Rex Galaxy Technology

"Innovating the future through technology."

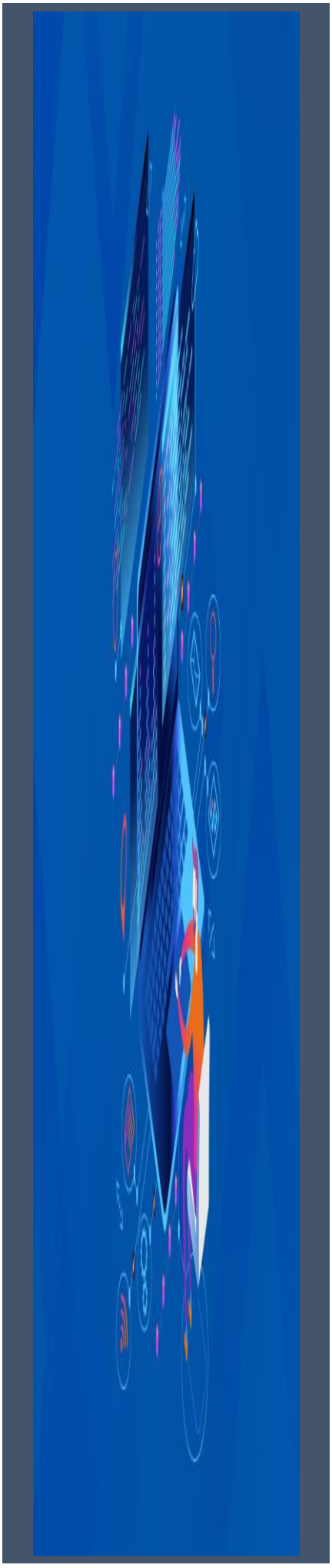
PYTHON LIST

''This is complete guide about python list that will help you to learn and understand all operations of list.''

06 May, 2023

PYTHON LIST GUIDE

1. Create a list:
2. Access an element:
3. Modify an element:
4. Get the length of a list:
5. Append an element to the end of the list:
6. Insert an element at a specific index:
7. Remove an element:
8. Check if an element exists in the list:
9. Concatenate two lists:
10. Slice a list:
11. Sort a list:
12. Reverse a list:
13. Count occurrences of an element:
14. List comprehensions:
15. Nested List



Rex Galaxy Technology
"Innovating the future through technology."

PYTHON LIST

In computer programming, a list is a data structure that represents a collection of items, where each item is identified by an index or a key. Lists are commonly used in programming languages to store and manipulate data, and they can hold any type of data, such as numbers, strings, or objects.

Lists are usually ordered, which means that the items in the list are arranged in a specific sequence. This allows programmers to access and modify the items in the list using their position or index.

Some programming languages have built-in support for lists, while others require the use of specialized libraries or modules. Lists are a fundamental concept in computer science and are used in many different programming contexts, such as data analysis, web development, and machine learning.

In Python, a list is a built-in data structure that represents a collection of items, where each item is separated by a comma and enclosed in square brackets ([]). Lists are one of the most used data structures in Python, and they can hold any type of data, such as numbers, strings, or objects.

Python lists are like arrays in other programming languages, but they are more flexible because they can be resized and can hold different types of data. Lists are also ordered, which means that the items in the list are arranged in a specific sequence. This allows programmers to access and modify the items in the list using their position or index.

Python lists support a wide range of operations, such as adding or removing elements, slicing, sorting, and searching. Lists can also be used to implement other data structures, such as stacks and queues.

Here is an example of creating a simple list in Python:

```
1 my_list = [1,"Python",2.5]
2 print(my_list)
```

```
[1, 'Python', 2.5]
```

Blank list Creation:

A blank list can be created just placing the elements in square brackets.

```
1 my_list = []
2
3 print(type(my_list))
```

```
<class 'list'>
```

Rex Galaxy Technology
"Innovating the future through technology."

Duplicate Members allowed in list?

```
my_list = [1,"Rexgalaxy","python","Rexgalaxy"]
print(my_list)
```

```
[1, 'Rexgalaxy', 'python', 'Rexgalaxy']
```

Yes, list allows duplicate members.

How to Access Element from list?

To access elements from list we have different methods.

1. We can use indexing for accessing elements from list.
2. We can use for loop for accessing elements from list.

Indexing:

We can access elements by using index number as below. As lists are orders collection so we get item just placing index number in square brackets.

In [1]:
```
my_list = [1,"Rexgalaxy","python","Rexgalaxy"]
print(my_list[0])
print(my_list[1])
print(my_list[2])
print(my_list[3])

```

```
1
Rexgalaxy
python
Rexgalaxy
```

Accessing elements from a nested list:

In Python, a nested list is a list that contains other lists as elements. This allows for the creation of multidimensional arrays, where each element can be accessed using multiple indices.

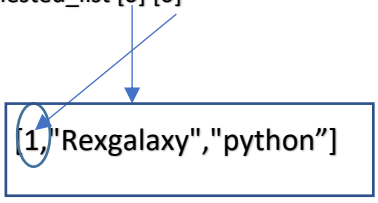
In [2]:
```
nested_list = [[1,"Rexgalaxy","python"],["Software testing","Machine Learning","Data Science"]]


```

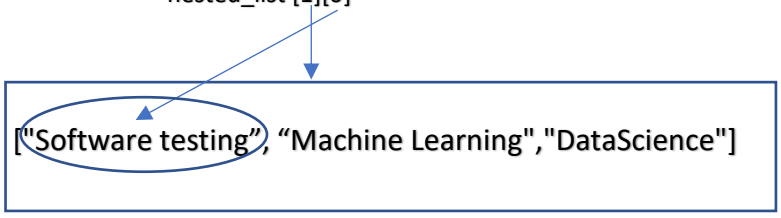
In [4]:
```
print(nested_list [0][0])
print(nested_list[1][0])
```

```
1
Software testing
```

nested_list [0] [0]



nested_list [1][0]



Rex Galaxy Technology
"Innovating the future through technology."

Negative indexing in list:

In Python, negative indexing is a way to access elements in a list by counting from the end of the list instead of from the beginning. Negative indices start from -1 and count backwards, with -1 referring to the last element in the list, -2 referring to the second-to-last element, and so on.

In [6]:
```
my_list = ["Python","Rexgalaxy","Machine Learning"]
```

In [7]:
```
print(my_list[-1]) # last element of list
print(my_list[-2]) # second last element of list
```

```
Machine Learning
Rexgalaxy
```

Accessing elements of List by for loop:

In [6]:
```
my_list = ["Python","Rexgalaxy","Machine Learning"]
```

In [8]:
```
for element in my_list:
    print(element)
```

```
Python
Rexgalaxy
Machine Learning
```

Taking input of a List:

In Python, you can take input of a list using the input () function and then converting it into a list using the split() method.

```
# Taking input of a list of integers
numbers = input("Enter a list of numbers separated by space: ")
numbers_list = numbers.split()
numbers_list = [int(num) for num in numbers_list]
print(numbers_list)
```

```
Enter a list of numbers separated by space: 1 2 3
[1, 2, 3]
```

Rex Galaxy Technology
"Innovating the future through technology."

How to add Elements to a Python List:

Three methods you can use to add elements in the list

1. append (): The append () method adds an item to the end of the list:
2. insert (): The insert () method adds an item at a specific index in the list:
3. extend (): You can also add multiple items to a list at once using the extend() method:

1. append (object) method:

only one item can be added at the end of the list at a time with append method. If we want to add more than one item we can use for loop.

In [3]:
```
my_list = ["Python","Datascience","Machine Learning"]

my_list.append("Mysql")

print(my_list)
```

```
['Python', 'Datascience', 'Machine Learning', 'Mysql']
```

2. insert (index, object):

insert method can insert element at the specific index. Example below will show you add “Mysql” at index 0.

In [5]:
```
my_list = ["Python","Datascience","Machine Learning"]

my_list.insert(0,"Mysql")

print(my_list)
```

```
['Mysql', 'Python', 'Datascience', 'Machine Learning']
```

Note: if specified index is not present in the list than it will add element at the last position of the list. See below…

In [27]:
```
my_list = ["Python","Datascience","Machine Learning"]

my_list.insert(14,"my_new_item")

print(my_list)
```

```
['Python', 'Datascience', 'Machine Learning', 'my_new_item']
```

3. extend (iterable) : extent method will add any iterable in the list. Lets see below example.

In [28]:
```
my_list = ["Python","Datascience","Machine Learning"]

my_list.extend(("new_item1",13,"new_item2"))

print(my_list)
```

```
['Python', 'Datascience', 'Machine Learning', 'new_item1', 13, 'new_item2']
```

Rex Galaxy Technology
"Innovating the future through technology."

How to remove elements from the list

1. pop () method: pop will remove specified index element. However, if your do not specify any index number inside method it will remove last element of the list.

2. remove () method: will remove specified element.

Let's see example for both….

1. pop () method:

Using pop () without specifying anything inside method. It will remove last element of the list.

```
In [33]:
1 my_list = ["Python","Datascience","Machine Learning"]
2
3 my_list.pop()
4
5 print(my_list)
```

```
['Python', 'Datascience']
```

Using pop () method specifying, specific index number to be deleted from list.

```
In [34]:
1 my_list = ["Python","Datascience","Machine Learning"]
2
3 my_list.pop(0) # will remove element at the zero index
4
5 print(my_list)
```

```
['Datascience', 'Machine Learning']
```

2. remove () method:

In Python, you can remove an element from a list using the remove() method. The remove() method removes the first occurrence of the specified element from the list.

Here is an example:

```
In [37]:
1 my_list = ["Python","Datascience","Machine Learning","Python"]
2
3 my_list.remove("Python") # will remove first occurrence of Python
4
5 print(my_list)

['Datascience', 'Machine Learning', 'Python']
```

Note: if item does not exist which you want to remove than it will throw an error.

```
In [38]:
1 my_list = ["Python","Datascience","Machine Learning","Python"]
2
3 my_list.remove("item") # will remove first occurrence of Python
4
5 print(my_list)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [38], in <cell line: 3>()
      1 my_list = ["Python","Datascience","Machine Learning","Python"]
----> 3 my_list.remove("item") # will remove first occurrence of Python
      5 print(my_list)

ValueError: list.remove(x): x not in list
```

Slicing of a List

Slicing is a way to extract a portion of a list in Python. It creates a new list that contains a portion of the original list. In other words, we can say that we can get a sub list from an original list. In Python, you can slice a list using the following syntax:

My_List[start:stop:setp]

1. **Start:** is the index of the first element you want to include in the slice. It is optional and defaults to 0 if not specified.
2. **Stop:** is the index of the first element you want to exclude from the slice. It is also optional and defaults to the length of the list if not specified.
3. **Step:** is the number of elements to skip between each included element. It is optional and defaults to 1 if not specified.

Rex Galaxy Technology
"Innovating the future through technology."

If we do not define anything after colon (:). It will use all default values.

Start: 0 index

Stop: last index

Step: 1

Let's see below example:

```
In [58]:
my_list = ["item0","item1","item2","item3"]

print(my_list[::]) # it will print all elements in the list beause no index has been spacified. It will take default values

['item0', 'item1', 'item2', 'item3']
```

Let's see other examples in which we will define specific index.

```
In [53]:
# Create a list of numbers from 0 to 9
my_list = list(range(10))
print("My list         :",my_list)

# Slice the list from index 2 to index 5
print("my_list[2:5]    :",my_list[2:5])  #[start at 2 : stop at (5-1)  : step of 1] Output: [2, 3, 4]

# Slice the list from the beginning up to index 4
print("my_list[:4]     :",my_list[:4]) # [start from 0 : stop at (4-1) : step of 1] # Output: [0, 1, 2, 3]

# Slice the list from index 4 to the end, skipping every other element
print("my_list[4::2]   :",my_list[4::2])  #[start at 4 : stop in the end : step of 2] Output: [4, 6, 8]

# Slice the list in reverse order
print("Reverse string  : ",my_list[::-1])  # Output: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

My list         : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
my_list[2:5]    : [2, 3, 4]
my_list[:4]     : [0, 1, 2, 3]
my_list[4::2]   : [4, 6, 8]
Reverse string  :  [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Negative Indexing slicing:

Negative indexing in Python lists allows you to access elements from the end of the list by using negative numbers as indices. You can also use negative indexing in list slicing to extract a range of elements from the list. Here are some examples to illustrate negative indexing and slicing:

Example Negative Indexing Slicing:

```
my_list = ["Python", "Mysql", "Power BI", "Mongo DB"]

# Accessing elements using negative indexing
print(my_list[-1])  # Output: 'Mongo DB' (last element)



print(my_list[-3])  # Output: 'Mysql' (third element from the end)

# Slicing using negative indices
print(my_list[-2:])  # Output: ['Power BI', 'Mongo DB'] (last two elements)


print(my_list[:-2])  # Output: ['Python', 'Mysql'] (all elements except the last two)

print(my_list[-3:-1])  # Output: ['Mysql', 'Power BI'] (elements from the third to the second last)
```

OUTPUT:

```
Mongo DB
Mysql
['Power BI', 'Mongo DB']
['Python', 'Mysql']
['Mysql', 'Power BI']
```

Note:

My_list[-3:-1] will not include last element.

Methods that can be applied on list. lets Discuss all one by one

Rex Galaxy Technology
"Innovating the future through technology."

1. Index() Method:

Syntax: list.index(item)

Returns the index of the first occurrence of a specified element in the list.

```
my_list = ['Python', 'Mysql', 'Power BI', 'Mongo DB','Mysql']

print(my_list.index("Mysql"))
```

output:

```
1
```

2. count() Method:

Syntax : list.count(item)

Returns the number of occurrences of a specified element in the list.

```
my_list = ['Python', 'Mysql', 'Power BI', 'Mongo DB','Mysql','Mysql']

print(my_list.count("Mysql"))
```

Output:

```
3
```

3. sort() Method:

Syntax : list.sort(reverse=True/False)

The sort() method sorts the list in ascending order. The key parameter can be used to specify a function to customize the sorting. If reverse is set to True, the list is sorted in descending order.

```
my_list = [3, 1, 2]
my_list.sort()
print(my_list)
```

Output:

```
[1, 2, 3]
```

If we keep the value of reverse = True than it will return the sorted list in descending order. Let's see below example.

```
my_list = [3, 1, 2]
my_list.sort(reverse=True)
print(my_list)
```

Output:

```
[3, 2, 1]
```

Rex Galaxy Technology
"Innovating the future through technology."

5. reverse() Method:

Syntax : list.reverse()

Reverses the order of elements in the list.

```
my_list = [1, 2, 3,"RexGalaxy"]
my_list.reverse()


print(my_list)
```

output:

```
['RexGalaxy', 3, 2, 1]
```

6. copy() Method:

Import copy

new_list = copy.deepcopy(old_list)

Where we want to copy a list we should use copy module. We should not use = operator to copy the list. Because it will create a reference of your old list. Meaning is that if you change anything in your any of list it will reflect in both lists. Let's see how…

```
old_list = ["ROhit","Noida"]
New_list = old_list

old_list.append("UP")
print(old_list)
print(New_list)
```

output:

```
['ROhit', 'Noida', 'UP']
['ROhit', 'Noida', 'UP']
```

As you can see in above example, we created a reference of old_list. So lets use copy module to copy the list.

```
import copy
old_list = ["rohit","noida"]
new_list = copy.deepcopy(old_list)
old_list.append("new_item")
new_list.append("new_item2")

print(old_list)

print(new_list)
```

Output:

```
['rohit', 'noida', 'new_item']
['rohit', 'noida', 'new_item2']
```

How to join two lists

To join or concatenate two lists, there are several methods you can use in Python. Here are common approaches:

1.Using the + operator:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
joined_list = list1 + list2
print(joined_list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
```

2. Using the extend method:

Extend Method we already discussed above

Add elements of two lists

1.Using map Method:

```
from operator import add
list1 = [10,20,30]
list2 = [1,1,1]

added_list = list(map(add,list1,list2))
print(added_list)
```

Output:

```
[11, 21, 31]
```

2.Using zip()

```
list1 = [100,200,300]
list2 = [1,1,1]

added_list = [sum(i) for i in zip(list1,list2)]
print(added_list)
```

Output:

```
[101, 201, 301]
```

Rex Galaxy Technology
"Innovating the future through technology."

List comprehensions

List comprehensions in Python provide a concise and efficient way to create lists based on existing lists, iterables, or other sequences. They allow you to generate new lists by applying an expression or transformation to each element of an existing list or iterable. List comprehensions have a compact syntax and can often replace the need for traditional for loops.

The general syntax of a list comprehension in Python is as follows:

```
new_list = [expression for item in iterable if condition]
```

Let's create a list using list comprehensions:

1.Here we would like to create a list with the help of range function.

```
new_list = [i for i in range(1,10)]
print(new_list)
```

Output:

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

2.Using list comprehension we will create a list of even numbers

```
new_list = [i for i in range(1,10) if i%2==0]
print(new_list)
```

Output:

```
[2, 4, 6, 8]
```

3.Using list comprehension we can filter list as below. We will filter only those strings which contain Noida.

```
old_list =["rohit_noida","Sachin_noida","ajay_jaipur","lalit_delhi"]
new_list = [i for i in old_list if "noida" in i]
print(new_list)
```

Output:

```
['rohit_noida', 'Sachin_noida']
```

4.Using if and else statement in list Comprehension. Let's see this…

We have a list in which we would like to change element with some other element.

We want to replace "empty" with "new_item".

```
old_list =["empty","rohit","empty","empty","sachin"]

new_list = [i if i!= "empty" else "new_item" for i in old_list]
print(new_list)

['rohit_noida', 'Sachin_noida']
```

Output:

```
['new_item', 'rohit', 'new_item', 'new_item', 'sachin']
```

5.We can create matrix using list comprehension.

```
matrix_list = [[j for j in range(5)] for i in range (4)]
print(matrix_list)
```

Output:

```
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

6. we can filter Items from the list. Let's see example:

We have a list from which we have to filter even and odd numbers:

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

even_list =[]
odd_list =[]

[even_list.append(i) if i%2==0 else odd_list.append(i) for i in my_list]

print(even_list)
print(odd_list)
```

Output:

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