{ Python }

Learning Path with Pankaj Chouhan

LISTS

{ Collection Data Type }

Python Lists

List is one of the most frequently used and versatile data type in Python.

A list is created by placing items (elements) inside square brackets `[]`, separated by commas. For example,

```
numbers = [1, 23, 43]
```

Here, we created a list named `numbers`. It contains 3 items.

We will learn more about creating lists next.

Create a List

A list can have any number of items. And, these items may be of different types (`int`, `float`, `str` etc.).

```
# empty list
list1 = []

# list of integers
list2 = [1, 2, 3]

# list with mixed data types
list3 = [1, "Hello", 3.4]
```

Also, a list can even have another list as an item. This is called a nested list.

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

Access Elements from a List

We can use the index operator `[]` to access an item in a list.

Suppose, a list has 5 items like this:

```
numbers = [1, 44, 10, 100, 3.33]
```

We access the first element using `numbers[0]`, the second element using `numbers[1]` and so on.

Index starts from **0 (not 1)**. So, a list having 5 items will have an index from 0 to 4.

```
numbers = [1, 44, 10, 100, 3.33]

print(numbers[0])    # 1
print(numbers[3])    # 100
print(numbers[4])    # 3.33
```

If you try to access the item outside of the index, you will get an `IndexError`. For example, if you try to access the sixth element in the above example with `numbers[5]`, you will get an error.

The index must be an integer. We cannot use float or other types. This results in `TypeError`.

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.

```
my_list = ['P', 'y', 't', 'h', 'o', 'n']

print(my_list[-1]) # Output: n
print(my_list[-5]) # Output: y
print(my_list[-6]) # Output: P
```

This list contains 6 items. In this case,

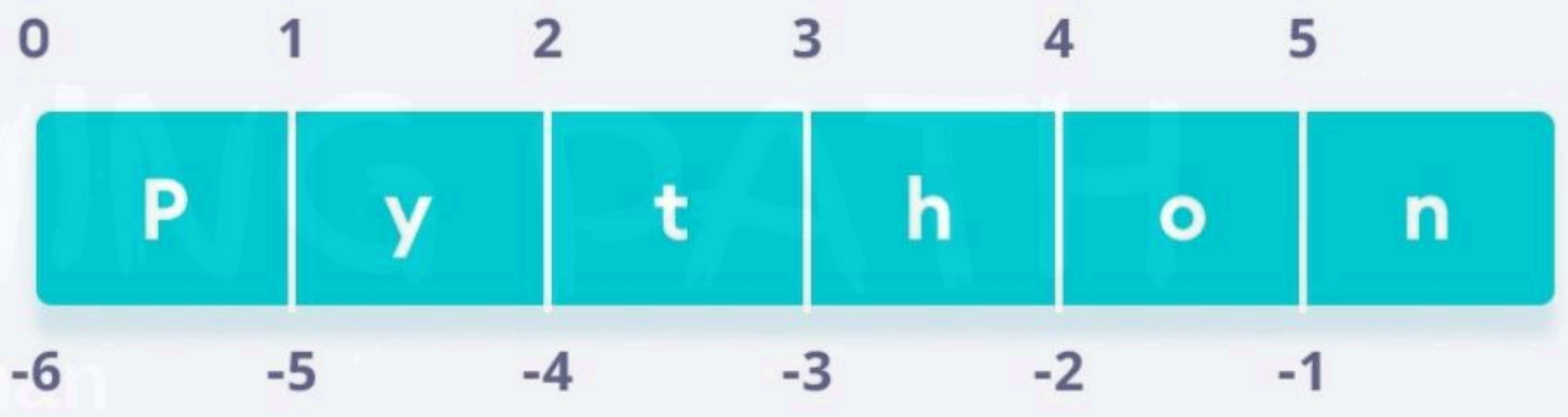
- Both `my_list[0]` and `my_list[-6]` gives us the first element `'P'`
- Both `my_list[1]` and `my_list[-5]` gives us the second element `'y'`
- Both `my_list[5]` and `my_list[-1]` gives us the last element `'n'`

Slicing of a List

In the previous few examples, we learn to access an item from a list. Now, we will learn to access a range of items. This is done by using the slicing operator `:`.

```
my_list = ['P', 'y', 't', 'h', 'o', 'n']

# elements 3rd to 5th
print(my_list[2:5])   # ['t', 'h', 'o']

# elements 4th to end
print(my_list[3:])    # ['h', 'o', 'n']

# elements beginning to 4th
print(my_list[:-5])   # ['P']

# elements beginning to end
print(my_list[:])     # ['P', 'y', 't',
'h', 'o', 'n']
```

Slicing can be best visualized by considering the index to be between the elements as shown below.



So if we want to access a range, we need two indexes that will slice that portion from the list.

Change Items of a List

Lists are mutable. Meaning, their items can be changed.

We can use the `=` operator to change an item or a range of items. Here's an example:

```
# mistake values
odd = [2, 4, 6, 8]

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

print(odd) # Output: [1, 4, 6, 8]

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

print(odd) # Output: [1, 3, 5, 7]
```

Add Elements to a List

To add a single item to a list, we can use the `append()` method. For example,

```
odd = [1, 3, 5]

odd.append(7)     # adding 7 to the list

print(odd)        # [1, 3, 5, 7]
```

If you need to add individual items from a list to another list, we use the `extend()` method. For example,

```
odd = [1, 3, 5]

odd.extend([7, 9, 11])

print(odd)        # [1, 3, 5, 7, 9, 11]
```

Using + and * Operators

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

If you need to repeat items of a list, you can use the `*` operator.

```
odd = [1, 3, 5]
print(odd + [9, 7, 5])

# Output: [1, 3, 5, 9, 7, 5]

print(['a', 'b'] * 3)

# Output: ['a', 'b', 'a', 'b', 'a', 'b']
```

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.

```
odd = [1, 9]

odd.insert(1, 3) # inserting 3 at 1st index
print(odd)  # Output: [1, 3, 9]

odd[2:2] = [5, 7]
print(odd) # Output: [1, 3, 5, 7, 9]
```

Delete Items from a List

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

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

del my_list[2]    # delete one item (3rd
item)

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

del my_list[1:5]  # delete multiple items
print(my_list)    # Output: ['p', 'm']

del my_list       # delete entire list
print(my_list)    # Error: List not defined
```

Delete Items from a List (Part II)

We can use the `remove()` method to remove the given item or `pop()` method 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).

We can also use the `clear()` method to empty a list.

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

# remove item 1 (not item at 1st index)
my_list.remove(1)

print(my_list)    # Output: [2, 3, 4, 5, 6,
7]

# pop item at index 1
print(my_list.pop(1)) # Output: 3

# my_list after the item is popped
print(my_list)    # Output: [2, 4, 5, 6, 7]

# pop last item
print(my_list.pop())  # Output: 7

# my_list after the last item is popped
print(my_list)        # Output: [2, 4, 5,
6]

my_list.clear()  # clear list
print(my_list)   # Output: []
```

List Copy

You can use the = operator to copy one list to another. For example,

```
list1 = [1, 2, 3]
list2 = list1
print(list2)  # Output: [1, 2, 3]
```

Here, list1 and list2 are two different variables that point to the same list. When you change items of the list1 list, list2 is also changed as they are pointing to the same list.

```
list1 = [1, 2, 3]
list2 = list1

# changing the first item of list1 to 'one'
list1[0] = 'one'

print(list1)   # Output: ['one', 2, 3]
print(list2)   # Output: ['one', 2, 3]
```

The actual way of copying a list is by using the copy() method.

```
list1 = [1, 2, 3]
list2 = list1.copy()  # copying list1 to
list2

# changing the first item of list1 to 'one'
list1[0] = 'one'

print(list1)   # Output: ['one', 2, 3]
print(list2)   # Output: [1, 2, 3]
```

Python List Method

Here's a list of list methods we have learned in this lesson.

- `append()` - to add an item to a list
- `extend()` - to add all elements of a list to another list
- `insert()` - to insert an item at a given index
- `remove()` - to remove an item from a list
- `pop()` - to remove an item at a given index
- `clear()` - to remove all items from a list
- `copy()` - returns a shallow copy of the list

To learn about all the list methods, you can go to Google and search "Python list methods".

We recommend you to continue with the course and learn about more list methods later, once you complete this course.

Loop through a List

The easiest way to iterate through each item in a list is by using the `for` loop.

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

Output

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

Sometimes, we need to check whether an item is in a list or not. Here's how we can do it.

```
my_list = ['P', 'y', 't', 'h', 'o', 'n']

print('P' in my_list)    # True
print('o' in my_list)    # True
print('p' in my_list)    # False
```

Nested Lists

A list can have another list within it, called a nested list.

```
my_list = [1, 2, [3, 4, 5]]
```

Here, the third item of `my_list` is a list `[3, 4, 5]`.

Items of nested lists are accessed using nested indexing. Here's how:

```
my_list = [1, 2, [3, 4, 5]]

# third element's second element
result = my_list[2][1]
print(result)    # Output: 4

# third element
result = my_list[2]
print(result)    # Output: [3, 4, 5]
```