Python Loops

Loops

- Repetitive execution of the same block of code over and over is referred to as **iteration**.
- There are two types of iteration:
    - **Definite** iteration, in which the number of repetitions is specified explicitly in advance
    - **Indefinite** iteration, in which the code block executes until some condition is met
- In Python, indefinite iteration is performed with a while loop.

While loop

- While (condition)
  - statements
  - Increment

- Count=1
- While count <11:
- Print (count)
- Count=count+1

Numeric Range Loop

The most basic for loop is a simple numeric range statement with start and end values

```
for i = 1 to 10
    <loop body>
```

Three-Expression Loop

- Another form of for loop popularized by the C programming language contains three parts:

- An initialization
- An expression specifying an ending condition
- An action to be performed at the end of each iteration.
- This type of has the following form:

- for (i = 1; i <= 10; i++)
-     <loop body>

Collection-Based or Iterator-Based Loop

- This type of loop iterates over a collection of objects, rather than specifying numeric values or conditions:
- for i in <collection>
-     <loop body>
- Each time through the loop, the variable i takes on the value of the next object in <collection>.

collection-based iteration

- for <var> in <iterable>:
-     <statement(s)>

- >>> a = ['foo', 'bar', 'baz']
- >>> for i in a:
- ...   print(i)
- ...
- foo
- bar
- baz

The range() Function

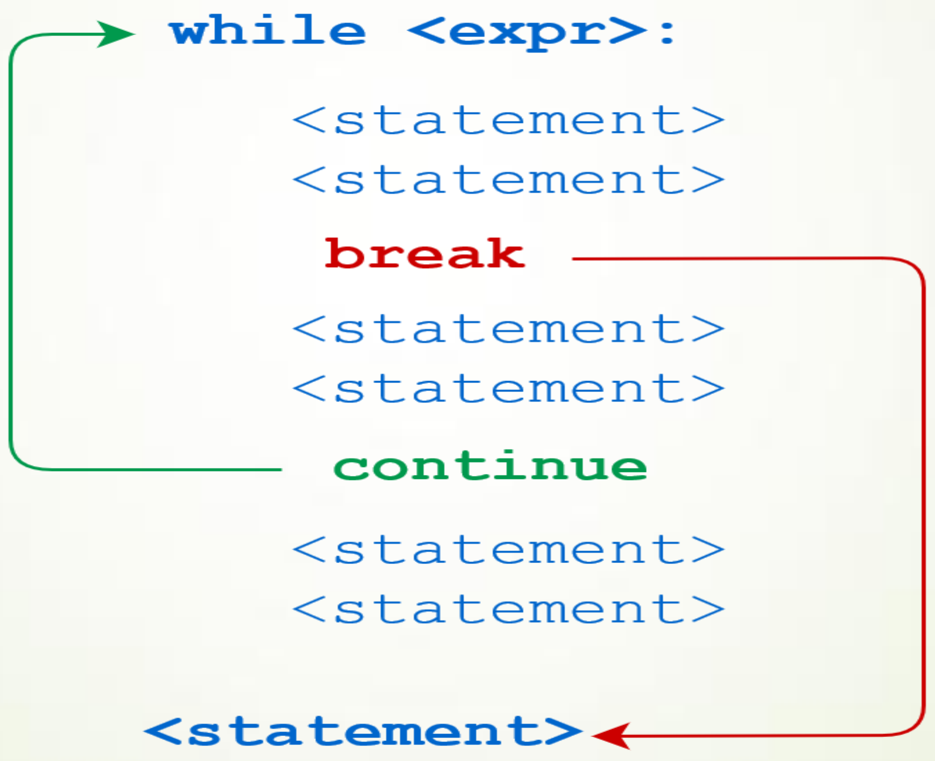
- the built-in range() function, which returns an iterable that yields a sequence of integers.

```
>>> x = range(5)
>>> x
range(0, 5)
>>> type(x)
<class 'range'>
```

Interruption of Loop Iteration

- **break** immediately terminates a loop entirely. Program execution proceeds to the first statement following the loop body.

- **continue** immediately terminates the current loop iteration. Execution jumps to the top of the loop, and the controlling expression is re-evaluated to determine whether the loop will execute again or terminate.



```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>
<statement>
```

If …..then ……else

- Age =80
- If age>60
  - Print (‘Senior citizen’)
  - Else
  - Print(‘ medium age’)

Splitting, Concatenating, and Joining Strings in Python

- >>> s = ' this   is my string '
- >>> s.split()
- ['this', 'is', 'my', 'string']
- String.lower()
- String.upper()
- Concatenation of strings
- >>> 'a' + 'b' + 'c'
- 'abc'
- LAB NOW ……………………..

Global vs Local Variables

- variables that are defined inside a function body have a local scope,
- and those defined outside have a global scope.
- That means that local variables are defined within a function block and can only be accessed inside that function,
- while global variables can be obtained by all functions that might be in your script:

Global and local variables…..

- X=100 // global variable
- Def myfunction()
  - K=78   // local variable
  - Print(k)

Functions in Python

- Function is a set of instructions that you want to use repeatedly
- self-contained in a sub-program
- Function is a piece of code written to carry out a specified task.
- To carry out that specific task, the function might or might not need multiple inputs.
- When the task is carried out, the function can or can not return one or more values.

Types of functions in Python

- **Built in function**
  - Help()
  - Min()
  - Max()
  - Print()
- **User defined function**
  - Created by users   by using **def** keyword
- **Anonymous functions,**
  - which are also called **lambda** functions because
  - they are not declared with the standard def keyword.

Anonymous functions
Lambda

- Anonymous functions are also called lambda functions in Python because
- instead of declaring them with the standard def keyword,
- we use the lambda keyword.
- p=lambda x: x*2
- print (p)

Main () function in python

- main function is required to execute the function
- # Define `main()` function
- def main():
  - hello()
- print("This is a main function")main()
- Main()

Main () function in python…….

- If __name__ == '__main__'

Overview of OOP Terminology
(Object-Oriented Programming)

- Python has been an object-oriented language since it existed.

- Class
- Data Member
- Function
- Instance or object

Classes

- A user-defined prototype for an object that defines a set of attributes that characterize any object of the class.
- The attributes are data members (class variables and instance variables) and methods, accessed via dot notation
- Classes provide a means of bundling data and functionality together.
- Creating a new class creates a new *type* of object, allowing new *instances* of that type to be made.

Class components …

- **Data member** – A class variable or instance variable that holds data associated with a class and its objects.
- **Function** -- function is a code that specify some task
- **Instance** – An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- objcircle=Circle()

Example

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

Creating Instance Objects

- "This would create first object of Employee class"
- emp1 = Employee("Zara", 2000)
- "This would create second object of Employee class"
- emp2 = Employee("Manni", 5000)

Accessing Attributes

- emp1.displayEmployee()
- emp2.displayEmployee()
- print "Total Employee %d" % Employee.empCount

The __init__() Function

- All classes have a function called __init__(), which is always executed when the class is being initiated

Example of __init__

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

  p1 = Person("John", 36)

  print(p1.name)
  print(p1.age)
  ```

- the __init__() function is called automatically every time the class is being used to create a new object

Modules in Python

- Consider a module to be the same as a code library.
- A file containing a set of functions you want to include in your application.

Create a Module

- File name ------ mymodule.ipynb

- def greeting(name):
-   print("Hello, " + name)

- USE a module

- **import mymodule**
- mymodule.greeting(“hello world”)

Inbuilt modules

- **Import** sys   ------ for system calls (hardware related)
- **Import** math
  - Math.pi
- **Import** os ---- for operating system
  - os.getcwd -------- get working directory
- **Import** re -----   regex functions -----regular functions

Packages in Python

- A package is a hierarchical file directory structure that defines a single Python application environment that consists of
- modules
- and subpackages
- and sub-subpackages,
- and so on.

Directory structure for packages

- Directory -- Mobile
  - File ---- Apple.ipynb having a function iphone()
  - File ---- LG.ipynb having a function work()

```
Import Mobile
From  Mobile import  Apple
From Mobile import LG
Apple.iphone()
LG.work()
```

Python File I/O operations

- What is a file?
- How to open a file?
- How to close a file Using Python?
- How to write to File Using Python?
- How to read files in Python?
- Python File Methods

What is a file?

- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

Python, a file operation takes place in the following order

- Open a file
- Read or write (perform operation)
- Close the file

open a file

- Python has a built-in function open() to open a file.
-
- >>> f = open("test.txt")   # open file in current directory
- >>> f = open("C:/Python33/README.txt")
    - # specifying full path

Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

- f = open("test.txt")   # equivalent to 'r' or 'rt'
- f = open("test.txt",'w') # write in text mode
- f = open("img.bmp",'r+b') # read and write in binary mode

What is Unicode?

- In computer systems, characters are transformed and stored as numbers (sequences of bits) that can be handled by the processor.
- A code page is an encoding scheme that maps a specific sequence of bits to its character representation.
- The pre-Unicode world was populated with hundreds of different encoding schemes that assigned a number to each letter or character.
- Many such schemes included code pages that contained only 256 characters - each character requiring 8 bits of storage.

The importance of Unicode

- From a translation/localization point of view, Unicode is an important step towards standardization, at least from a tools and file format standpoint.
- Unicode enables a single software product or a single website to be designed for multiple platforms, languages and countries (no need for re-engineering) which can lead to a significant reduction in cost over the use of legacy character sets.

- ASCII (American Standard Code for Information Interchange) became the first widespread encoding scheme. However, it's limited to only 128 character definitions. This is fine for the most common English characters, numbers, and punctuation, but is a bit limiting for the rest of the world.
- Naturally, the rest of the world wants the same encoding scheme for their characters too. However, for a little, while depending on where you were, there might have been a different character displayed for the same ASCII code.

UTF -8

- It became apparent that a new character encoding scheme was needed, which is when the Unicode standard was created. The objective of Unicode is to unify all the different encoding schemes so that the confusion between computers can be limited as much as possible.
- These days, **the Unicode standard defines values for over 128,000 characters** and can be seen at the Unicode Consortium. It has several character encoding forms:
- **UTF-8:** Only uses one byte (8 bits) to encode English characters. It can use a sequence of bytes to encode other characters. UTF-8 is widely used in email systems and on the internet.
- **UTF-16:** Uses two bytes (16 bits) to encode the most commonly used characters. If needed, the additional characters can be represented by a pair of 16-bit numbers.
- **UTF-32:** Uses four bytes (32 bits) to encode the characters. It became apparent that as the Unicode standard grew, a 16-bit number is too small to represent all the characters. UTF-32 is capable of representing every Unicode character as one number.

Utf -8 (file encoding format)

- **UTF-8** - the most popular type of Unicode encoding. It uses one byte for standard English letters and symbols, two bytes for additional Latin and Middle Eastern characters, and three bytes for Asian characters.
- Additional characters can be represented using four bytes. UTF-8 is backwards compatible with ASCII, since the first 128 characters are mapped to the same values.
- Moreover, the default encoding is platform dependent
- Hence, when working with files in text mode, it is highly recommended to specify the encoding type.
- f = open("test.txt",mode = 'r',encoding = 'utf-8')
- **UTF-8** is a variable width character encoding capable of encoding all 1,112,064 valid code points in Unicode using one to four **8**-bit bytes. The encoding is defined by the Unicode Standard,

close a file

- f = open("test.txt",encoding = 'utf-8')
- # perform file operations
- f.close()

Pickle in Python: Object Serialization

- Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening.
- Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network.
- Later on, this character stream can then be retrieved and de-serialized back to a Python object.

Use of Pickle

- Pickling is useful for applications where you need some degree of persistency in your data.
- Your program's state data can be saved to disk, so you can continue working on it later on.
- It can also be used to send data over a Transmission Control Protocol (TCP) or socket connection, or to store python objects in a database.
- Pickle is very useful for when you're working with machine learning algorithms,

Storing data with pickle

- You can pickle objects with the following data types:
- Booleans,
- Integers,
- Floats,
- Complex numbers,
- (normal and Unicode) Strings,
- Tuples,
- Lists,
- Dictionaries