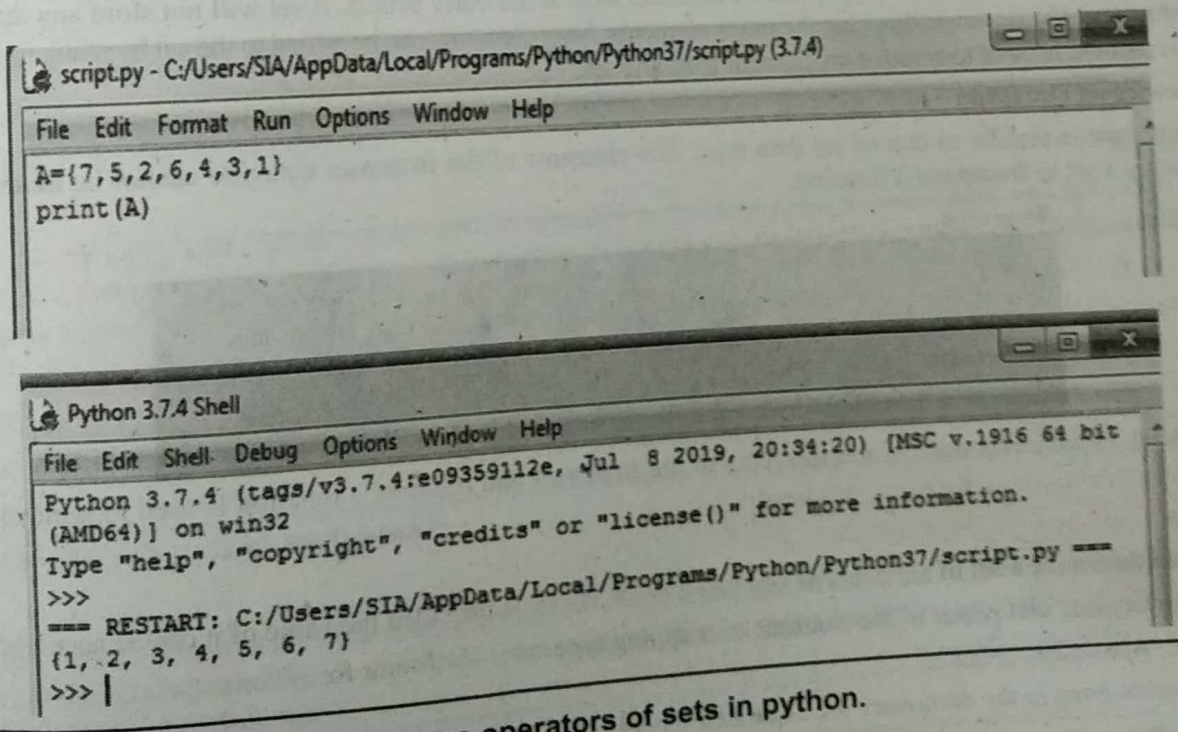
>>>Languages = {'c', 'c++', 'java', 'oracle'}

The elements in a set can be shown as >>> Languages in interactive mode or print (Languages) in a program. The items in a set are stored in a particular order but, they are shown in different orders.

Methods of Sets

1. **add( ):** This method is used to add item to a set.
   Syntax: b.add(a)
2. **discard( ):** This method is used to remove an item if it is present in another set.
   Syntax: b.discard(a)
3. **remove( ):** This method is used to remove an item if it is present in another set or an exception is raised.
   Syntax: b.remove(a)
4. **pop( ):** This method is used to return and remove an item from set. If set is empty then, an exception is raised.
   Syntax: a = b.pop( )
5. **clear( ):** This method is used to remove all items from set.
   Syntax: b.clear( )
6. **copy( ):** This method is used to copy a set to a new set.
   Syntax: p = b.copy( )

Program



Q43. Discuss about various standard type operators of sets in python.

Answer :

Operators of sets are classified into standard type operators and mapping type operators.

1. **Standard Type Operators**

Python supports various standard type operators for sets. They are the comparison operators such as less than (<), greater than (>), equal to (= =), not equal to (! =) etc. It is difficult to perform these operations on numbers and strings like lists and tuples.

2. **Mapping Type Operators**

Various mapping type operators are as follows,

(i) **Dictionary Key-Lookup Operator ([ ])**

The key lookup operator is specific to dictionaries and it used as single element slice operator for sequence types. An index offset for sequences is considered as single argument or subscript for accessing an element from sequence. The lookups are performed by key for dictionaries. It can be used for assigning as well as retrieving values from dictionary.

```
>>> dict [x] = a
>>> dict [x]
```

(ii) **Membership (in, not in)**

The membership operators in and not in are used to check for membership.

```
>>> 'Good' in d1
>>> 'Hello' not in d3
```

Q44. Write in short about functions of sets and mapping types.

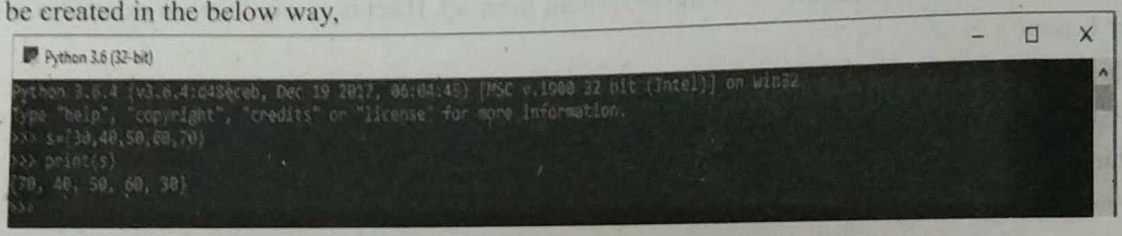
Answer :

Sets

A set can be defined as a group unordered elements. The elements in a set might not appear in the order they are entered into it. There are two types of sub types, they are,

1. **Set Datatype**

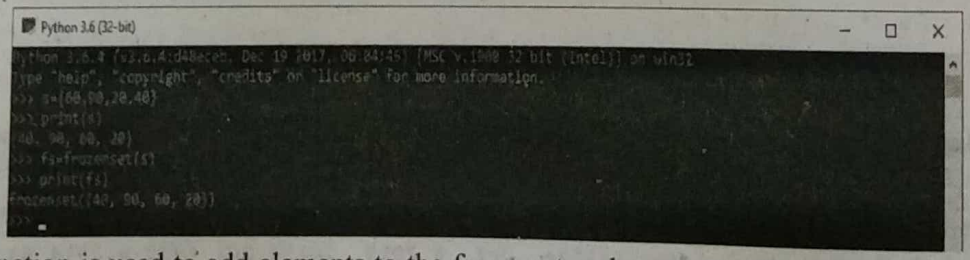
A set can be created in the below way,



The elements in the set must be separated by commas with in the curly braces. A set will not store any duplicate values. Elements of sets cannot be retrieved by using the index elements. New elements can be added to the set by using update( ) method. And remove( ) method is used to remove an element from the set.

2. **The Frozenset Datatype**

This data type is similar to that of set data type. The elements of the frozenset datatype cannot be modified. It can be created by passing a set to frozenset( ) function.



The update( ) function is used to add elements to the frozenset and remove( ) function is used to remove the element from the set.

Mapping Types

A map can determine a set of elements as key value pairs. If key is provided the value of it can be accessed.

The dictionary i.e., dict is one of the example of mapping type.

dt = {20 : 'ABC', 30 : 'XYZ'}

The key value pairs in the dictionary are separated by colon(:).