**PYTHON LIST**

In python, list is the generic term for **an ordered set**. Each element of a list 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.

List is a datatype/datastructure in python.

[8,34,77,53]🡪 index=0,1,2,3 or -1,-2,-3,-4

**The basic list operations**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Explanation** | **Example** |
| + (concatenation) | Combines 2 sequences | [1,2,3]+[4,5] will evaluate to [1,2,3,4,5]. |
| \* (repeat) | repeats a sequence a (positive integral) number of times. | [1,11]\*3 will evaluate to [1,11,1,11,1,11]. |
| in | x in mySeq will return True if x is an element of mySeq, and False otherwise. You can negate this statement with either not (x in mySeq) or x not in mySeq. | ‘wozti’ in [‘wozti’,’ip’,’abc’] |
| index | mySeq[i] will return the i'th character of mySeq  mySeq[-i] will return the i'th element from the end of mySeq | Lis = [‘a’,’bc’,’def’]  Lis[0] will return ‘a’ |
| Slicing | mySeq[i:j] | S =”I Love India”  S[2:6] will return ‘Love’ |

**Functions useful for Sequences**

|  |  |
| --- | --- |
| **Function** | **Explanation** |
| len | len(mySeq) returns the number of elements in the sequence mySeq |
| index | mySeq.index(x) returns the index of the first occurrence of x in mySeq. |
| min(mySeq) | Returns the smallest element in sequence |
| max(mySeq) | Returns the largest element in sequence |
| count | mySeq.count(x) returns the number of occurrences of x in mySeq |

**List**

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. We can have strings, integers, booleans, another list etc in a list.

**Creating a list** is as simple as putting different comma-separated values between square brackets. For example –

**list1 = ['physics', 'chemistry', 1997, 2000, True, [1,2]];**

**list2 = [1, 2, 3, 4, 5 ];**

**list3 = ["a", "b", "c", "d"]**

**list4 = [] #empty list**

**list5 = list() #creates empty list**

list() accepts either a sequence or tuple as the argument and converts into a Python list.

**Using List Comprehension – An Intuitive Way To Create Lists**

Python supports a concept known as “List Comprehension.” It helps in constructing lists in a completely natural and easy way.

A list comprehension has the following syntax:

**theList = [expression(iter) for iter in oldList if filter(iter)]**

*Examples*

>>> theList = [x for x in range(5)]

>>> print(theList) #output is [0,1,2,3,4]

>>>countries= ["India", "America", "England", "Germany", "Brazil", "Vietnam"]

>>> firstLetters = [ country[0] for country in countries ]

>>> print(firstLetters) #prints ['I', 'A', 'E', 'G', 'B', 'V']

>>> print ([x+y for x in 'get' for y in 'set'])

['gs', 'ge', 'gt', 'es', 'ee', 'et', 'ts', 'te', 'tt']

**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. Any attempt to access an item beyond the range would result in an IndexError. The index is always an integer. Using any other type of value will lead to TypeError.

**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]**

**Reverse Indexing**

Python enables reverse (or negative) indexing for the sequence data type. So, for Python list to index in the opposite order, you need to set the index using the minus (-) sign.

Indexing the list with “-1” will return the last element of the list, -2 the second last and so on.

**Slicing a List**

**Syntax:** list[start:end:step]

Default value for start is 0.

Default value for step is 1

**Reverse A Python List Using The Slice Operator**

theList[::-1]

The ‘-1’ after the second colon intends to increment the index every time by -1 and directs to traverse in the backward direction.

**Iterating List elements**

Python provides a traditional for-in loop for iterating the list. The for statement makes it super easy to process the elements of a list one by one.

**for element in theList:**

**print(element)**

If you wish to use both the index and the element, then call the enumerate() function.

**for index, element in enumerate(theList):**

**print(index, element)**

If you only want the index, then call the range() and len() methods.

**for index in range(len(theList)):**

**print(index)**

The list elements support the iterator protocol. To intentionally create an iterator, call the built-in **iter** function.

**it = iter(theList)**

**element = next(it) # fetch first value**

**element = next(it) # fetch second value**

**Update Elements in a List**

Unlike the string or tuple, Python list is a mutable object, so the values at each index can be modified.

You can use the assignment operator (=) to update an element or a range of items.

**Example**

theList = ['Python', 'C', 'C++', 'Java', 'CSharp']

theList[4]=‘C#’

theList[1:4]=[‘Perl’, ‘DotNet’, ‘JS’]

**Add elements to a List**

*Using* ***“+”*** *operator*

L1 = [1,2,3]

L2=[4,5]

L3= L1+L2

print(L3) # prints [1,2,3,4,5]

*Using* ***extend*** *method*

L1 = [1,2,3]

L2 = [4,5,6]

L1.extend(L2)

print(L1) # prints [1,2,3,4,5,6]

*Using* ***append*** *method*

L1 = [1,2,3]

L2=[4,5,6]

L1.append(L2)

print(L1) # prints [1,2,3,[4,5,6]]

*Using* ***insert*** *method*

L1=[1,2,3]

L1.insert(0,10)

print (L1) # prints [10,1,2,3]

**Remove/Delete elements from a list**

*Using del keyword*

vowels = ['a','e','i','o','u']

del vowels[2] #deletes the element at 2nd index

del vowels[1:3] #deletes the elements at 1st & 2nd index

del vowels #deletes the list

*Using remove() method*

vowels.remove(‘a’) #removes element ‘a’. It throws ValueError if element is not in list

*Using pop() –* removes and returns the element at the specified index. Generates IndexError if given index is out of range

print(vowels.pop(1))

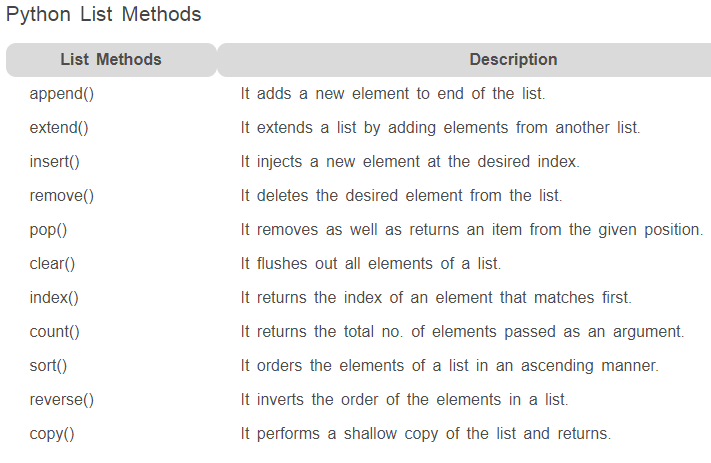
print(vowels.pop())

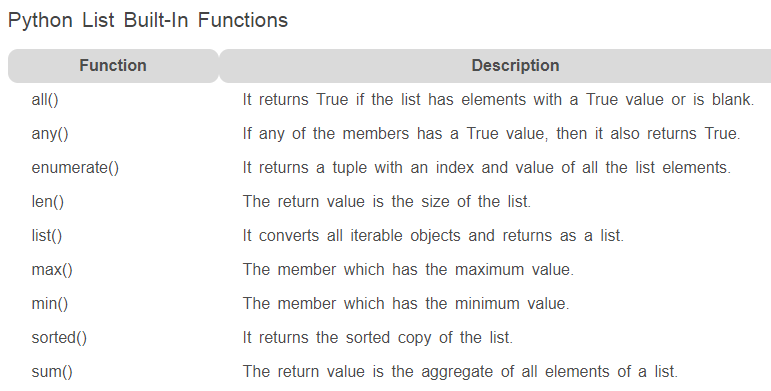
*Using clear()* – removes all elements from list

vowels.clear()

*By assigning a blank list* – removes the elements in the given range

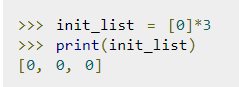
vowels[1:3]=[]





**Creating a Multi-Dimensional List**

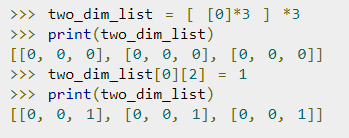
You can create a sequence with a pre-defined size, by specifying an initial value for each element.



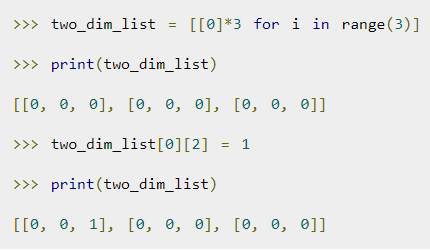
With the above concept, you can build a two-dimensional list.



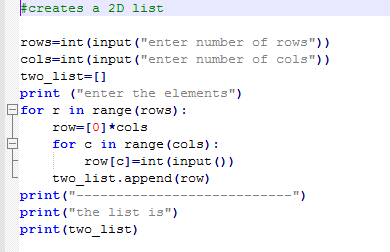
The above statement works but Python will only create the references as sublists instead of creating separate objects. So, if we change the value of the third item in the first row, but the same column in other rows also got affected.



So, we can use list comprehensions to get around of the above issue.



**Example**



**Example:**

r = int(input("row: "))

c = int(input("column: "))

tb = []

print("enter elements")

for rr in range(r):

r1 = [0]\*c

for cc in range(c):

r1[cc] = int(input())

tb.append(r1)

print("The list is: ",tb)

**Result:**

row: 3

column: 4

enter elements

12

13

14

15

16

17

18

19

20

21

22

23

The list is: [[12, 13, 14, 15], [16, 17, 18, 19], [20, 21, 22, 23]]

///////////////////////////////////////////////

The *any()* function returns *True* if any item in an iterable are true, otherwise it returns *False*. However, if the iterable object is empty, the *any ()* function will return *False*.

Syntax:

*any(iterable)*

The iterable object can be a list, tuple or dictionary.

*any()* returns:

* True – if atleast one item of the iterable is True.
* False – if all the items are False or if an iterable is empty.

| **When** | **Return Value** |
| --- | --- |
| All values are true | True |
| At least one value is True | True |
| All values are false | False |
| Empty iterable | False |

**Example:**

a = [False, True, False]

x = any(a)

print(x)

print(any(a))

**Result:**

True

True

**Example:**

a = (0, 1, 0, False) # Tuple

x = any(a)

print(x)

**Result:**

True

**Example:**

s = {0, 3, 0 } # Set

x = any(s)

print(x)

**Result:**

True

**Example:**

d = { 0 : "Apple", 1: "Banana"} # Dictionary

x = any(d) # Checking keys of dictionary only

print(x)

**Result:**

True

/////////////////////////////////////////////

The *all()* function returns *True* if all items in an iterable are true, otherwise it returns *False*. If the iterable object is empty, the *all()* function returns *True*.

### Syntax:

*all(iterable)*

The iterable object can be *list*, *tuple* or *dictionary*.

Return value from *all()*

The *all()* method returns

* True – if all elements in an iterable are true
* False – if any element in an iterable is false

| When | Return Value |
| --- | --- |
| All values are true | True |
| At least one value is True | False |
| All values are false | False |
| Empty iterable | True |

////////////////////////////////////////////////

**Example of *all()* to check if all items are *True***

a = [True, True, False]

x = all(a)

print(x)

print(all(a))

**Result:**

False

False

**Example:**

a = (-9, True, False) # Tuple

x = all(a)

print(x)

**Result:**

False

**Example:**

a = {True, 1, 1} #set

x = all(a)

print(x)

**Result:**

True

**Example:**

a = {0: "Apple", 1:"Banana"} # dictionary

x = all(a) # Checking keys of dictionary

print(x)

**Result:**

False

**Example:**

vowels = ['e', 'a', 'u', 'o', 'i']

vowels.sort()

print('Sorted list:', vowels)

**Result:**

Sorted list: ['a', 'e', 'i', 'o', 'u']

////////////////////////////////////////////

*sort()* method accepts a *reverse* parameter as an optional argument.

Setting *reverse=True* sorts the list in the descending order.

list.sort(reverse=True)

Alternately for *sorted()*, you can use the following code.

sorted(list, reverse=True)

//////////////////////////////////////////////////////////////////////////

**Example : Sort the list in Descending order**

vowels = ['e', 'a', 'u', 'o', 'i']

vowels.sort(reverse = True)

print('Sorted list (in Descending):', vowels)

**Result:**

Sorted list (in Descending): ['u', 'o', 'i', 'e', 'a']

///////////////////////////////////////////////////

If you want your own implementation for sorting, *sort()* also accepts a *key* function as an optional parameter.

Based on the results of the *key* function, you can sort the given list.

list.sort(key=len)

Alternatively for sorted

sorted(list, key=len)

Here, *len* is the Python's in-built function to count the length of an element.

The list is sorted based on the length of its each element, from lowest count to highest.

//////////////////////////////////////////////

## Example: Sort the list using ‘key’

def takeSecond(elem):

return elem[1]

random = [(2, 2), (3, 4), (4, 1), (1, 3)] # sort this list according to 2nd item of tuple

random.sort(key = takeSecond) # 'key' is in-built

print('Sorted list:', random)

**Result:**

Sorted list: [(4, 1), (2, 2), (1, 3), (3, 4)]

**Example:**

random = (1,2,3,5,4) # 'sort' function is applicable only for 'list'

random.sort()

print('Sorted: ', random)

**Result:**

random.sort()

AttributeError: 'tuple' object has no attribute 'sort'

**Example:**

random = [1,2,3,5,4]

a = random.sort() # sort() will not assign value to a variable

print('Sorted: ', a)

**Result:**

Sorted: None

//////////////////////////////////////////////////

*Sorted()* sorts any sequence (list, tuple) and always returns a list with the elements in sorted manner, without modifying the original sequence.

**Syntax :** sorted(iterable, key, reverse)

**Parameters :** sorted takes three parameters from which two are optional.

*Iterable* : sequence (list, tuple, string) or collection (dictionary, set, frozenset) or any other iterator that needs to be sorted.

*Key*(optional) : A function that would serve as a key or a basis of sort comparison.

*Reverse*(optional) : If set True, then the iterable would be sorted in reverse (descending) order, by default it is set as False.

///////////////////////////////////////////

**Example:**

x = [2, 8, 1, 4, 6, 3, 7]

print (sorted(x))

print (sorted(x, reverse = True))

print ("\nOriginal list not modified: ",x)

**Result:**

[1, 2, 3, 4, 6, 7, 8]

[8, 7, 6, 4, 3, 2, 1]

Original list not modified: [2, 8, 1, 4, 6, 3, 7]

**Example: Sorting different data types**

x = ['q', 'w', 'r', 'e', 't', 'y']

print (sorted(x))

# Tuple

x = ('q', 'w', 'e', 'r', 't', 'y')

print (sorted(x))

# String

x = "python"

print (sorted(x))

# Dictionary

x = {'q':1, 'w':2, 'e':3, 'r':4, 't':5, 'y':6}

print (sorted(x))

# Set

x = {'q', 'w', 'e', 'r', 't', 'y'}

print (sorted(x))

# Frozen Set

x = frozenset(('q', 'w', 'e', 'r', 't', 'y'))

print (sorted(x))

**Result:**

['e', 'q', 'r', 't', 'w', 'y']

['e', 'q', 'r', 't', 'w', 'y']

['h', 'n', 'o', 'p', 't', 'y']

['e', 'q', 'r', 't', 'w', 'y']

['e', 'q', 'r', 't', 'w', 'y']

['e', 'q', 'r', 't', 'w', 'y']

//////////////////////////////////////////////////

**Custom Sorting using the *key* parameter**

*sorted()* function has an optional parameter called ‘key’ which takes a function as its value. This key function transforms each element before sorting, it takes the value and returns one value which is then used within sort instead of the original value.

For example, if we pass a list of strings in *sorted()*, it gets sorted alphabetically. But if we specify *key = len*, i.e, give *len* function as key, then the strings would be passed to *len*, and sort according to length of strings.

/////////////////////////////////////////////////

**Example:**

L = ["cccc", "b", "dd", "aaa"]

print ("Normal sort :", sorted(L))

print ("Sort with len :", sorted(L, key = len))

**Result:**

Normal sort : ['aaa', 'b', 'cccc', 'dd']

Sort with len : ['b', 'dd', 'aaa', 'cccc']

/////////////////////////////////////////

Key also takes user-defined functions as its value for the basis of sorting.

//////////////////////////////////////////

**Example:**

def func(x):

return x % 7

L = [15, 3, 11, 7]

print ("Normal sort :", sorted(L))

print(15%7,' ',3%7,' ',11%7,' ',7%7)

print('15', ' 3', ' 11', ' 7')

print ("Sorted with key:", sorted(L, key = func)) # Prints in ascending order of the result after modulo operation

**Result:**

Normal sort : [3, 7, 11, 15]

1 3 4 0

15 3 11 7

Sorted with key: [7, 15, 3, 11]

**Example:**

"""When anything other than list is used in place of list,

then it returns an AttributeError"""

# Python3 program to demonstrate the error in reverse() method.

a = "gth"

a.reverse()

print(a) #Attribute error

a = (1,2,3)

a.reverse()

print(a) #Attribute error

a = [1,2,3]

a.reverse()

print(a) # No error. That is, *reverse()* will work only for ‘list’

///////////////////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[-len(a)])

**Result:**

h

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[:3])

**Result:**

['h', 'i', 'j']

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[0:-1])

**Result:**

['h', 'i', 'j']

////////////////////////////////

# Striding is nothing but giving step-size during slicing. The number given to step-size is called stride.

**Example of ‘striding’:**

a = ['h','i','j','k']

print(a[0:4:2])

**Result:**

['h', 'j']

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[:])

**Result:**

['h', 'i', 'j', 'k']

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[::])

**Result:**

['h', 'i', 'j', 'k']

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[::-2])

**Result:**

['k', 'i']

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[0:3:-1]) # Doesn’t work

**Result:**

[]

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[2:0:-1])

**Result:**

['j', 'i']

////////////////////////////////

**Example:**

a = ['h','i','j','k']

print(a[2:0]) # doesn’t work

**Result:**

[]

////////////////////////////////

**Example:**

a = ['h','i','j','k']

for i in range(len(a)):

print(a[0:i])

**Result:**

[]

['h']

['h', 'i']

['h', 'i', 'j']

////////////////////////////////

**Example:**

a = ['h','i','j','k','l','m','n','o']

for i in range(len(a)):

print(a[i:i+3])

**Result:**

['h', 'i', 'j']

['i', 'j', 'k']

['j', 'k', 'l']

['k', 'l', 'm']

['l', 'm', 'n']

['m', 'n', 'o']

['n', 'o']

['o']

////////////////////////////////

**Example:**

b = 5

a = ['h','i','j','k','l','m','n','o']

for i in range(len(a)-4):

print(a[i:i+b])

**Result:**

['h', 'i', 'j', 'k', 'l']

['i', 'j', 'k', 'l', 'm']

['j', 'k', 'l', 'm', 'n']

['k', 'l', 'm', 'n', 'o']

////////////////////////////////

**Example:**

b = 4

a = ['h','i','j','k','l','m','n','o']

for i in range(len(a)-3):

print(a[i:i+b])

**Result:**

['h', 'i', 'j', 'k']

['i', 'j', 'k', 'l']

['j', 'k', 'l', 'm']

['k', 'l', 'm', 'n']

['l', 'm', 'n', 'o']

////////////////////////////////

**Example:**

b = 4

a = ['h','i','j','k','l','m','n','o']

for i in range(len(a)-(b-1)):

print(a[i:i+b])

**Result:**

['h', 'i', 'j', 'k']

['i', 'j', 'k', 'l']

['j', 'k', 'l', 'm']

['k', 'l', 'm', 'n']

['l', 'm', 'n', 'o']

////////////////////////////////

**Example:**

b = 2

a = ['h','i','j','k','l','m','n','o']

for i in range(len(a)-(b-1)):

print(a[i:i+b])

**Result:**

['h', 'i']

['i', 'j']

['j', 'k']

['k', 'l']

['l', 'm']

['m', 'n']

['n', 'o']

////////////////////////////////

**Example:**

b = 1

a = ['h','i','j','k','l','m','n','o']

for i in range(len(a)-(b-1)):

print(a[i:i+b])

**Result:**

['h']

['i']

['j']

['k']

['l']

['m']

['n']

['o']

////////////////////////////////

**‘Zip’ function in python**

Zip function take any iterables in python. For ‘zip’ function, iterables should be equal in length.

/////////////////////////////////////////////////

**Example:**

a = [1,2,3]

b = ['one', 'two', 'three']

c = list(zip(a,b))

print(c)

**Result:**

[(1, 'one'), (2, 'two'), (3, 'three')]

///////////////////////////////////////////////

Make two lists where first list contains more items.

///////////////////////////////////////////////

**Example:**

a = [1,2,3,4]

b = ['one', 'two', 'three']

c = list(zip(a,b))

print(c)

**Result:**

[(1, 'one'), (2, 'two'), (3, 'three')]

///////////////////////////////////////////////

Make two lists where second list contains more items.

///////////////////////////////////////////////

**Example:**

a = [1,2,3]

b = ['one', 'two', 'three', 'four']

c = list(zip(a,b))

print(c)

**Result:**

[(1, 'one'), (2, 'two'), (3, 'three')]

///////////////////////////////////////////////

Make three lists and zip them

///////////////////////////////////////////////

**Example:**

a = [1,2,3, 4]

b = ['one', 'two', 'three', 'four']

c = ['won','to','thrii','phour']

d = list(zip(a,b,c))

print(d)

**Result:**

[(1, 'one', 'won'), (2, 'two', 'to'), (3, 'three', 'thrii'), (4, 'four', 'phour')]

///////////////////////////////////////////////

Unzipping lists after zipping.

///////////////////////////////////////////////

**Example:**

a = [1,2,3, 4]

b = ['one', 'two', 'three', 'four']

c = ['won','to','thrii','phour']

d = list(zip(a,b,c))

e = list(zip(\*d))

print(e)

**Result:**

[(1, 2, 3, 4), ('one', 'two', 'three', 'four'), ('won', 'to', 'thrii', 'phour')]

///////////////////////////////////////////////

Printing items in llists using ‘for’ loop after zipping

///////////////////////////////////////////////

**Example:**

a = [1,2,3, 4]

b = ['one', 'two', 'three', 'four']

c = ['won','to','thrii','phour']

for d,e,f in zip(a,b,c):

print(d,' : ',e,' : ',f)

**Result:**

1 : one : won

2 : two : to

3 : three : thrii

4 : four : phour

///////////////////////////////////////////////

Creating sentences from different lists using ‘for’ loop

///////////////////////////////////////////////

**Example:**

items = ['apple','banana','orange']

counts = ['3','6','4']

prices = ['220','25','30']

a = []

for it,con,pri in zip(items, counts, prices):

b = 'I bought ' + con + ' ' + it + 's at ' + pri + ' rupees each'

a.append(b)

print(a)

**Result:**

['I bought 3 apples at 220 rupees each', 'I bought 6 bananas at 25 rupees each', 'I bought 4 oranges at 30 rupees each']

**Example:**

x = [1,2,3]

y = (4,5,6)

print(list(reversed(x)))

print(tuple(reversed(y)))

print(set(reversed(y)))

**Result:**

[3, 2, 1]

(6, 5, 4)

{4, 5, 6}

**Example**

list1 = [1, 2, 3, 2, 1]

list2 = list1.copy()

print(list2)

list2.reverse()

print(list2)

if list1 == list2:

print("Palindrome")

else:

print("Not Palindrome")

**Result:**

[1, 2, 3, 2, 1]

[1, 2, 3, 2, 1]

Palindrome

**Example:**

x=int(input('Please insert a number: '))

f = str(x)

y = reversed(f)

print(type(y))

k = ""

for i in y:

k+=i

j = int(k)

if x == j:

print("palindrome")

else:

print("Not palindrome")

**Result:**

Please insert a number: 1221

<class 'reversed'>

palindrome

///////////////////////////////////

The *reverse()* method does not return any value but

reverse the given object from the list.

/////////////////////////////////////

**Example:**

list1 = [1, 2, 3, 4, 1, 2, 6]

list1.reverse()

print(list1)

list2 = ['a', 'b', 'c', 'd', 'a', 'a']

list2.reverse()

print(list2)

**Result:**

[6, 2, 1, 4, 3, 2, 1]

['a', 'a', 'd', 'c', 'b', 'a']

///////////////////////////////////

When anything other than list is used in place of list,

then it returns an AttributeError

///////////////////////////////////

**Example:**

a = "abgedge"

a.reverse()

print(a)

**Result:**

a.reverse()

AttributeError: 'str' object has no attribute 'reverse'

///////////////////////////////////////

Given a list of numbers, check if the list is palindrome.

**Note:** palindrome-sequence that reads the same backwards as forwards.

//////////////////////////////////////

**Example:**

list1 = [1, 2, 3, 2, 1]

list2 = list1.copy()

list2.reverse()

if list1 == list2:

print("Palindrome")

else:

print("Not Palindrome")

**Result:**

Palindrome

//////////////////////////////////

**reversed()** method returns an iterator that accesses the given sequence in the reverse order.

**Syntax :**

reversed(sequ)

**sequ :** Sequence to be reversed.

*reversed()* returns an iterator that accesses the given sequence in the reverse order.

/////////////////////////////////////////////////////////

**Example:**

class AppDividend:

title = ['a', 'p', 'p', 'd', 'i', 'v', 'i', 'd', 'e', 'n', 'd']

def \_\_reversed\_\_(self):

return reversed(self.title)

obj = AppDividend()

print(list(reversed(obj)))

**Result:**

['d', 'n', 'e', 'd', 'i', 'v', 'i', 'd', 'p', 'p', 'a']

**Example:**

a = "ABCDEF"

print(''.join(reversed(a)))

**Result:**

FEDCBA

**Example of deepcopy():**

import copy

li1 = [1, 2, [3,5], 4]

li2 = copy.deepcopy(li1)

print ("The original elements before deep copying")

print(li1)

print("\r")

li2[2][0] = 7

print ("The new list of elements after deep copying ")

print(li2)

print("\r")

print ("The original elements after deep copying")

print(li1)

**Result:**

The original elements before deep copying

[1, 2, [3, 5], 4]

The new list of elements after deep copying

[1, 2, [7, 5], 4]

The original elements after deep copying

[1, 2, [3, 5], 4]

**Example of shallowcopy:**

import copy

li1 = [1, 2, [3,5], 4]

li2 = copy.copy(li1)

print ("The original elements before shallow copying")

print(li1)

print("\r")

li2[2][0] = 7

print ("The new list of elements after shallow copying ")

print(li2)

print("\r")

print ("The original elements after shallow copying")

print(li1)

**Result:**

The original elements before shallow copying

[1, 2, [3, 5], 4]

The new list of elements after shallow copying

[1, 2, [7, 5], 4]

The original elements after shallow copying

[1, 2, [7, 5], 4]

**Assignments**

1

Comparing and analysing ages.

2

Rearranging a list of IT companies.

3

Assignment to compare different scenarios of any() and all(). Slicing list in different ways.

4

Assignment on list filtering.