Strings, Lists, Tuples and Loops

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

29 March 2021

Introduction to Strings

- Strings are amongst the most popular types in Python.
- We can create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable.
- For example –
  var1 = ‘Hello World!’
  var2 = “Python Programming”

Accessing Values in Strings

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.
- To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.
- For example –

```
#!/usr/bin/python
var1 = ‘Hello World!’
var2 = “Python Programming”
print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Updating Values in Strings

- You can "update" an existing string by (re)assigning a variable to another string.
- The new value can be related to its previous value or to a completely different string altogether.
- For example –

```
#!/usr/bin/python
var1 = ‘Hello World!’
print "Updated String :- ", var1[:6] + 'Python'
```

String Special Operators

- Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then –

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell

String Formatting Operator

- This operator (%) is unique to strings and makes up for the pack of having functions from C's printf() family.
- For example –
  #!/usr/bin/python
  print "My name is %s and age is %d." % (‘ABC', 15)

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i or %d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x or %X	hexadecimal integer (lowercase and uppercase letters)
%e or %E	exponential notation (lowercase and uppercase)
%f	floating point real number

Raw Strings

- Raw strings do not treat the backslash (\) as a special character at all.
- Every character you put into a raw string stays the way you wrote it.
- For example –

```
#!/usr/bin/python
print r‘C:\\somepath’
```

String Methods

- **capitalize()** – Capitalizes first letter of string
- **center(width, fillchar)** – Returns a space-padded string with the original string centered to a total of width columns.
- **count(str, beg=0, end=len(string))** – Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
- **encode(encoding='UTF-8', errors='strict')** – Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
- **decode(encoding='UTF-8', errors='strict')** – Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.

String Methods

- **endswith(suffix, beg=0, end=len(string)) -** Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
- **find(str, beg=0, end=len(string)) -** Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
- **len(string)** - Returns the length of the string
- **lower()** - Converts all uppercase letters in string to lowercase.
- **upper()** – Converts lowercase letters in string to uppercase.

String Methods

- **lstrip() -** Removes all leading whitespace in string.
- **rstrip()** – Removes all trailing whitespace in string.
- **split(str=“”, num=string.count(str))** – Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
- **max(str)** – Returns the max alphabetical character from the string str.
- **min(str)** – Returns the min alphabetical character from the string str.

Introduction to Lists

- The list is the most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.
- Items in a list need not be of the same type.
- Creating a list is as simple as putting different comma-separated values between square brackets.
- For example –
  list1 = ['physics', 'chemistry', 1997, 2000]
  list2 = [1, 2, 3, 4, 5]
  list3 = ["a", "b", "c", "d"]
- Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists

- To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.
- For example –

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

Updating Values in Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.
- For example –

```
#!/usr/bin/python
list = ['physics', 'chemistry', 1997, 2000]
print "Value available at index 2 : “
print list[2]
list[2] = 2001
print "New value available at index 2 : “
print list[2]
```

Basic List Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership

for x in [1, 2, 3]:
    print x,
	1 2 3	Iteration

Indexing, Slicing and Matrices

- Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.
- Assuming following input –
  L = ['spam', 'Spam', 'SPAM!']

Python Expression	Results	Description
L[2]	SPAM!	Offsets start at zero
L[-2]	Spam	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

List Methods

- **cmp(list1, list2)** – Compares elements of both lists.
- **len(list) -** Gives the total length of the list.
- **max(list) -** Returns item from the list with max value.
- **min(list) -** Returns item from the list with min value.
- **list(seq) -** Converts a tuple into list.
- **list.append(obj) -** Appends object obj to list
- **list.count(obj) -** Returns count of how many times obj occurs in list
- **list.extend(seq) -** Appends the contents of seq to list

List Methods

- **list.insert(index, obj)** – Inserts object obj into list at offset index.
- **list.pop(obj=list[-1]) -** Removes and returns last object obj from list
- **list.remove(obj) -** Removes object obj from list
- **list.reverse() -** Reverses objects of list in place
- **list.sort([func])** – Sorts objects of list, use compare function if given

Introduction to Tuples

- A tuple is a collection of objects which is ordered and immutable.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
- Creating a tuple is as simple as putting different comma-separated values.
- Optionally you can put these comma-separated values between parentheses also.

Introduction to Tuples

- For example –
  tup1 = ('physics', 'chemistry', 1997, 2000)
  tup2 = (1, 2, 3, 4, 5)
  tup3 = "a", "b", "c", "d“
- The empty tuple is written as two parentheses containing nothing –
  tup1 = ()
- To write a tuple containing a single value you have to include a comma, even though there is only one value –
  tup1 = (50,)
- Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

- To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.
- For example –

```
#!/usr/bin/python
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership

for x in (1, 2, 3):
    print x,
	1 2 3	Iteration

Basic Tuples Operations

- Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings.
- Assuming following input –
  L = ('spam', 'Spam', 'SPAM!')

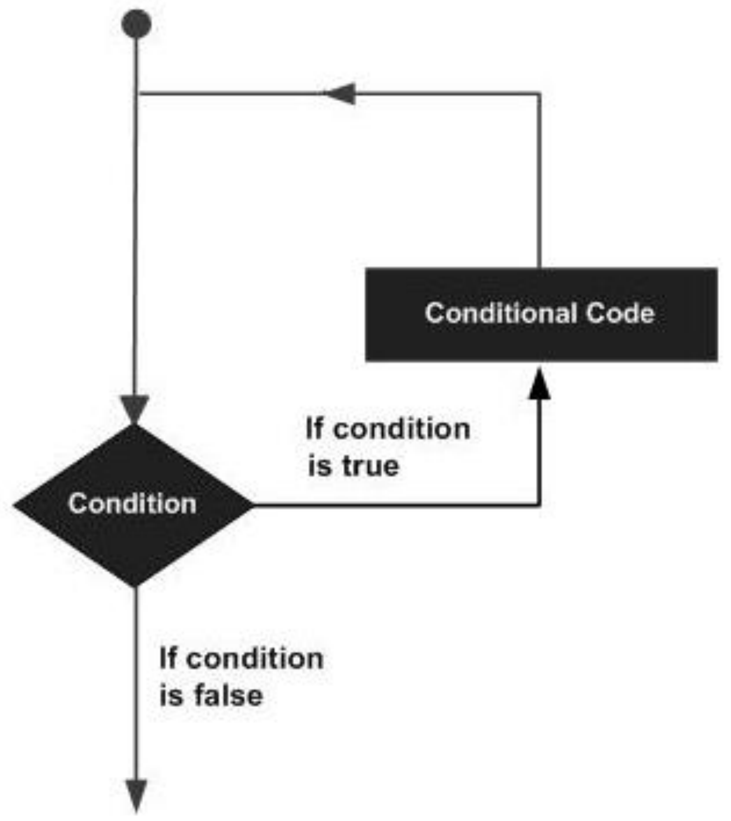
Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Tuple Methods

- **cmp(tuple1, tuple2)** – Compares elements of both tuples.
- **len(tuple) -** Gives the total length of the tuple.
- **max(tuple) -** Returns item from the tuple with max value.
- **min(tuple) -** Returns item from the tuple with min value.
- **tuple(seq) -** Converts a list into tuple.

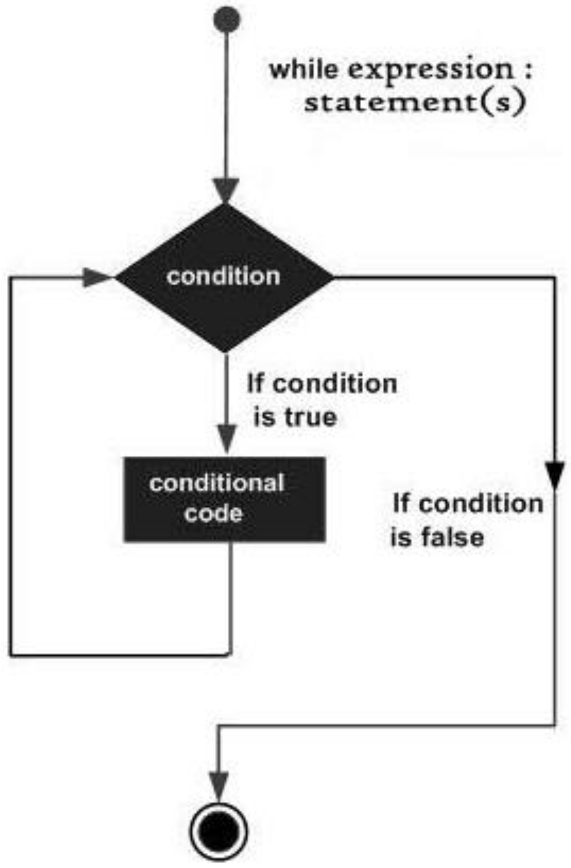
Loops

- A loop statement allows us to execute a statement or group of statements multiple times.
- The following diagram illustrates a loop statement –



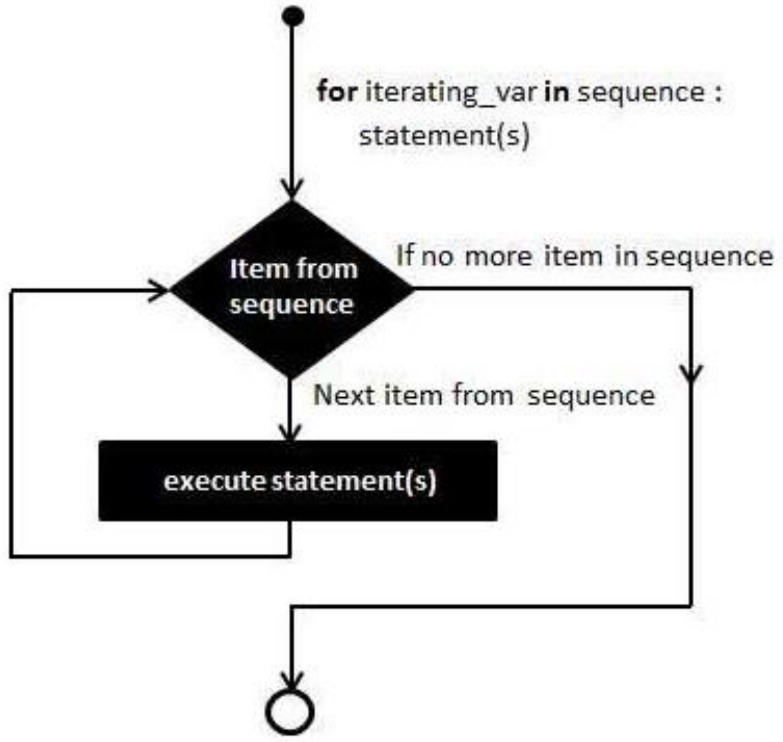
while Loop

- Repeats a statement or group of statements while a given condition is TRUE.
- It tests the condition before executing the loop body.
- Syntax:
  while expression:
  statement(s)



for Loop

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- Syntax:
  for iterating_var in sequence:
  statement(s)



Thank you!