Lists in Python

Dr. Amer Almahdawi

Lists (Sequence)

- The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
- There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.
- The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Lists

- Creating a list is as simple as putting different comma-separated values between square brackets. For example:
- List1 = [‘Computer’, ‘Physics’, ‘Chemistry’, 1997, 2000]
- List2 = [1, 2, 3, 4, 5, 6]
- List3 = [“a”, “b”, “c”, “d”]
- Accessing Values in Lists
- Print (list1[0])
- Computer
- Print (list2[1:5])
- [2, 3, 4, 5]

Lists

- Updating 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:
- List1 = [‘Computer’, ‘Physics’, ‘Chemistry’, 1997, 2000]
- Print (“ Available value at index 3 is: “, list1[3])
- Available value at index 3 is: 1997
- List1[3] = 2001
- Print (“ Available value at index 3 is: “, list1[3])
- Available value at index 3 is: 2001

Lists

- Delete List Elements
- List1 = ['physics', 'chemistry', 'computer', 2000]
- del list1[3]
- List1 = ['physics', 'chemistry', 'computer']

Basic List Operations

- Length:
- Len([1,2,3]) = 3
- Concatenation:
- [1,2,3] + [4,5,6] = [1,2,3,4,5,6]
- Repetition
- [‘Hi’]*4 = ['Hi', 'Hi', 'Hi', 'Hi']
- Membership
- 3 in [1,2,3] =True
- Iteration
- For x in [1,2,3]: print (x)

1
2
3

Basic List Operations

- L = [‘car’, ‘Car’, ‘CAR!’]
- L[2] = CAR!
- L[-2] = Car
- L[1:] = [‘Car’, ‘CAR!’]
- alist=['Ahmed', 2, 123, 'Jeep']
- alist.append(2000)
- ['Ahmed', 2, 123, 'Jeep', 2000]
- alist.count(2)
- 1

Basic List Operations

- alist= ['Ahmed', 2, 123, 'Jeep', 2000]
- alist.index(123)
- 2
- alist.insert(3, 'Laptop')
- alist=['Ahmed', 2, 123, 'Laptop', 'Jeep', 2000]
- alist.pop()
- 2000
- alist=['Ahmed', 2, 123, 'Laptop', 'Jeep']
- alist.remove(2)
- alist=['Ahmed', 123, 'Laptop', 'Jeep']

Basic List Operations

- alist=['Ahmed', 123, 'Laptop', 'Jeep']
- alist.reverse()
- alist=['Jeep', 'Laptop', 123, 'Ahmed']
- alist=['Jeep', 'Laptop', 'Ahmed']
- alist.sort()
- alist=['Ahmed', 'Jeep', 'Laptop']

Python Classes, objects

- Python is an object-oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor.
- Create a Class
- To create a class, use the keyword class:
- Create a class named MyClass, with a property named x:
- class Myclass:
- x = 5
- print(Myclass)
- <class '__main__.Myclass'>

Python Classes, objects

- Create Object
- Create an object named p1, and print the value of x:
- p1 = Myclass()
- print(p1.x)
- 5

Python Classes, objects

- The __init__() Function
- The examples above are classes and objects in their simplest form and are not really useful in real-life applications.
- To understand the meaning of classes we have to understand the built-in __init__() function.
- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Python Classes, objects

- **Create a class named Person, use the __init__() function to assign values for name and age:**
- class Person:
- def __init__(self, name,age):
- self.name = name
- self.age = age
- p1 = Person("John",36)
- print(p1.name)
- print(p1.age)
- John
- 36

Python Classes, objects

- **Insert a function that prints a greeting, and execute it on the p1 object:**
- class Person:
- def __init__(self, name,age):
- self.name = name
- self.age = age
- def myfunc(self):
- print("Hello my name is " + self.name)
- p1 = Person("John",36)
- p1.myfunc()
- Hello my name is John

Python Classes, objects

- Modify Object Properties
- P1.age = 40
- Delete Object Properties
- del p1.age
- Delete Objects
- del p1

Thank you