Lab 2

Iterative Structures in Python

Python programming language provides following types of loops to handle looping requirements. Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

While Loop:

In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

Syntax :

```
while expression:
    statement(s)
```

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Example:

```
# Python program to illustrate
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

Single statement while block:

Just like the if block, if the while block consists of a single statement the we can declare the entire loop in a single line as shown below:

Example:

```
# Python program to illustrate
```

```
# Single statement while block
count = 0
while (count == 0): print("Hello Geek")
```

for in Loop:

For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to for each loop in other languages. Let us learn how to use for in loop for sequential traversals.

Syntax:

```
for iterator_var in sequence:
    statements(s)
```

It can be used to iterate over iterators and a range

Example 1

```
# Python program to illustrate
# Iterating over a list
print("List Iteration")
l = ["geeks", "for", "geeks"]
for i in l:
        print(i)
```

Example 2

```
# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
        print(i)
```

Example 3

```
# Iterating over a String

print("\nString Iteration")

s = "Geeks"

for i in s :

        print(i)
```

Iterating by index of sequences:

We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and in iterate over the sequence within the range of this length. See the below example:

```
# Python program to illustrate

# Iterating by index

list = ["geeks", "for", "geeks"]

for index in range(len(list)):

        print list[index]
```

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Continue Statement:

It returns the control to the beginning of the loop.

```
# Prints all letters except 'e' and 's'

for letter in 'geeksforgeeks':

        if letter == 'e' or letter == 's':

                continue
```

```
        print 'Current Letter :', letter

        var = 10
```

Break Statement:

It brings control out of the loop

```
for letter in 'geeksforgeeks':

        # break the loop as soon it sees 'e'

        # or 's'

        if letter == 'e' or letter == 's':

                break

print 'Current Letter :', letter
```

Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

Example

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():
  print("Hello from a function")

my_function()
```

Parameters

Information can be passed to functions as parameter.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without parameter, it uses the default value:

Example

```
def
  my_functi
  on(countr
  y =
```

```
    "Norway"
    ): print("I
    am from "
    + country)

my_function("Sweden")

my_function("India")

my_function()

my_function("Brazil")
```

Passing a List as a Parameter

You can send any data types of parameter to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as a parameter, it will still be a List when it reaches the function:

Example

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"] my_function(fruits)
```

Return Values

To let a function return a value, use the return statement:

Example

```
def my_function(x):
  return 5 * x
```

```
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Keyword Arguments

You can also send arguments with the *key* = *value* syntax. This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class,

use the

keyword

class:

Example

Create a class named MyClass, with a property named x:

```
class MyClass: x = 5
```

Create Object

Now we can use the class named myClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass() print(p1.x)
```

The_init_() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in_init_() function.

All classes have a function called_init_(), which is always executed when the class is being initiated.

Use the_init_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
def_init_(self, name, age): self.name = name
```

```
self.age = age


p1 = Person("John", 36)

print(p1.name)

 print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create

a method in

the Person

class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
def __init__(self, name, age): self.name = name

self.age = age


def myfunc(self):
print("Hello my name is " + self.name)

p1 = Person("John", 36) p1.myfunc()
```