Module 1: Object Oriented Programming using Python
Copyright Guideline

© 2017 Infosys Limited, Bangalore, India. All Rights Reserved.

Infosys believes the information in this document is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. Except as expressly permitted, neither this documentation nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior permission of Infosys Limited and/ or any named intellectual property rights holders under this document.
Confidential Information

- This Document is confidential to Infosys Limited. This document contains information and data that Infosys considers confidential and proprietary ("Confidential Information").

- Confidential Information includes, but is not limited to, the following:
  - Corporate and Infrastructure information about Infosys
  - Infosys’ project management and quality processes
  - Project experiences provided included as illustrative case studies

- Any disclosure of Confidential Information to, or use of it by a third party, will be damaging to Infosys.

- Ownership of all Infosys Confidential Information, no matter in what media it resides, remains with Infosys.

- Confidential information in this document shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.

- This document also contains third party confidential and proprietary information. Such third party information has been included by Infosys after receiving due written permissions and authorizations from the party/ies. Such third party confidential and proprietary information shall not be disclosed, duplicated or used – in whole or in part – for any purpose other than reading without specific written permission of an authorized representative of Infosys.
Course Information

Course Code: FP4.1 - M1
Course Name: Object Oriented Programming using Python
Document Number: 003
Version Number: 1.0
Learning Objectives

On completion of this module, the learner should be able to:

1) Understand Problem Solving skills
2) Choose most appropriate Collections
3) Learn basic vocabulary of Python Programming
4) Apply Object oriented thought process for real life problems
5) Implement OO features using Python
6) Build complex and large software systems using Object oriented programming
Topics covered in “OO Programming using Python” course

• Introduction

• Python Basics

• Python Data Variables & Operators
  – Data Variables and its types
  – Operators
  – id() and type() functions
  – Coding Standards

• Control Structures
  – If else
  – elif, Nested if
  – Iteration Control Structures
  – Break, Continue & Pass
Topics covered in “OO Programming using Python” course

• Collections
  – Strings
  – Tuples
  – Lists
  – Sets
  – Dictionary

• Functions
  – Defining & Calling a function
  – Passing Arguments to functions – Mutable & Immutable Data types
  – Different types of arguments
  – Scope of variables
  – Introduction to Python Tutor
Topics covered in “OO Programming using Python” course

• Standard Library
  – Math Module
  – String Module
  – List Module

• Modules and Package
  – Modules
  – Packages

• Exception Handling
  – Try….except
  – Try….finally

• Introduction to Regular Expressions
Topics covered in “OO Programming using Python” course

• Basic OOP concepts
• Creating classes and objects
• Class variables and Object Variables
• Method Invocation
• Using default arguments in Methods
• Static, Class and Instance Methods
• Relationships
  – Inheritance
  – Aggregation
  – Association
Introduction to Python
Why Python for beginners?

• Easy – to - learn
  – Code is 3-5 times shorter than Java
  – 5-10 times shorter than C++

• Stepping Stone to Programming universe
  – Python’s methodologies can be used in a broad range of applications

• Bridging the Gap between abstract computing and real world applications
  – Python is used as main programming language to do projects using Raspberry Pi

• Rising Demand for Python Programmers
  – Google, Nokia, Disney, Yahoo, IBM use Python

• Open- Source, Object – Oriented, procedural and functional
  – Not only a Scripting language, also supports Web Development and Database Connectivity
Evolution of Python

• **Guido Van Rossum** developed Python in early 1990s at National Research Institute for Mathematics and Computer Science, Netherlands.

• Named after a circus show Monty Python show.

• Derives its features from many languages like **Java, C++, ABC, C, Modula-3, Smalltalk, Algol-68, Unix shell** and other scripting languages.

• Available under the GNU General Public License (GPL) – Free and open-source software

• **Python v3.0** – 2008 being the latest version
Python Features

• Python is a High-Level, Interpreted, Interactive and Object-Oriented Programming Language

• Features include:
  – Beginners Language
  – Extensive Standard Library
  – Cross Platform Compatibility
  – Interactive Mode
  – Portable and Extendable
  – Databases and GUI Programming
  – Scalable and Dynamic Semantics
  – Automatic Garbage Collection
Configuration

• Download and Install Python 3.5: [https://www.python.org/downloads/](https://www.python.org/downloads/)
• Download PyDev_3.8.0 or higher version: [http://www.pydev.org/download.html](http://www.pydev.org/download.html)
• Download Eclipse Juno or higher version: [https://www.eclipse.org/downloads/index.php](https://www.eclipse.org/downloads/index.php)
• Install Python on your machine

**Note: You need eclipse locally installed in your machine**

**Procedure:**

• Copy paste the contents of the plugin folder of PyDev into plugin folder of Eclipse
• In Eclipse open PyDev perspective (Window -> Open Perspective -> Other -> Pydev)
• Create a PyDev Project
• Select Grammar as 3.0
• Configure interpreter by choosing the .exe file of Python installed in your machine
Commenting Style in Python!

Types of Comments:

• A single line comment starts with hash symbol ‘#’ and end as the line ends.
  – These lines are never executed and are ignored by the interpreter.
    • Single → # This is a single line comment

• Multi-line comments starts and ends with triple single quotes ‘’’ or triple “”” double quotes
  – Used for documentation
    • Triple → ‘’’ or “””

"""
Contents here can be used for documentation
"""

""
An example for multi-line comments with single quotes
""
Multiline statements

• Python statements always end up with a new line, but it also allows multiline statement using “\” character at the end of line as shown below:

```python
result = (8+5)*\ 
2+\ 
9/5
```

• Statements which have (), [], {} brackets and comma, do not need any multiline character to go to next line.

```python
customer_details = [101, 'kevin', 
'165', 498.24]
```
Print Statement

- Displays the output on the screen of user

```python
PI = 3.1417
print("The value of PI is: ", PI)
Output: The value of PI is : 3.1417
```

User Input in Python

```python
name = input("Enter your name")
print("Welcome to session on Programming in Python," , name)
num1 = int(input("Enter a Number"))
num2 = int(input("Enter a Number"))
total = num1 + num2
print("The value of Total is:", total)
```

Guided Activity: Assignment 1: Using Eclipse IDE to create and execute Python Program

Input statement will accept the data as String by default. Hence we need to typecast it to int.
Python Data Variables
Programming Constructs in Python

- Given a real world problem, to solve the problem using a program, we need:
  - Logic
  - High level programming language
    - Programming Fundamentals
      - Identifiers
      - Variables
      - Data types
      - Operators etc
Identifiers

- Are names given to anything that you want to identify in a program
- Helps to refer to that item from any place in the program
- Can start with an underscore (_) or a upper or lower case alphabet
- Can have digits
- Identifiers cannot match any of Python's reserved words
- Are case-sensitive

bill_id
customer_id
bill_amount

Identify the identifiers needed to solve the problem discussed as part of previous guided activity?
Variables

- An identifier for the data and it holds data in your program
- Is a location (or set of locations) in memory where a value can be stored
- A quantity that can change during program execution
- No declaration of variables

- Data type of a variable can change during program execution compared to other strongly typed languages such as Java, C++, C

```python
customer_id = 101  # Integer
customer_name = "John"  # String
bill_amount = 675.45  # Floating-point
x = 5.3 + 0.9j  # complex number
print(customer_id, customer_name, bill_amount)
print(x.real)  #prints 101 John 675.45
print(x.imag + 3)  #prints 5.3
```
# Data Types in Python

<table>
<thead>
<tr>
<th>Category</th>
<th>Data Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Type</td>
<td>int</td>
<td>675</td>
</tr>
<tr>
<td></td>
<td>complex</td>
<td>2 + 5i</td>
</tr>
<tr>
<td>Floating Type</td>
<td>float</td>
<td>642.43</td>
</tr>
<tr>
<td>Textual</td>
<td>String</td>
<td>Infosys</td>
</tr>
<tr>
<td>Logical</td>
<td>boolean</td>
<td>True, False</td>
</tr>
</tbody>
</table>

We will now understand different types of operators and how to write a simple program.

Guided Activity: Programming constructs in Python - Assignment 3, 4
Operators (1 of 6)

• Used to perform specific operations on one or more operands (or variables) and provide a result

- Arithmetic Operators
- Relational Operators
- Assignment Operators
- Logical Operators
- Membership Operators
- Identity Operators
Operators (2 of 6)

- Arithmetic Operators
  - Used for performing arithmetic operations

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Additive operator (also used for String concatenation)</td>
<td>2 + 3 = 5</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction operator</td>
<td>5 – 3 = 2</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication operator</td>
<td>5 * 3 = 15</td>
</tr>
<tr>
<td>/</td>
<td>Division operator</td>
<td>6 / 2 = 3.0</td>
</tr>
<tr>
<td>%</td>
<td>Modulus operator</td>
<td>7 % 2 = 1</td>
</tr>
<tr>
<td>//</td>
<td>Truncation division (also known as floor division)</td>
<td>10 // 3 = 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.0 // 3 = 3.0</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
<td>10 ** 3 = 1000</td>
</tr>
</tbody>
</table>
Operators (3 of 6)

- Relational Operators
  - Also known as **Comparison operators**
  - Used in conditional statements to compare values and take action depending on the result

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Lesser than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Similar to Not equal to</td>
</tr>
</tbody>
</table>
Operators (4 of 6)

• Assignment Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Assignment from right side operand to left side</td>
<td>c = 50; c = a;</td>
<td></td>
</tr>
<tr>
<td>+=</td>
<td>Add &amp; assigns result to left operand</td>
<td>c += a</td>
<td>c = c + a</td>
</tr>
<tr>
<td>-=</td>
<td>Subtract &amp; assigns result to left operand</td>
<td>c -= a</td>
<td>c = c − a</td>
</tr>
<tr>
<td>*=</td>
<td>Multiply &amp; assigns result to left operand</td>
<td>c *= a</td>
<td>c = c * a</td>
</tr>
<tr>
<td>/=</td>
<td>Divide &amp; assigns result to left operand</td>
<td>c /= a</td>
<td>c = c / a</td>
</tr>
<tr>
<td>%=</td>
<td>Calculates remainder &amp; assigns result to left operand</td>
<td>c %= a</td>
<td>c = c % a</td>
</tr>
<tr>
<td>//=</td>
<td>Performs floor division &amp; assigns result to left operand</td>
<td>c //= a</td>
<td>c = c // a</td>
</tr>
<tr>
<td>**=</td>
<td>Performs exponential calculation &amp; assigns result to left operand</td>
<td>c **= a</td>
<td>c = c ** a</td>
</tr>
</tbody>
</table>

• Multiple Assignments – Same value can be assigned to more than one variable

**Ex.1:** Students Ram, Sham, John belong to semester 6
Ram = Sham = John = 6

**Ex.2:** a, b, c = 10, 20, 30 is same as a = 10, b = 20, c = 30
Operators (5 of 6)

• Logical Operators
  – Are based on Boolean Algebra
  – Returns result as either True or False

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>Short Circuit-AND</td>
</tr>
<tr>
<td>or</td>
<td>Short Circuit-OR</td>
</tr>
<tr>
<td>not</td>
<td>Unary NOT</td>
</tr>
</tbody>
</table>

Demo: Assignment 5: Programming constructs in Python

Guided Activity: Assignment 6, 7, 8: Programming constructs in Python
Operators (6 of 6)

- Membership Operators
  - Checks for membership in a sequence of Strings, Lists, Dictionaries or Tuples

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Returns to true if it finds a variable in given sequence else false</td>
</tr>
<tr>
<td>not in</td>
<td>Returns to true if it does not find a variable in given sequence else false</td>
</tr>
</tbody>
</table>

- Identity Operators
  - Are used to compare memory locations of 2 objects

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>Returns to true if variables on either side of operator are referring to same object else false</td>
</tr>
<tr>
<td>is not</td>
<td>Returns to false if variables on either side of operator are referring to same object else true</td>
</tr>
</tbody>
</table>
Built-in function: id()

- **Id(object)**
  - Returns identity of an object. It is the address of object in memory.
  - It will be unique and constant throughout the lifetime of an object.

*Example:*

```python
a = 10
b = a
print("Value of a and b before increment")
print("id of a: ", id(a))
print("id of b: ", id(b))
b = a + 1
print("Value of a and b after increment")
print("id of a: ", id(a))
print("id of b: ", id(b))
```

*Output*

```
Value of a and b before increment
id of a:  1815592664
id of b:  1815592664
Value of a and b after increment
id of a:  1815592664
id of b:  1815592680
```

Note the change in address of variable ‘b’ after increment.
**Built-in function: type()**

- Used to identify the type of object

**Example:**

```python
int_a = 10
print("Type of 'int_a':", type(int_a))

str_b = "Hello"
print("Type of 'str_b':", type(str_b))

list_c = []
print("Type of 'list_c':", type(list_c))
```

**Output:**

```
Type of 'int_a': <class 'int'>
Type of 'str_b': <class 'str'>
Type of 'list_c': <class 'list'>
```

**Note:** Every variable in Python is a object

Guided Activity: Assignment 9: id() and type() functions - Quiz
Coding Standards in Python

• Set of guidelines
  – To Enhance the readability and Clarity of the program
  – Make it easy to debug and maintain the program
• All the letters in a variable name should be in lowercase
• When there are more than two words in variable name, underscore can be used between internal words
• Use meaningful names for variables
• Limit all lines to a maximum of 79 characters
• A function and class should be separated by 2 blank lines
• Methods within classes should be separated by single blank line
• Always surround binary operators with a space on either side:
  Ex: a = a + 1;
## Coding Standards in Python

<table>
<thead>
<tr>
<th>Bad Code</th>
<th>Good Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 10</td>
<td>marks1 = 10</td>
</tr>
<tr>
<td>b = 23</td>
<td>marks2 = 23</td>
</tr>
<tr>
<td>C = 24</td>
<td>marks3 = 24</td>
</tr>
<tr>
<td>sum = a + b + c</td>
<td>marks = marks1 + marks2 + marks3</td>
</tr>
<tr>
<td>avg = sum/3</td>
<td>marks = sum_of_marks / 3</td>
</tr>
<tr>
<td>print(&quot;Average: &quot;, avg)</td>
<td>print(&quot;Average: &quot;, avg_of_marks)</td>
</tr>
</tbody>
</table>

**Usage of keyword ‘sum’ as variable name will lead to warning**

---

**Guided Activity: Assignment 10: Coding Standards**
Control Structures
Indentation in Python

- Python uses offside rule notation for coding
- Uses indentation for blocks, instead of curly brackets
- The delimiter followed in Python is a colon (:) and indented spaces or tabs.

**Example:**

```python
x = 3
if x > 5:
    print("true")
    print(x)
else:
    print ("false")
print ("Out of if block")
```

1st level indent ➔ if block
2nd level indent ➔ statements within if or else block
Control Structures

• Decision making statements
  – if statement:
    • if statement checks for a condition and if that is found true a particular set of instructions gets executed

  **Example:**
  
  ```python
  x = 8
  if x < 10:
      print("Value of x is \%d" \%x)
  var = 10
  if var > 5:
      print ("Hi")  # line belongs to if block
  print("I'm out of if")
  ```

  **Output:**
  
  Value of x is 8
  Hi
  I'm out of if

  **Syntax:**
  
  ```python
  if condition1:
      statement(s)
  else:
      statement(s)
  ```

  Predict the output of this code snippet when value of x = 15?
Control Structures…

• **elif statement:**
  
  – **elif** statement is used when there is more than one condition to be checked separately

  **Example:**

  ```python
  var=10
  if var > 10 :
      print("Hello")
      print(var)
  elif var < 10: 
      print("Hola in Spanish for Hello")
      print(var)
  else:
      print("Hi")
      print(var)
  print("End of Program")
  ```

  **Syntax:**

  ```python
  if condition1:
      statement(s)
  elif condition2:
      statement(s)
  else:
      statement(s)
  ```

  **Output:**

  Hi
  10
  End of Program

  **Note:** There is no switch case statement in Python unlike C/C++ language

Guided Activity: Assignment 13, 14, 15, 16: Control Structures
Iterative Statements

• Loop statements:
  – Allows us to execute a statement or group of statements multiple times.
    • While Loop
    • For Loop
    • Range

• Loop Control Statements:
  – Are used to change flow of execution from its normal sequence.
    • Break
    • Continue
    • Pass
Iterative Statements

– **while loop:**
  
  • Repeats a statement or group of statements while a given condition is TRUE.
  
  • Tests the condition before executing the loop body.

**Syntax:**

```
while condition:
    statement(s)
```

**Example:**

```
n = 5
result = 0
counter = 1
while counter <= n:
    result = result + counter
    counter += 1
print("Sum of 1 until %d: %d" % (n, result))
```

**Output:**

```
Sum of 1 until 5: 15
```
Iterative Statements

– for loop:
  • Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

**Syntax:**

```python
for iterating_var in sequence:
    statement(s)
```

**Example:**

```python
for counter in 1,2,'Sita', 7,'Ram',5:
    print(counter)
```

**Output:**

```
1
2
Sita
7
Ram
5
```
**Iterative Statements**

- **range function in loops**

  - Used in case the need is to iterate over a specific number of times within a given range in steps/intervals mentioned

  ```python
  for value in range(1, 6): print(value)
  ```

  Output: 
  
  ```
  1 2 3 4 5
  ```

  Remarks: Prints all the values in given range exclusive of upper limit

  ```python
  for value in range(0, 6, 2): print(value)
  ```

  Output: 
  
  ```
  0 2 4
  ```

  Remarks: Prints values in given range in increments of 2

  ```python
  for value in range(6, 1, -2): print(value)
  ```

  Output: 
  
  ```
  6 4 2
  ```

  Remarks: Prints values in given range in decrements of 2

  ```python
  for ch in "Hello World": print(ch.upper())
  ```

  Output: 
  
  ```
  HELLO WORLD
  ```

  Remarks: Prints all the characters in the string converting them to upper case

**Syntax:** \[\text{range}(\text{lower limit}, \text{upper limit}, \text{Increment/decrement by})\]
Iterative Statements- break

- Loop Control Statements - break and continue
  - When an external condition is triggered, Exits a loop immediately.
  - **Break Statement:**
    - Terminates the loop statement and transfers execution to the statement immediately following the loop.

**Example:**

```python
var = 3
while var > 0:
    print("I'm in iteration ", var)
    var -= 1
    if var == 2:
        break
    print("I'm still in while")
print("I'm out of while loop")
```

**Output:**

```
I'm in iteration  3
I'm out of while loop
```

Demo: Assignment 18: break statement

Observe the output of this code snippet when value of var = 5?
Iterative Statements - continue

- **continue** statement:
  - Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

**Example:**

```python
var = 3
while var > 0:
    print("I'm in iteration ", var)
    var -= 1
    if var == 2:
        continue
    print("I'm still in if block")
    print("I'm still in while")
    print("I'm out of while loop")
```

**Output:**

```
I'm in iteration  3
I'm in iteration  2
I'm still in while
I'm in iteration  1
I'm still in while
I'm out of while loop
```

Demo: Assignment 19: continue statement

Guided Activity: Assignment 20: Iteration Control Structure - Debugging
Iterative Statements - pass

- **pass** statement:
  - *pass* statement is never executed.
  - Used when a statement is required syntactically but do not want any command or code to execute or if the code need to be implemented in future.
  - Behaves like a placeholder for future code

**Example:**

```python
x = "Joy"
if x == "John":
    print ("Name:",x)
elif x == "Joy":
    pass
else:
    print ("in else")
```

**Output:**

No Output
Collections in Python

• Dynamic way of organizing data in memory

• Some of the Collections available in Python are:
  – Strings
  – List
  – Tuples
  – Sets
  – Dictionaries
Strings

• Accepts 3 types of quotes to assign a string to a variable.
  – single (‘), double ("), and triple (""") or """
  – String starts and ends with same type of quote
  – Triple quotes are used to span string across multiple lines.

• Index starts from zero.

• Can be accessed using negative indices. Last character will start with -1 and traverses from right to left.

Syntax:

```python
word = 'Programming'
sentence = "Object Oriented Programming."
paragraph = """"Python is a Object Oriented Programming Language. It is a Beginner’s language."""
```

Demo: Assignment 22: Strings
String Operators and Functions

- **Concatenation**
  - Strings can be concatenated with ‘+’ operator
    - “Hello” + “World” will result in **HelloWorld**

- **Repetition**
  - Repeated concatenation of string can be done using asterisk operator “*”
    - “Hello” * 3 will result in **HelloHelloHello**

- **Indexing**
  - “Python”[0] will result in “P”

- **Slicing**
  - Substrings are created using two indices in a square bracket separated by a ‘:’
    - “Python”[2:4] will result in “th”

- **Size**
  - prints length of string
    - len(“Python”) will result in **6**

Guided Activity: Assignment 23, 24, 25, 26, 27 : Strings
Tuples
Tuples

- An ordered group of sequences separated by symbol , and enclosed inside the parenthesis
- Tuples are immutable.

**Syntax:**

```python
# Creation of empty tuple
tuple1 = ()
tuple2 = (Sequence1,)
tuple3 = (Sequence1, Sequence2)
```

**Examples:**

```python
customer = ("John",)
customers = ('John', 'Joe', 'Jack', 'Jill', 'Harry')
customers[0] = "Sam"  # Compilation Error
```

Cannot modify existing values - Immutable

, symbol is mandatory without which it becomes just a string assignment operation

Demo: Assignment 28: Operations on Tuples

Guided Activity: Assignment 29: Tuples
Lists

• An ordered group of sequences enclosed inside square brackets and separated by symbol ,
• Lists are mutable

Syntax:

```python
list1 = [] #Creation of empty List
list2 = [Sequence1,]
list3 = [Sequence1, Sequence2]
```

Examples:

```python
language = ['Python']
languages = ['Python','C','C++','Java']
Languages[1] = "Perl"
```

In this case symbol , is NOT mandatory

Mutable – Can change existing values
## List Notation and Examples

<table>
<thead>
<tr>
<th>List Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>An empty list</td>
</tr>
<tr>
<td>[1, 3, 7, 8, 9, 9]</td>
<td>A list of integers</td>
</tr>
<tr>
<td>[7575, “Shyam”, 25067.56]</td>
<td>A list of mixed data types</td>
</tr>
<tr>
<td>[[7575, “John”, 25067.56], [7531, “Joe”, 56023.2], [7821, “Jill”, 43565.23]]</td>
<td>A nested list</td>
</tr>
<tr>
<td>[“India”, [“Karnataka”, [“Mysore”, [GEC1, GEC2]]]]</td>
<td>A deeply nested list</td>
</tr>
</tbody>
</table>

**Demo: Assignment 30: Accessing Elements from Lists**
## Basic List Operations

<table>
<thead>
<tr>
<th>Python Expression</th>
<th>Result</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len([4, 5, 6])</code></td>
<td>3</td>
<td>Length</td>
</tr>
<tr>
<td><code>[1, 3, 7] + [8, 9, 9]</code></td>
<td><code>[1, 3, 7, 8, 9, 9]</code></td>
<td>Concatenation</td>
</tr>
<tr>
<td><code>[ 'Hello' ] * 4</code></td>
<td>['Hello', 'Hello', 'Hello', 'Hello']</td>
<td>Repetition</td>
</tr>
<tr>
<td><code>7 in [1, 3, 7]</code></td>
<td>True</td>
<td>Membership</td>
</tr>
<tr>
<td><code>for n in [1, 3, 7] : print(n)</code></td>
<td>1 3 7</td>
<td>Iteration</td>
</tr>
<tr>
<td><code>n = [1, 3, 7] print(n[2])</code></td>
<td>7</td>
<td>Indexing: Offset starts at 0</td>
</tr>
<tr>
<td><code>n = [1, 3, 7] print(n[-2])</code></td>
<td>3</td>
<td>Negative slicing: Count from right</td>
</tr>
<tr>
<td><code>n = [1, 3, 7] print(n[1:])</code></td>
<td>[3, 7]</td>
<td>Slicing</td>
</tr>
</tbody>
</table>

Guided Activity: Assignment 31, 32, 33: Lists
Sets

• An un-ordered collection of unique elements
• Are lists with no index value and no duplicate entries
• Can be used to identify unique words used in a paragraph

**Syntax:**

```python
set1 = {}  # Creation of empty set
set2 = {"John"}  # Set with an element
```

**Example:**

```python
s1 = set("my name is John and John is my name".split())
s1 = {'is', 'and', 'my', 'name', 'John'}
```

• Operations like intersection, difference, union, etc. can be performed on sets

**Demo: Assignment 34: Sets**
Dictionary

• A list of elements with key and value pairs (separated by symbol :) inside curly braces.
• Keys are used instead of indexes
• Keys are used to access elements in dictionary and keys can be of type – strings, number, list, etc
• Dictionaries are mutable, i.e., it is possible to add, modify and delete key-value pairs

Syntax:

```python
phonebook = {}  # Creation of empty Dictionary
phonebook={"John":938477565}  # Dictionary with one key-value pair
phonebook={"John":938477565, "Jill":938547565}  # 2 key-value pairs
```
Mutable v/s Immutable Data Types

<table>
<thead>
<tr>
<th>Mutable Data Type</th>
<th>Immutable Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequences can be modified after creation</td>
<td>Sequences cannot be modified after creation</td>
</tr>
<tr>
<td>Ex: Lists, Sets, Dictionary</td>
<td>Ex: Strings, Tuples</td>
</tr>
<tr>
<td>Operations like add, delete and update can be performed</td>
<td>Operations like add, delete and update cannot be performed</td>
</tr>
</tbody>
</table>

**Example**

```python
int_list = [12, 14, 9]
int_tuple = (12, 14, 19)

int_list[0] = 7  # prints [7, 14, 9]
print("List: ", int_list)

int_tuple[0] = 7  # Prints TypeError: 'tuple' object does not support item assignment
print("Tuple: ", int_tuple)
```

**Output**

```
List: [7, 14, 9]
Tuple: TypeError: 'tuple' object does not support item assignment
```
Functions
Functions

• Are blocks of organized, reusable code used to perform single or related set of actions
• Provide better modularity and high degree of reusability
• Python supports:
  – Built-in functions like print() and
  – User-defined functions
Functions (Cont…)

• Defining a function:
  – Function blocks starts with a keyword ‘def’ followed by function_name, parenthesis (() and a colon :
  – Arguments are placed inside these parenthesis
  – Function block can have optional statement/comment for documentation as its first line
  – Every line inside code block is indented
  – return [expression] statement exits the function by returning an expression to the caller function.
  – return statement with no expression is same as return None.

Syntax:

```python
def function_name( parameters ):
    "—optional: Any print statement for documentation"
    function_suite
    return [expression]
```

– Parameters exhibit positional behavior, hence should be passed in the same order as in function definition
Functions (Cont…)

• Calling a Function
  – Defining a function gives it a name, specifies function parameters and structures the blocks of code.
  – Functions are invoked by a function call statement/code which may be part of another function

• Example:

```python
# Defining function print_str(str1)
def print_str(str1):
    print("This function prints string passed as an argument")
    print(str1)
    return

# Calling user-defined function print_str(str1)
print_str("Calling the user defined function print_str(str1)")
```

• Output:

```
This function prints string passed as an argument
Calling the user defined function print_str(str1)
```
Functions (Cont…)

• Pass arguments to functions:
  – Arguments are passed by reference in Python
  – Any change made to parameter passed by reference in the called function will reflect in the calling function based on whether data type of argument passed is mutable or immutable
  – In Python
    • **Mutable Data types** include Lists, Sets, Dictionary
    • **Immutable Data types** include Number, Strings, Tuples
Functions (Cont…)

• Pass arguments to functions: Immutable Data Type - Number

Example:

```python
# Function Definition
def change(cust_id):
    cust_id += 1
    print("Customer Id in function definition: ", cust_id)
    return

# Function Invocation with arguments of immutable data type
cust_id = 100
print("Customer Id before function invocation: ", cust_id)
change(cust_id)
print("Customer Id after function invocation: ", cust_id)
```

Output:

Customer Id before function invocation: 100
Customer Id in function definition: 101
Customer Id after function invocation: 100

Observe that customer id remains unchanged even after function invocation
Functions (Cont…)

- Pass arguments to functions: Mutable Data Type - List

Example:

```python
#Function Definition
def change(list_cust_id):
    #Assign new values inside the function
    list_cust_id.append([10, 20, 30])
    print("Customer Id in function definition: ", list_cust_id)
    return

# Function Invocation with arguments of immutable data type
list_cust_id = [100, 101, 102]
print("List of Customer Id before function invocation: ", list_cust_id)
change(list_cust_id)
print("Customer Id after function invocation: ", list_cust_id)
```

Output:

List of Customer Id before function invocation: [100, 101, 102]
List of Customer Id in function definition: [100, 101, 102, [10, 20, 30]]
List of Customer Id after function invocation: [100, 101, 102, [10, 20, 30]]
Functions (Cont…)

- Different types of formal arguments:
  - Required arguments
  - Keyword arguments
  - Default arguments
  - Variable – length arguments

Demo: Assignment 38: Functions – Pass by Reference

Guided Activity: Assignment 39: Functions – Pass by Reference
Functions (Cont…)

- **Required arguments**
  - Arguments follow positional order
  - No. of arguments and the order of arguments in the function call should be exactly same as that in function definition

**Example:**

```python
# Function Definition
def print_str(str1):
    print("This function prints the string passed as an argument")
    print(str1)
    return

# Function Invocation without required arguments
print_str()
```

**Output:**

```
TypeError: print_str() missing 1 required positional argument: 'str1'
```
Functions (Cont…)

• Keyword arguments
  – when used in function call, the calling function identifies the argument by parameter name
  – Allows you to skip arguments or place them out of order
  – Python Interpreter uses the keyword provided to match the values with parameters

Example:

# Function Definition
def customer_details (cust_id, cust_name):
    print("This function prints Customer details")
    print("Customer Id: ",cust_id)
    print("Customer Name: ",cust_name)
    return

# Function Invocation with Keyword arguments
customer_details(cust_name = "John", cust_id = 101)

Output:

This function prints Customer details
Customer Id:  101
Customer Name:  John

Observe the change in positional order of arguments
Functions (Cont…)

- Default Arguments:
  - Assumes a default value if the value is not specified for that argument in the function call

**Example:**

```python
# Function Definition
def customer_details(cust_name, cust_age = 30):
    print("This function prints Customer details")
    print("Customer Name: ", cust_name)
    print("Customer Age: ", cust_age)
    return

# Function Invocation with Default arguments
customer_details(cust_age = 25, cust_name = "John")
customer_details(cust_name = "John")
```

**Output:**

- This function prints Customer details
  - Customer Name: John
  - Customer Age: 25
- This function prints Customer details
  - Customer Name: John
  - Customer Age: 30

Observe the usage of default value for cust_age argument
Functions (Cont…)

• Variable-length arguments
  – Used to execute functions with more arguments than specified during function definition
  – unlike required and default arguments, variable arguments are not named while defining a function

Syntax:

```python
def functionname([formal_args,] *var_args_tuple):
    """—optional: Any print statement for documentation"
    function_suite
    return [expression]
```

– An asterisk `*` is placed before variable name to hold all non-keyword variable arguments
– `*var_args_tuple` is empty if no additional arguments are specified during function call
Functions (Cont…) 

• Variable-length arguments

**Example:**

```python
# Function Definition
def customer_details(cust_name, *var_tuple):
    print("This function prints Customer Names")
    print("Customer Name: ", cust_name)
    for var in var_tuple:
        print(var)
    return

# Function Invocation with Variable length arguments
customer_details("John", "Joy", "Jim", "Harry")
customer_details("Mary")
```

**Output:**

- This function prints Customer Names
- Customer Name: John
- Joy
- Jim
- Harry

- This function prints Customer Names
- Customer Name: Mary

Invocate this function without arguments and observe the output
Functions (Cont…)

• Scope of variables
  – Determines accessibility of a variable at various portions of the program

• Different types of variables
  – Local variables
    • Variables defined inside the function have local scope
    • Can be accessed only inside the function in which it is defined
  – Global variables
    • Variables defined outside the function have global scope
    • Variables can be accessed throughout the program by all other functions as well

Example:

total = 0

# Function Definition
def add( arg1, arg2 ):
    # Add both the parameters and return total
    total = arg1 + arg2; # total is local variable
    print ("Value of Total(Local Variable): ", total)
    return total;

# Function Invocation
add( 25, 12 );
print("Value of Total(Global Variable): ", total)

Output:

Value of Total(Local Variable): 37
Value of Total(Global Variable): 0
Usage of keyword ‘Global’

• Used to access the variable outside the function

Example:

```python
total = 0

# Function Definition
def add( arg1, arg2 ):
    # Add both the parameters and return total
    global total
    total = arg1 + arg2;  # Here total is made global variable
    print("Value of Total(inside the function): ", total)
    return total;

# Function Invocation
add( 25, 12 );
print("Value of Total(outside the function): ", total)
```

Output:

Value of Total(inside the function): 37
Value of Total(outside the function): 37
Introduction to Python Tutor (Self Study)

• A visualization tool Created by Philip Guo

• Helps in understanding step-by-step execution of each line of a program’s source code.

• It can be used to write and visualize programs in Python, Java, JavaScript, Ruby, C and C++

• Click on the link below to visualize your code using Python Tutor and follow the steps in following 3 slides

  http://www.pythontutor.com/

• Suggested Sections of examples on Python Tutor:

  – Basic
    • Intro – a 10min introduction to Python Programming Fundamentals
  – Math
  – Objects
  – Linked Lists
Introduction to Python Tutor...

Click here to start coding and visualizing.
Introduction to Python Tutor...

Select programming language Python 3.3 from drop-down menu.

Write your code here.

Click on Visualize execution to visualize your code.
Introduction to Python Tutor...

Click here to visualize the step-by-step execution of your code

Observe the change in values of variables during execution and contents of frames and objects
Math Module

- Provides access to mathematical functions like power, logarithmic, trigonometric, hyperbolic, angular conversion, constants etc;
- Few functions are described below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>Absolute value of x: the (positive) distance between x and zero</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>Ceiling of x: smallest integer not less than x</td>
</tr>
<tr>
<td>cmp(x, y)</td>
<td>-1 if x &lt; y, 0 if x == y, or 1 if x &gt; y</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Exponential of x: (e^x)</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Floor of x: the largest integer not greater than x</td>
</tr>
<tr>
<td>max(x1, x2,...)</td>
<td>Largest of its arguments: the value closest to positive infinity</td>
</tr>
<tr>
<td>min(x1, x2,...)</td>
<td>Smallest of its arguments: the value closest to negative infinity</td>
</tr>
<tr>
<td>pow(x, y)</td>
<td>Value of (x^{**y})</td>
</tr>
<tr>
<td>round(x [,n])</td>
<td>x rounded to n digits from the decimal point.</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Square root of x for (x &gt; 0)</td>
</tr>
</tbody>
</table>
### String Module

- Includes built-in methods to manipulate strings. Consider the string, `str = Infosys`

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str.count(&quot;s&quot;)</code></td>
<td>Returns count of occurrence of character “s” in string str</td>
<td>2</td>
</tr>
<tr>
<td><code>str.startswith(&quot;s&quot;)</code></td>
<td>Returns true if string str starts with character “s”</td>
<td>false</td>
</tr>
<tr>
<td><code>str.endswith(&quot;s&quot;)</code></td>
<td>Returns true if string str ends with character “s”</td>
<td>true</td>
</tr>
<tr>
<td><code>str.find(&quot;s&quot;)</code></td>
<td>Returns index position of character “s” in string str if found else -1</td>
<td>4</td>
</tr>
<tr>
<td><code>str.replace(&quot;s&quot;, &quot;S&quot;)</code></td>
<td>Replaces all occurrences of character “s” with character “S” in string str</td>
<td>InfoSyS</td>
</tr>
<tr>
<td><code>str.isdigit()</code></td>
<td>Checks if all the characters in string str are digits and returns true or false accordingly</td>
<td>false</td>
</tr>
<tr>
<td><code>str.upper()</code></td>
<td>Converts all the characters in string str to uppercase</td>
<td>INFOSYS</td>
</tr>
<tr>
<td><code>str.lower()</code></td>
<td>Converts all the characters in string str to lowercase</td>
<td>infosys</td>
</tr>
</tbody>
</table>

**Guided Activity: Assignment 40: Strings built-in functions**
List module

- Built-in functions and methods in lists

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp(list1, list2)</td>
<td>Compares elements of both lists</td>
</tr>
<tr>
<td>len(list)</td>
<td>Gives total length of list</td>
</tr>
<tr>
<td>max(list)</td>
<td>Returns item from the list with maximum value</td>
</tr>
<tr>
<td>min(list)</td>
<td>Returns item from the list with minimum value</td>
</tr>
<tr>
<td>list(seq)</td>
<td>Converts a tuple to list</td>
</tr>
<tr>
<td>list.append(obj)</td>
<td>Appends object obj to list</td>
</tr>
<tr>
<td>list.count(obj)</td>
<td>Returns count of how many times obj occurs in list</td>
</tr>
<tr>
<td>list.insert(index, obj)</td>
<td>Inserts object obj into list at offset index</td>
</tr>
<tr>
<td>obj = list.pop()</td>
<td>Removes the item at position -1 from list and assigns it to obj</td>
</tr>
<tr>
<td>list.remove(obj)</td>
<td>Removes object obj from list</td>
</tr>
<tr>
<td>list.reverse()</td>
<td>Reverses the order of items in list</td>
</tr>
<tr>
<td>sorted(list)</td>
<td>Sorts items in list</td>
</tr>
</tbody>
</table>

Guided Activity: Assignment 41: Lists
Module

- Allows logical organization of code.
- Grouping related code into module makes it easier to understand and use.
- Can be used to define functions, classes and variables.
- It is a file and may have runnable code.

**import** Statement

- Any Python source file can be used as a module by executing an import statement in any other Python source file

**Syntax:**

```python
import module1[, module2[,.... moduleN]]
```

- When the interpreter encounters the import statement, it imports the module if it is present
- Module is loaded only once regardless of the number of times it is imported
Packages and its elements
Packages

- Packages are namespaces which contain multiple packages and modules
- Collection of modules in directory
- Must have `__init__.py` file
- May contain subpackages
- `__init__.py` can be empty or it contains valid python code
- `__init__.py` indicates that the directory it contains is a package and it can be imported the same way as a module
  - **Ex: foo.abc**
    - Module abc belongs to package named foo
- Users of the package can import individual modules from the package
  - **Ex: import sound.effects.echo**
    - This loads the submodule sound.effects.echo. It must be referenced with its full name.
Exception

• Is a runtime error that abnormally terminates the program

• Consider the following code:

  1. num1 = 4
  2. num2 = 0
  3. num3 = num1 / num2
  4. print (num3)

• Exception occurs on Line 3 causing program to terminate abnormally. Hence Line 4 is not executed.
Exception handling

• Used to handle any unexpected error in Python programs
• Few Standard Exceptions:

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>Base class for all exceptions</td>
</tr>
<tr>
<td>Arithmetic Error</td>
<td>Base class for all errors that occur for numeric calculation</td>
</tr>
<tr>
<td>Floating Point Error</td>
<td>Raised when a floating point calculation fails.</td>
</tr>
<tr>
<td>Zero Division Error</td>
<td>Raised when division or modulo by zero takes place for all numeric types.</td>
</tr>
<tr>
<td>IO Error</td>
<td>Raised when an input / output operation fails, such as print() or open() functions when trying to open a file that does not exist.</td>
</tr>
<tr>
<td>Syntax error</td>
<td>Raised when there is a error on Python syntax</td>
</tr>
<tr>
<td>Indentation error</td>
<td>Raised when indentation is not specified properly</td>
</tr>
<tr>
<td>Value Error</td>
<td>Raised when built-in-function for a data type has a valid type of arguments, but the arguments have invalid values specified</td>
</tr>
<tr>
<td>Runtime Error</td>
<td>Raised when a generated error does not fall into any category</td>
</tr>
</tbody>
</table>
Handling an Exception

• Exception is an event, which occurs during the execution of program and disrupts the normal flow of program’s instructions.

• When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

• If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block.

• After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

• Different ways of Exception Handling in Python are:
  – try...except
  – try...finally
Handling an Exception…

• **try...except**
  - A single try statement can have multiple except statements
  - Useful when we have a try block that may throw different types of exceptions
  - Code in else-block executes if the code in the try: block does not raise an exception

**Syntax:**
```python
try:
    You do your operations here;
    ......................
except ExceptionA:
    If there is ExceptionA, then execute this block.
except ExceptionB:
    If there is ExceptionB, then execute this block.
except:
    If none of the exception matches then execute this block
```

**Example:**
```python
num1 = 4
num2 = 0
try:
    num3 = num1 / num2
    print(num3)
except:
    print("zero division error")
```

**Output:**
```
zero division error
```
Handling an Exception…

• try…except..
  – Catches all exceptions that occur

```python
num1 = 10
num2 = 2
list1 = [10,20,30]
try:
    num3 = num1 / num2
    print(num3)
    print(list1[4])
except ZeroDivisionError:
    print("Division by zero")
except IndexError:
    print("Index out of range")
except:
    print("Not specific error")
print(num1)
print(num2)
```

Predict the output of this code?
Handling an Exception...

• `try...finally..`
  - finally block is a place to put any code that must execute irrespective of try-block raised an exception or not.
  - except block can be used with finally block

**Syntax:**
```
try:
    You do your operations here;
    ......................
    Due to any exception, this may be skipped.
except:
    ......................
finally:
    This would always be executed.
    ......................
```

**Example:**
```python
num = 4
try:
    res = num / 0
    print(res)
except:
    print("zero division error")
finally:
    print("Inside finally")
    print("out of try-except")
```

**Output:**
```
zero division error
Inside finally
out of try-except
```
Regular Expression

• Special sequence of characters and/or symbols used to express the string or pattern to be searched within a given text

• Widely used in languages like UNIX, PHP, Perl

• Module `re` provides support for regular expressions in Python

• `re` module raises `re.error` exception if it encounters an error while compiling the regular expressions

• Import `re` module to access methods in `re` module

• `search` function
  – Searches for the first occurrence of a pattern within a string with optional flags.

**Syntax:**

```
re.search(pattern, string, flags=0)
```
# Regular Expression Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Matches start of a string</td>
</tr>
<tr>
<td>$</td>
<td>Matches end of string</td>
</tr>
<tr>
<td>.</td>
<td>Matches any single character except newline</td>
</tr>
<tr>
<td>()</td>
<td>Matches sub-pattern inside parenthesis as a group and stores the matched substring for later use</td>
</tr>
<tr>
<td>*</td>
<td>Matches zero or more occurrences of preceding sub-pattern, Creates greedy expression</td>
</tr>
<tr>
<td>+</td>
<td>Matches one or more occurrences of preceding sub-pattern, Creates greedy expression</td>
</tr>
<tr>
<td>?</td>
<td>Matches zero or one occurrence of preceding sub-pattern, Creates greedy expression</td>
</tr>
<tr>
<td>*?</td>
<td>Matches zero or more occurrences of preceding sub-pattern, Creates Non-greedy expression</td>
</tr>
<tr>
<td>{m, n}</td>
<td>Matches from m repetitions to n occurrences of the preceding sub-pattern</td>
</tr>
<tr>
<td>[ ]</td>
<td>Matches any one of the character in the set</td>
</tr>
</tbody>
</table>
Special Character ^, $

- ^ - Matches start of a string

**Example1**

import re
sample_string = ("Python is Object Oriented Language")
if re.search('^Python', sample_string):
    print ('Pattern Matched')
else:
    print ('Pattern Unmatched')

**Example2**

import re
sample_string = ("Python is Object Oriented Language")
if re.search('Language$', sample_string):
    print ('Pattern Matched')
else:
    print ('Pattern Unmatched')

- $ - Matches end of a string

Output
Pattern Matched

Output
Pattern Matched
Object Oriented Programming
Need for Object Oriented Approach

• Challenges in developing a business application

• If these challenges are not addressed it may lead to **Software Crisis**

• Features needed in the business application to meet these challenges:
  - Modularity
  - Extendibility
  - Reusability
  - Interoperability
  - Security

• Challenges can be addressed using object oriented approach
Need for Object Oriented Approach

• Properties of a business application

  - Clear separation of functionalities
  - Exhibits hierarchy
  - Composed of subsystems

• These properties can be implemented using object oriented approach

  Easy Shop application is a complex business application &
  object oriented approach may be used to develop this system
  (Refer Case Study I)

Road Ahead - Need to understand object oriented concepts
**Object**
Real world entities, which has two characteristics namely, state (attributes) and behavior (method). It is an active entity.

**Class**
A description of what data is accessible through a particular kind of object, and how that data may be accessed. It is a passive entity.

**Method**
The means by which an object’s data is accessed, modified, or processed.

**Abstraction**
Hides all but the relevant data about an object so as to reduce complexity and increase efficiency. Focus on what the object does instead of how it does.

**Encapsulation**
This wraps code and data into a single unit. And separates what from the how. Or implementation of abstraction.

**Inheritance**
The way in which existing classes of object can be upgraded to provide additional data or methods.

**Polymorphism**
The way that distinct objects can respond differently to the same message, depending on the class they belong to.
Let us look at some customers

Cameron & Alphonso are two instances /examples /objects of Easy Shop customers

**Attributes of Customer**

**Customer Id:**
1001

**Customer Name:**
Cameron

**Telephone Number:**
9901911445

**Address:**
No.31, Silver Shine, Bangalore, India

**Customer Id:**
1002

**Customer Name:**
Alphonso

**Telephone Number:**
9496244655

**Address:**
No.255, Brigade, Bangalore, India
Classes & Objects (2 of 2)

• A class is a prototype / design that describes the common attributes (properties) and activities (behaviors) of objects

Attributes
- Example: Customer Id, Name, Telephone number and Address

Behavior/Activity
- Activities (behavior) exhibited by the class to external world
- Example: Purchasing items from the retail shop

Guided Activity: Object Oriented Fundamentals - Assignment 43
Everything in Python is an object

• When you are working with Python, always remember that everything i.e. variable, class, function, method etc. in Python is an object

• Thus Python embraces OOP at a fundamental level

• An object consists of:
  1. A collection of related information i.e. attributes.
  2. A set of operations to manipulate that information i.e. behaviors / methods.
Abstraction – Guided Activity

- Users of the retail application – Billing staff, Admin, Retail outlet manager
- Each user needs to know some details and need not know other details

Who are the users of the retail application?
What are the things each user must know to perform their activities?

Billing staff (Billing of customers)
Admin (Registration of customers)
Retail Outlet Manager (Registration of users)

ABSTRACTION: Process of identifying the essential details to be known and ignoring the non-essential details from the perspective of the user of the system
Encapsulation – Guided Activity

- Swipe machine in a retail store
  - Used by billing staff to key the amount
  - Used by admin to record payment

ENCAPSULATION: A mechanism of hiding the members from the external world. Swipe machine has all the control to perform the operations.

How is a swipe machine used for payment of bill in a retail store?
Inheritance – Guided Activity

• Customers are of two kinds
  – Regular
  – Privileged

All customers have Customer Id, Name, Telephone Number and Address

The regular customer in addition is given discounts

The privileged customer gets a membership card based on which gifts are given

INHERITANCE: Is a mechanism which allows to define generalized characteristics and behavior and also create specialized ones. The specialized ones automatically tend to inherit all the properties of the generic ones

What are the two different types of customers you can see in the retail application?
Types of Inheritance

- **Single Inheritance**
  - Father
  - Son

- **Multi-level Inheritance**
  - Grand Father
  - Father
  - Son

- **Multiple Inheritance**
  - Bird
  - Mammal
  - Bat

- **Hierarchy Inheritance**
  - Animal
    - Wild
    - Domestic
    - Lion
Polymorphism – Guided Activity

• Payment of bill - Two modes
  – Cash (Calculation includes VAT)
  – Credit card (Calculation includes processing charge and VAT)

What do you observe in this retail store scenario?

POLYMORPHISM: Refers to the ability of an object/operation to behave differently in different situations

Guided Activity: OO Concepts Quiz - Assignment 44
Object Oriented Approach – Benefits

- Leads to development of smaller but stable subsystems
- The subsystems are resilient to change
- Reduces the risk factor in building large systems as they are built incrementally from subsystems which are stable

Hence Object Orientation is suitable for developing extremely complex business systems

Guided Activity: OO Concepts Rapid Fire - Assignment 45
What is UML?

“The Unified Modelling Language (UML) is a language for visualizing, specifying, constructing and documenting the software system and its components” [OMG03a]
UML Diagrams

• There are thirteen standard diagrams used in different phases

<table>
<thead>
<tr>
<th>Use Case Diagram</th>
<th>Deployment Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Diagram</td>
<td>State Machine Diagram</td>
</tr>
<tr>
<td>Object Diagram</td>
<td>Activity Diagram</td>
</tr>
<tr>
<td>Component Diagram</td>
<td>Sequence Diagram</td>
</tr>
<tr>
<td>Composite Structure Diagram</td>
<td>Communication Diagram</td>
</tr>
<tr>
<td>Package Diagram</td>
<td>Timing Diagram</td>
</tr>
<tr>
<td></td>
<td>Interaction Overview Diagram</td>
</tr>
</tbody>
</table>

We will be focusing on **class diagrams** in this course.
Class Diagram

• Classes are the basic components of an object oriented system
• This diagram shows the collection of classes and the relationships among them
• In UML, any class is represented by a rectangular box divided with three compartments:

Class Name
Attributes
Behaviors

Access Specifiers
+ public
- private
# protected

Guided Activity: Class Diagram - Assignment 46

Links: http://www.uml-diagrams.org/
Refer Appendix Section for more
Creating Classes

- Classes are OOP tools/templates used to create objects.
- Definition of a **class** starts with keyword `class` followed by `classname` and `:`. It contains attributes and methods.
- An indented block of statements forms the body of the class.

```python
class Demo:
    pass  # An empty block

p = Demo()
```

- Like functions and modules, classes are also python program units but they are more useful while building new objects.
Creating Classes (contd…)

- **class** `classname` :
  - Contains the code block for defining the class
  - Suitable class name, should follow the rules of naming variables
  - Mandatory to begin the code block
Instance Variables – Guided Activity

• Member / Instance variables used to represent the state of an object / attributes of a class

Identify the member variables of the Customer class?

- customerid
- telephoneno

customerid can be Integer and telephoneno can be Long

Identify the data types to be used for these member variables?

Valid names/attributes:
- __custno
- _cname
- address2
- emp_no

Invalid names/attributes:
- 12test
- &test
- office address
- test-no

Python doesn’t need variable/attribute declaration. Reduces the job for programmer.
Access Specifiers

- Used to expose or hide the attribute and behavior of a class
- Used to specify access permissions on a member variable/function/method

<table>
<thead>
<tr>
<th>Naming</th>
<th>Type</th>
<th>UML Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Public</td>
<td>+</td>
<td>These attributes can be freely used inside or outside of a class definition. A declaration that is accessible to all classes</td>
</tr>
<tr>
<td>_name</td>
<td>Protected</td>
<td>#</td>
<td>Protected attributes are accessible only within the class in which it is declared and to its inherited sub-classes.</td>
</tr>
<tr>
<td>__name</td>
<td>Private</td>
<td>-</td>
<td>This kind of attribute is inaccessible and invisible. It's neither possible to read nor write to those attributes, except inside of the class definition itself. A declaration that is accessible only to the class in which it is declared</td>
</tr>
</tbody>
</table>
Methods (1 of 2)

- Methods define the behavior of an object
- All the methods should be defined inside the class with an access specifier
- Values that are passed to methods are known as arguments/parameters
- Value that is returned from a method is known as return value
- A method can return only one value at a time and it can be done using the return statement
- Implementation of a method requires the following:

  ![Method Header](image1)
  ![Method Definition](image2)
  ![Method Invocation/Method Call](image3)
Methods (2 of 2)

**def** is the keyword which is needed to define a method inside a class.

**Method header**

- *def* keyword is mandatory to define a method for a class.
- **self** is mandatory for a method which is a member of the class.
- However, no need to pass argument for **self** while calling the method using object.

**Access specifier** (Public as no underscore)

**Arguments/parameters**

Some methods do not return explicit values and for those methods **return** is not needed.
class Customer:
    def setcustomerid(self, id):
        self.customerid = id

    def settelephoneno(self, teleno):
        self.telephoneno = teleno

    def getcustomerid(self):
        return self.customerid

    def gettelephoneno(self):
        return self.telephoneno

How can this class be used to create an instance of a customer?

By creating objects of Customer class
Creation of Objects

• Class is a blueprint for the creation of objects
• To realize a class, an object or an instance of the class needs to be created
• There can be many instances for a class and each instance will have its own data
• In Python, Class name with brackets allocates memory for objects during run time i.e. dynamic memory allocation e.g.

    Customer()

    Customer(cid=1001, name=“Kevin”)  

• Also one or more arguments could be passed for object creation e.g.

    Customer(1001, “Kevin”)   

This will be discussed more in __init__() method.
Reference Variables

• The reference returned by a newly created object must be tagged to a variable and that is known as reference variable.

• Following syntax can be used to create a reference variable for the Customer class and make it point to a Customer object.

```python
custobj = Customer()
```

• As there is no need for declaration in Python, custobj will be identified as reference type automatically during the runtime.

• In python, if you end with semicolon (;) it will not display any error. However, ending semicolon is not in the statement syntax of python.

```python
custobj = Customer();
```

No ERROR!
Method Invocation

Bound Method Invocation

• To invoke a method, the instance has to be followed by a period and
  then followed by the method to be invoked. This is **Bound method
  invocation**.
• Ex:  `print (“Customer Name: ”, objcust.getcustomername())`

Unbound Method Invocation

• In **Unbound method invocation**, the method is invoked by accessing
  it through its containing class. Being a little inconvenient it is not much
  used.
• Ex:  `print (“Customer Name: ”, Customer.getcustomername(objcust))`
A complete Python Program

Guided Activity: Classes & Objects Hands On - Assignment 47

- Access Specifiers
- Variables – Local and Instance variables
- Methods
- Creation of objects & Storing Reference
- Compilation and Execution of a python program
Importance of ‘self’

- Explicit reference to refer the current object, i.e. the object which invoked the method
- Used to create and initialize instance variables of a class i.e. it creates the attribute for the class
- ‘self’ reference must be used as a first parameter in all instance methods of a class otherwise the methods are known as simply “class methods”
- Moreover, “self” is not a keyword and has no special meaning in Python. We can use any name in that place. **However, it is recommended not to use any name other than “self”** (merely a convention and for readability)
__init__() method

• __init__() method helps in initializing the variable during the creation of object. Hence, it is also called as 'initializer method' or Default Constructor.

• Before __init__() method, the object is been already constructed

  – Simple __init__() method

  – Parameterized __init__(argument 1, argument 2, ...) method
**Instance Methods and Static Methods**

**Instance Method**

- **Instance / Regular methods** require an instance (self) as the first argument and when the method is invoked (bound), self is automatically passed to the method.

**Static Method**

- **Static methods** are functions which do not require instance but are part of class definitions.
When to use Static and Instance methods?

Instance method

• Useful when method needs access to the values that are specific to the instance and needs to call other methods that have access to instance specific values.

Static method

• Useful when method does not need access to either the class variables or the instance variables.
Using default parameters in Method

- Python doesn’t have compile-time polymorphism as data type selection is not happening during compile-time.
- Python is a dynamically typed language and hence the concept of method overloading is not supported.
- However, illusion of method overloading could be created by the usage of default arguments in method.
- When user doesn’t pass any value for the argument, the default value will be taken for that argument during execution.

Demo: Using default parameters in Method - Assignment 53 & 54
Relationships

• Objects should collaborate each other for which relationships help in connecting to each other

• Few kinds of relationships between classes:
  – Generalization and specialization (is-a relationship)
  – Aggregation (has-a relationship)
  – Association (uses-a relationship)

Helps in reusing existing classes

Demo: Relationships - Assignment 55
Inheritance

- Classes are customized by **Inheritance**. Concept wherein a class shares some common structure or behavior with one or more classes. New classes can be created by making use of the existing classes.

- The **new class** (*derived/sub/child class*) inherits attributes of the existing class (*base/super/parent class*).

- Generalized class also known as parent class or base class or super class

- Specialized class also known as child class or derived class or sub class

- The **child class attributes** can be used just like they were created in the child class itself. A child class can **override inherited methods and the attributes of the parent class**.
Inheritance (Contd…)

• **Syntax:**
  – list of parent classes which needs to be inherited are given after the child class name.

```python
class ChildClass_Name(SuperClass1_Name[, SuperClass2_Name, ...]):
    .......
```

• Base class `__init__` should be invoked explicitly by the derived class during object creation otherwise, the members of the base class will not be initialized.

• Base class `__init__` can be invoked using “super” keyword.

• The super keyword can be used to refer members and `__init__()` of a base class from a derived class.

Quiz: OO Concepts Assignment 57
Inheritance – protected access specifier

• If an instance variable in the base class has the protected access specifier
  – In Python, member starts with _ (single underscore) is protected
  – It can be directly accessed inside the subclasses

• This is useful as methods of the child class can access the parent class variables directly

• In UML notation # is used to represent the protected access specifier
Inheritance Demo

class Customer:
    def __init__(self, id=0, name=None):
        self.__customerid = id
        self.__customercname = name

class RegularCustomer(Customer):
    def __init__(self, id=0, name=None, dis=0):
        super().__init__(id, name)
        self.__discount = dis

Python supports Multi level, Multiple, Hierarchical and Hybrid inheritance.
Multiple inheritance

```python
class father:
    def land(self):
        print("father's land")
    def money(self):
        print("father's money")

class mother:
    def jewels(self):
        print("mother's jwellery")
    def money(self):
        print("mother's money")

class son(father, mother):
    pass

class daughter(mother, father):
    pass

john = son()
mother = daugther()

john.land()
john.money()
mother.jewels()
mother.money()
```

Analyze the output for this code?
Aggregation

• An object is available inside another object, known as aggregation

• If the lifetime of an object is aligned with the lifetime of the parent object, known as composition

• A simplified version of the demo is provided here to summarize aggregation

```python
class Address:
    def __init__(self, addressline):
        self.__addressline = addressline

    def getaddressline(self):
        return self.__addressline

class Customer:
    def __init__(self, address):
        self.__address = address

    def getaddress(self):
        return self.__address
```

```python
add = Address("No.333")
custobj = Customer(add)
# Check if address is correctly saved
temp = custobj.getaddress()
print("Address:")
print(temp.getaddressline())
```
Association

• Loosely coupled relationship

• Objects are connected only at the time of need. Once the purpose is done, objects are disconnected. Thus, embraces uses-a relationship
Summary

- OOP using Python
  - Basic Python Programming
  - Basic OOP concepts
  - Creating classes and objects
  - Class variables and Object Variables
  - Method Invocation
  - Using default arguments in Methods
  - Static, Class and Instance Methods
- Relationships
  - Inheritance
  - Aggregation
  - Association
References

Books:
- Head First Programming, Apress Publications
- Head First Python, Apress Publications
- Beginning Python, Apress Publications

E-Books:

Tutorials:
- [http://www.pythontutor.com/](http://www.pythontutor.com/)
- [http://www.learnpython.org/](http://www.learnpython.org/)
- [https://docs.python.org/2/tutorial/index.html](https://docs.python.org/2/tutorial/index.html)
- [https://developers.google.com/edu/python/introduction](https://developers.google.com/edu/python/introduction)
Learn by Video:

- **Object Oriented Programming: Part 1**
  
  **Link:** https://www.youtube.com/watch?v=4YU3J5yq24A&index=46&list=PL_RGaFnxSHWpX_byHyTEj9hecPngl2DqR
  
  **Author:** Bad Tutorials
  
  **Duration:** 14:38

- **Object Oriented Programming: Part 2**
  
  **Link:** https://www.youtube.com/watch?v=sCUDcYMj_m0&index=47&list=PL_RGaFnxSHWpX_byHyTEj9hecPngl2DqR
  
  **Author:** Bad Tutorials
  
  **Duration:** 10:56
Thank You