Python Lists

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List & Assignment

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

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

OUTPUT:

['apple', 'banana', 'cherry']

Negative Indexing

Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

OUTPUT:

Cherry

List Slicing / Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

OUTPUT:

['cherry', 'orange', 'kiwi']

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to "orange":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi",
            "melon", "mango"]
print(thislist[:4])
```

OUTPUT:

['cherry', 'orange', 'kiwi']

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" and to the end:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

OUTPUT:

['cherry', 'orange', 'kiwi', 'melon', 'mango']

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

OUTPUT:

['orange', 'kiwi', 'melon']

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

OUTPUT:

['apple', 'blackcurrant', 'cherry']

Loop Through a List

You can loop through the list items by using a `for` loop:

Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

OUTPUT:

apple

banana

cherry

You will learn more about `for` loops in our Python For Loops Chapter.

LINEAR SEARCHING LISTS

The **in** operator can be used to check if an item is present in the list:

```
if value in L:
    print "list contains", value
```

To get the index of the first matching item, use **index**:

```
i = L.index(value)
```

The **index** method does a linear search, and stops at the first matching item. If no matching item is found, it raises a **ValueError** exception.

```
try:
    i = L.index(value)
except ValueError:
    i = -1 # no match
```

To get the index for all matching items, you can use a loop, and pass in a start index:

```
L = [1,2,3,4,5]
i = -1
try:
    while 1:
        i = L.index(4, i+1)
        print ("match at", i)
except ValueError:
    pass
```

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

OUTPUT:

Yes, 'apple' is in the fruits list

List Length

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

Example

Print the number of items in the list:

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

OUTPUT:

3

Add Items

To add an item to the end of the list, use the `append()` method:

Example

Using the `append()` method to append an item:

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

OUTPUT:

['apple', 'banana', 'cherry', 'orange']

To add an item at the specified index, use the `insert()` method:

Example

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

OUTPUT:

['apple', 'banana', 'cherry', 'orange']

Remove Item

There are several methods to remove items from a list:

Example

The `remove()` method removes the specified item:

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

OUTPUT:

['apple', 'cherry']

Example

The `pop()` method removes the specified index, (or the last item if index is not specified):

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

OUTPUT:

['apple', 'banana']

Example

The `del` keyword removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

OUTPUT:

['banana', 'cherry']

Example

The `del` keyword can also delete the list completely:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

OUTPUT:

Traceback (most recent call last):
File "demo_list_del2.py", line 3, in <module>
print(thislist) #this will cause an error because you have succsesfully deleted "thislist".
NameError: name 'thislist' is not defined

Example

The `clear()` method empties the list:

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

OUTPUT:

[]

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

There are ways to make a copy, one way is to use the built-in List method `copy()`.

Example

Make a copy of a list with the `copy()` method:

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

OUTPUT:

['apple', 'banana', 'cherry']

Another way to make a copy is to use the built-in method `list()`.

Example

Make a copy of a list with the `list()` method:

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

OUTPUT:

['apple', 'banana', 'cherry']

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the `+` operator.

Example

Join two list:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

OUTPUT:

['a', 'b', 'c', 1, 2, 3]

Another way to join two lists are by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

OUTPUT:

['a', 'b', 'c', 1, 2, 3]

Or you can use the `extend()` method, which purpose is to add elements from one list to another list:

Example

Use the `extend()` method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

OUTPUT:

['a', 'b', 'c', 1, 2, 3]

The list() Constructor

It is also possible to use the `list()` constructor to make a new list.

Example

Using the `list()` constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

OUTPUT:

['apple', 'banana', 'cherry']

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 sort() Method

Example

Sort the list alphabetically:

```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort()
```

OUTPUT:

['BMW', 'Ford', 'Volvo']

Definition and Usage

The `sort()` method sorts the list ascending by default.

You can also make a function to decide the sorting criteria(s).

Syntax

```
list.sort(reverse=True|False, key=myFunc)
```

Parameter Values

Parameter	Description
Reverse	Optional. reverse=True will sort the list descending. Default is reverse=False
Key	Optional. A function to specify the sorting criteria(s)

More Examples

Example

Sort the list descending:

```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort(reverse=True)
```

OUTPUT:

['Volvo', 'Ford', 'BMW']

Example

Sort the list by the length of the values:

```
# A function that returns the length of the value:
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(key=myFunc)
```

OUTPUT:

['VW', 'BMW', 'Ford', 'Mitsubishi']

Example

Sort a list of dictionaries based on the "year" value of the dictionaries:

```
# A function that returns the 'year' value:
def myFunc(e):
  return e['year']

cars = [
  {'car': 'Ford', 'year': 2005},
  {'car': 'Mitsubishi', 'year': 2000},
  {'car': 'BMW', 'year': 2019},
  {'car': 'VW', 'year': 2011}
]

cars.sort(key=myFunc)
```

OUTPUT:

[{'car': 'Mitsubishi', 'year': 2000}, {'car': 'Ford', 'year': 2005}, {'car': 'VW', 'year': 2011}, {'car': 'BMW', 'year': 2019}]

Example

Sort the list by the length of the values *and* reversed:

```
# A function that returns the length of the value:
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(reverse=True, key=myFunc)
```

OUTPUT:

['Mitsubishi', 'Ford'', 'BMW', 'VW']

PRIME GENERATION WITH LIST

```
a = [1,2,3,4,5,6,7,8,9,10]

lower = a[0]

upper = a[9]


for i in range(lower,upper + 1):
        count = 0
        for j in range(1,i+1):
                if (i % j) == 0:
```

```
            count = count + 1
    if (count == 2):
        print(i)
```

Output –

2

3

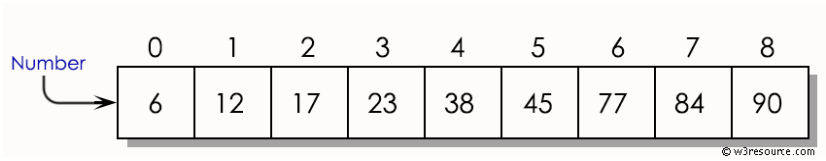
5

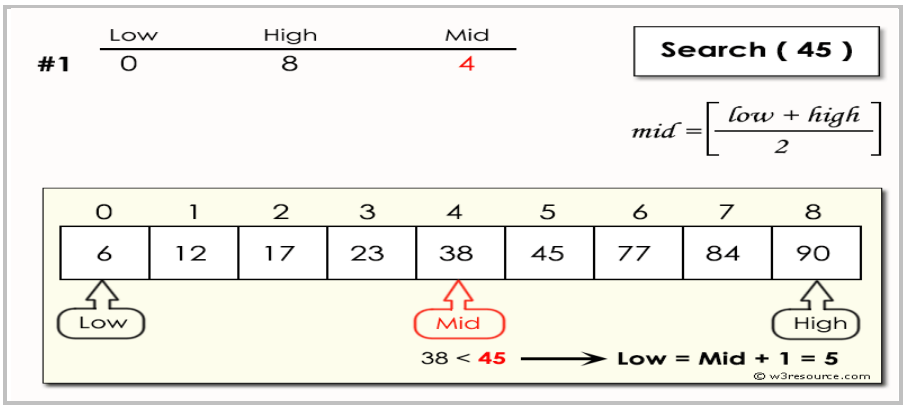
7

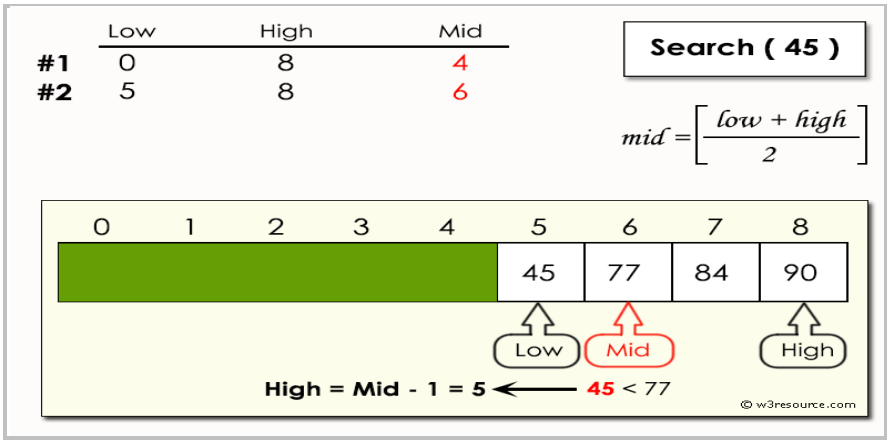
BINARY SEARCH

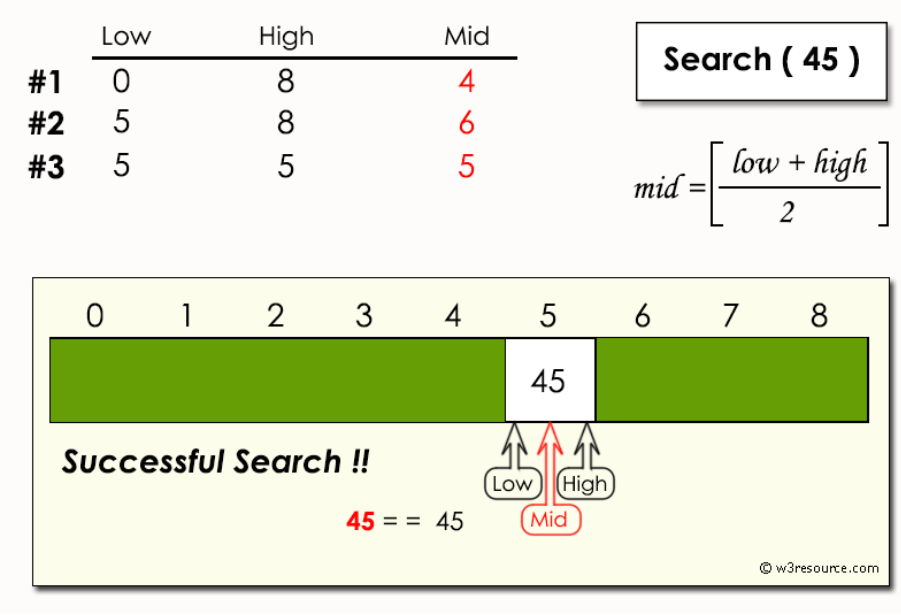
Write a Python program for binary search.
Binary Search : In computer science, a binary search or half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

Step by step example :






Low
High
Mid
#1
0
8
4
#2
5
8
6
Search (45)
$mid = \left[\frac{low + high}{2} \right]$
0 1 2 3 4 5 6 7 8
45
77
84
90
Low
Mid
High
High = Mid - 1 = 5
45 < 77
© w3resource.com


Low
High
Mid
#1
0
8
4
#2
5
8
6
#3
5
5
5
Search (45)
$mid = \left[\frac{low + high}{2} \right]$
0 1 2 3 4 5 6 7 8
45
Successful Search !!
Low
High
Mid
45 = = 45
© w3resource.com

Sample Solution:

Python Code:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 27 21:38:34 2019

@author: admin
"""

def binary_search(a,x):
    first_pos=0
    #print(len(a))
    last_pos= len(a)-1
    flag=0

    count=0

    while(first_pos <= last_pos and flag==0):
        count = count + 1
        mid = (first_pos + last_pos)//2
        if (x==a[mid]):
            flag=1
            print ("The element present in the position: "+str(mid))
            print ("The number of iteration are: "+str(count))
            return
        else:
            if(x<a[mid]):
                last_pos = mid-1
            else:
                first_pos = mid+1

    print (" The number is not present")

a=[]
for i in range (1,501):
    a.append(i)

binary_search(a,70)

```

OUTPUT

```
In [6]: runfile('D:/Phyton-Programme/Binary-Search.py', wdir='D:/Phyton-Programme')
The element present in the position: 69
The number of iteration are: 9
```

LIST PERMUTATION IN PYTHON

Python provide direct methods to find permutations and combinations of a sequence. These methods are present in itertools package.

Permutation

First import itertools package to implement permutations method in python. This method takes a list as an input and return an object list of tuples that contain all permutation in a list form.

PYTHON PROGRAM FOR PERMUTATION

```
15
16 from itertools import permutations
17
18 # Get all permutations of [1, 2, 3]
19 perm = permutations([1, 2, 3])
20
21 # Print the obtained permutations
22 for i in list(perm):
23     print (i)
```

OUTPUT

```
In [2]:

In [2]: runfile('D:/Phyton-Programme/LinearSearch.py', wdir='D:/Phyton-Programme')
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

Random Permutations

Random Permutations of Elements

A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa.

The NumPy Random module provides two methods for this: shuffle() and permutation().

Shuffling Arrays

Shuffle means changing arrangement of elements in-place. i.e. in the array itself.

Example

Randomly shuffle elements of following array:

```
from numpy import random
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
random.shuffle(arr)
print(arr)
```

OUTPUT

```
[3 4 1 5 2]
```

Generating Permutation of Arrays

Example

Generate a random permutation of elements of following array:

```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))
```

OUTPUT

[5 3 4 2 1]
