Department of Information Technology
Statistics with Python

Module 2: **Lists and Tuples**-Arrays, Creating and Accessing Lists, Manipulating Lists, Creating and Accessing Tuples, Data Frames.

Arrays

An array is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
The solution is an array!

Array is created as follows:

```
cars = ["Ford", "Volvo", "BMW"]
```

Access the Elements of an Array -An array can hold many values under a single name, and you can access the values by referring to an index number.

```
x = cars[0]
print(x)
```

Output: Ford

Modify the value of the first array item:

```
cars[0] = "Toyota"
```

The Length of an Array-Use the len() method to return the length of an array (the number of elements in an array).

Example
Return the number of elements in the cars array:

```
x = len(cars)
```

Adding Array Elements-You can use the append() method to add an element to an array.
Example
Add one more element to the cars array:

cars.append("Honda")

Removing Array Elements-You can use the pop() method to remove an element from the array.
Example
Delete the second element of the cars array:
cars.pop(1)

You can also use the remove() method to remove an element from the array.
Example
Delete the element that has the value "Volvo":
cars.remove("Volvo")

Creating and Accessing Lists

A list is a collection of similar or different types of data. For example,
Suppose we need to record the age of **5** students. Instead of creating **5** separate variables, we can simply create a list:

List of Age

Create a Python List-A list is created in Python by placing items inside [], separated by commas . For example,

```
# A list with 3 integers
numbers = [1, 2, 5]
print(numbers)
# Output: [1, 2, 5]
```

Here, we have created a list named numbers with **3** integer items.
A list can have any number of items and they may be of different types (integer, float, string, etc.). For example,

```
# empty list
my_list = []
```

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

Access Python List Elements -In Python, each item in a list is associated with a number. The number is known as a list index.

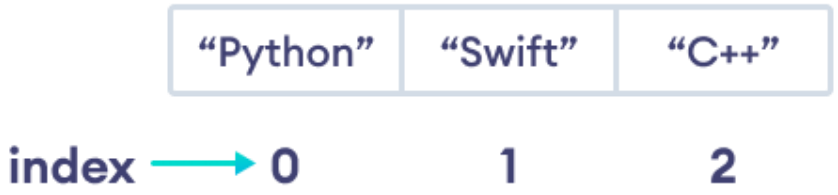
We can access elements of an array using the index number **(0, 1, 2 …)**. For example,

```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[0])   # Python

# access item at index 2
print(languages[2])   # C++
```

In the above example, we have created a list



Note: The list index always starts with 0. Hence, the first element of a list is present at index 0, not 1.

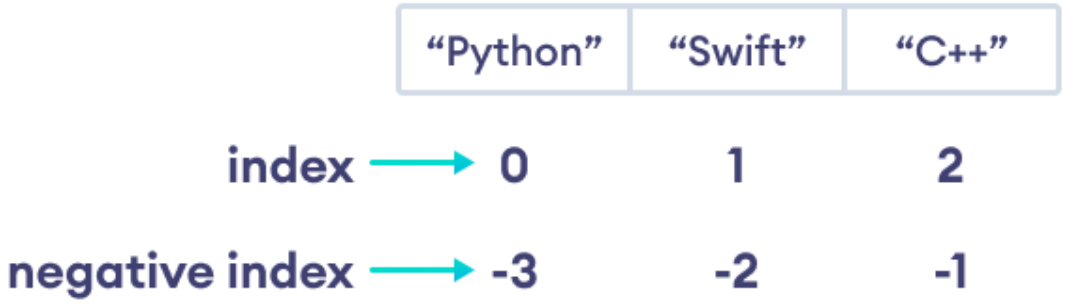
Negative Indexing in Python-Python allows negative indexing for its sequences. The index of **-1** refers to the last item, **-2** to the second last item and so on.

Let's see an example,

```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[-1])   # C++

# access item at index 2
print(languages[-3])   # Python
```



Slicing of a Python List-In Python it is possible to access a section of items from the list using the slicing operator :, not just a single item.

For example,

```
# List slicing in Python

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

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

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

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

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

Here,
`my_list[2:5]` returns a list with items from index **2** to index **4**.
`my_list[5:]` returns a list with items from index **1** to the end.
`my_list[:]` returns all list items

Note: When we slice lists, the start index is inclusive but the end index is exclusive.

Manipulation of Lists

Add Elements to a Python List

Python List provides different methods to add items to a list.
1. Using append()

The append() method adds an item at the end of the list. For example,

```
numbers = [21, 34, 54, 12]
print("Before Append:", numbers)
# using append method
numbers.append(32)
print("After Append:", numbers)
```

Output
Before Append: [21, 34, 54, 12]
After Append: [21, 34, 54, 12, 32]

In the above example, we have created a list named `numbers`. Notice the line,
`numbers.append(32)`
Here, `append()` adds **32** at the end of the array.

2. Using extend()
We use the extend() method to add all items of one list to another. For example,

```
prime_numbers = [2, 3, 5]
print("List1:", prime_numbers)


even_numbers = [4, 6, 8]
print("List2:", even_numbers)

# join two lists
prime_numbers.extend(even_numbers)

print("List after append:", prime_numbers)
```

Output
List1: [2, 3, 5]
List2: [4, 6, 8]
List after append: [2, 3, 5, 4, 6, 8]
In the above example, we have two lists named `prime_numbers` and `even_numbers`.

Change List Items

Python lists are mutable. Meaning lists are changeable. And, we can change items of a list by assigning new values using `=` operator. For example,

```
languages = ['Python', 'Swift', 'C++']
# changing the third item to 'C'
languages[2] = 'C'
print(languages)  # ['Python', 'Swift', 'C']
```

Here, initially the value at index **3** is `'C++'`. We then changed the value to `'C'` using

```
languages[2] = 'C'
```

Remove an Item From a List

1. Using del()
In Python we can use the del statement to remove one or more items from a list. For example,

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']
# deleting the second item
del languages[1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust', 'R']

# deleting the last item
del languages[-1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust']

# delete first two items
del languages[0 : 2]  # ['C', 'Java', 'Rust']
print(languages)
```

2. Using remove()

We can also use the remove() method to delete a list item. For example,

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']
# remove 'Python' from the list
languages.remove('Python')
print(languages) # ['Swift', 'C++', 'C', 'Java', 'Rust', 'R']
```

Here, `languages.remove('Python')` removes `'Python'` from the `languages` list.

Python List Methods

Python has many useful list methods that makes it really easy to work with lists.

Method	Description
append()	add an item to the end of the list
extend()	add items of lists and other iterables to the end of the list
insert()	inserts an item at the specified index
remove()	removes item present at the given index
pop()	returns and removes item present 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 specified item in the list
sort()	sort the list in ascending/descending order
reverse()	reverses the item of the list
copy()	returns the shallow copy of the list

Creating and Accessing Tuples

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

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

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Output

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
```

In the above example, we have created different types of tuples and stored different data items inside them.

we can also create tuples without using parentheses:

```
my_tuple = 1, 2, 3
my_tuple = 1, "Hello", 3.4
```

Create a Python Tuple With one Element

In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.

We will need a trailing comma to indicate that it is a tuple,

```
var1 = ("Hello") # string
var2 = ("Hello",) # tuple
```

Access Python Tuple Elements

Each element of a tuple is represented by index numbers **(0, 1, ...)** where the first element is at index **0**.We use the index number to access tuple elements. For example,

1. Indexing

We can use the index operator [] to access an item in a tuple, where the index starts from 0. So, a tuple having **6** elements will have indices from **0** to **5**.

```
# accessing tuple elements using indexing
letters = ("p", "r", "o", "g", "r", "a", "m", "i", "z")

print(letters[0])   # prints "p"
print(letters[5])   # prints "a"
```

In the above example,

letters[0] - accesses the first element

letters[5] - accesses the sixth element

2. 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. For example,

```
# accessing tuple elements using negative indexing
letters = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(letters[-1])   # prints 'z'
print(letters[-3])   # prints 'm'
```

In the above example,

letters[-1] - accesses last element

letters[-3] - accesses third last element

3. Slicing

We can access a range of items in a tuple by using the slicing operator colon :.

```
# accessing tuple elements using slicing
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# elements 2nd to 4th index
print(my_tuple[1:4])  #  prints ('r', 'o', 'g')

# elements beginning to 2nd
print(my_tuple[:-7]) # prints ('p', 'r')

# elements 8th to end
print(my_tuple[7:]) # prints ('i', 'z')

# elements beginning to end
print(my_tuple[:]) # Prints ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Output

```
('r', 'o', 'g')
('p', 'r')
('i', 'z')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Here,
`my_tuple[1:4]` returns a tuple with elements from index **1** to index **3**.
`my_tuple[:-7]` returns a tuple with elements from beginning to index **2**.
`my_tuple[7:]` returns a tuple with elements from index **7** to the end.
`my_tuple[:]` returns all tuple items.

Note: When we slice lists, the start index is inclusive but the end index is exclusive.

Data Frame

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.
Example

Create a simple Pandas DataFrame:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

Result

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the Data Frame is like a table with rows and columns.
Pandas use the loc attribute to return one or more specified row(s)

Example 1:
Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

Result

```
calories    420
duration     50
```

Name: 0, dtype: int64

Example 2:
Return row 0 and 1:

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

Result

```
   calories  duration
0       420        50
1       380        40
```

Named Indexes

With the index argument, you can name your own indexes.

Example-Add a list of names to give each row a name:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

Result

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Sources:

https://www.w3schools.com/python/python_arrays.asp
https://www.programiz.com/python-programming/list
https://www.programiz.com/python-programming/tuple
https://www.w3schools.com/python/pandas/pandas_dataframes.asp