Python lists are one of the most versatile data types that allow us to work with multiple elements at once. For example,

```
# a list of programming languages
['Python', 'C++', 'JavaScript']
```

Create Python Lists

In Python, a list is created by placing elements inside square brackets `[]`, separated by commas.

```
# list of integers
my_list = [1, 2, 3]
```

A list can have any number of items and they may be of different types (integer, float, string, etc.).

```
# empty list
my_list = []

# list with mixed data types
my_list = [1, "Hello", 3.4]
```

A list can also have another list as an item. This is called a nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Access List Elements

There are various ways in which we can access the elements of a list.

List Index

We can use the index operator `[]` to access an item in a list. In Python, indices start at 0. So, a list having 5 elements will have an index from 0 to 4.

Trying to access indexes other than these will raise an `IndexError`. The index must be an integer. We can't use float or other types, this will result in `TypeError`.

Nested lists are accessed using nested indexing.

```
my_list = ['p', 'r', 'o', 'b', 'e']

# first item
print(my_list[0])   # p

# third item
print(my_list[2])   # o

# fifth item
print(my_list[4])   # e

# Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing
print(n_list[0][1])

print(n_list[1][3])

# Error! Only integer can be used for indexing
print(my_list[4.0])
```

Output

```
p
o
e
a
5
Traceback (most recent call last):
  File "<string>", line 21, in <module>
TypeError: list indices must be integers or slices, not float
```

Negative indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

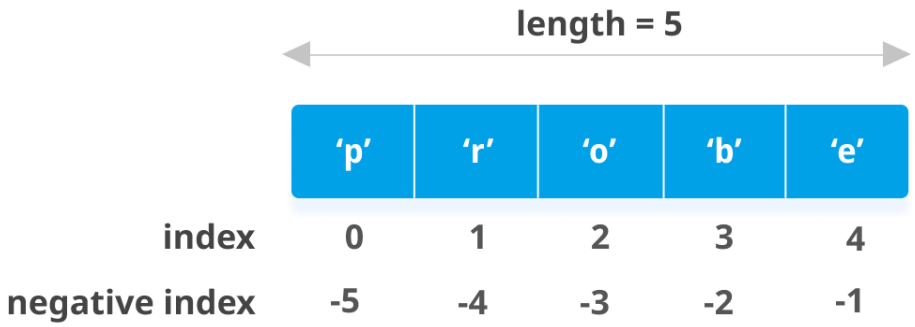
```
# Negative indexing in lists
my_list = ['p','r','o','b','e']

# last item
print(my_list[-1])

# fifth last item
print(my_list[-5])
```

Output

```
e
p
```



List indexing in Python

List Slicing in Python

We can access a range of items in a list by using the slicing operator `:`.

```
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# elements from index 2 to index 4
print(my_list[2:5])

# elements from index 5 to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

Output

```
['o', 'g', 'r']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

> **Note:** When we slice lists, the start index is inclusive but the end index is exclusive. For example, `my_list[2: 5]` returns a list with elements at index 2, 3 and 4, but not 5.

Add/Change List Elements

Lists are mutable, meaning their elements can be changed unlike string or tuple.

We can use the assignment operator `=` to change an item or a range of items.

```
# Correcting mistake values in a list
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

print(odd)
```

Output

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

We can add one item to a list using the `append()` method or add several items using the `extend()` method.

```
# Appending and Extending lists in Python
odd = [1, 3, 5]

odd.append(7)

print(odd)

odd.extend([9, 11, 13])

print(odd)
```

Output

```
[1, 3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
```

We can also use `+` operator to combine two lists. This is also called concatenation.

The `*` operator repeats a list for the given number of times.

```
# Concatenating and repeating lists
odd = [1, 3, 5]

print(odd + [9, 7, 5])

print(["re"] * 3)
```

Output

```
[1, 3, 5, 9, 7, 5]
['re', 're', 're']
```

Furthermore, we can insert one item at a desired location by using the method `insert()` or insert multiple items by squeezing it into an empty slice of a list.

```
# Demonstration of list insert() method
odd = [1, 9]
odd.insert(1,3)

print(odd)

odd[2:2] = [5, 7]

print(odd)
```

Output

```
[1, 3, 9]
[1, 3, 5, 7, 9]
```

Delete List Elements

We can delete one or more items from a list using the Python del statement. It can even delete the list entirely.

```
# Deleting list items
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# delete one item
del my_list[2]

print(my_list)

# delete multiple items
del my_list[1:5]

print(my_list)

# delete the entire list
del my_list

# Error: List not defined
print(my_list)
```

Output

```
['p', 'r', 'b', 'l', 'e', 'm']
['p', 'm']
Traceback (most recent call last):
  File "<string>", line 18, in <module>
NameError: name 'my_list' is not defined
```

We can use `remove()` to remove the given item or `pop()` to remove an item at the given index.

The `pop()` method removes and returns the last item if the index is not provided. This helps us implement lists as stacks (first in, last out data structure).

And, if we have to empty the whole list, we can use the `clear()` method.

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')
```

```
# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

Output

```
['r', 'o', 'b', 'l', 'e', 'm']
o
['r', 'b', 'l', 'e', 'm']
m
['r', 'b', 'l', 'e']
[]
```

Finally, we can also delete items in a list by assigning an empty list to a slice of elements.

```
>>> my_list = ['p','r','o','b','l','e','m']
>>> my_list[2:3] = []
>>> my_list
['p', 'r', 'b', 'l', 'e', 'm']
>>> my_list[2:5] = []
>>> my_list
```

```
['p', 'r', 'm']
```

Python List Methods

Python has many useful list methods that makes it really easy to work with lists. Here are some of the commonly used list methods.

Methods	Descriptions
append()	adds an element to the end of the list
extend()	adds all elements of a list to another list
insert()	inserts an item at the defined index
remove()	removes an item from the list
pop()	returns and removes an element at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the number of items passed as an argument
sort()	sort items in a list in ascending order
reverse()	reverse the order of items in the list
copy()	returns a shallow copy of the list

```
# Example on Python list methods

my_list = [3, 8, 1, 6, 8, 8, 4]
```

```
# Add 'a' to the end
my_list.append('a')

# Output: [3, 8, 1, 6, 8, 8, 4, 'a']
print(my_list)

# Index of first occurrence of 8
print(my_list.index(8))     # Output: 1

# Count of 8 in the list
print(my_list.count(8))   # Output: 3
```

List Comprehension: Elegant way to create Lists

List comprehension is an elegant and concise way to create a new list from an existing list in Python.

A list comprehension consists of an expression followed by for statement inside square brackets.
Here is an example to make a list with each item being increasing power of 2.

```
pow2 = [2 ** x for x in range(10)]
print(pow2)
```

Output

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

This code is equivalent to:

```
pow2 = []
for x in range(10):
```

```
    pow2.append(2 ** x)
```

A list comprehension can optionally contain more `for` or if statements. An optional `if` statement can filter out items for the new list. Here are some examples.

```
>>> pow2 = [2 ** x for x in range(10) if x > 5]
>>> pow2
[64, 128, 256, 512]
>>> odd = [x for x in range(20) if x % 2 == 1]
>>> odd
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> [x+y for x in ['Python ','C '] for y in ['Language','Programming']]
['Python Language', 'Python Programming', 'C Language', 'C Programming']
```

Visit Python list comprehension to learn more.

Other List Operations in Python

List Membership Test

We can test if an item exists in a list or not, using the keyword `in`.

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

Output

```
True
False
True
```

Iterating Through a List

Using a `for` loop we can iterate through each item in a list.

```
for fruit in ['apple','banana','mango']:
    print("I like",fruit)
```

Output

```
I like apple
I like banana
I like mango
```