MANIPAL
ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

MDS5103/ MSBA5104 Segment 04

LISTS -TUTORIAL

Table of Contents

Introduction

Lists are a single value for collections of elements containing any type of values such as float, integers, booleans, strings and even lists within the lists. In this topic, we will explore the list datatype, the operations that can be performed on it and so on.

1. Creation of List

A list can be created using a squared bracket. For example - [a,b,c]. In this, the squared bracket can have various elements – strings, integers and so on. For example, ['a','1','b','2','c','3']. Integer and boolean are single value datatypes. However, a list is a compound data type that can group values together.

In the code shown below, Income1, Income2 till Income5 are integers. They can be grouped together within square brackets and assigned to a list named Income. The output below displays the contents of the list.

```
#Income of a certain group
Income1 = 50000
Income2 = 75000
Income3 = 65000
Income4 = 45000
Income5 = 25000
Income = [Income1, Income2, Income3, Income4, Income5]
print(Income)
```

Output

```
[50000, 75000, 65000, 45000, 25000]
```

The list can contain heterogenous types. In the code below, the variable Income contains Strings and Integers.

```
Income = ["Sumit",50000,"Amit",75000,"Tina",65000,"Heena",45000,"Reena",25000]
Income
```

Output

```
['Sumit', 50000, 'Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]
```

The related elements in the above list can be grouped into a sublist and this can be fit into an outerlist as shown below,

```
# Lists can also contain Lists themselves
#Instead of putting strings between the number one can create sublists for each
# Income entry
Income2 = [['Sumit',50000],
['Amit',75000],
['Tina',65000],
['Heena',45000],
['Reena',25000]]
Income2 #The main list contains 5 sub lists
```

Output

```
[['Sumit', 50000],
 ['Amit', 75000],
 ['Tina', 65000],
 ['Heena', 45000],
 ['Reena', 25000]]
```

As shown in the code below, both Income and Income2 are of type list.

```
type(Income)
type(Income2)
```

Output

```
list
```

2. Slicing

Slicing refers to accessing a particular element in the list. This is also referred to as indexation. The index can be a single entry or a range of values.

The code below demonstrates the accessing of the sixth element of the list Income. This corresponds to the income of the user – Tina. Note that Python uses zero based indexing. The first element has index 0.

```
# Accessing Information from a List - Use Index
Income[5] #to access Income corresponding to Tina
#Zero Based Indexing
```

Output

```
65000
```

The syntax for slicing the index of a list is:

Syntax

```
[start:end]
```

'Start index' is inclusive while 'end index' is exclusive. The code below demonstrates the use of index ranges in a list.

```
#Slicing using :
#[start:end]
print(Income[2:6])
print(Income[:])
print(Income[2:])
print(Income[:2])
 #Income of males
Income[1] + Income[3]
```

Output

```
['Amit', 75000, 'Tina', 65000]
['Sumit', 50000, 'Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]
['Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]
['Sumit', 50000]
125000
```

If both the start position and end position are specified, for example, 'print(Income[2:6])' we get ['Amit', 75000, 'Tina', 65000].

If both the start and end position are not specified, for example, 'print(Income[:])' we get the whole list ['Sumit', 50000, 'Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000].

If the start position is specified, but end position is not specified, for example, 'print(Income[2:])', we get the output ['Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]. This implies that all elements starting from the second position till the end.

If only the end position is specified, for example, 'print(Income[:2])', we get ['Sumit', 50000].

3. Negative Index

Python programming language supports negative indexing of arrays, something which is unavailable in arrays in other programming languages. This means that the index value of -1 gives the last element, and -2 gives the second last element of an array. The negative indexing starts from where the array ends.

Let us take a list ['Sumit', 50000, 'Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]. Here, 25000 is Index [-1] , 45000 is Index[-3] and 65000 is Index[-5] and so on. The code below shows the sum of these numbers.

```
#Income of females
Income[-1] + Income[-3] + Income[-5]
```

Output
```
135000
```

The code below demonstrates the access of the elements from a list within a list.

```
Income2[2][:] #Accessing all elements from list of list
```

Output
```
['Tina', 65000]
```

Here, two indexes are used. The first one is for the outer list and the next is for the inner list.

4. List Manipulations

The following are the manipulations that can be done on a list.

- Changing elements using '=' operator
- Adding new elements using '+' operator
- Removing elements using del

4.1 Changing Elements

We can assign new values to the element specified by the index using the '=' symbol. In the code below, the element specified by index 1, that is the second element, is changed to 55000.

```
#List Manipulations
Income[1] = 55000 #Change
Income
```

Output
```
['Sumit', 55000, 'Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]
```

In the output above, the second element is changed to 55000.

The code below demonstrates replacing elements in a range of values. In this the first two elements are replaced with the new values. The new Income list is displayed below.

```
Income[0:2] = ['Likhit',60000] #Replacing elements
Income
```

Output

```
        ['Likhit', 60000, 'Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000]
```

4.2 Adding Elements

To add new elements to the list, the '+' operator can be used. In the code below, the elements 'Sumit' and 50000 are added to the list 'Income_add'. The new 'Income_add' is displayed in the output.

```
Income_add = Income + ['Sumit',50000] #Adding elements
Income_add
```

Output

```
        ['Likhit',
         60000,
         'Amit',
         75000,
         'Tina',
         65000,
         'Heena',
         45000,
         'Reena',
```

4.3 Deleting Elements

To delete the element, the 'del' command is used. The list with the index is provided as a parameter. In the code below, the first two elements are deleted using the del command.

```
del Income_add[0:2] #Deleting elements
Income_add
```

Output

```
['Amit', 75000, 'Tina', 65000, 'Heena', 45000, 'Reena', 25000, 'Sumit', 50000]
```

4.4 Modifying Elements in Assigned Lists

The code below demonstrates the behaviour of the list when it is assigned to another list and modified. The code below creates a list 'age'. This is assigned to another list named 'age_new'.

```
# Create list Age
age = [11, 18, 20, 10, 9]
age_new = age
age_new
```

Output

```
[11, 18, 20, 10, 9]
```

The code below replaces the first element of the new list with 24, now the new list has the values that were replaced. The original list 'age' is displayed in the code below.

```
age_new[1] = 24
age
```

Output

```
[11, 24, 20, 10, 9]
```

The problem here is, the original value age also has the replaced value 24.

This is because the names of the list are only a reference to where a list is stored in the computer memory. It does not contain all the list elements but just the reference of the list elements.

To avoid this, we use ‘copy’ method as shown below. Now even if we change the new list, the original elements remain unaffected. The code below demonstrates the same.

```
age_final = age_new.copy()
age_final
```

Output

```
[11, 24, 20, 10, 9]
```

```
age_final[1] = 32
age_final
```

Output

```
[11, 32, 20, 10, 9]
```

```
age_new
```

Output

```
[11, 24, 20, 10, 9]
```

From the above output, we can see that the value remains unchanged, this is possible because of the ‘copy()’ function.

The behaviour is similar to the 'copy()' function, if we use slicing behaviour to assign the values to the new list. In the code below, 'age_final' is created using the slicing operator. Even if one of the lists is updated, the others remain unaffected.

```
age_final = age_new[:]
print(age_final)
age_final[1] = 32
print(age_final)
print(age_new)
```

Output

```
        [11, 24, 20, 10, 9]
        [11, 32, 20, 10, 9]
        [11, 24, 20, 10, 9]
```

E-references:

- Dive into Python by Mark Pilgrim

  Link: https://diveintopython3.net/
- Automate the Boring Stuff with Python by Al Sweigart

  Link: https://automatetheboringstuff.com/#toc