PART-1

Function : Parts of a Function, Execution of a Function, Keyword and Default Arguments, Scope Rules.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. Define function and write its advantages.

Answer

1. Functions are self-contained programs that perform some particular tasks.
2. Once a function is created by the programmer for a specific task, this function can be called anytime to perform that task.
3. Each function is given a name, using which we call it. A function may or may not return a value.
4. There are many built-in functions provided by Python such as dir (), len ( ), abs ( ), etc.
5. Users can also build their own functions, which are called user-defined functions.

Advantages of using functions :

1. They reduce duplication of code in a program.
2. They break the large complex problems into small parts.
3. They help in improving the clarity of code (*i.e.*, make the code easy to understand).
4. A piece of code can be reused as many times as we want with the help of functions.

Que 3.2. How to define and call function in Python ? Explain different parts of a function.

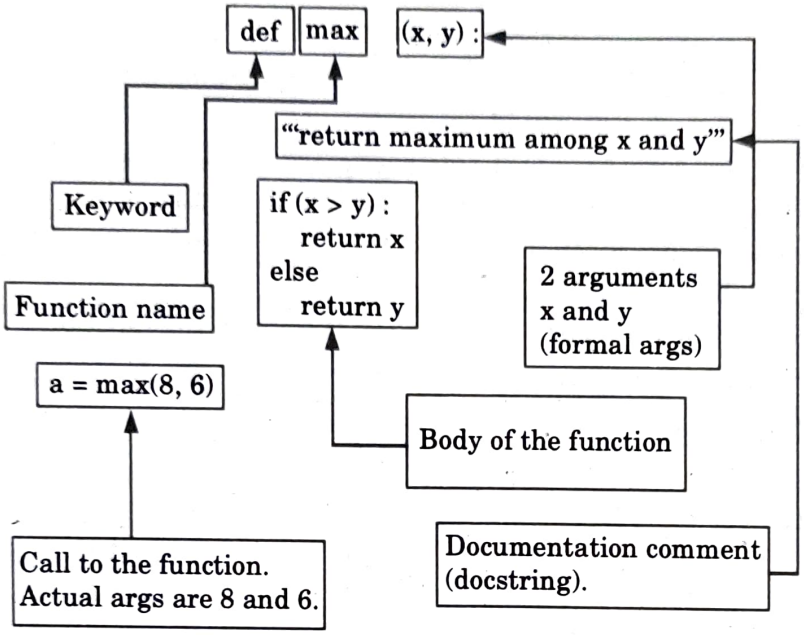
Answer

Function is defined by "def" keyword following by function name and parentheses.

Syntax of function definition : def function_name ( ) :

Syntax of functional call : function_name ( )

For example :



1. **Keyword :** The keyword 'def' is used to define a function header.
2. **Function name :** We define the function name for identification or to uniquely identify the function. In the given example, the function name is max. Function naming follows the same rules of writing identifiers in Python.
3. A colon (:) to mark the end of function header.
4. **Arguments :** Arguments are the values passed to the functions between parentheses. In the given example, two arguments are used, x and y. These are called formal arguments.
5. **Body of the function :** The body processes the arguments to do something useful. In the given example, body of the function is intended w.r.t. the def keyword.
6. **Documentation comment (docstring) :** A documentation string (docstring) to describe what the function does. In the given example, "return maximum among x and y" is the docstring.
7. An optional return statement to return a value from the function.
8. **Function call :** To execute a function, we have to call it. In the given example, a = max (8, 6) is calling function with 8 and 6 as arguments.

Que 3.3. Discuss the execution of a function with the help of example.

Answer

Let consider an example to understand the execution of function :

Step 1 : When function without "return" :

a. For example, we want the square of 4, and it should give answer "16" when the code is executed.

b. Which it gives when we simply use "print x*x" code, but when we call function "print square" it gives "None" as an output.

c. This is because when we call the function, recursion does not happen and leads to end of the function.

d. Python returns "None" for failing off the end of the function.

For example :

```
def square(x) :
    print(x*x)
print(square(4))
```

Output :

```
16
None
```

Step 2 : When we retrieve the output using "return" command :

When we use the "return" function and execute the code, it will give the output "16".

For example :

```
def square(x) :
    return(x*x)
print(square(4))
```

Output :

```
16
```

Step 3 : When function is treated as an object :

a. Functions in Python are themselves an object, and an object has some value.

b. When we run the command "print square" it returns the value of the object.

c. Since we have not passed any argument, we do not have any specific function to run hence it returns a default value (0x021B2D30) which is the location of the object.

For example :

```
def square(x) :
    return(x*x)
```

```
print(square)
```

Output :

```
<function square at 0x021B2D30>
```

Que 3.4. What do you mean by keyword and default arguments?

Answer

Keyword arguments :

1. Keyword arguments are related to the function calls.
2. When we use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
3. This allows us to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

For example :

```
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;
# Now you can call printme function
printme( str = 'My string')
```

Output :

```
My string
```

Default arguments :

1. A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

For example :

```
# Function definition is here
def printinfo( name, age = 35 ):
    "This prints a passed info into this function"
    print "Name:", name
    print "Age", age
    return ;
# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
printinfo( name = "miki")
```

Output :

```
Name : miki
```

Age 50

Name : miki

Age 35

Que 3.5. Discuss the scope rules in Python.

Answer

1. Scope of a name is the part of the program in which the name can be used.
2. Two variables can have the same name only if they are declared in separate scopes.
3. A variable cannot be used outside its scopes.
4. Fig. 3.5.1 illustrates Python's four scopes.

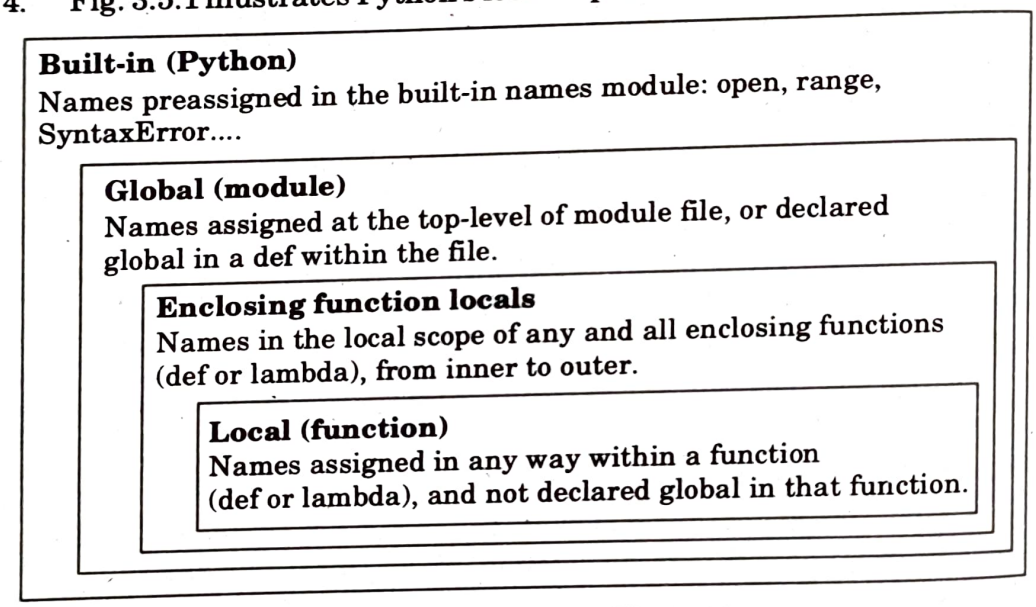


Fig. 3.5.1. The LEGB scope.

5. The LEGB rule refers to local scope, enclosing scope, global scope, and built-in scope.
6. Local scope extends for the body of a function and refers to anything indented in the function definition.
7. Variables, including the parameter, that are defined in the body of a function are local to that function and cannot be accessed outside the function. They are local variables.
8. The enclosing scope refers to variables that are defined outside a function definition.
9. If a function is defined within the scope of other variables, then those variables are available inside the function definition. The variables in the enclosing scope are available to statements within a function.

Que 3.6. **Discuss function in Python with its parts and scope. Explain with example. (Take simple calculator with add, subtract, division and multiplication).** **AKTU 2019-20, Marks 10**

Answer

Function : Refer Q. 3.1, Page 3–2T, Unit-3.

Parts of function : Refer Q. 3.2, Page 3–2T, Unit-3.

Scope : Refer Q. 3.5, Page 3–6T, Unit-3.

For example : Simple calculator using python :

```
# This function adds two numbers
def add(x, y) :
    return x + y
# This function subtracts two numbers
def subtract(x, y) :
    return x – y
# This function multiplies two numbers
def multiply(x, y) :
    return x * y
# This function divides two numbers
def divide(x, y) :
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
# Take input from the user
choice = input("Enter choice(1/2/3/4) : ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1' :
print(num1,"+",num2,"=", add(num1,num2))
elif choice == '2' :
print(num1,"–",num2,"=", subtract(num1,num2))
elif choice == '3' :
print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
print(num1,"/",num2,"=", divide(num1,num2))
else :
print("Invalid input")
```

PART-2

Strings : Length of the String and Perform Concatenation and Repeat Operations in it, Indexing and Slicing of Strings.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.7. What do you mean by the term strings ?

Answer

1. Strings are created by enclosing various characters within quotes. Python does not distinguish between single quotes and double quotes.
2. Strings are of literal or scalar type. The Python interpreter treats them as a single value.
3. Strings, in Python, can be used as a single data type, or, alternatively, can be accessed in parts. This makes strings really useful and easier to handle in Python.

For example :

```
>>> var1 = 'Hello Python!'
>>> var2 = "Welcome to Python Programming!"
>>> print var1
Hello Python!   #   Output
>>> print var2
Welcome to Python Programming!   #   Output
```

Que 3.8. What is len( ) function ?

OR

Explain length of the string.

Answer

1. len ( ) is a built-in function in Python. When used with a string, len returns the length or the number of character in the string.
2. Blank symbol and special characters are considered in the length of the string.

For example :

```
>>> var = "Hello Python!"
```

```
>>> len(var)
13  # Output
```

Here, we took a string 'Hello Python!' and used the len function with it. The len function returned the value 13 because not only characters, but also the blank space and exclamation mark in our string will also be counted as elements.

Que 3.9. Discuss the concatenation and repeat operation in Python with example.

Answer

Concatenation :

1. Concatenation means joining two operands by linking them end-to-end.
2. In list concatenation, + operator concatenate two lists with each other and produce a third list.
3. For example, to concatenate two lists :

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> z = x + y                    # concatenate two lists
>>> print z
[1, 2, 3, 4, 5, 6]
```

Repeat / replicate :

1. Lists can be replicated or repeated or repeatedly concatenated with the asterisk (*) operator.
2. For example,

```
>>> aList = [1, 2, 3]
>>> print aList*3                # list aList repeats three times
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Here, the list aList multiplied by 3 and printed three times.

Que 3.10. What do you mean by string slices ?

Answer

1. A piece or subset of a string is known as slice.
2. Slice operator is applied to a string with the use of square braces ([]).
3. Operator $[n : m]$ will give a substring which consists of letters between n and m indices, including letter at index n but excluding that at m, *i.e.*, letter from n^{th} index to $(m - 1)^{th}$ index.

For example :

```
>>> var = 'Hello Python'
```

```
>>> print var [0 : 4]
Hell # Output
```

In the above example, we can see that in the first case the slice is [0 : 4], which means that it will take the 0^{th} element and will extend to the 3^{rd} element, while excluding the 4^{th} element.

4. Similarly, operator [n : m : s] will give a substring which consists of letters from n^{th} index to $(m-1)^{th}$ index, where s is called the step value, *i.e.*, after letter at n, that at $n+s$ will be included, then $n+2s$, $n+3s$, etc.

   For example :

```
>>> alphabet = "abcdefghij"
>>> print alphabet [1 : 8 : 3]
beh # Output
```

In the given example, the slice is [1 : 8 : 3], which means it will take the element at 1^{st} index which is *b* and will extend till 7^{th} element. Since step is 3, it will will print 1^{st} element, then 4^{th} element and then 7^{th} element *i.e.*, beh.

Que 3.11. Explain the term indexing in Python.

Answer

1. Indexing means referring to an element of an iterable by its position within the iterable.
2. **For example :** Create a list using a list comprehension :

```
my_list = [_for_in 'abcdefghi']
my_list
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

3. Now to retrieve an element of the list, we use the index operator ([ ]).
   For example :

```
my_list[0]
'a'
```

4. In Python, lists are "zero indexed", so [0] returns the zeroth (*i.e.*, the left-most) item in the list.
5. In the given example there are 9 elements in list ([0] through [8]) and if we want to access my_list[9] then it throws an indexError : list index out of range.
6. Python also allows to index from the end of the list using a negative number. In negative indices, we start counting from the right instead from the left.
7. Negative indices start from – 1.

PART-3

Pythons Data Structure : Tuples, Unpacking Sequences, Lists.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.12. Define tuples. How are tuples created in Python ?

Answer

1. Tuples are the sequence or series values of different types separated by commas (,).
2. Values in tuples can also be accessed by their index values, which are integers starting from 0.

For example :

The names of the months in a year can be defined in a tuple :

```
>>> months = ('January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December')
```

Creating tuples in Python :

1. To create a tuple, all the items or elements are placed inside parentheses separated by commas and assigned to a variable.
2. Tuples can have any number of different data items (that is, integer, float, string, list, etc.).

For examples :

1. **A tuple with integer data items :**

```
>>> tuple = (4, 2, 9, 1)
>>> print tuple
(4, 2, 9, 1) # Output
```

2. **A tuple with items of different data types :**

```
>>>tuple_mix = (2, 30, "Python", 5.8, "Program")
>>>print tuple_mix
(2, 30, 'Python', 5.8, 'Program')  #  Output
```

3. **Nested tuple :**

```
>>>nested_tuple = ("Python", [1, 4, 2], ["john", 3.9])
>>> print nested_tuple
('Python', [1, 4, 2], ['john', 3.9])  #  Output
```

4. **Tuple can also be created without parenthesis :**

```
>>>tuple = 4.9, 6, 'house'
>>>print tuple
(4.9, 6, 'house')  #  Output
```

Que 3.13. **How are the values in a tuple accessed ?**

Answer

Accessing values in tuples :

1. In order to access the values in a tuple, it is necessary to use the index number enclosed in square brackets along with the name of the tuple.

   For example : Using square brackets

```
>>>tup1 = ('Physics', 'Chemistry', 'Mathematics')
>>>tup2 = (10, 20, 30, 40, 50)
>>>print tup1 [1]
Chemistry  #  Output
>>>print tup2 [4]
50  #  Output
```

2. We can also use slicing in order to print the continuous values in a tuple.

   For example :

```
>>>tup1 = ('Physics', 'Chemistry', 'Mathematics')
>>>tup2 = (10, 20, 30, 40, 50)
>>>tup2 [1 : 4]
(20, 30, 40)  # Output
>>>tup1[ : 1]
('Physics',)  #  Output
```

Que 3.14. **Explain tuples as return values and variable-length arguments tuples with the help of example.**

Answer

Tuple as return values :

1. Tuples can also be returned by the function as return values.
2. Generally, the function returns only one value but by returning tuple, a function can return more than one value.

   For example : If we want to compute a division with two integers and want to know the quotient and the remainder, both the quotient and the remainder can be computed at the same time. Two values will be returned, *i.e.*, quotient and remainder, by using the tuple as the return value of the function.

```
>>>def div_mod(a, b) :  #  defining function
```

```
        quotient = a/b
        remainder = a%b
        return quotient, remainder   #   function returning two values
    # function calling
    >>>x = 10
    >>>y = 3
    >>>t = div_mod(x, y)
    >>>print t
    (3, 1)   #   Output
    >>>type(t)
    <type 'tuple'>   #   Output
```

Variable length argument tuples :

1. Variable number of arguments can also be passed to a function.
2. A variable name that is preceded by an asterisk (*) collects the arguments into a tuple.

   For example : In the given example, we have defined a function traverse with argument *t*, which means it can take any number of arguments and will print each of them one by one.

```
>>>def traverse (* t) :
...         i = 0
...     while i<len(t) :
...         print t[i]
...         i = i + 1
>>>traverse(1, 2, 3, 4, 5)
```

Output :

```
1
2
3
4
5
```

Que 3.15. Explain the basic operation of tuples with example.

Answer

Basic operations of tuples are :

1. **Concatenation :** The concatenation in tuples in used to concatenates two tuples. This is done by using + operator in Python.

For example :

```
>>>t1 = (1, 2, 3, 4)
>>>t2 = (5, 6, 7, 8)
>>>t3 = tl + t2
>>>print t3
(1, 2, 3, 4, 5, 6, 7, 8)  #  Output
```

2. **Repetition :** The repetition operator repeats the tuples a given number of times. Repetition is performed by the * operator in Python.

For example :

```
>>>tuple = ('ok',)
>>>tuple * 5
('ok', 'ok', 'ok', 'ok', 'ok')  # Output
```

3. **In operator :**
   i. The in operator also works on tuples. It tells user that the given element exists in the tuple or not.
   ii. It gives a Boolean output, that is, true or false.
   iii. If the given input exists in the tuple, it gives the true as output, otherwise false.

For example :

```
>>>tuple = (10, 20, 30, 40)
>>>20 in tuple
True  #  Output
>>>50 in tuple
False  #  Output
```

4. **Interation :** It can be done in tuples using for loop. It helps in traversing the tuple.

For example :

```
>>>tuple = (1, 2, 3, 4, 5, 6)
>>> for x in tuple :
... print x
```

Output :

1
2
3
4
5
6

Que 3.16. What do you mean by unpacking sequences ? Give example to explain.

Answer

1. Unpacking allows to extract the components of the sequence into individual variable.
2. Several different assignments can be performed simultaneously.
3. We have multiple assignments in Python where we can have multiple LHS assigned from corresponding values at the RHS. This is an example of unpacking sequence.
4. There is one restriction, the LHS and RHS must have equal length. That is, every value that is created at RHS should be assigned to LHS.
5. Strings and tuples are example of sequences. Operations applicable on sequences are: Indexing, repetition, concatenation.

   For example :

   >>> student

   ('Aditya', 27, ('Python', 'Abha', 303))

   >>> name, roll, regdcourse = student

   >>> name

   Output : Aditya

   >>> roll

   27

   >>> regdcourse

   ('Python', 'Abha', 303)
6. Since strings are also sequences, we can also get individual characters in the string using unpacking operation.

   For example :

   >>> x1, x2, x3, x4 = 'abha'

   >>> print (x1, x2, x3, x4)

   a b h a

Que 3.17. Write short note on list.

Answer

1. A list is also a series of values in Python.
2. In a list, all the values are of any type.
3. The values in a list are called elements or items.
4. A list is a collection of items or elements.
5. The sequence of data in a list is ordered and can be accessed by their positions, *i.e.*, indices.

   For example :

   >>> list = [1, 2, 3, 4]

   >>> list [1]

```
2  #  Output
>>> list [3]
4  #  Output
```

Que 3.18. Explain the copying method in list.

Answer

We can make a duplicate or copy of an existing list.

There are two ways to make copy of a list.

1. **Using [:] Operator :**

```
>>> list_original = [1, 2, 3, 4]
>>> list_copy = list_original [:]   #   Using [:] operator
>>> print list_copy
[1, 2, 3, 4]
```

2. **Using built-in function :** Python has a built-in copy function which can be used to make copy of an existing list. In order to use the copy function, first we have to import it.

For example :

```
>>> from copy import copy   #   Import library
>>> list_original = [1, 2, 3, 4]
>>> list_copy = copy(list_original)   #   Copying list
>>> print list_copy
[1, 2, 3, 4]
```

Que 3.19. How elements are deleted from a list ?

Answer

1. Python provides many ways in which the elements in a list can be deleted.

Methods of deleting element from a list :

1. **pop operator :**
   a. If we know the index of the element that we want to delete, then we can use the pop operator.

For example :

```
>>> list = [10, 20, 30, 40,]
>>> a = list.pop (2)
>>> print list
[10, 20, 40]   #   Output
```

   b. The pop operator deletes the element on the provided index and stores that element in a variable for further use.

2. **del operator :** The del operator deletes the value on the provided index, but it does not store the value for further use.

For example :

```
>>> list = ['w', 'x', 'y', 'z',]
>>> del list (1)
>>> print list
['w', 'y', 'z']  #  Output
```

3. **Remove operator :**
   a. We use the remove operator if we know the item that we want to remove or delete from the list.

   For example :

   ```
   >>> list = [10, 20, 30, 40,]
   >>> list.remove (10)
   >>> print list
   [20, 30, 40]  #  Output
   ```

   b. In order to delete more than one value from a list, del operator with slicing is used.

   For example :

   ```
   >>>list = [1, 2, 3, 4, 5, 6, 7, 8]
   >>>del list [1 : 3]
   >>>print list
   [1, 4, 5, 6, 7, 8] # Output
   ```

Que 3.20. Discuss built-in list operators in detail.

Answer

Built-in list operators are as follows :

1. **Concatenation :** The concatenation operator is used to concatenate two lists. This is done by the + operator in Python.

   For example :

   ```
   >>>list1 = [10, 20, 30, 40]
   >>>list2 = [50, 60, 70]
   >>>list3 = list1 + list2
   >>>print list3
   [10, 20, 30, 40, 50, 60, 70]  #  Output
   ```

2. **Repetition :** The repetition operator repeats the list for a given number of times. Repetition is performed by the * operator.

   For example :

   ```
   >>>list1 = [1, 2, 3]
   >>>list 1 * 4
   [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]  # Output
   ```

3. **In operator :**
   a. The in operator tells the user whether the given string exists in the list or not.

b. It gives a Boolean output *i.e.*, true or false.

c. If the given input exists in the string, it given true as output, otherwise, false.

For example :

```
>>>list = ['Hello', 'Python', 'Program']
>>> 'Hello' in list
True  #  Output
>>>'World' in list
False  #  Output
```

Que 3.21. Write various built-in list methods. Explain any three with example.

Answer

Python includes many built-in methods for use with list as shown in Table 3.21.1.

Table 3.21.1. List of built-in list methods.

S. No.	Method	Description
1.	cmp (list1, list2)	It compares the elements of both lists, list1 and list2.
2.	max (list)	It returns the item that has the maximum value in a list.
3.	min (list)	It returns the item that has the minimum value in a list.
4.	list (seq)	It converts a tuple into a list.
5.	list.append (item)	It adds the item to the end of the list.
6.	list.count (item)	It returns number of times the item occurs in the list.
7.	list.extend (seq)	It adds the elements of the sequence at the end of the list.
8.	list.remove (item)	It deletes the given item from the list.
9.	list.reverse ( )	It reverses the position (index number) of the items in the list.
10.	list.sort ([func])	It sorts the elements inside the list and uses compare function if provided.

1. **Append method :** This method can add a new element or item to an existing list.

For example :

```
>>>list = [1, 2, 3, 4]
>>>list.append (0)
[1, 2, 3, 4, 0]                    # Output
```

2. **Extend method :** This method works like concatenation. It takes a list as an argument and adds it to the end of another list.

For example :

```
>>>list1 = ['x', 'y', 'z']
>>>list2 = [1, 2, 3]
>>>list1.extend (list2)
>>>print list1
['x', 'y', 'z', 1, 2, 3]           # Output
```

3. **Sort method :** This method arranges the list in ascending order.

For example :

```
>>>list = [4, 2, 5, 8, 1, 9]
>>>list.sort ( )
>>>print list
[1, 2, 4, 5, 8, 9]                 # Output
```

PART-4

Python Data Structure : Mutable Sequences, List Comprehension, Sets and Dictionaries.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.22. What are mutable sequences ? Discuss with example.

Answer

1. Python represents all its data as objects. Mutability of object is determined by its type.
2. Some of these objects like lists and dictionaries are mutable, meaning we can change their content without changing their identity.
3. Other objects like integers, floats, strings and tuples are immutable, meaning we cannot change their contents.

4. Dictionaries are mutable in Python :

a. Dictionaries in Python are mutable.

b. The values in a dictionary can be changed, added or deleted.

c. If the key is present in the dictionary, then the associated value with that key is updated or changed; otherwise a new key : value pair is added.

For example :

```
>>> dict1 = {'name' : 'Akash', 'age' : 27}
>>> dict1['age'] = 30   #   updating a value
>>> print dict
{'age' : 30, 'name': 'Akash'}   #   Output
>>> dict1['address'] = 'Alaska'   #   adding a key : value
>>>print dict1
{'age': 30, 'name': 'Akash', 'address': 'Alaska'}   #   Output
```

In the given example, we tried to reassign the value '30' to the key 'age', Python interpreter first searches the key in the dictionary and then update it. Hence, the value of 'age' is updated to 30. However, in the next statement, it does not find the key 'address'; hence, the key: value 'address' : 'Alaska' is added to the dictionary.

5. Strings are immutable :

a. String are immutable which means that we cannot change any element of a string.

b. If we want to change an element of a string, we have to create a new string.

For example :

```
>>> var = 'hello Python'
>>> var [0] = 'p'
```

Output :

Type error : 'str' object does not support item assignment

Here, we try to change the 0^{th} index of the string to a character *p*, but the python interpreter generates an error.

Now, the solution to this problem is to generator a new string rather than change the old string.

For example :

```
>>> var = 'hello Python'
>>> new_var = 'p' + var[1 :]
>>> print new_var
pello Python   #   Output
```

In the given example, we cut the slice from the original string and concatenate it with the character we want to insert in the string. It does not have any effect on the original string.

6. **Lists are mutable :**
   a. Lists are mutable means that the value of any element inside the list can be changed at any point of time.
   b. The elements of the list are accessible with their index value.
   c. The index always starts with 0 and ends with $n - 1$, if the list contains n elements.
   d. The syntax for accessing the elements of a list is the same as in the case of a string. We use square brackets around the variable and index number.

For example :

```
>>> list = [10, 20, 30, 40]
>>> list [1]
20  #  Output
```

In the given example, we access the 2nd element of the list that has 1 as index number and the interpreter prints 20.

Now, if we want to change a value in the list given in the example :

For example :

```
>>> list [3] = 50
>>> print list
[10, 20, 30, 50]  #  Output
```

Note that the value of the 4th element is changed to 50.

Que 3.23. Define the term list comprehension.

Answer

1. List comprehension is used to create a new list from existing sequences.
2. It is a tool for transforming a given list into another list.
3. Using list comprehension, we can replace the loop with a single expression that produces the same result.
4. The syntax of list comprehension is based on set builder notation in mathematics.
5. Set builder notation is a notation is a mathematical notation for describing a set by stating the property that its members should satisfy. The syntax is

   [<expression> for <element> in <sequence> if <conditional>]

The syntax is read as "Compute the expression for each element in the sequence, if the conditional is true".

For example :

```
>>> List1 = [10, 20, 30, 40, 50]
>>> List1
      [10, 20, 30, 40, 50]
>>> for i in range (0, len(List1)) :
      List1 [i] = List1[i] + 10
>>> List1
[20, 30, 40, 50, 60]
```

Without list comprehension

```
>>> List1 = [10, 20, 30, 40, 50]
>>> List1= [x + 10 for x in List1]
>>> List1
[20, 30, 40, 50, 60]
```

Using list comprehension

5. In the given example, the output for both without list comprehension and using list comprehension is the same.
6. The use of list comprehension requires lesser code and also runs faster.
7. From above example we can say that list comprehension contains :
   a. An input sequence
   b. A variable referencing the input sequence
   c. An optional expression
   d. An output expression or output variable

Que 3.24. What do you mean by sets ? Explain the operations performed on sets.

Answer

1. In Python we also have one data type which is an unordered collection of data known as set.
2. A set does not contain any duplicate values or elements.
3. **Operations performed on sets are :**
   i. **Union :** Union operation performed on two sets returns all the elements from both the sets. It is performed by using and operator.
   ii. **Intersection :** Intersection operation performed on two sets returns all the element which are common or in both the sets. It is performed by using '|' operator.
   iii. **Difference :** Difference operation performed on two sets set1 and set2 returns the elements which are present on set1 but not in set2. It is performed by using '–' operator.
   iv. **Symmetric difference :** Symmetric difference operation performed on two sets returns the element which are present in either set1 or set2 but not in both. It is performed by using ^ operator.

For example :

```
# Defining sets
>>> set1 = set([1, 2, 4, 1, 2, 8, 5, 4])
>>> set2 = set ([1, 9, 3, 2, 5])
>>> print set1   #   Printing set
set([8, 1, 2, 4, 5])
>>> print set2
set([1, 2, 3, 5, 9])   #   Output
>>> intersection = set1 & set2   #   intersection of set1 and set2
>>> print intersection
set([1, 2, 5])   #   Output
>>> union = set1 | set2   #   Union of set1 and set2
>>> print union
set ([1, 2, 3, 4, 5, 8, 9])   #   Output
>>> difference = set1 – set2   # Difference of set 1 and set2
>>> print difference
set ([8, 4])   #   Output
>>> symm_diff = set1 ^ set2 # symmetric difference of set1 and set2
>>> print symm_diff
set ([3, 4, 8, 9])   #   Output
```

Que 3.25. What is dictionary in Python ?

Answer

1. The Python dictionary is an unordered collection of items or elements.
2. All other compound data types in Python have only values as their elements or items whereas the dictionary has a key : value pair. Each value is associated with a key.
3. In dictionary, we have keys and they can be of any type.
4. Dictionary is said to be a mapping between some set of keys and values.
5. The mapping of a key and value is called as a key-value pair and together they are called one item or element.
6. A key and its value are separated by a colon (:) between them.
7. The items or elements in a dictionary are separated by commas and all the elements must be enclosed in curly braces.
8. A pair of curly braces with no values in between is known as an empty dictionary.
9. The values in a dictionary can be duplicated, but the keys in the dictionary are unique.

Que 3.26. How do we create a dictionary in Python ?

Answer

1. Creating a dictionary is simple in Python.
2. The values in a dictionary can be of any data type, but the keys must be of immutable data types (such as string, number or tuple).

For example :

1. **Empty dictionary :**

```
>>> dict1 = {}
>>>print dict1
{}  #  Output
```

2. **Dictionary with integer keys :**

```
>>> dict1 = {1 : 'red', 2 : 'yellow', 3 : 'green'}
>>>print dict1
{1 : 'red', 2 : 'yellow', 3 : 'green'}  #  Output
```

3. **Dictionary with mixed keys :**

```
>>> dict1 = {'name' : 'Rahul', 3 : ['Hello', 2, 3]}
>>>print dict1
{3 : ['Hello', 2, 3], 'name' : 'Rahul'}  #  Output
```

Que 3.27. Explain the operations performed on dictionary with example.

Answer

Operations in dictionary :

1. **Traversing :**
   a. Traversing in dictionary is done on the basis of keys.
   b. For traversing, for loop is used, which iterates over the keys in the dictionary and prints the corresponding values using keys.

For example :

```
>>>defprint_dict(d):
...     for c in d:
...         print c,d[c]
>>> dictl = {1 : 'a', 2 : 'b', 3 : 'c' 4 : 'd'}
>>> print_dict(dict1)
```

Output :

```
1 a
2 b
3 c
4 d
```

This example prints the key: value pairs in the dictionary dict.

2. **Membership :**
   i. Using the membership operator (in and not in), we can test whether a key is in the dictionary or not.
   ii. In operator takes an input key and finds the key in the dictionary.

iii. If the key is found, then it returns true, otherwise, false.

For example :

```
>>>cubes = {1 : 1, 2 : 8, 3 : 27, 4 : 64, 5 : 125, 6 : 216}
>>> 3 in cubes
True                                    # Output
>>> 17 not in cubes
True                                    # Output
```

Que 3.28. Write some built in dictionary methods used in Python with example.

Answer

There are some built-in dictionary methods which are included in Python given in Table 3.28.1.

Table 3.28.1. Built-in dictionary methods.

S. No.	Function	Description
1.	all (dict)	It is a Boolean type function, which returns true if all keys of dictionary are true (or the dictionary is empty).
2.	any (dict)	It is also a Boolean type function, which returns true if any key of the dictionary is true. It returns false if the dictionary is empty.
3.	len (dict)	It returns the number of items (length) in the dictionary.
4.	cmp (dict1, dict2)	It compares the items of two dictionaries.
5.	sorted (dict)	It returns the sorted list of keys.
6.	dict.clear ( )	It deletes all the items in a dictionary at once.
7.	dict. copy ( )	It returns a copy of the dictionary.
8.	dict.get (key, default = None)	For key key, returns value or default if key not in dictionary.
9.	dict.items ( )	It returns a list of entire key : value pair of dictionary.
10.	dict.keys ( )	It returns the list of all the keys in dictionary.
11.	dict.update (dict2)	It adds the items from dict2 to dict.
12.	dict.values ( )	It returns all the values in the dictionary.

For example :

```
>>>cubes = {1 : 1, 2 : 8, 3 : 27, 4 : 64, 5 : 125, 6 : 216}
>>>all (cubes)
True                                    # Output
>>>any (cubes)
True                                    # Output
>>>len (cubes)
6                                       # Output
>>>sorted (cubes)
[1, 2, 3, 4, 5, 6]                      # Output
>>>str (cubes)
'{1 : 1, 2 : 8, 3 : 27, 4 : 64, 5 : 125, 6 : 216}'          # Output
```

PART-5

Higher Order Functions : Treat Functions as First-Class Objects, Lambda Expressions.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.29. Explain treat functions as first-class objects in detail.

Answer

1. In Python both functions and classes are treated as objects which are called as first class objects, allowing them to be manipulated in the same ways as built-in data types.
2. By definition, first class objects are the following which is :
   a. Created at runtime
   b. Assigned as a variable or in a data structure
   c. Passed as an argument to a function
   d. Returned as the result of a function
3. For example, let create and test a function, then read its__doc__and check its type.

```
>>> def factorial(n):
... '''returns n!'''
        ... return 1 if n < 2 else n * factorial(n – 1)
...
>>> factorial (42)
```

```
1405006117752879898543142 60$2445115699363840000000000
>>> factorial. __doc __
'returns n!'
>>> type(factorial)
<class 'function'>
```

4. The console session in example shows that Python functions are objects. Here we create a function, call it, read its__doc__attribute, and check that the function object itself is an instance of the function class.

Que 3.30. Explain Lambda expression.

Answer

1. Lambda expressions is used to create the anonymous function.
2. The anonymous functions are the functions created using a lambda keyword.
3. They are not defined by using def keyword. For this reason, they are called anonymous functions.
4. We can pass any number of arguments to a lambda form functions, but still they return only one value in the form of expression.
5. An anonymous function cannot directly call print command as the lambda needs an expression.
6. It cannot access the parameters that are not defined in its own namespace.
7. An anonymous function is a single line statement function.
8. **Syntax :**

   lambda [arg1 [,arg2, ..., argn]] : expression

   The syntax of the lambda function is a single statement

   For example :

```
# function definition here
>>>mult = lambda val1, val2 : val1*val2 ;
# function call here
>>> print "value :",  mult(20,40)
Value : 800   # Output
```

In the given example, the lambda function is defined with two arguments val1 and val2. The expression val1*va12 does the multiplication of the two values. Now, in function call, we can directly call the mult function with two valid values as arguments and produce the output as shown in given example.

Que 3.31. **Explain higher order function with respect to lambda expression. Write a Python code to count occurrences of an element in a list.** **AKTU 2019-20, Marks 10**

Answer

1. Reduce(), filter(), map() are higher order functions used in Python.
2. Lambda definition does not include a "return" statement, it always contains an expression which is returned.
3. We can also put a lambda definition anywhere a function is expected, and we do not have to assign it to a variable at all.
4. Lambda functions can be used along with built-in higher order functions like filter(), map() and reduce().

Use of lambda with filter() :

1. The filter() function in Python takes in a function and a list as arguments.
2. This function helps to filter out all the elements of a sequence "sequence", for which the function returns true.

For example : Python program that returns the odd numbers from an input list :

```
# Python code to illustrate filter() with lambda
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambdax: (x%2!=0), li))
print(final_list)
```

Output :

```
[5, 7, 97, 77, 23, 73, 61]
```

Use of lambda() with reduce() :

1. The reduce() function in Python takes in a function and a list as argument.
2. The function is called with a lambda function and a list and a new reduced result is returned. This performs a repetitive operation over the pairs of the list.
3. This is a part of functools module.

For example :

```
# Python code to illustrate reduce() with lambda() to get sum of a list
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y : x + y), li)
print (sum)
```

Output :

```
193
```

Here the results of previous two elements are added to the next element and this goes on till the end of the list like (((((5+8)+10)+20)+50)+100).

Program to count occurrences of an element in a list :

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# count element 'i'
count = vowels.count('i')
# print count
print('The count of i is:', count)
# count element 'p'
count = vowels.count('p')
# print count
print('The count of p is:', count)
```

Output :

The count of i is: 2

The count of p is: 0

Que 3.32. **Explain unpacking sequences, mutable sequences, and list comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.**

AKTU 2019-20, Marks 10

Answer

Unpacking sequences : Refer Q. 3.16, Page 3–14T, Unit-3.

Mutable sequences : Refer Q. 3.22, Page 3–19T, Unit-3.

List comprehension : Refer Q. 3.23, Page 3–21T, Unit-3.

Program :

```
# Initializing list of dictionaries
lis = [{"name" : "Nandini", "age" : 20},
{"name" : "Manjeet", "age" : 20 },
{"name" : "Nikhil" , "age" : 19 }]
# using sorted and lambda to print list sorted by age
print "The list printed sorting by age :"
print sorted(lis, key = lambda i: i['age'])
print ("\r")
# using sorted and lambda to print list sorted by both age and name
print "The list printed sorting by age and name:"
print sorted(lis, key = lambda i: (i['age'], i['name']))
print ("\r")
# using sorted and lambda to print list sorted
# by age in descending order
print "The list printed sorting by age in descending order:"
print sorted(lis, key = lambda i: i['age'],reverse=True)
```

Output :
The list printed sorting by age:
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}]
The list printed sorting by age and name :
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Manjeet'}, {'age': 20, 'name': 'Nandini'}]
The list printed sorting by age in descending order:
[{'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}, {'age': 19, 'name': 'Nikhil'}]

Que 3.33. **Write a function in find the HCF of some given numbers.**

Answer

```
>>>def hcf (a, b) :
    if a > b :
        small = b
    else :
        small = a
    for i in range (1, small + 1) :
        if (a % i == 0) and (b % i == 0)) :
            hcf = i
    return hcf
>>> hcf (20, 40)
20                                  #Output
>>>hcf (529, 456)
1                                   #Output
```

Que 3.34. **Write a function to find the ASCII value of the character.**

Answer

```
>>>def ascii_val_of (a) :
    print ("The ASCII value of '" + a + "' is", ord (a))
>>>ascii_val_of ('A')
("The ASCII value of 'A' is", 65)       #Output
>>>ascii_val_of (' ')                   #Finding ASCII value of space
("The ASCII value of ' ' is", 32)       #Output
```

Que 3.35. **Write a function to convert a decimal number to its binary, octal and hexadecimal equivalents.**

Answer

```
>>>def bin_oct_hex (a) :
      print (bin (a), "binary equivalent")
      print (oct (a), "octal equivalent")
      print (hex (a), "hexadecimal equivalent")
>>>bin_oct_hex (10)                    #Finding binary, octal, hex value of 10
('0b1010', 'binary equivalent')        #Output
('012', 'octal equivalent')
('0xa', 'hexadecimal equivalent')
```