Python Django training

UNIT Content

Unit 1: Introduction to Python

Unit 2:Object-Oriented Programming with Python

Unit 3: Introduction to Django

Unit 4: Django REST

UNIT 1

Unit 1: Introduction to Python

Python is by far the easiest language programming to learn, especially when you are a beginner and this is your first experience.

It actually doesn't have main limitations; it is in fact very powerful. And the living proof to that, is the existing hundreds of success stories in major companies all around the world using python.

The biggest advantage of Python is that it is easily readable. Just like reading English.

Sequential Types: Strings & List

In Python everything is object and string are an object too. Python string can be created simply by enclosing characters in the double quote.

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Print("Hello")

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

a = "Hello"

Print(a)

Reversing String

By using the reverse function, you can reverse the string. For example, if we have string "12345" and then if you apply the code for the reverse function as shown below.

```
string="12345"
print(''.join(reversed(string)))
```

Output

```
54321
```

Sequential Types: Strings & List

Split Strings

Split strings is another function that can be applied in Python let see for string "guru99 career guru99". First here we will split the string by using the command word.split and get the result.

```
word="guru99 career guru99"
print(word.split(' '))
```

Output

```
['guru99', 'career', 'guru99']
```

Using "join" function for the string

The join function is a more flexible way for concatenating string. With join function, you can add any character into the string.

For example, if you want to add a colon (:) after every character in the string "Python" you can use the following code.

```
print(":".join("Python"))
```

Output

```
P:y:t:h:o:n
```

Sequential Types: Strings & List

Python List

A list is exactly what it sounds like, a container that contains different Python objects, which could be integers, words, values, etc.

It is the equivalent of an array in other programming languages. It is represented by square brackets (and this is one of the attributes that differentiates it from tuples, which are separated by parentheses).

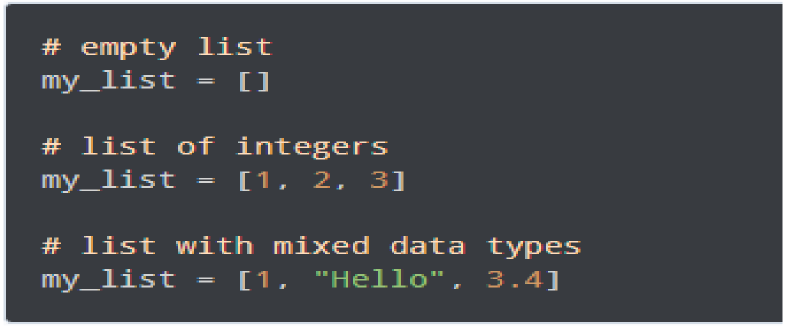
It is also mutable, that is, it can be modified or updated; unlike tuples, which are immutable.

In this tutorial, we'll learn everything about Python lists, how they are created, slicing of a list, adding or removing elements from them and so on.

How to create a list

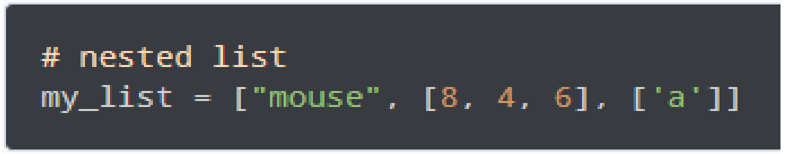
In Python programming, a list is created by placing all the items (elements) inside square brackets [], separated by commas.

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



```python
# empty list
my_list = []

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

# 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.



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

Sequential Types: Strings & List

Accessing values within lists

To access values within lists, the index of the objects inside the lists can be used. An index in Python lists refers to the position of an element within an ordered list. For example:

list = [3, 22, 30, 5.3, 20]

The first value in the list above, 3, has an index of 0

The second value, 22, has an index of 1

The third value, 30, has an index of 2

and so on. To access each of the values from the list, you would use:

list[0] to access 3

list[1] to access 22

list[2] to access 30

list[3] to access 5.3

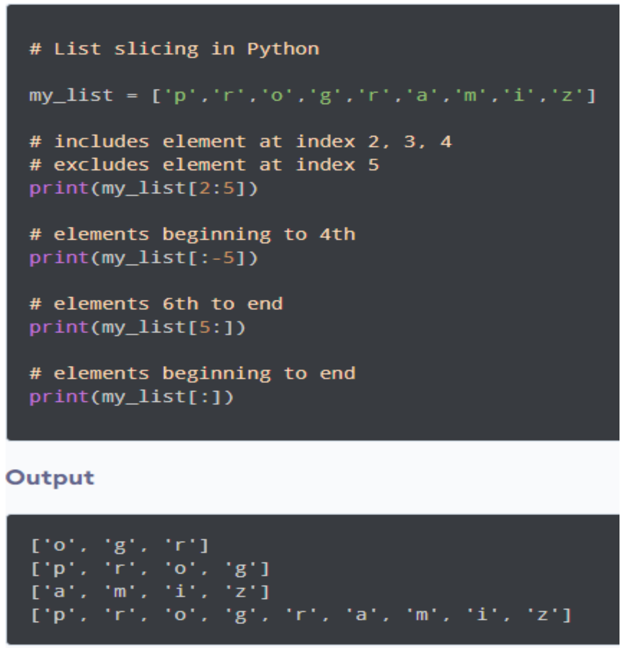
list[4] to access 20

The last member of a list can also be accessed by using the index -1. For example,

list[-1] = 20

How to slice lists in Python?

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



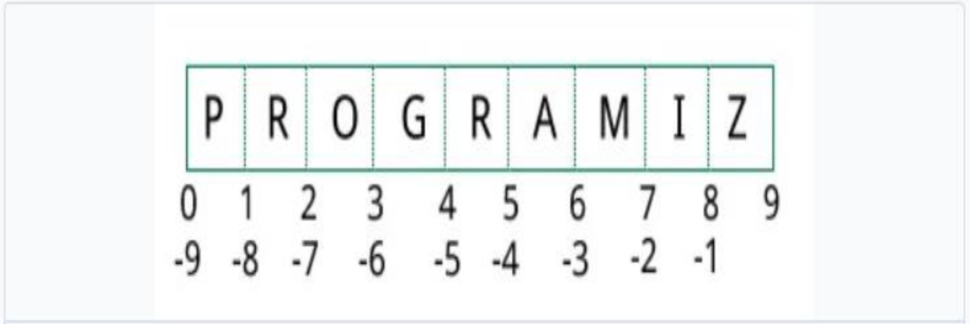
```
# List slicing in Python

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

# includes element at index 2, 3, 4
# excludes element at index 5
print(my_list[2:5])

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

# elements 6th to end
print(my_list[5:])

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

Output

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

Sequential Types: Strings & List

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 indices that will slice that portion from the list.

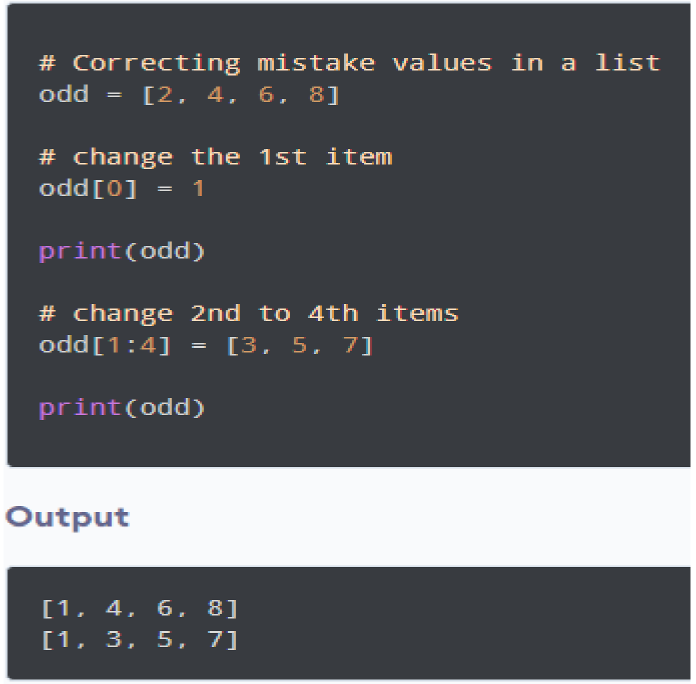


Element Slicing from a list in Python

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]
```

Sequential Types: Strings & List

Delete/Remove List Elements

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

If-Elif-else statements in python

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false.

Conditional statements are handled by IF statements in Python.

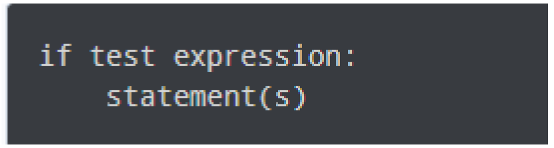
In this tutorial, we will see how to apply conditional statements in Python.

What is if...else statement in Python?

Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if...elif...else statement is used in Python for decision making.

Python if Statement Syntax



```
if test expression:
    statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the test expression is True.

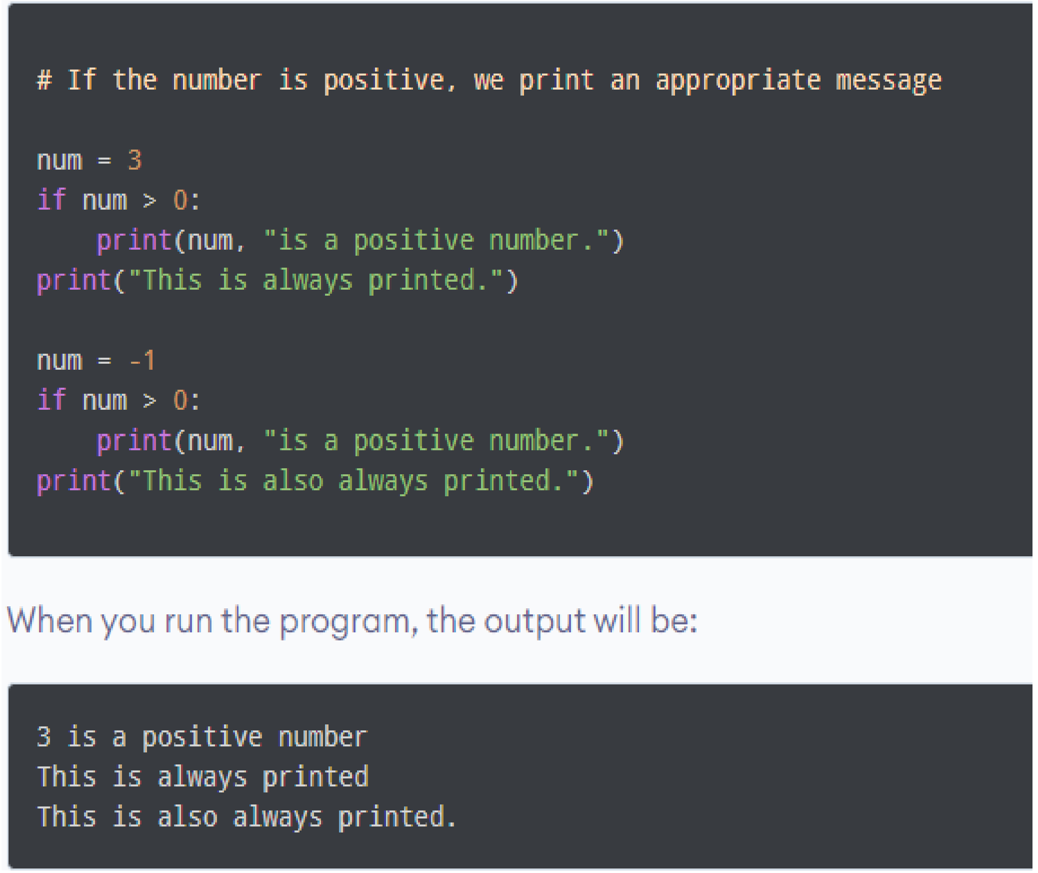
If the test expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation. The body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as True. None and 0 are interpreted as False.

If-Elif-else statements in python

Example: Python if Statement



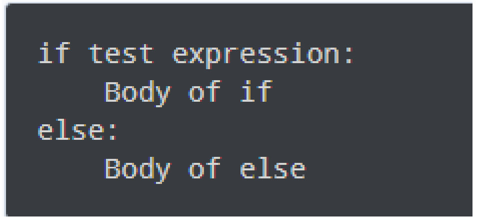
```
# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")

num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

When you run the program, the output will be:

```
3 is a positive number
This is always printed
This is also always printed.
```

Python if...else Statement

syntax



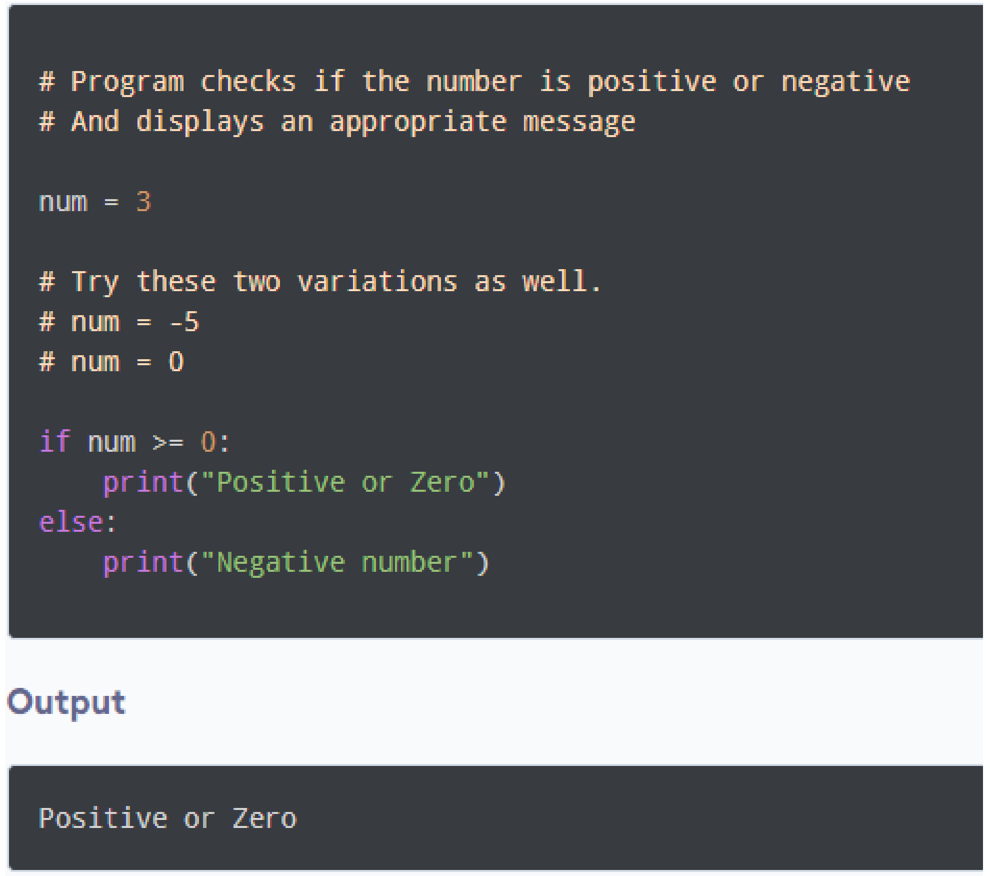
```
if test expression:
    Body of if
else:
    Body of else
```

The if..else statement evaluates test expression and will execute the body of if only when the test condition is True.

If the condition is False, the body of else is executed. Indentation is used to separate the blocks.

If-Elif-else statements in python

Example of if...else Statement



```
# Program checks if the number is positive or negative
# And displays an appropriate message

num = 3

# Try these two variations as well.
# num = -5
# num = 0

if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

Output

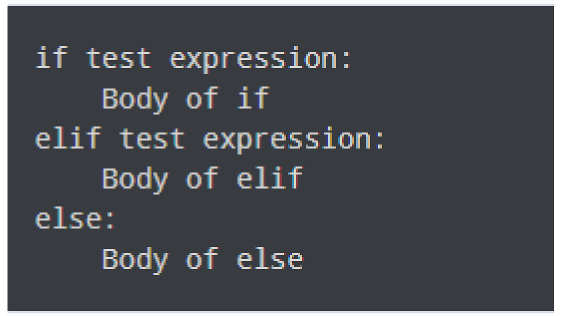
```
Positive or Zero
```

In the above example, when num is equal to 3, the test expression is true and the body of if is executed and the body of else is skipped.

If num is equal to -5, the test expression is false and the body of else is executed and the body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

Python if...elif...else Statement



```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

If-Elif-else statements in python

Cont. if...elif...else

The elif is short for else if. It allows us to check for multiple expressions.

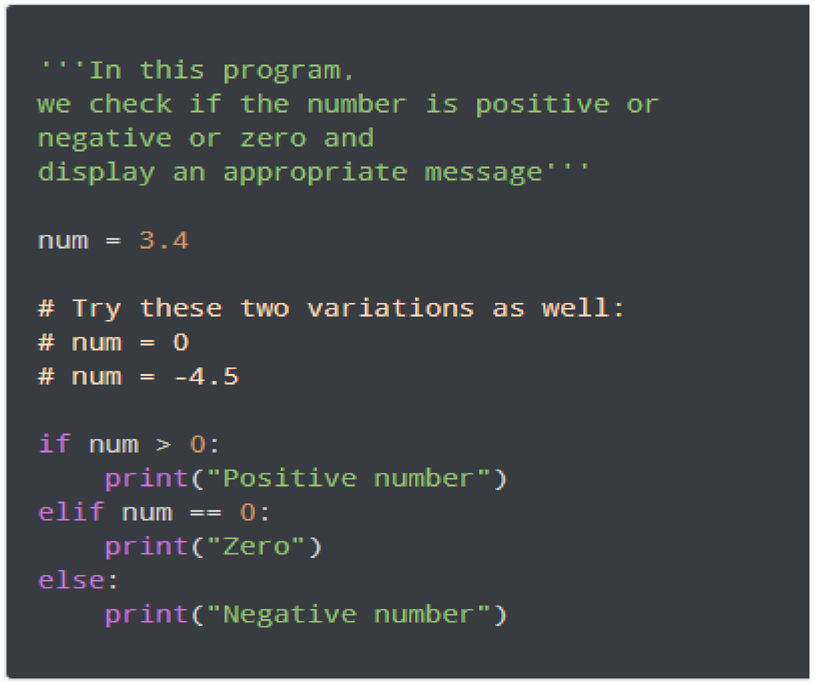
If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, the body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

Example of if…elif…else Statement



```python
'''In this program,
we check if the number is positive or
negative or zero and
display an appropriate message'''

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

When variable num is positive, Positive number is printed.

If num is equal to 0, Zero is printed.

If num is negative, Negative number is printed.

Definite loops: For loops

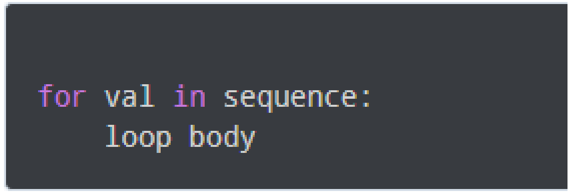
Python for Loop

In this article, you'll learn to iterate over a sequence of elements using the different variations of for loop.

What is for loop in Python?

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

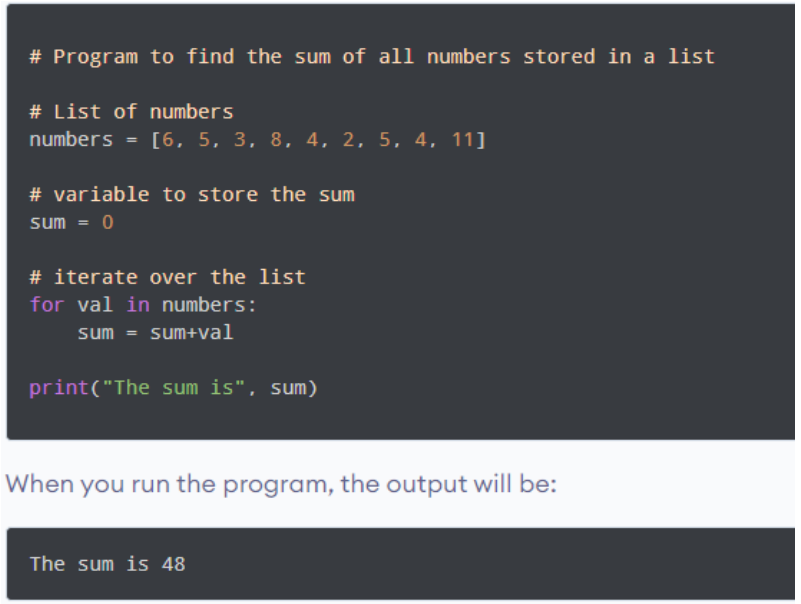


```
for val in sequence:
    loop body
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Example: Python for Loop



```
# Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

When you run the program, the output will be:

```
The sum is 48
```

Unit 1.2: Advanced Data Types in Python

Python Tuple

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.

For example –

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5);

tup3 = "a", "b", "c", "d";

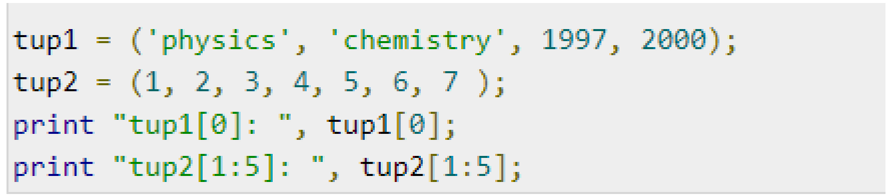
To write a tuple containing a single value you have to include a comma, even though there is only one value –

tup1 = (50,);

Accessing Values in Tuples

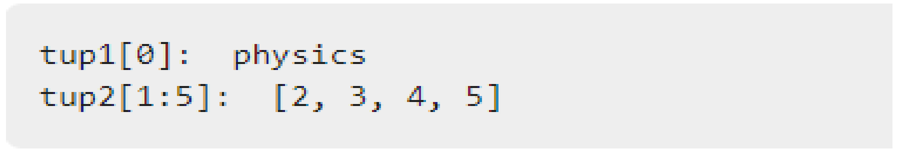
To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

For example –



```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0];
print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result –



```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

Data Types: Tuples

Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements.

You are able to take portions of existing tuples to create new tuples as the following example demonstrates

–For example –

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```

Output

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement.

For example –

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

Data Types: Dictionaries

Python Dictionary

Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.

A Dictionary in python is declared by enclosing a comma-separated list of key-value pairs using curly braces({}).

Syntax for Python Dictionary

```
Dict = { ' Tim': 18,  xyz,.. }
```

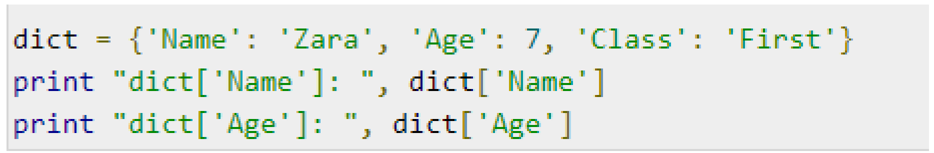
Dictionary is listed in curly brackets, inside these curly brackets, keys and values are declared. Each key is separated from its value by a colon (:), while commas separate each element.

Properties of Dictionary Keys

- ✓ More than one entry per key is not allowed (no duplicate key is allowed)
- ✓ The values in the dictionary can be of any type, while the keys must be immutable like numbers, tuples, or strings.
- ✓ Dictionary keys are case sensitive- Same key name but with the different cases are treated as different keys in Python dictionaries.

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example



```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Data Types: Dictionaries

Cont

When the above code is executed, it produces the following result –

```
dict['Name']:  Zara
dict['Age']:  7
```

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Reading and Writing txt and csv files

Read & Write txt/csv files

To create a new file in Python, use the open() method, with one of the following parameters:

- "x" - Create - will create a file, returns an error if the file exist
- "a" - Append - will create a file if the specified file does not exist
- "w" - Write - will create a file if the specified file does not exist

Example: to create a filled called “myfile.txt”

```
f = open("myfile.txt", "x")
```

Read a txt file

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

Example:

```
f = open("demofile.txt", "r")
print(f.read())
```

Reading and Writing txt and csv files

How to Read a CSV File

To read data from CSV files, you must use the reader function to generate a reader object.

The reader function is developed to take each row of the file and make a list of all columns. Then, you have to choose the column you want the variable data for.

```
#import necessary modules
import csv
with open('X:\data.csv','rt')as f:
  data = csv.reader(f)
  for row in data:
        print(row)
```

How to write CSV File

When you have a set of data that you would like to store in a CSV file you have to use writer() function. To iterate the data over the rows(lines), you have to use the writerow() function.

Consider the following example. We write data into a file “writeData.csv” where the delimiter is an apostrophe

```
#import necessary modules
import csv

with open('X:\writeData.csv', mode='w') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)

    #way to write to csv file
    writer.writerow(['Programming language', 'Designed by', 'Appeared', 'Extension'])
    writer.writerow(['Python', 'Guido van Rossum', '1991', '.py'])
    writer.writerow(['Java', 'James Gosling', '1995', '.java'])
    writer.writerow(['C++', 'Bjarne Stroustrup', '1985', '.cpp'])
```

Unit 1.3: Writing iterative Code and algorithms

Python while loops

What is while loop in Python?

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know the number of times to iterate beforehand.

Syntax of while Loop in Python

```
while test_expression:
    Body of while
```

In the while loop, test expression is checked first. The body of the loop is entered only if the test expression evaluates to True.

After one iteration, the test expression is checked again. This process continues until the test expression evaluates to False.

Example: Python while Loop

```
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))

n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

Runtime Complexity

In computer science, the time complexity is the computational complexity that describes the amount of time it takes to run an algorithm. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform.

Time Complexity

Time Complexity of an algorithm is the representation of the amount of time required by the algorithm to execute to completion. Time requirements can be denoted or defined as a numerical function t(N), where t(N) can be measured as the number of steps, provided each step takes constant time.

For example, in case of addition of two n-bit integers, N steps are taken. Consequently, the total computational time is t(N) = c*n, where c is the time consumed for addition of two bits. Here, we observe that t(N) grows linearly as input size increases.

More on:

- https://www.tutorialspoint.com/time-and-space-complexity-in-data-structure
- https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/
- https://adrianmejia.com/how-to-find-time-complexity-of-an-algorithm-code-big-o-notation/
- https://stackabuse.com/big-o-notation-and-algorithm-analysis-with-python-examples/

Searching and sorting algorithms

Sorting Algorithms

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Below we see five such implementations of sorting in python.

- ❖ Bubble Sort
- ❖ Merge Sort
- ❖ Insertion Sort
- ❖ Shell Sort
- ❖ Selection Sort

Bubble Sort

It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Example

```
def bubblesort(list):

# Swap the elements to arrange in order
   for iter_num in range(len(list)-1,0,-1):
      for idx in range(iter_num):
         if list[idx]>list[idx+1]:
            temp = list[idx]
            list[idx] = list[idx+1]
            list[idx+1] = temp
list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

Output

```
[2, 6, 11, 19, 27, 31, 45, 121]
```

Searching and sorting algorithms

Searching Algorithms

Searching is a very basic necessity when you store data in different data structures. The simplest approach is to go across every element in the data structure and match it with the value you are searching for. This is known as Linear search. It is inefficient and rarely used, but creating a program for it gives an idea about how we can implement some advanced search algorithms.

Linear Search

In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data structure.

Example:

```
def linear_search(values, search_for):
    search_at = 0
    search_res = False
# Match the value with each data element
    while search_at < len(values) and search_res is False:
        if values[search_at] == search_for:
            search_res = True
        else:
            search_at = search_at + 1
    return search_res
l = [64, 34, 25, 12, 22, 11, 90]
print(linear_search(l, 12))
print(linear_search(l, 91))
```

Output:

```
True
False
```

Custom functions in Python Args and Kwargs Importing files

*args and **kwargs in Python

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

Special Symbols Used for passing arguments:-

1.)*args (Non-Keyword Arguments)

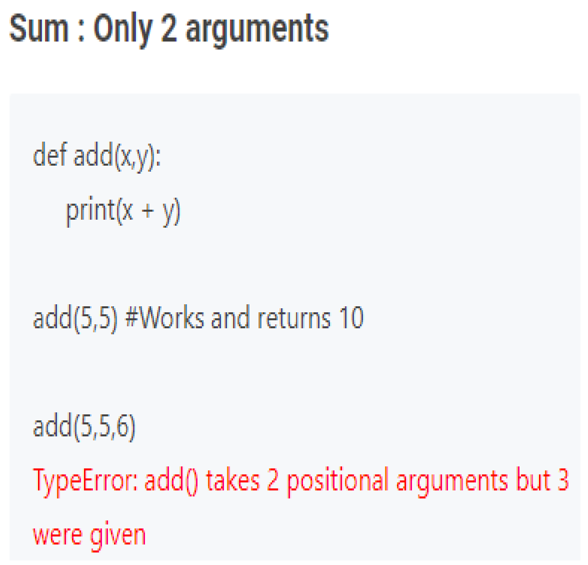
2.)**kwargs (Keyword Arguments)

What is *args?

args is a short form of arguments. With the use of *args python takes any number of arguments in user-defined function and converts user inputs to a tuple named args. In other words, *args means zero or more arguments which are stored in a tuple named args.

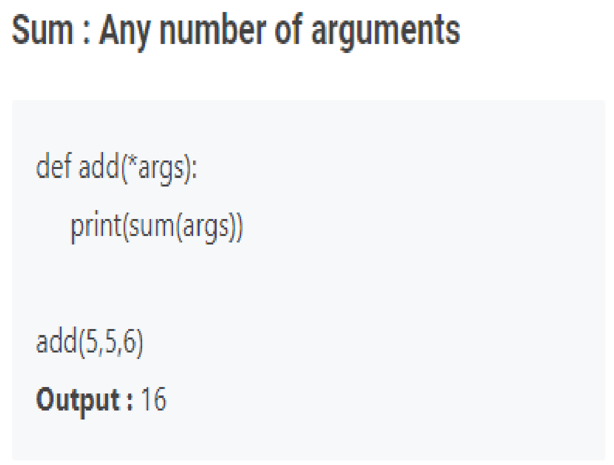
When you define function without *args, it has a fixed number of inputs which means it cannot accept more (or less) arguments than you defined in the function.

In the example code below, we are creating a very basic function which adds two numbers. At the same time, we created a function which can sum more than 2 numbers. Even the function can take less than 2 arguments (zero argument will also work!)



Sum : Only 2 arguments

```
def add(x,y):
    print(x + y)

add(5,5) #Works and returns 10

add(5,5,6)
TypeError: add() takes 2 positional arguments but 3
were given
```



Sum : Any number of arguments

```
def add(*args):
    print(sum(args))

add(5,5,6)
Output : 16
```

Custom functions in Python Args and Kwargs Importing files

What is **kwargs?

**kwargs stands for keyword arguments. It is very similar to *args in terms of its objective but there are some differences explained below –

**kwargs creates dictionary whereas *args creates tuple.

By using **kwargs you can name your arguments which is not possible in *args.

```
def test2(**kwargs):
    print(kwargs)
    print(kwargs.keys())
    print(kwargs.values())

test2(a=1, b=3)
```

Output :

```
{'a': 1, 'b': 3}
dict_keys(['a', 'b'])
dict_values([1, 3])
```

Strings of each keyword are the keys of the dictionary and values assigned after = symbol are the values of the dictionary. It is also called named arguments.

Examples : Uses of kwargs

Suppose you need to build a conditional statement which says sum of all the arguments if a fixed named argument called flag is set 'Yes'. Otherwise ignore the calculation and don't sum values and also throws a message which hints users why calculation has been ignored.

```
def calc(flag='Yes', **i):
    if flag == 'Yes' :
        return sum(i.values())
    else :
        print("Flag is set No")

calc(x=17, y=20)
```

Output : 37

```
calc(flag='No', x=17, y=20)
```

Output : Flag is set No

END OF UNIT 1