Python - Data Structure

Data structures deal with how the data is organized and held in the memory when a program processes it. It is important to note that the data that is stored in the disk as part of persistent storages (like relational tables) are not referred as data structure here.

An Algorithm is step by step set of instruction to process the data for a specific purpose. So an algorithm utilizes various data structures in a logical way to solve a specific computing problem.

**General Data Structures**

The various data structures in computer science are divided broadly into two categories shown below. We will discuss about each of the below data structures in detail in subsequent chapters.

**Liner Data Structures**

These are the data structures which store the data elements in a sequential manner.

* **Array:** It is a sequential arrangement of data elements paired with the index of the data element.
* **Linked List:** Each data element contains a link to another element along with the data present in it.
* **Stack:** It is a data structure which follows only to specific order of operation. LIFO(last in First Out) or FILO(First in Last Out).
* **Queue:** It is similar to Stack but the order of operation is only FIFO(First In First Out).
* **Matrix:** It is two dimensional data structure in which the data element is referred by a pair of indices.

**Non-Liner Data Structures**

These are the data structures in which there is no sequential linking of data elements. Any pair or group of data elements can be linked to each other and can be accessed without a strict sequence.

* **Binary Tree:** It is a data structure where each data element can be connected to maximum two other data elements and it starts with a root node.
* **Heap:** It is a special case of Tree data structure where the data in the parent node is either strictly greater than/ equal to the child nodes or strictly less than it’s child nodes.
* **Hash Table:** It is a data structure which is made of arrays associated with each other using a hash function. It retrieves values using keys rather than index from a data element.
* **Graph: .**It is an arrangement of vertices and nodes where some of the nodes are connected to each other through links.

**Python Specific Data Structures**

These data structures are specific to python language and they give greater flexibility in storing different types of data and faster processing in python environment.

* **List:** It is similar to array with the exception that the data elements can be of different data types. You can have both numeric and string data in a python list.
* **Tuple:** Tuples are similar to lists but they are immutable which means the values in a tuple cannot be modified they can only be read.
* **Dictionary:** The dictionary contains Key-value pairs as its data elements.

# Python - Arrays

Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

* **Element**− Each item stored in an array is called an element.
* **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.

## Array Representation

Arrays can be declared in various ways in different languages. Below is an illustration.



As per the above illustration, following are the important points to be considered.

* Index starts with 0.
* Array length is 10 which means it can store 10 elements.
* Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

## Basic Operations

Following are the basic operations supported by an array.

* **Traverse** − print all the array elements one by one.
* **Insertion** − Adds an element at the given index.
* **Deletion** − Deletes an element at the given index.
* **Search** − Searches an element using the given index or by the value.
* **Update** − Updates an element at the given index.

Array is created in Python by importing array module to the python program. Then the array is declared as shown eblow.

from array import \*

arrayName = array(typecode, [Initializers])

Typecode are the codes that are used to define the type of value the array will hold. Some common typecodes used are:

|  |  |
| --- | --- |
| **Typecode** | **Value** |
| b | Represents signed integer of size 1 byte/td> |
| B | Represents unsigned integer of size 1 byte |
| c | Represents character of size 1 byte |
| i | Represents signed integer of size 2 bytes |
| I | Represents unsigned integer of size 2 bytes |
| f | Represents floating point of size 4 bytes |
| d | Represents floating point of size 8 bytes |

Before lookign at various array operations lets create and print an array using python.

The below code creates an array named array1.

from array import \*

array1 = array('i', [10,20,30,40,50])

for x in array1:

print(x)

When we compile and execute the above program, it produces the following result −

## Accessing Array Element

We can access each element of an array using the index of the element. The below code shows how

from array import \*

array1 = array('i', [10,20,30,40,50])

print (array1[0])

print (array1[2])

## Insertion Operation

Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Here, we add a data element at the middle of the array using the python in-built insert() method.

from array import \*

array1 = array('i', [10,20,30,40,50])

array1.insert(1,60)

for x in array1:

print(x)

## Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Here, we remove a data element at the middle of the array using the python in-built remove() method.

from array import \*

array1 = array('i', [10,20,30,40,50])

array1.remove(40)

for x in array1:

print(x)

When we compile and execute the above program, it produces the following result which shows the element is removed form the array.

## Search Operation

You can perform a search for an array element based on its value or its index.

Here, we search a data element using the python in-built index() method.

from array import \*

array1 = array('i', [10,20,30,40,50])

print (array1.index(40))

## Update Operation

Update operation refers to updating an existing element from the array at a given index.

Here, we simply reassign a new value to the desired index we want to update.

from array import \*

array1 = array('i', [10,20,30,40,50])

array1[2] = 80

for x in array1:

print(x)

# Python - Lists

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.

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 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]

## Delete List Elements

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know. For example −

#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000];

print list1

del list1[2];

print "After deleting value at index 2 : "

print list1

## Basic List Operations

Lists respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

|  |  |  |
| --- | --- | --- |
| **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 |

# Python - Tuples

A tuple is a sequence of immutable Python objects. 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. 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];

## Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

#!/usr/bin/python

tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

# Following action is not valid for tuples

# tup1[0] = 100;

# So let's create a new tuple as follows

tup3 = tup1 + tup2;

print tup3;

## Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example −

#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);

print tup;

del tup;

print "After deleting tup : ";

print tup;

## Basic Tuples Operations

Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

|  |  |  |
| --- | --- | --- |
| **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 |

# Python - Dictionary

In Dictionary 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. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

## Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']

## Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

## Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example −

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

## Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

**(a)** More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']

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

dict['Name']: Manni

**(b)** Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']