Introduction To Python

Agenda

1
- History of python

2
- Features of python

3
- Basic syntax

History of Python

- **Python** was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- **Python** is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- **Python** is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- **Python** is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Cons..

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features of Python

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Cons..

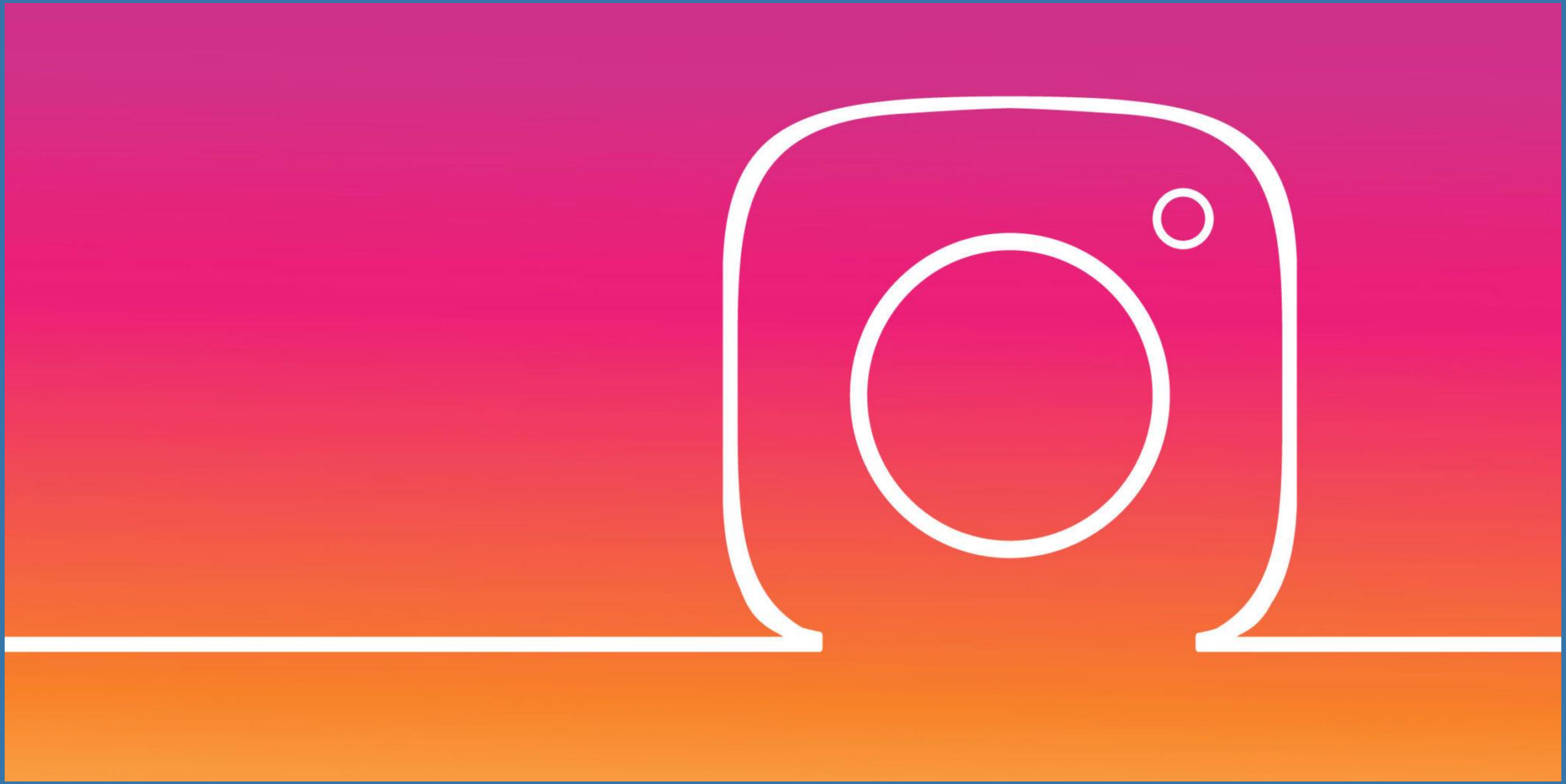
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix

Amazing Projects Using Python

Google

YouTube

Instagram



Battlefield 2

```
SLAYER 1C4RUS09 [AK-101]  ImtellingMom
 GPalleja [SVD]  BLUTORAIDERS
SLAYER 1C4RUS09 [AK-101] SgtMaj J.Harris[22ndMEU
 Devius_Howius [M249 SAW] SLAYER 1C4RUS09
 GPalleja [SVD]  Devius_Howius

 marcusbr08: Enemy units on the move
 aparicedo: Clear!
 aparicedo: Recharging
 aparicedo: You'll live buddy, now get back to the fight!
```

The Sims 4

ANACONDA

Basic Syntax

1. Variables Types
2. Basic Operators
3. Decision Making
4. Loops
5. Numbers
6. Strings
7. Lists
8. Function
9. Classes & Objects
10. Files I/O

Variable Types

- As in every language, a variable is the name of a memory location
- **Python** is weakly typed That is, you don't declare variables to be a specific type
- A variable has the type that corresponds to the value you assign to it
- Variable names begin with a letter or an underscore and can contain letters, numbers, and underscores
- **Python** has reserved words that you can't use as variable names

Python Reserved Words

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Cons..

- At the >>> prompt, do the following:

```
x=5

type(x)

x=“this is text”

type(x)

x=5.0

type(x)
```

Input

- The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action

```
input("\n\nPress the enter key to exit.")
```

Printing

- You’ve already seen the print statement
- You can also print numbers with formatting
- These are identical to Java or C format specifiers

Print (“Hello World”)

Comments

- All code must contain comments that describe what it does
- In Python, lines beginning with a # sign are comment lines

➢You can also have comments on the same line as a statement

```
# This entire line is a comment
          x=5        # Set up loop counter
```

- Multiple Line comment “”” `Comments` “””

Exercise

- Write a Python program which accepts the radius of a circle from the user and compute the area.

Day 2 ☺

Basic Operator

Types of Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators

Arithmetic operators

- **Arithmetic operators** we will use:

+ - * /	addition, subtraction/negation, multiplication, division
%	modulus, a.k.a. remainder
**	exponentiation

- **precedence**: Order in which operations are computed.
  - * / % ** have a higher precedence than + -

    1 + 3 * 4 is

- Parentheses can be used to force a certain order of evaluation.

  (1 + 3) * 4 is

Arithmetic operators

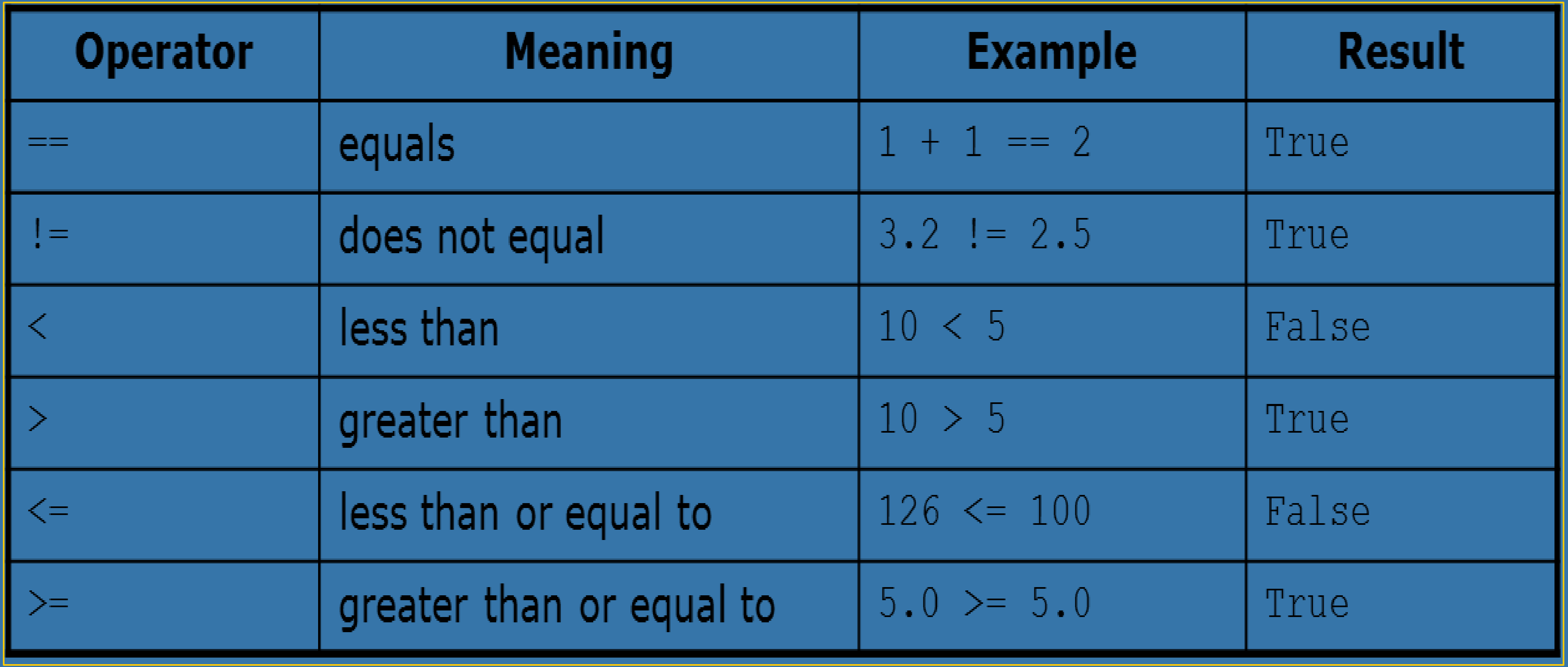
- Arithmetic **operators** we will use:
  - + - * / addition, subtraction/negation, multiplication, division
  - % modulus, a.k.a. remainder
  - ** exponentiation
- **precedence**: Order in which operations are computed.
  - * / % ** have a higher precedence than + -

    1 + 3 * 4 is 13
- Parentheses can be used to force a certain order of evaluation.

  (1 + 3) * 4 is 16

Relational Operators

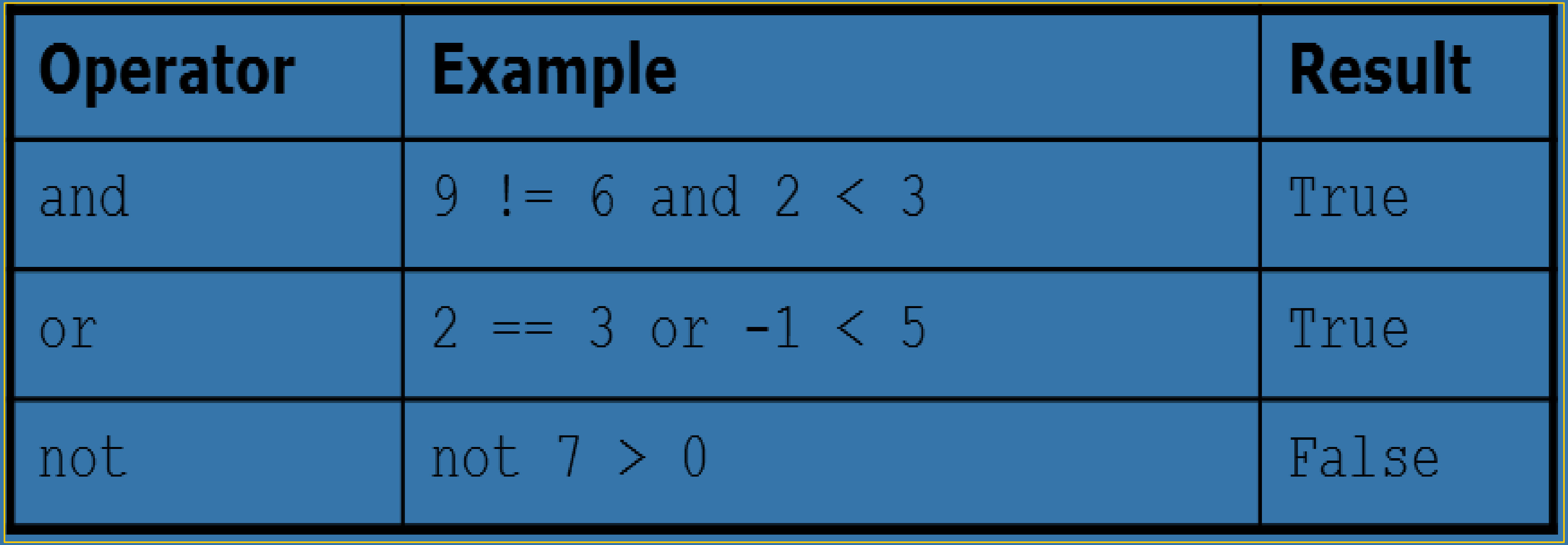
- Many logical expressions use *relational operators*:



Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Logical Operators

- These operators return true or false

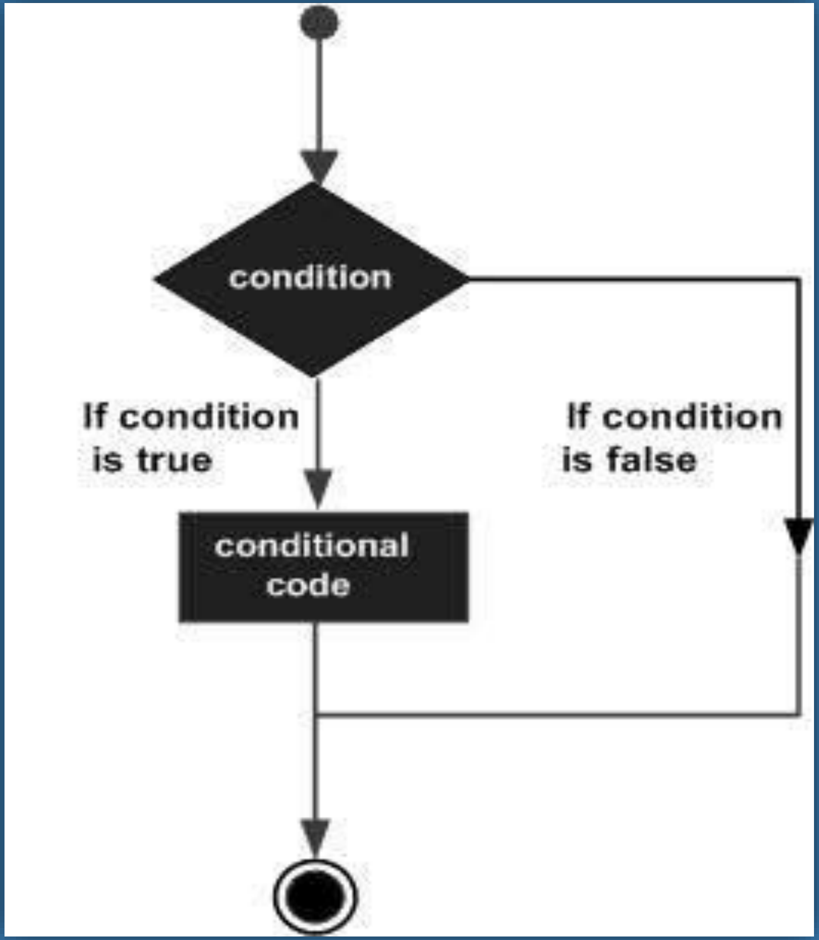


Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

Indentation

- **Python** provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the **indentation** is **variable**, but all statements within the block must be indented the same amount.

Decision Making



The if Statement

- Syntax:

```
if <condition>:
      <statements>


x = 5
if x > 4:
      print(“x is greater than 4”)
print(“This is not in the scope of the if”)
```

The if Statement

- The **colon** is required for the **`if statement`**
- Note that all statement indented one level in from the **`if`** are within it scope:

```
x = 5
if x > 4:
        print(“x is greater than 4”)
        print(“This is also in the scope of the if”)
```

The if/else Statement

```
if <condition>:
        <statements>
else:
        <statements>
```

- Note the **colon** following the **else**
- This works exactly the way you would expect

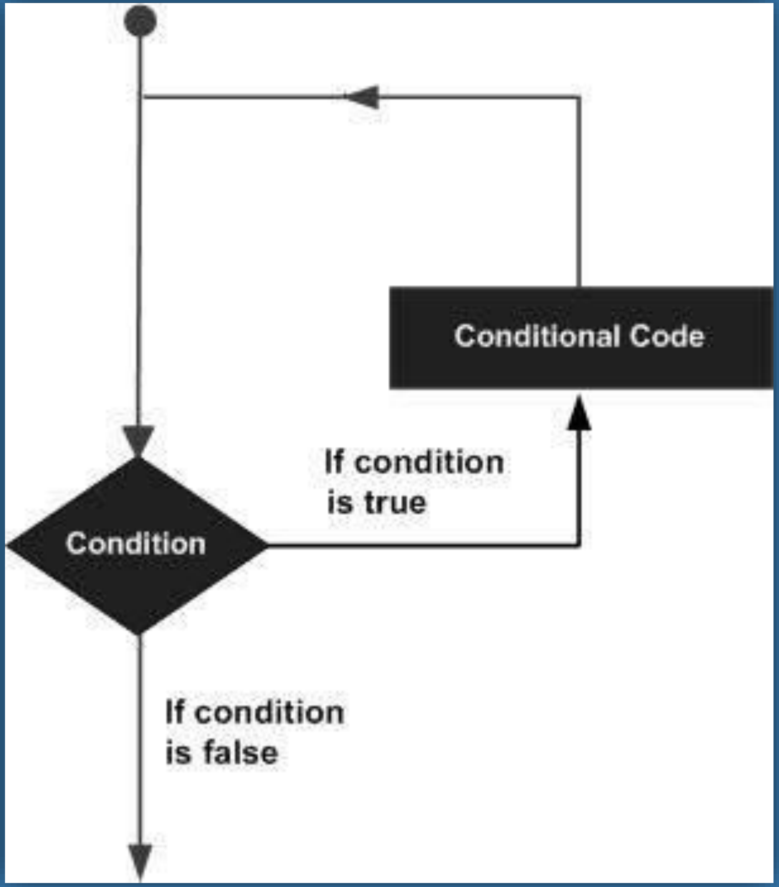
Exercise

- Write a **Python** program to find whether a given number (accept from the user) is **even or odd**, print out an appropriate message to the user.

Exercise

- Write a **Python** program to **sum** of three given integers. However, if two values are **equal** sum will be **zero**.

Loops


Conditional Code
If condition is true
Condition
If condition is false

The for Loop

- This is similar to what you're used to from C or Java, but not the same Syntax:

```
for variableName in groupOfValues:
    <statements>
```

- **Variable Name** gives a name to each value, so you can refer to it in the statements.
- **Group Of Values** can be a range of integers, specified with the range function.

- Example:

```
for x in range(1, 6):
    print (x, "squared is", x * x)
```

Range function

- The **range** function specifies a range of integers:

`range(start, stop)` - the integers between start (**inclusive**)
and stop (**exclusive**)

- It can also accept a **third value** specifying the change between values.

`range(start, stop, step)` - the integers between start (**inclusive**)
and stop (**exclusive**) by step

The `while` Loop

- Executes a group of statements as long as a condition is True.
- Good for indefinite loops (repeat an unknown number of times)
- Syntax:

```
while <condition>:
    <statements>
```

- Example:

```
number = 1
while number < 200:
    print (number)
    number = number * 2
```

Exercise

- Write a **Python** program to compute and display the first **16** numbers **powers of 2**, starting with **1**

Assignment 1

- Write a **Python** program to calculate the **sum** of three given numbers, if the values are **equal** then return **thrice** of their **sum**.

Assignment 2

- Write a **Python** program to print all **even** numbers from **1** to **n** using for loop. **Python** program to generate all even numbers between given range.

Day 3 ☺

Numbers & Strings

Numbers

- **Number** data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.
- **Number** objects are created when you assign a value to them. For example

```
var1 = 1
var2 = 10
```

Cons..

- **Python** supports four different numerical types –
- **int (signed integers)** – They are often called just integers or ints, are positive or negative whole numbers with no **decimal point**.
- **long (long integers)** – Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase **L**.
- **float (floating point real values)** – Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5×10^2 = 250).
- **complex (complex numbers)** – are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a, and the imaginary part is b. Complex numbers are not used much in Python programming.

Math Functions

- Use this at the top of your program: `from math import *`

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1, value2***`)`	larger of two values
`min(`***value1, value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

Strings

- **String**: A sequence of text characters in a program.
- **Strings** start and end with **quotation mark** " or **apostrophe** ‘ characters.
- Examples:

"hello"
‘This is a string’
“This, too, is a string.   It can be very long!”

- A string may not span across multiple lines or contain a " character.
  "This is not
  a legal String."

  "This is not a "legal" String either."

Strings

- A **string** can represent characters by preceding them with a backslash.
  - \t tab character
  - \n new line character
  - \" quotation mark character
  - \\ backslash character

- Example: "Hello\tthere\nHow are you?"

Indexing Strings

- As with other languages, you can use square brackets to index a string as if it were an array:

```
name = “John Dove”

print(name, “starts with “, name[0])
```

- Output

```
        John Dove starts with J
```

String Functions

- `len(`*`string`*`)` - number of characters in a string
- `str.lower(`*`string`*`)` - lowercase version of a string
- `str.upper(`*`string`*`)` - uppercase version of a string
- `str.isalpha(`*`string`*`)` - True if the string has only alpha chars
- Many others: split, replace, find, format, etc.

- Note the “dot” notation: These are **static methods.**

Exercise

- Write a **Python** program to compute **number** of characters in a given **string**.

Exercise

- Write a **Python** program to get a **string** and **n** (non-negative integer) **copies** of a given string.

Exercise

- Write a **Python** program to test whether a passed letter is a **vowel** or **not**.

- **[aeiou]**

Lists

- The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
- The **list** is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list **need not be of the same type**.

➢list1 = ['physics', 'chemistry', 1997, 2000]

➢list2 = [1, 2, 3, 4, 5]

Lists

- To access values in **lists**, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example
  - `list1 = ['physics', 'chemistry', 1997, 2000]`
  - `list2 = [1, 2, 3, 4, 5, 6, 7 ]`
  - `print ("list1[0]: ", list1[0])`
  - `print ("list2[1:5]: ", list2[1:5])`

Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the **append**() method. For example

- `list = ['physics', 'chemistry', 1997, 2000];`
- `print ("Value available at index 2 : “)`
- `print (list[2])`
- `list[2] = 2001;`
- `print "New value available at index 2 : "`
- `print (list[2])`

Lists

- To **remove** a list element, you can use either the **del** statement if you know exactly which element(s) you are deleting or the remove() method if you do not know. For example

- `list1 = ['physics', 'chemistry', 1997, 2000];`
- `print (list1)`
- `del (list1[2])`
- `print ("After deleting value at index 2 : “)`
- `print (list1)`

Basic Lists Operation

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

- L = ['spam', 'Spam', 'SPAM!’]

Python Expression	Results	Description
L[2]	SPAM!	Offsets start at zero
L[-2]	Spam	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Exercise

- Write a **Python** program to first print all elements in a given list of **integers** then sum of its element and print max value in this list.

Exercise

- Write a **Python** program to create a histogram from a given list of **integers**.

- **Ex. [6,3,7]**
- ******
- ***
- *******

Day 4 ☺

Tuples & Dictionary

Tuples

- A **tuple** is a sequence of immutable Python objects. **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.

- tup1 = ('physics', 'chemistry', 1997, 2000)
- tup2 = (1, 2, 3, 4, 5)
- tup3 = "a", "b", "c", "d"

Tuples

- A **tuple** is a sequence of immutable Python objects. **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.

- tup1 = ('physics', 'chemistry', 1997, 2000)
- tup2 = (1, 2, 3, 4, 5)
- tup3 = "a", "b", "c", "d"

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.

- 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]

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.

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.

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

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x	1 2 3	Iteration

Indexing, Slicing, and Matrixes

- L = ('spam', 'Spam', 'SPAM!')

Python Expression	Results	Description
L[2]	SPAM!	Offsets start at zero
L[-2]	Spam	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Dictionaries

- A **dictionary** is a data type similar to arrays, but works with **keys** and **values** instead of indexes. Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it. **Keys** are unique within a dictionary while **values** may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

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

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.

- 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.

- 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']

Properties of Dictionary Keys

- **Dictionary** values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.
- There are two important points to remember about dictionary keys :
  - (a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.
  - **(b)** Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed

Exercise

- Add "Jake" to the phonebook with the phone number 938273443, and remove Jill from the phonebook.

- phonebook = {"John" : 938477566, "Jack" : 938377264, "Jill" : 947662781}

Function

- A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Function

- Define a **function**:

```
def <function name>(<parameter list>)
```

- The function body is indented one level:

```
def computeSquare(x):
        return x * x
# Anything at this level is not part of the function
```

Exercise

- Write a python **program** that calculates power of number Ex- 6^5. Declare your own function to do this.

Assign

- Write a python **program** that simulate a simple calculator to calculate basic arithmetic operations.

python

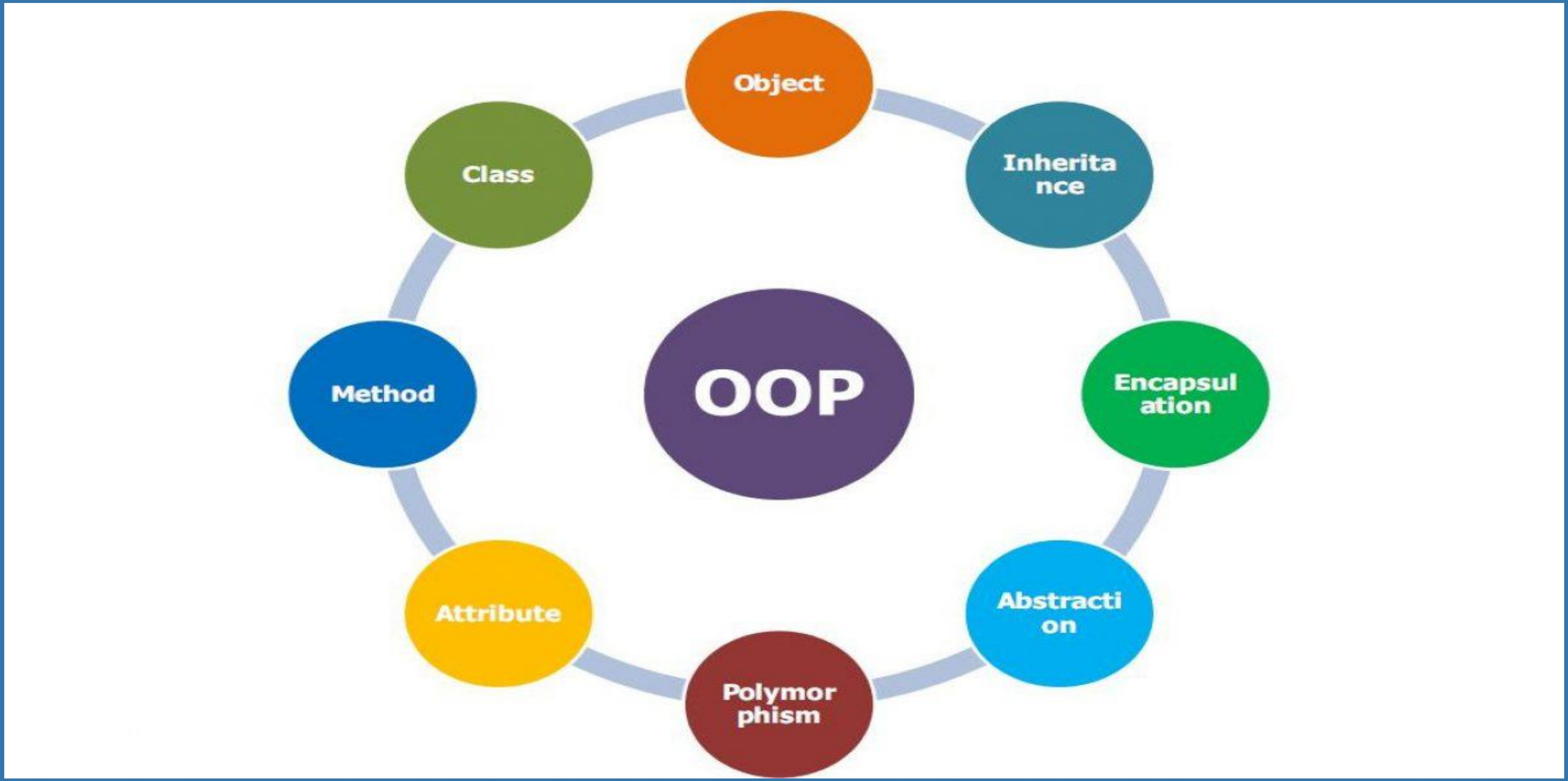
Day 5 ☺

Classes and Objects

Classes and Objects

- **Python** has been an **object-oriented** language since it existed. Because of this, creating and using **classes** and **objects** are downright easy

OOP Terminology



Creating Classes

- The **class** statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon

- class **ClassName**:

```
'Optional class documentation string'
class_suite
```

Class Example

- class MyClass:

  x = 5

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

__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

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

p1 = Person("John", 36)

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

The self Parameter

- The **self** parameter is a reference to the current instance of the class, and is used to access variables that **belongs** to the **class**.

- It does not have to be named **self** , you can call it whatever you like, but it has to be the **first parameter** of **any function** in the class

Class Methods

- 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()

Destroying Objects

- You have to define **destructor**
- **def __del__(self):**

print self.__class__.__name__, "destroyed"

Class Inheritance

- Instead of starting from scratch, you can create a class by **deriving** it from a **preexisting** class by listing the parent class in **parentheses** after the new class name.

- The child class **inherits** the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also **override data members and methods from the parent**

Class Inheritance

- class **SubClassName** (ParentClass1[, ParentClass2, ...]):

    'Optional class documentation string'

    class_suite

Multiple Inheritance

- class A: # define your class A
- .....

- class B: # define your class B
- .....

- class C(A, B):   # subclass of A and B
- .....

Data Hiding

- An object's attributes may or may not be visible outside the class definition. You need to name attributes with a **double underscore** prefix, and those attributes then are not be directly visible to outsiders.

Overriding Methods

- You can always **override** your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

- **class Parent**:   # define parent class
  ```
  def myMethod(self):
     print 'Calling parent method'
  ```

- **class Child**(Parent): # define child class
  ```
  def myMethod(self):
     print 'Calling child method'
  ```

Exercise

- Create a base class, **Telephone**, and derive a class **ElectronicPhone** from it. In Telephone, create a string member **phonetype**, and a function **Ring()** that outputs a text message like this: "Ringing the <phonetype>." In **ElectronicPhone**, the constructor should set the phonetype to "Digital.“ then call Ring() on the ElectronicPhone to test the inheritance

Day 5 ☺

Files I/O

Files I/O

- Python provides basic functions and methods necessary to manipulate **files** by default. You can do most of the file manipulation using a file object.

Opening and Closing Files

- Before you can read or write a file, you have to open it using Python's built-in **open()** function. This function creates a file object, which would be utilized to call other support methods associated with it.

- file object = open(file_name [, access_mode][, buffering])

Modes of opening a file

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing

Modes	Description
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing

The close() Method

- The **close**() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.
- Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the **close**() method to close a file.

- fileObject.close()

Reading and Writing Files

- The **file** object provides a set of access methods to make our lives easier. We would see how to use **read()** and **write**() methods to read and write files.

The write() Method

- The **write()** method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.
- The **write()** method does not add a newline character ('\n') to the end of the string.

- fo = open("foo.txt", "w")
- fo.write("Python is a great language.\nYeah its great!!\n")

The read() Method

- The **read()** method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

- f = open("demofile.txt", "r")
- print(f.read(5))

The read() Method

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

- f = open("demofile.txt", "r")
- print(f.readline())

- f = open("demofile.txt", "r")
- for x in f:
-   print(x)

File Remove

- import os
- os.remove("demofile.txt")

File Positions

- The **tell()** method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.
- The **seek**(offset[, from]) method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.
- If **from** is set to **0**, it means use the **beginning** of the file as the reference position and **1** means use the **current** position as the reference position and if it is set to **2** then the **end** of the file would be taken as the reference position.

Exercise

- Write a python **program** that computes the size of file called “student.txt” which has the following statements.
- My name is ali
- I am 20 years old

Useful Links

- https://docs.python.org/3/
- https://www.udacity.com/course/introduction-to-python--ud1110
- https://www.geeksforgeeks.org/python-programming-language/

Thanks ☺