Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

Introduction to Python Programming

Introduction to Python Programming

a. History of Python
b. Python Basics – variables, identifier, indentation
c. Data Structures in Python (list , string, sets, tuples, dictionary)
d. Statements in Python (conditional, iterative, jump)
e. OOPS concepts
f. Exception Handling
g. Regular Expression

Overview of Scripting Language

A scripting language is a “wrapper” language that integrates OS functions.

The interpreter is a layer of software logic between your code and the computer hardware on your machine.

Wiki Says:	"Scripts" are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted from source code or bytecode.

The “program” has an executable form that the computer can use directly to execute the instructions.
The same program in its human-readable source code form, from which executable programs are derived *(e.g., compiled)*

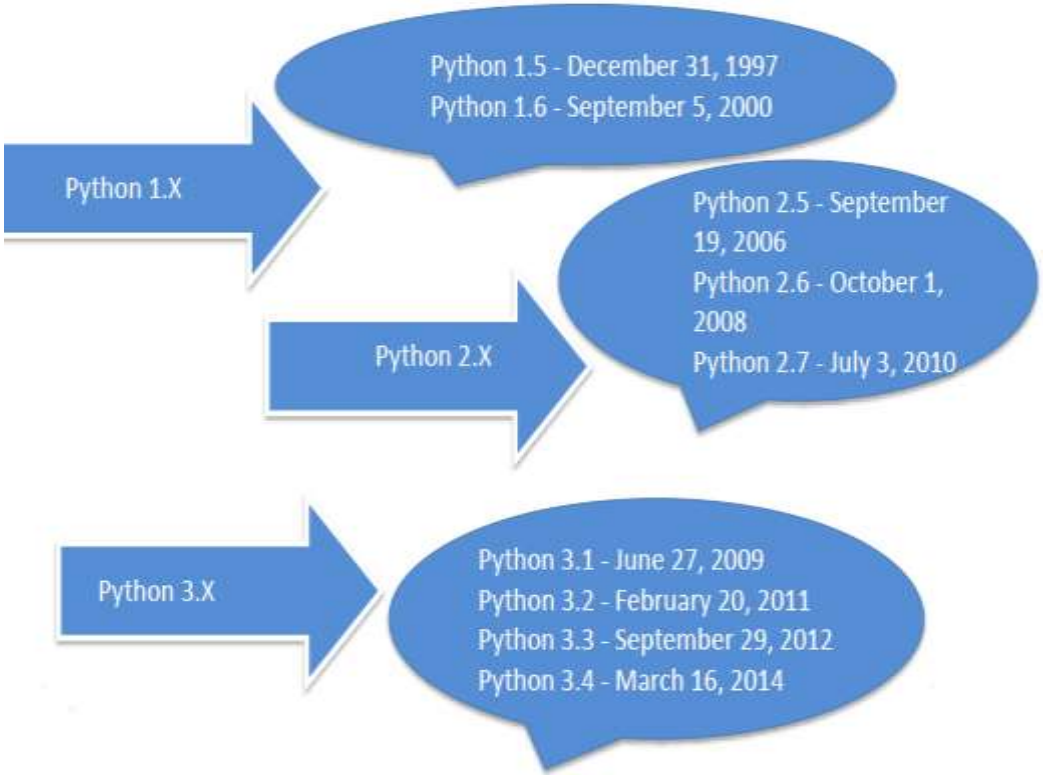
Python is scripting language, fast and dynamic. It is called so because of it’s scalable interpreter, but actually it is much more than that.

History of Python

Python is an interpreted high-level programming language for general-purpose programming Created by Guido van Rossum and first released in 1991.

Timeline

- Python was conceived in the late 1980s and its implementation began in December 1989 by Guido at Centrum Wiskunde & Informatica (CWI) in Netherlands.



Launch of different versions

2.0	Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode.
3.0	Python 3.0 (py3k) was released on 3 December 2008 after a long testing period.

Python is inherited from ABC programming and has many features including Interactive programming, object oriented, exceptional handling and many more.

Major differences between Python 2 and Python 3

- ✓ Changing `print` so that it is a built-in function, not a statement.
- ✓ Moving **reduce** (but not **map** or **filter**) out of the built-in namespace and into functools
- ✓ Adding support for optional function annotations that can be used for informal type declarations or other purposes.
- ✓ A change in integer division functionality. (In Python 2, **5 / 2** is **2**. In Python 3, **5 / 2** is **2.5**, and **5 // 2** is **2**)
- ✓ Unifying the **str**/**unicode** types, representing text, and introducing a separate immutable **bytes** type; and a mostly corresponding mutable **bytearray** type, both of which represent arrays of bytes

Understanding Identifiers

A Python Identifier is a name used to identify a variable, function, class, module or other project. Following rules need to be kept in mind while declaring Identifiers:

- Identifiers starts with a letter capital A to Z or small a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Invalid Identifiers: special characters such as @, $ and %
- Should not start with number
- Should not be named as pre-defined reserved keywords
- Identifiers are unlimited in length. Case is significant.

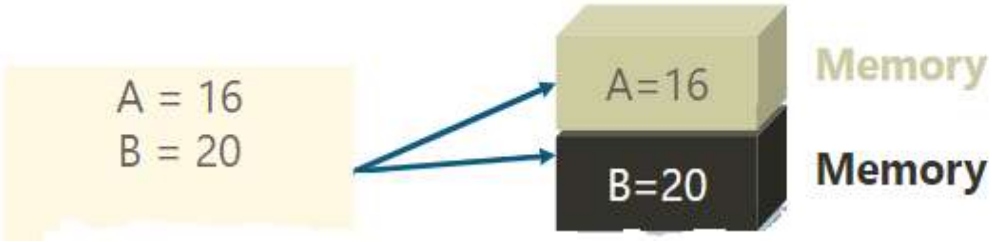
Identifier naming Convention

- Class names start with uppercase. All other identifiers involves lower case.
- Starting an identifier with 2 leading underscores indicates a strongly private identifier.
- A private identifier starts with a single leading Underscore

.

Variables are reserved memory locations to store values.

Whenever a variable is created a relevant space in memory is reserved for the created variable.



Assigning values to variables

```
A=10
B='Hello World'
Print(A,B)
```

Output
10 Hello World

Python Basics
Pre-defined reserved keywords

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

Python Basics Indenting Code

- ✓ Python blocks of code are defined by line indentation.
- ✓ The number of spaces in the indentation in block should be same with others statements within the Block
- ✓ Indenting starts a block and unindenting ends it.
- ✓ No explicit braces, brackets or keywords needed.
- ✓ Whitespace is significant.
- ✓ Code blocks are started by a ':'

Correct Example

```
N=10
If N:
   print "True"
else:
   print "False"
```

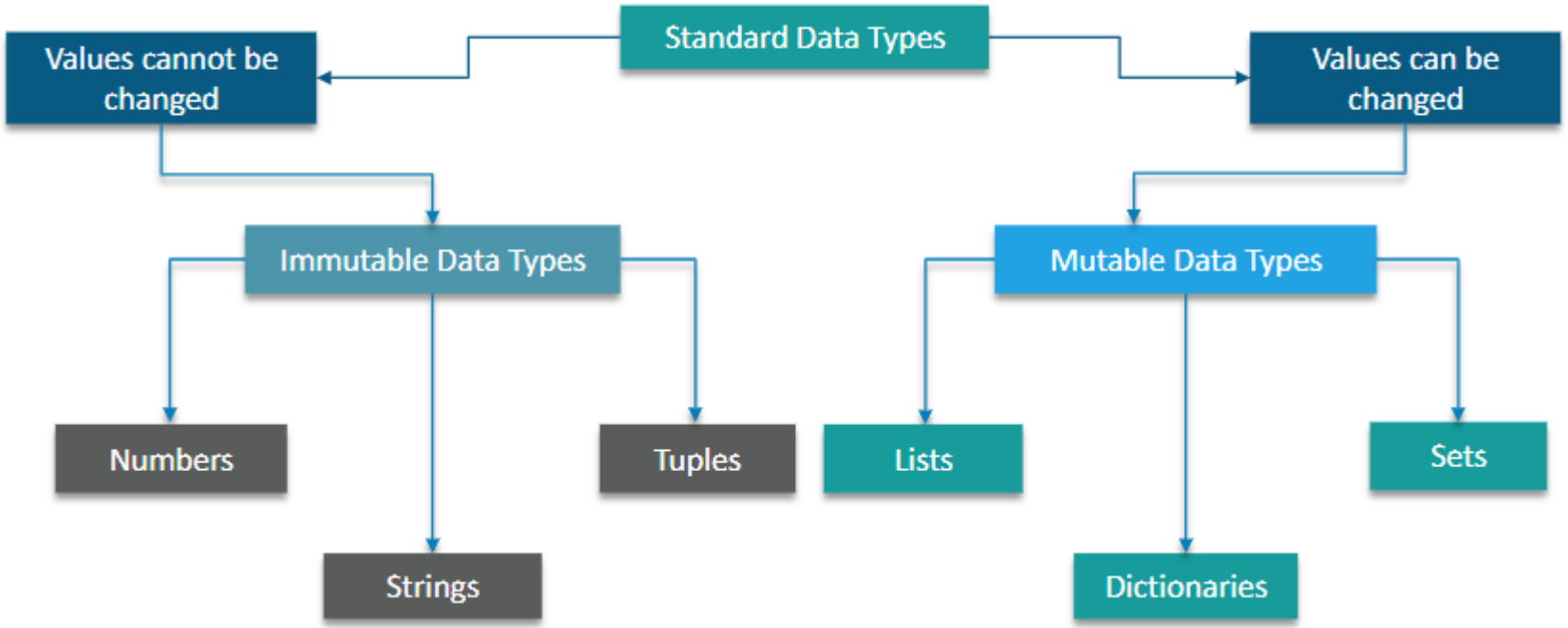
In-correct Example

```
N=10
If N:
   print "Answer"
          print "True"
else:
   print "Answer"
  print "False"
```

- ✓ Statements in Python end with a new line. However, Python allows the use of the line continuation character (\) to denote that the line should continue.
- ✓ Comments in Python is created with # sign for single line comment
- ✓ For multi line comment we will have to put three double quotes and insert comment in between. """ comment """
- ✓ Statements contained within the [], {} or () brackets do not need to use the line continuation character.

Python is dynamically and strongly typed which means that the data types in python are discovered at execution time and cannot be interchanged without conversion.

Python data types at a glance



Python supports 3 numerical value types:

- ✓ Int (Signed Integers)
- ✓ Float (Real Numbers)
- ✓ Complex Numbers

Numbers can be represented in following ways:

- ❑ Binary
- ❑ Octal
- ❑ Hexadecimal

```
A=10                    #(Integer)
B=9.75                  #(Float)
C=10 + 6j               #(Complex)

print(A,B)

Output
10 9.75 (10+6j)
```

A continuous set of characters within quotation (either single or double) is known as String. Python does not support a character type instead a single character is read as string of length one.

```
UserName='Sam Wilson'
Pwd='Jones123'

print(A)

print(B)

Output
Sam Wilson
Jones123
```

Tuples consists of a number of values which are separated by comma. The values are enclosed within parenthesis.
A tuple can have objects of different data types.

```
A=(4,2,3.14,'jones')

print(A)
```

Output
(4,2,3.14,'jones')

Lists is an ordered set of elements enclosed within square brackets. The main difference between Lists and Tuples are:

- Lists are Mutable whereas Tuples are Immutable.
- Tuples work faster than Lists.
- Lists are enclosed in brackets[] and Tuples are enclosed within parenthesis().

```
A=[4,2,3.14,'jones']

print(A)
```

Output
(4,2,3.14,'jones')

Dictionaries contain key value pairs. Each key is separated from its value by colon (:). Different Keys are separated by comma and all the values in a Dictionary are enclosed within curly braces.

```
A={'Name':'John', 'Marks': 79}

print(A)
```

Output
{'Name':'John', 'Marks': 79}

A set is an unordered collection of items with every item being unique.
A set is created by placing all the items or elements inside { } and each element is separated by comma.

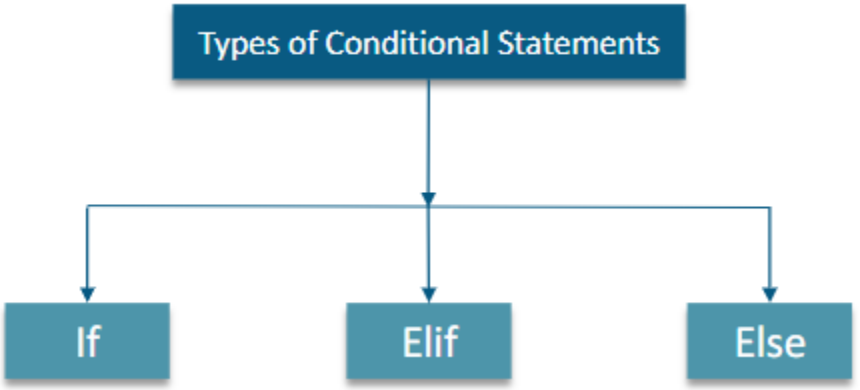
```
A={2,3,4,4}

print(A)
```

Output
{2,3,4}

Statements in Python – Conditional

Conditional Statements are used to execute a statement or a group of statements, when certain condition is true.



I and E small

The syntax and example for conditional statements are mentioned below:

Syntax:

```
if condition1:
   statements
elif condition2:
      statements
else:
      statements
```

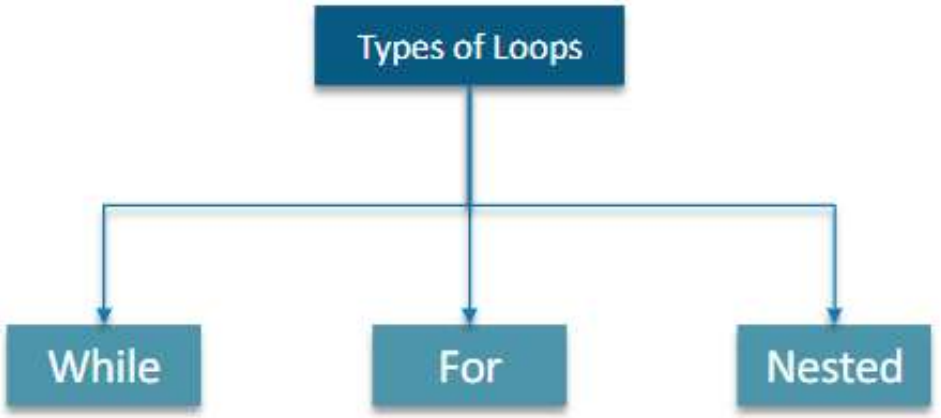
Sample code:

```
A=5
B=9

if (A<B):
    print ('A is less than B')
elif (A>B):
    print ('A is greater than B')
else:
    print ('A is equal to B')
```

Statements in Python – Loops

A loop statement enables the user to execute a statement or a group of statements, multiple times.



Statements in Python – While Loop

While loop is a conditional loop which will keep on iterating until certain conditions are met.
However there is uncertainty regarding how many times the loop will iterate.

Syntax:

```
while expression:
        statements
```

Sample code:

```
i=0
while (i<5):
        print (i)
        i=i+1
print ("End of loop")
```

Output

```
0
1
2
3
4
End of loop
```

Statements in Python – For Loop

For loop repeats a group of statements a specified number of times. The syntax of for loop involves following:

- Boolean Condition
- The initial value of the counting variable
- Increment of counting variable

Syntax:

```
For <variable> in <range>:
        statement1
        statement2
        .
        .
        Statement
```

Sample code:

```
name=['AK','RT','SK']
For count in range(len(name)):
        print(name[count])
```

Statements in Python – Nested Loop

Nested loop indicates a loop inside a loop, It could be a while loop inside a while loop, for loop inside a while loop, for loop inside a for loop and similar other cases.

Sample code:

```
index=1
for i in range(9):
      print(name(i)*i)
      for j in range(0,i):
           index=index+1
```

Output

```
1
22
333
4444
And so on till 8 is reached
```

Loop control statements have the ability to change the sequence of execution of statements. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Control Statement	Description
Break statement	Terminates the loop statement and transfers execution to statement immediately following the loop
Continue statement	Causes the loop to skip the remainder of its body and resets the condition prior to iterating
Pass statement	Pass statement is used when a statement is required syntactically but no command or code is needed to be executed.

An OOPS program is divided into parts called objects and follows a bottom up approach.
In OOP, objects can move and communicate with each other through member functions. Python follows OOP concepts.

Benefits of OOP concepts in Python:

- ❑ Data becomes active
- ❑ Code is reusable
- ❑ Ability to simulate real world events much more effectively
- ❑ Ability to code faster and accurate written applications

Python is an interpreted language which means it executes code line-by-line. So if the program has some error, that error will be shown when the line of code will execute.

Python Basics OOPS Concepts

Class and Objects

Class is a blueprint used to create objects having same property or attribute as its class.

An object is an instance of a class which contains variables and methods.

Relation between Classes and objects

- A class contains the code for all the object's methods.
- A class describes the abstract characteristics of a real-life thing.
- An instance is an object of a Class created at run-time.
- There can be multiple instances of a class.

Create a class

```
# Create a class
class number():
        pass

#Defining an instance of class
t=number()
print(t)
```

Variables and Attributes in a class

Variables which are declared within a class can be used within that class only and are referred to as 'Local' variables. While the ones which can be used within any class are called 'Global' variables.

Class attributes are shared by all the instances of the class. There are built-in as well as user defined attributes in python.

Built-in class attributes

Every Python class keeps following built-in attributes and they can be accessed using dot (.) operator like any other attribute:

- __dict__ : Dictionary containing the class's namespace.
- __doc__ : Class documentation string or None if undefined.
- __name__: Class name.
- __module__: Module name in which the class is defined.
  – This attribute is "__main__" in interactive mode.
- __bases__ : A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list

Python Basics
OOPS Concepts – Types of Attributes

Private Attributes	Can only be accessed inside of the class definition.
Public Attributes	Can and should be freely used.
Protected Attributes	Are accessible only from within the class and its subclass and main for debugging purpose.

More about Attributes

Attributes may be read-only or writable. In the latter case, assignment to attributes is possible. Module attributes are writable: you can write **modname.the_answer = 42**.
Writable attributes may also be deleted with the del statement.
For example, **del modname.the_answer** will remove the attribute **the_answer** from the object named by **modname**.

Python Basics
OOPS Concepts – Private Methods

When the attributes of an object can only be accessed inside the class, it is called a Private Class.
Python uses two underscores to hide a method or hide a variable.

Sample code

```
Class sample:
        def apublicmethod(self):
              print('public method')
        def __aprivatemethod(self):
               print('private method')
Obj=sample()
Obj.apublicmethod()
Obj._sample__aprivatemethod()
```

Sample code

Declaring class variable

```
Class sample:
    domain = ("Analytics")
    def Setcourse (self,name):
        self.name=name
```

Instance of class

```
Ob1=sample()
Ob2=sample()
```

Class variable is shared by both instances Ob1 and Ob2

```
Ob1.Setcourse ("Module1")
Ob2.Setcourse ("Module2")

print (Ob1.domain)
Ob1.domain='AI'
print (Ob1.domain)
print (Ob2.domain)
```

Output

```
Analytics
AI
Analytics
```

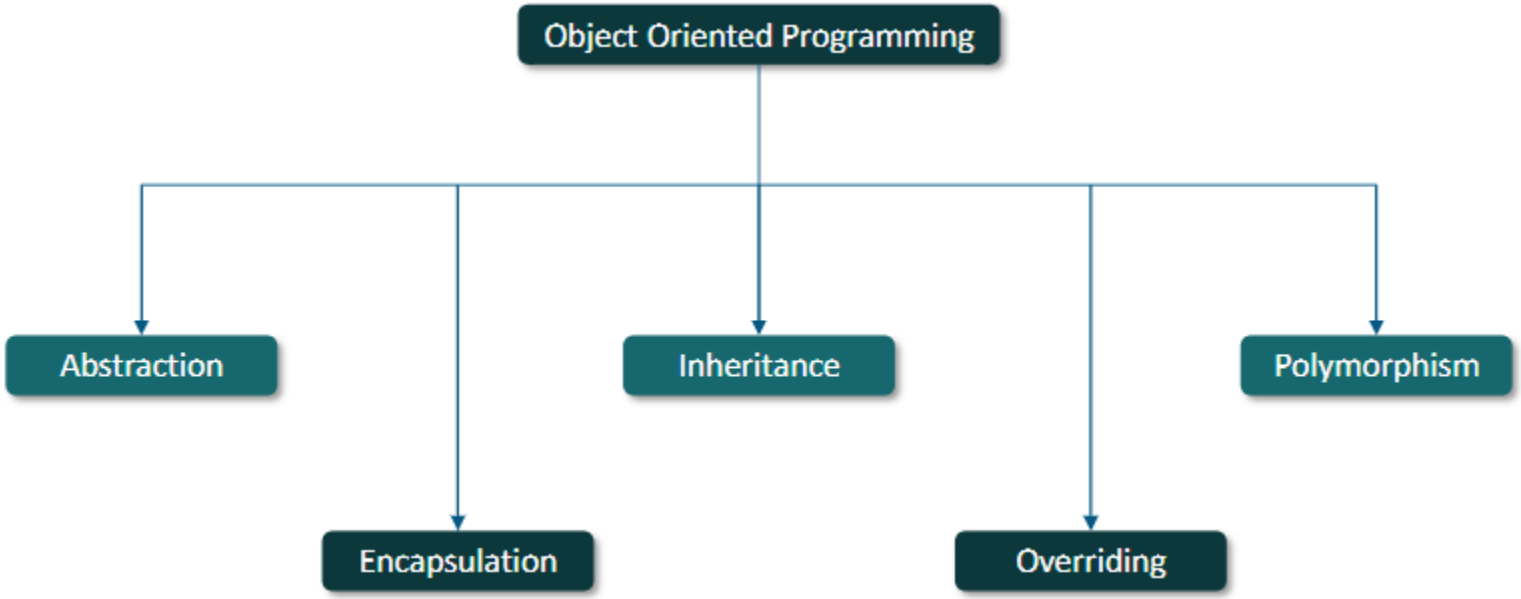
Sample code

```
Class sample:
        def __init__ (self):
            print('constructor')
        def __del__ (self):
            print('destructor')
If __name__ == "__main__":
      obj = TestClass ()
       del obj
```

Output

```
Constructor
Destructor
# obj is created and manually deleted after displaying both the messages.
```

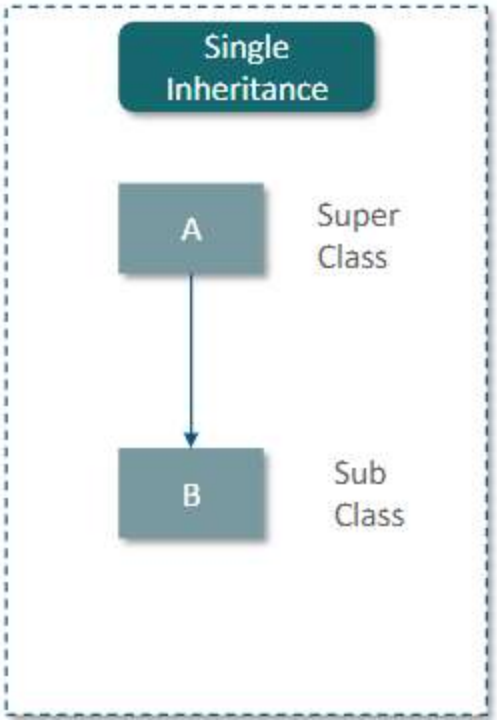
Python Basics
OOPS Concepts – Key concepts

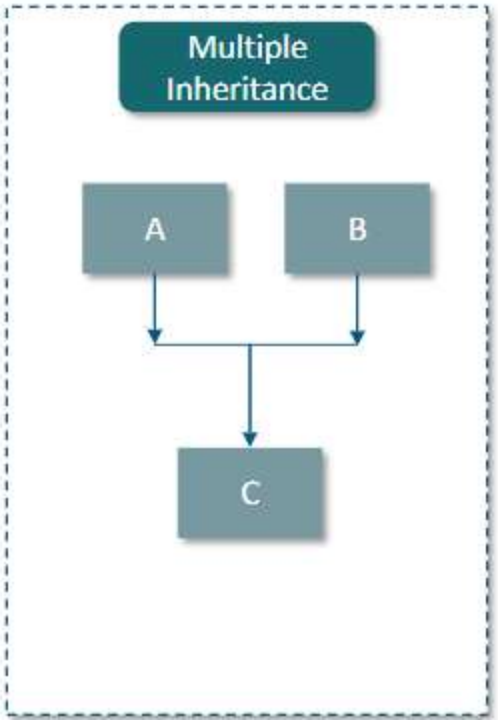


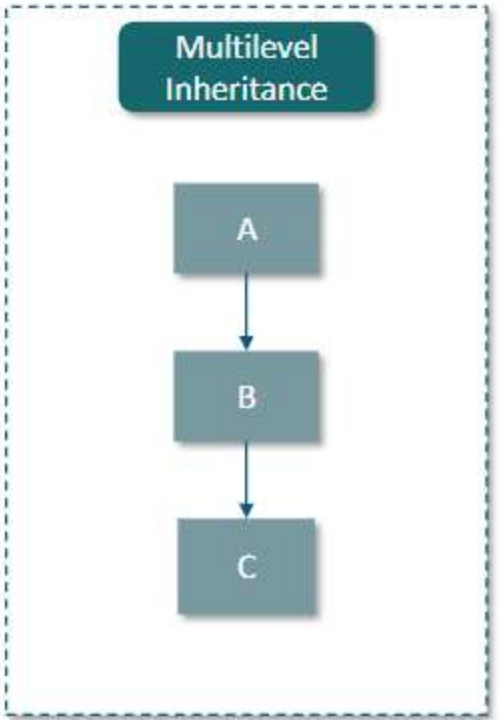
OOPS Concepts – Key Concepts

Abstraction	❑ Abstraction means simplifying complex reality by modeling classes relevant to the problem. ❑ Class abstraction is defined by separating class implementation from the use of the class.
Encapsulation	❑ Encapsulation is a mechanism for restricting access to some of an object's components. This means that the internal components of an object can not be seen from outside of object's definition.
Inheritance	❑ Refers to deriving a class from the base class with little or no modification in it.

Python Basics
OOPS Concepts – types of inheritance

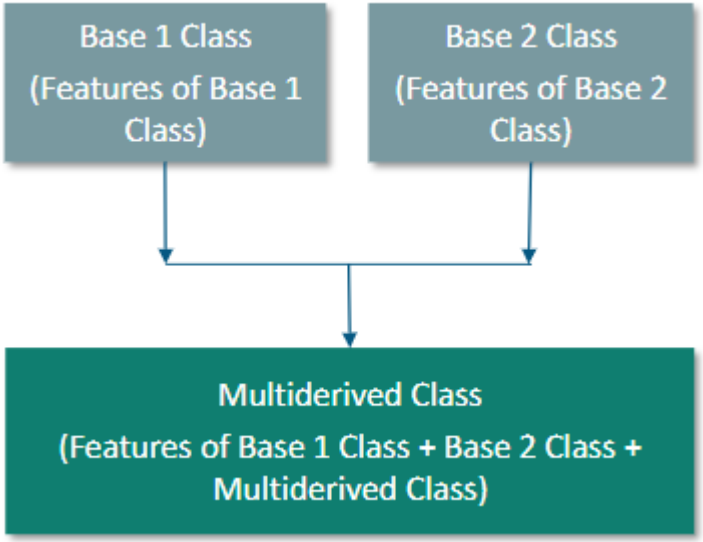




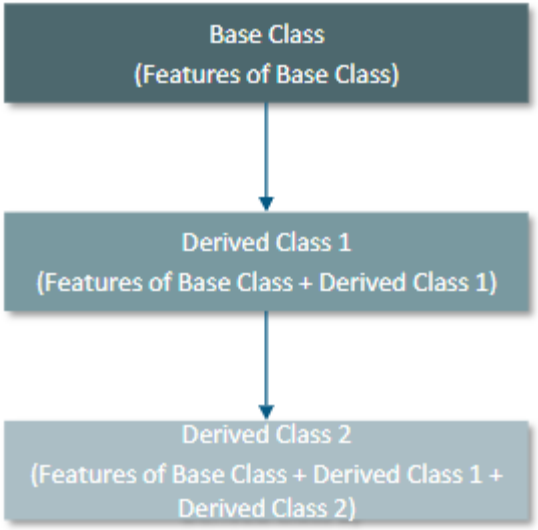


Python Basics
OOPS Concepts – types of inheritance

Multiple Inheritance



Multilevel Inheritance



Python Basics
OOPS Concepts – Overriding Method

Overriding is a very important part of OOP since it makes inheritance utilize its full power. By using method overriding a class may "copy" another class, avoiding duplicated code, and at the same time enhance or customize part of it. Method overriding is thus a part of the inheritance mechanism.

```
class Parent(object):
    def __init__(self):
        self.value = 4
    def get_value(self):
        return self.value
 class Child(Parent):
    def get_value(self):
        return self.value + 1
c = Child()
c.get_value()
5
```

Python Basics
OOPS Concepts – Polymorphism

Sometimes an object comes in many types or forms. If we have a button, there are many different draw outputs (round button, check button, square button, button with image) but they do share the same logic: onClick(). We access them using the same method . This idea is called *Polymorphism*.

```
class Bear(object):
    def sound(self):
        print "Groarrr"
 class Dog(object):
    def sound(self):
        print "Woof woof!"
def makeSound(animalType):
    animalType.sound()
bearObj = Bear()
dogObj = Dog()
makeSound(bearObj)
makeSound(dogObj)
```

Output

```
Groarrr
Woof woof!
```

OOPS Concepts – Getter and Setter methods

Getters and setters are used in many object oriented programming languages to ensure the principle of data encapsulation. They are known as mutator methods as well.

```
class P:
    def __init__(self,x):
        self.__x = x
    def get_x(self):
        return self.__x
    def set_x(self, x):
        self.__x = x
```

```
>>> from mutators import P
>>> p1 = P(42)
>>> p2 = P(4711)
>>> p1.get_x()
42
>>> p1.set_x(47)
>>> p1.set_x(p1.get_x()+p2.get_x())
>>> p1.get_x()
4758
```

Python has many built-in exceptions which forces a program to output an error when something in it goes wrong.

When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, the program will crash.
For example, if function A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.

If never handled, an error message is spit out and program comes to a sudden, unexpected halt.

Python Basics
OOPS Concepts – Catching Exceptions

In Python, exceptions can be handled using a try statement.
A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause. **(add one more slide on assert)**

```
# import module sys to get the type of exception
import sys
randomList = ['a', 0, 2]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print("The reciprocal of",entry,"is",r)
```

Output

```
The entry is a
Oops! <class 'ValueError'> occured.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError' > occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

Explaining code on previous slide

In this program, we loop until the user enters an integer that has a valid reciprocal. The portion that can cause exception is placed inside try block.

If no exception occurs, except block is skipped and normal flow continues. But if any exception occurs, it is caught by the except block.

Here, we print the name of the exception using ex_info() function inside sys module and ask the user to try again. We can see that the values 'a' and '1.3' causes ValueError and '0' causes ZeroDivisionError.

Python Basics
OOPS Concepts – Raising Exceptions

In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.

```
>>> raise KeyboardInterrupt
Traceback (most recent call last):
...
KeyboardInterrupt
>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument
>>> try:
...     a = int(input("Enter a positive integer: "))
...     if a <= 0:
...         raise ValueError("That is not a positive number!")
... except ValueError as ve:
...     print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```

The try statement in Python can have an optional **finally** clause. This clause is executed no matter what, and is generally used to release external resources.

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

finally:
  f.close()
```

Python Basics
OOPS Concepts – Regular Expressions

A Regular Expression is a special text string for describing a search pattern. The module **re** provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

```
re.match(pattern, string, flags=0)      # re.match function returns a match object on success, None on failure.
```

pattern
This is the regular expression to be matched.

string
This is the string, which would be searched to match the pattern at the beginning of string.

flags
You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

Sample Code

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
   print "matchObj.group() : ", matchObj.group()
   print "matchObj.group(1) : ", matchObj.group(1)
   print "matchObj.group(2) : ", matchObj.group(2)
else:
   print "No match!!"
```

Output

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

The search function:

```
re.search(pattern, string, flags=0)
```

Sample Code

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
if searchObj:
   print "searchObj.group() : ", searchObj.group()
   print "searchObj.group(1) : ", searchObj.group(1)
   print "searchObj.group(2) : ", searchObj.group(2)
else:
   print "Nothing found!!"
```

Output

```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```


Cognixia
A Collabera LEARNING SOLUTIONS COMPANY
THANK YOU
45%
www.cognixia.com