CSC552
Python Programming

Prepared by

Dr. Ramesh Saha

Assistant professor

Computer Science and Engineering

Indian Institute of Information Technology (IIIT), Sonepat

SEQUENCE

- A sequence is an **ordered collection of items**, indexed by positive integers.
- It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.
- There are three types of sequence data type available in Python, they are

- **Strings**
- **Lists**
- **Tuples**

STRINGS

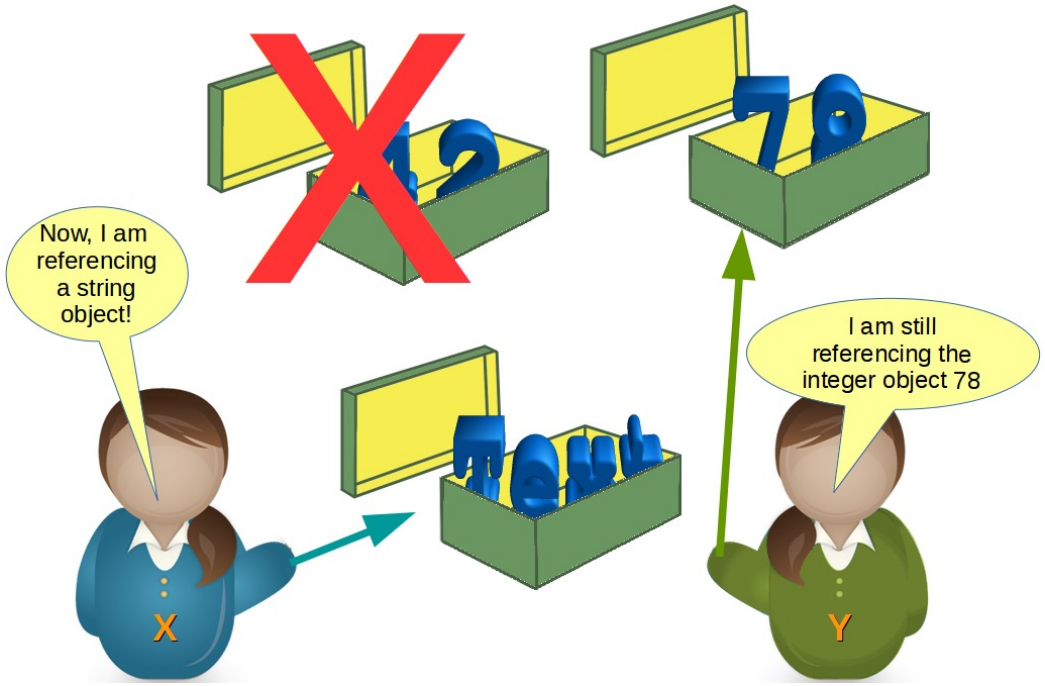
- A String in Python consists of a **series or sequence of characters - letters, numbers, and special characters.**
- Strings are marked by quotes:
  - single quotes (' '), Eg, **'This a string in single quotes'**
  - double quotes (" "), Eg, **"This a string in double quotes“**
  - triple quotes ("""“ """), Eg, This is a paragraph. It is made up of multiple lines and sentences."""
- Individual character in a string is accessed using a subscript **(index).**
- Characters can be accessed using indexing and slicing operations

Strings are immutable i.e. the contents of the string cannot be changed after it is created.

STRINGS

- Strings and Operations

1. **Indexing**
2. **Concatenation**
3. **Slicing**
4. **Repetition**
5. **Membership**



String Special Operators

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

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
in	**Membership** - Returns true if a character exists in the given string	H in a will give 1
not in	**Membership** - Returns true if a character does not exist in the given string	M not in a will give 1
r/R	**Raw String** - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' prints \n and print R'\n'prints \n
%	Format - Performs String formatting	See at next section

Indexing

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 −

```
var1 = 'Hello World!' var2 = "Python Programming"
print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

When the above code is executed, it produces the following result −
var1[0]: H
var2[1:5]: ytho

Indexing

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "Hello, World!"
print(b[-5:-2])
```

Output: orl

Slicing

You can return a range of characters by using the slice syntax.
Specify the start index and the end index, separated by a colon, to return a part of the string.

```
b = "Hello, World!"
print(b[2:5])                    Output:  llo
```

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 5 (not included):

```
b = "Hello, World!"
print(b[:5])                     Output: Hello
```

Slice To the End

By leaving out the *end* index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

```
b = "Hello, World!"
print(b[2:])                     Output: llo, World!
```

Concatenation

Updating 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 −

```
var1 = 'Hello World!'
print "Updated String :- ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result –

Updated String :- Hello Python

String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family.

Following is a simple example

```
print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

When the above code is executed, it produces the following result −

My name is Zara and weight is 21 kg!

Membership

- Returns true if a character exists in the given string or character.
- To check if a certain phrase or character is present in a string, we can use the keyword **in**.
- Example:

```
txt = "The best things in life are free!"
print("free" in txt)
```

- To check if a certain phrase or character is NOT present in a string, we can use the keyword **not in**.
- Example:

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

Upper Case

Example

The upper() method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())            //HELLO, WORLD!
```

Lower Case

Example

The lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())            //hello, world!
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip())            # returns "Hello, World!"
```

Python - Modify Strings

Replace String

Example

The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(","))                    # returns ['Hello', ' World!']
```

Here is the list of complete set of symbols which can be used along with %

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

Lists

- List is an **ordered sequence of items**. Values in the list are called elements / items.
- It can be written as a list of terms (values) between **square brackets[ ]**.
- It is comma-separated.
- Items in the lists can be of different data types.
- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists

- List is an **ordered sequence of items**. Values in the list are called elements / items.
- It can be written as a list of terms (values) between **square brackets[ ]**.
- It is comma-separated.
- Items in the lists can be of different data types.
- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

Lists

Ordered: When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.

List Methods : Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python List append() Method

Example

Add an element to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
print(fruits)                #['apple', 'banana', 'cherry', 'orange’]
```

```
a = ["apple", "banana", "cherry"]
b = ["Ford", "BMW", "Volvo"]
a.append(b)
print(a)          #['apple', 'banana', 'cherry', ["Ford", "BMW", "Volvo"]]
```

Python List clear() Method

Example

Remove all elements from the fruits list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']
fruits.clear()
```

Python List copy() Method

Example

Copy the fruits list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']
x = fruits.copy()                    #['apple', 'banana', 'cherry']
```

Python List count() Method

Return the number of times the value "cherry" appears in the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.count("cherry")           #1
```

Parameter Values

Parameter	Description
value	Required. Any type (string, number, list, tuple, etc.). The value to search for.

Return the number of times the value 9 appears int the list:

```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]
x = points.count(9)
print(x)                    #2
```

Python List extend() Method

Add the elements of cars to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
cars = ['Ford', 'BMW', 'Volvo']
fruits.extend(cars)
print(fruits)       #['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']

fruits = ['apple', 'banana', 'cherry']
points = (1, 4, 5, 9)
fruits.extend(points)
print(fruits)               #['apple', 'banana', 'cherry', 1, 4, 5, 9]
```

Python List index() Method

What is the position of the value "cherry":

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.index("cherry")
print(x)                              #2
```

```
fruits = [4, 55, 64, 32, 16, 32]
x = fruits.index(32)
print(x)                              #3
```

Python List insert() Method

```
fruits = ['apple', 'banana', 'cherry']
fruits.insert(1, "orange")
print(fruits)                          #['apple', 'orange', 'banana', 'cherry’]
```

Python List pop() Method

Remove the second element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.pop(1)
print(fruits)                  #['apple', 'cherry’]
```

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.pop(1)
print(x)                       #banana
```

Python List remove() Method

Remove the "banana" element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")
print(fruits)                    #['apple', 'cherry’]
```

Python List reverse() Method

Reverse the order of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.reverse()
print(fruits)                    #['cherry', 'banana', 'apple’]
```

Python List sort() Method

Sort the list alphabetically:

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort()
print(cars)                      #['BMW', 'Ford', 'Volvo']
```

Sort the list descending:

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort(reverse=True)
print(cars)                                    #['Volvo', 'Ford', 'BMW']
```

List Length

To determine how many items a list has, use the len() function:

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))          #3
```

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
print(list1)                  ['apple', 'banana', 'cherry']
print(list2)                  [1, 5, 7, 9, 3]
print(list3)                  [True, False, False]
```

```
list1 = ["abc", 34, True, 40, "male"]
print(list1)          #['abc', 34, True, 40, 'male']
```

type()

From Python's perspective, lists are defined as objects with the data type 'list’

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))                    #<class 'list’>
```

End