Chapter 1 Python and Its Features

1. Introduction

Python is a powerful high-level, dynamic object-oriented programming that is implemented in C, and relies on the extensive portable C libraries.

Python is an interpreted programming language, which means that it can be executed by the computer without compiling. Compiling is the conversion of a programing language into machine instructions that can then be executed by the computer.

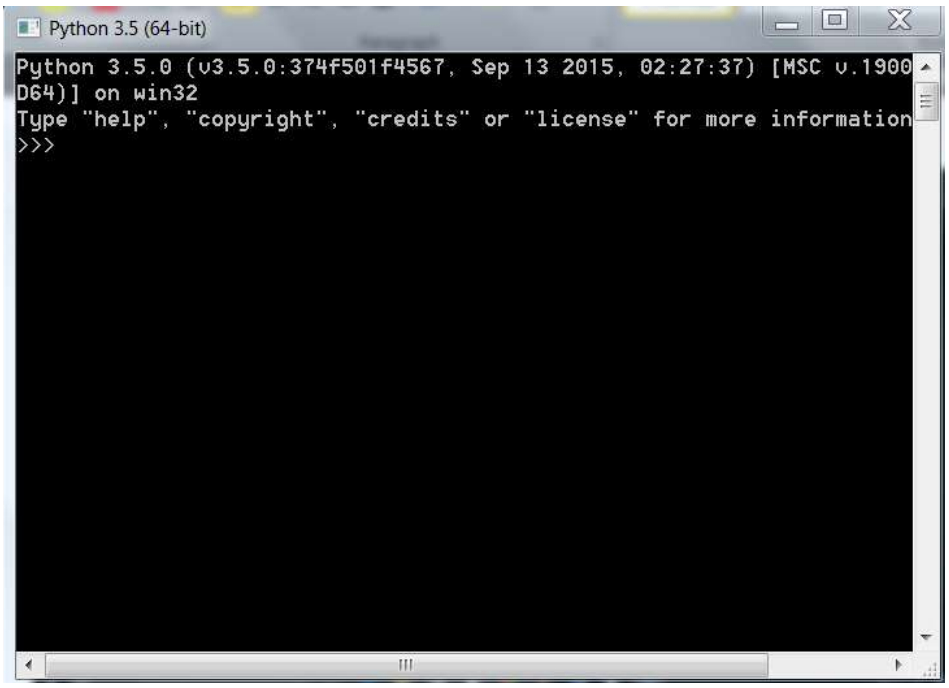
C is a general-purpose, computer programming language, supporting structured programming.

The implementation of Python studied in the course is called CPython, or Classic Python (often just called Python) other implementations are Jthon and IronPython.

2. Using Python

Python can be used on through the command line

The command line is used to execute one line of Python code at a time. Note that the commands executed this way cannot be saved.

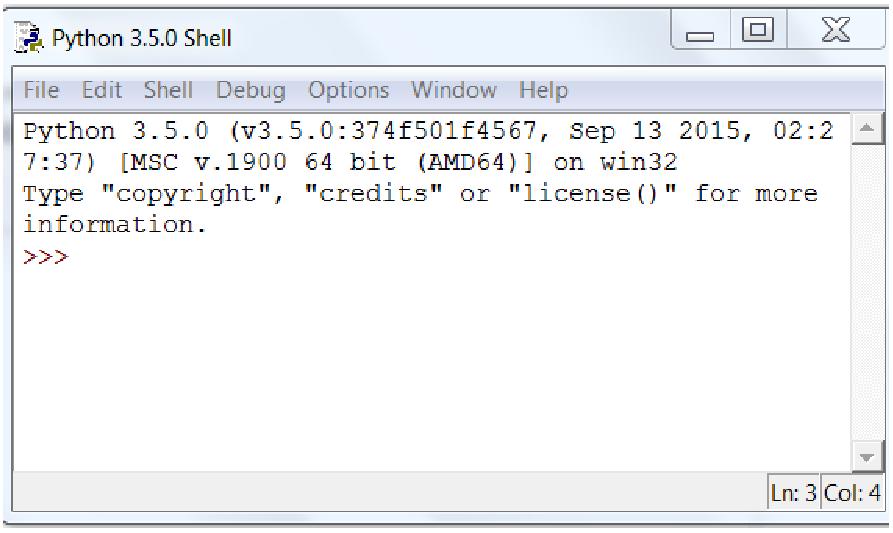


```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900
D64)] on win32
Type "help", "copyright", "credits" or "license" for more information
>>>
```

To ensure that Python can be executed from any folder on a computer, the Path variable in the Environmental Variables must be updated with the path of the Python installation.

Or the IDLE IDE

The IDLE enables the programmer to save Python code in a file and execute all the code at once. One can edit, debug and preview code in the IDLE.



```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:2
7:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>>
```

3. Basic Elements in a Program

Programs consist of the following basic elements, a collection of literals, variables, and keywords.

Literals

A *literal* is a number or string that appears directly in a program, e.g.

```
10 # Integer literal
10.50 # Floating-point literal
'Hello' # String literal
"World!" # String literal
'''Hello World!
It might rain today # Triple-quoted string literal
Tomorrow is Sunday'''
```

In Python, you can use both single and double quotes to represent strings. The strings that run over multiple lines are represented by triple quotes.

Variables

Variables are used for storing data in a program. E.g.

```
l = 10
Length = 10
k= "Hello World!"

a = True # Boolean variable
b = [2, 9, 4] # List variable
c = ('apple', 'mango', 'banana') # tuple variable
A tuple in python language refers to an ordered, immutable (non-
changeable) set of values of any data type.
```

Keywords

Python has 30 keywords: ensure you're familiar with the ones used in the textbook examples and assignments.

Comments
These are preceded by #, they are ignored by the interpreter and are used to make the code more readable.

Continuation Lines

Two types of lines in Python are called physical (e.g. the 7 lines in the code below.
A logical line is a combination of several physical lines, e.g. line 2, 3 and 4.

```
1. print (10)
2. print ('Hello World! \
3. It might rain today. \
4. Tomorrow is Sunday.')
5. print ('''Hello World!
6. It might rain today.
7. Tomorrow is Sunday.''')
```

Check page 33 of the textbook for the ways in which physical lines can be joined into one logical line.

Printing

```
print (["message"][variable list])
```

The print statement above prints the string in message and a list of variables in the optional variable list.

Chapter 2 Getting Wet in Python

1. Performing Arithmetic Operations

The operations that can be performed on a variable in Python depends on the data type of the variable. The different data types are described on page 29 of the textbook. Numerical data types can be operated on by arithmetic operators. Review the numerical data types listed in the textbook.

In addition to the normal arithmetic operators, +; -; x and /, the following operators are also used in Python

// the truncation or floor division operator which ignores the fraction in the result.

** the exponentiation operator, 2^3 is written as 2** 3 in Python

% the modulus operator gives the remainder of one number divided by another, e.g. 5%2 gives 1.

+x or –x the unary operators increase or decrease x by one respectively, e.g. if x = 10 then –x gives 9 and +x gives 11.

2. Multiple Assignment Statement

Python allows the assignment of multiple variables to multiple values in one statement, i.e.

x, y,z = 1,2,3 will assign 1 to x, 2 to y and 3 to z. This eliminates the need to write 3 assignment statements in the code.

3. Escape or nonprintable characters

Familiarise yourself with the escape characters used throughout the exercises, assignments and examples in the book. These are listed on page 42 of the textbook.

4. Other operations

Type(x) gives the data type of the variable x.

0o expresses a number in base 8 (octal).
0x expresses a number in base 6 (hexal).

variable=input ('Message') prompts a user of your program to input data.

Int(x), float(x), str(x) all converts a variable x into an integer, float or string respectively.

X << y equivalent to x multiplied by 2**y

X >> y equivalent to x divided by 2**y

X & Y compares corresponding bits of x and y if both are 1, returns 1 otherwise returns 0.

X | Y compares corresponding bits of x and y if both are 0, returns 0 otherwise returns 1.

x ^ y compares corresponding bits of x and y if either is 1, returns 1 otherwise returns 0.

~ x converts binary 1 to 0 and vice versa (inversion).

Addition of Complex numbers – complex numbers are made up of a real and imaginary component, when adding complex numbers, add real components and imaginary components separately.

5. Making Decisions

The **if … else** statement is used to control the flow of logic in a program

```
if (logical expression):
statement(s)
else:
statement(s)
```

Comparison operators are used in the logical expression. Comparison operators are listed on page 55 of the textbook.

The **if-elif-else** statement is used to execute code based on several conditions that are related, e.g if the marks in a class are compared and different statements are to be executed based on the range in which a particular mark falls.

Logical operators – and, or, not are used to combine if statements, e.g. if the test mark is < 50 and the assignment mark is < than 50 then execute statements 1 and 2.

Comparison operators can also be chained to reduce if statements, e.g.

if 1<= m <=10 which means if m is between 1 and 10, including the numbers 1 and 10.

6. Loops

Loops are used to execute the same statements more than once.

The **while loop**

```
        while expression :
statement1
statement2
statement3
```

expression represents a condition that must be checked every time the statements in the loop are executed. Once the expression is true, the loop will end. The comparison operators discussed above are used in the expression. Alternatively, the **break statement** can be used in an if … else statement in the loop to end the loop.

The **continue statement** is used to skip part of the code in the loop. If the continue statement is executed, the remained of the statements below it are skipped.

The **pass statement** is used to represent an empty block of code. It tells the program to do nothing for the condition in which the pass statement is.

The range () function is used with the loop for returning values of integers.

The for loop

```
for iterating_var in sequence:
        statement1
        statement2
        statement3
```

used to process sequences, e.g. a tuple containing many records.

The for loop can be used with the **membership operators**, **in** and **not in.**

The **choice ()** function is used with the for loop and picks a random item from the sequence.

Chapter 3 Sequences

1. Sequences

There are three types of sequences;

i. **Strings** which are sequences of Unicode characters, presented in Python in quotes i.e. 'Hello', "Hello", '"Hello"',"""Hello"". Strings are immutable—they cannot be modified after they are created.

ii. **Lists** are ordered sequences of values, presented in Python as ["John", "Kelly", 1, 2, [Sugar, Butter, 10]]. A list is mutable; you can append elements, remove existing elements, or rearrange elements.

iii. **Tuples** are ordered sequences of values, presented in Python in brackets i.e. (Tiger, Coffee, 1,'a',10.50, 'b'). Tuples are immutable—they cannot be modified after they are created.

2. Sequence operators

+ concatenates sequences.
* repeats sequences.
[ ] fetches a particular element from the sequence (indexing) or
a subset of elements from the sequence (slicing). The index is a number that represents an element in a string. All sequences start with an index of 0, i.e. the first element has index equal to zero.

3. String Methods and functions

A list of functions and methods that can be used with strings is given in the textbook on page 77 to 84. Familiarise yourself with the ones used in examples and assignments in the module.

A **function** is a part of the code that is called by name, e.g. min (), max () and len (), which does some computation on the attributes of an object and returns a value.

A **method** is a section of the code called by name that returns the object (or part of it) to which it is associated, e.g. str (), sorted () and capitalise ().

4. Arrays

Arrays are type of variable used to store numerical values. The location of values in the array is determined by indices (starting with index 0 for the first element).

One dimensional arrays consist of a list of values, similar to the strings and lists discussed above.

Two dimensional arrays consist of rows and columns. Each element in the array is identified by the number of the row and the number of the column. Thus element [0] [0] is the first element in the array, element [1][0] is the second element in the first column. Note the first index always refers to the row and the second index refers to the column.

5. Dictionary

A dictionary is a mutable combination of key/value pairs in which every key has to be unique.

```
d = {key1 : value1, key2 : value2 }
```

The dictionary is used in Python to store and retrieve values quickly and efficiently, e.g. you may you it to store the exchange rate for currencies, where your key is values “Euro”, “Rand”, “Pound” etc. The value would then be the exchange rates of the above currencies against the dollar for example.

Dictionary methods are listed on page 97 of the textbook.

6. Sets

A set is a collection of values on which the following operations can be performed;

union (|) - an element appears in the union if it exists in one set or the other.

intersection (&) - the elements that appear in both sets appear in the intersection.

difference (-) - all the elements that are in the left set but not in the right set
will appear in the difference operation.

Sets are used in Python to manipulate groups of data, e.g. students’ marks from two modules. You would use the intersection operation to find students who wrote both module tests (all students appearing in both sets), the union operation to find all students who wrote both module tests (all students in the two sets).

Sets methods are listed on page 101 of the textbook.

Chapter 4 Functions and Modules

1. Functions

A **function** is a part of the code that is called by name (as defined in chapter 3), the code calling the function is called a function call. Refer back to chapter 3 for more features of a function.

When defining a function in Python, the following syntax is used:

```
def function-name(parameters):
statement(s)
```

where parameters are the values the function will receive from the function call and statements are the tasks the function will perform.

For example, to create a function that calculates interest on a given amount for a given period, the function would be defined as follows:

```
Def calc_interest (principal_amt, num_months):
        int_rate = 11
        Interest_amt = (principal_amt * num_months * int_rate)/100
        return interest_amt
```

Note that the principal_amt and the num_months in the function definition are called **parameters.**

The function call would be calc_interest (10000, 24), note that 10000 the principal amount and 24 the period, the **arguments** can be variables instead of values.

In the function call, number of **arguments** must always equal number of **parameters**.

Default value parameters are optional parameters that do not have to be supplied in the arguments.

The calc_interect function above can be defined with a default value parameter as shown below.

```
Def calc_interest (principal_amt, num_months, int_rate = 11):
        Interest_amt = (principal_amt * num_months * int_rate)/100
        return interest_amt
```

Then the function call can be;
calc_interest (1000, 24) in which case 11 will be used as the interest.

Or it can be calc_interest (1000,24,12) in which case the interest will be 12.

In the above examples the order of the arguments must be the same as the order of the parameters. If **keyword parameters** are used, the order of arguments and parameters is no longer important. Keyword parameters can be used by defining **global parameters** as described on page 109 of the textbook. The parameters defined in the examples above are **local** to the function and can only be used within the function.

Lambda functions are single use functions that do can take any number of parameters, they do not have a name.

2. Applying Functions to Sequences

For this module were are going to apply the following functions to sequences

i. filter function – select elements from a sequence that satisfy the criteria defined in the function

filter (function,sequence) where the function specifies the criteria.

ii. The map function – calls the given function for all elements in a sequence and returns a list of the returned values.

map (function,sequence)

iii. The reduce function – calls the given function using the first and second element, then the result of the first call and the third element are used to make the next call and that result and the fourth element are used to make the next call until the end of the sequence.

reduce (function, sequence)

3. Function Attributes

Note the attributes of a function as described on page 115 of the textbook.

4. Recursion

Recursion occurs when a function calls itself. To ensure that the call does not result in an endless loop, an exit condition must be included in the function, through the use of a the return statement.

5. Iterators

Iterators are used to loop through data, an iter object is created as follows:

Iter (object)

The next () method of the object is then used to loop through elements in the original object.

6. Generators

A generator is a function that returns a value using the yield statement. The generator iterator can be called using the _next_ () method.

7. Generator Expression

Is created by enclosing an expression in parenthesis. The expression will contain either an if statement or a for loop.

e.g. (squarenum(x) for x in range(6))
The _next_() method can be used to loop through the elements in the object.

8. Modules

A module is a file containing several functions that can be imported into a program using the import statement.

A particular function in a module can be imported as follows:

from module import function

Or all the functions in a module can be imported using:

```
from module import *
```

If a different name is required for the module in the current program it can be imported as follows:

```
import module as newname
```

9. The math Module

The math Module contains common trigonometric functions as well as mathematical constants. It is imported as follows:

```
import math
```

10. The dir () function

The dir () function is used to display the identifiers (classes, functions and variables) in a module.

11. Command-line arguments

Command-line arguments are used to pass arguments to a program during execution. They are stored in the sys.argv variable

Chapter 5 Classes

1. The class statement

A class is a template or blueprint for data and operations. It consists of the following:

i. Data members – consisting of class variables (outside of any method of the class) and instance variables which are inside a method.

ii. Member functions – operations that can be applied to the data members.

A class is created using the following syntax:

```
class classname (base-classes):
statement(s)
```

The **base-class** above is the class from which the current class inherits from, it is also called a **super class**.
The inheriting class is called a **derived class** or a **subclass**.
The statements are called the body of the class.

The **attributes of a class** are objects that are contained in the body of the class (as defined in the statements) or are **built-in attributes**. Refer to page 132 of the textbook for built-in class attributes.

2. Defining functions in a class

Functions defined in a class are known as methods and they always start with the parameter named self. These methods are called instance methods

Instance methods are defined as follows:

```
class classname(base-classes):
class variable(s)
def method 1(
statement(s) self):
instance variable(s)
```

3. Accessing Class Variables in Instance Methods

To access class variables, the methods defined in a class body must use a fully qualified name; the class object must be fully prefixed with the class variables.

4. Instances

An instance is a variable that acts as a replica of a class and an unlimited number of instances can be created and each gets a separate copy of the methods and attributes defined in the class.

To create an instance of a class called classname, you call the class object as if it were without parameters as shown below:

```
r=classname()
```

The _ _init_ _() method is used to initiate the class.

Python allows for the assigning of one instance to another.

```
Inst1 = inst2
```

This will create the instance if it does not exist and all the variables in inst1 will be set to the values in inst2.

Arguments can be passed to the __init__ using the following statement def __init__ (self, x,y)

5. Class Methods

A class method has no self argument and receives a class (called cls) as its first argument.
The class method is defined as follows:

```
@classmethod
Def f (cls, parm1, parm2, ….):
    body of method
```

where warm1, parm2 … are parameters of the object.

6. Static Methods

Static methods are ordinary functions that bind their results to a class attribute. It does not have a self or cls parameter and is automatically inherited by any child classes. The static method is immutable via inheritance.

The static method is defined as follows:

```
@staticmethod
Def name        (parm …):
        body of the method
```

7. Garbage Collection

Garbage collection is a procedure for freeing up the memory that is used by the variables or instances that are no longer required.
Python uses a reference count, any object whose reference count is zero is garbage collected.

8. Inheritance

Inheritance is a technique of copying the data members and member functions of an existing class into another class.
The members in a class can be defined as:
Public – accessed from within or outside of the class.
Private – can be accessed from outside of the class.

Member functions in the inheriting class that have the same name as the base class' function will override the base class' function, i.e. the function in the sub class will be executed. This is called **method overriding.**

The methods of a base class can be called from the sub class by using fully qualified method names, i.e. base_classname.methodname.

i. Single inheritance – one class is derived from another single class, implemented as follows:
Class sub-class-name (base-class-name)
ii. Multilevel inheritance – when a class inherits a class that in turn is inherited by some other class.
iii. Multiple Inheritance – one class inherits from more than one base class. If two classes with the same function name are inherited from, the function from the first base class will be executed.

9. Operator overloading

Overloading is when arithmetic operators are used to perform operations on classes.

The comparison operator == can be used to determine if two instances have instance variables with the same value.

Polymorphism allows different methods with the same name in different classes to perform different tasks.

Properties are used to manage attributes with get **and** set **methods.**

10. Descriptors

Descriptors are used to manage instance attributes. There are two types of descriptors;

i. **Non-data descriptors – implements the _get_ method.**
ii. **Data descriptor –** implements the _delete_ , _set_ and _get_ methods.

The __setattr__ method is called whenever you assign a value to an instance variable
The __getattr__method fetches an attribute of an instance using a string object and is called when attribute look up fails, that is, when you access an undefined attribute.

Chapter 6 File Handling

1. File Handling

A file is a container for a sequence of data objects, represented as sequences of bytes. The following are the three types of files in Python;

Text files are encoded and stored in a format that is viewable by many programs as well as people. Text files are difficult to update in place.

Binary files are formatted to optimize processing speed. A binary file will typically place data at known offsets, making it possible to access any particular byte using the seek() method.

Pickled files are formatted to store objects. The objects are stored in binary format to optimize performance.

Opening a file

```
Open (file_name, mode)
```

The mode specified determines the actions that will be taken when the file is opened e.g. reading, writing, appending etc. Note the different modes on page 174 of the textbook.

Once the file is opened, several methods (actions) can be invoked including write, read, close etc. check the full list of methods in the textbook on page 175 and 185.

2. Exception Handling

Exception handling is used in programing to catch errors that may occur whilst a program is executed so that an error that can be understood by a user is displayed.

The try/except statement is used to execute code in the try block, should an error (exception) be encountered, the code for handling the exception is executed (usually information displayed for the user), the else statement(s) are executed when the try statements do not raise an error.

```
try:
        statement(s)
except SomeException:
        code for handling exception
[else:
        statement(s)]
```

The try/finally statement is used to handle an error by executing code in the finally statement. The code in the finally statement is executed regardless of whether the statements in the try section are successful or not. This statement is usually used to ensure that some action is taken whether the try statement(s) succeed or not, e.g. closing a file or collecting garbage

```
try:
        statement(s)
finally:
```

```
statement(s)
```

3. Raising an exception

The exceptions raised by Python include execution (runtime) errors such as dividing by 0 or trying to open a file that does not exist etc. if for example you are writing code for a business rule that says a manager can only travel once a month on business trips, and you want to raise an exception when a second trip is captured, then the following code can be used to raise an exception;

```
try:
    if condition:
        raise customException, statement for customException
except customException, e:
    statements for customException
```

The if condition would in this case check if the manager has travelled already in the current month.

4. The Assert Statement

The assert statement is used to check that the values in variables are correct, e.g. a numerical value is entered and not a string. The syntax of the statement is as follows;

```
Assert (variable comparison)
```

Chapter 7 PyQt

1. Qt tool kit

The Qt tool kit is a framework for developing **user interfaces (UI)** using a **graphical user interface (GUI)**. This means you can develop forms by dragging and dropping objects such as buttons, text boxes etc.

2. PyQt

PyQt is a version of Qt that works with Python. PyQt is installed by configuring it to work with the version of Python installed on your workstation. Thus it is important to ensure that the version of Python installed is compatible with the version of PyQt you want to install. Additionally the configuration should take into account the folders in which both Python and PyQt are installed. Where possible, an installation package .exe file should be used to install both PyQt. Usually in the installation package, all the necessary configurations have been included. For this course, a separate installation instruction manual is provided.

3. Creating a GUI Application with Code

The code required to create a GUI application can be written without using a drag and drop **integrated development environment (IDE**) such as Qt (see example on page 203 of the textbook).

Note that the interpreter for GUI programs is **Pythonw.exe**, and because of that, GUI programs are saved with extension .**pyw**, compared to **.py** for the console (command line) programs which are interpreted by **Python.exe**.

4. Using Qt Designer

Qt designer is the GUI developer that comes with PyQt.

The objects in QT are called widgets. Qt uses widgets to define the top level class in the program created, i.e. your program will inherit from this class. The following are the three top level widgets used; **QDialog, Qwidget and QMainWindow.** Each of these are used as options for templates for the creation of a new application.

Qt is made up of the following main windows;

Object Inspector - Displays a hierarchical list of all the objects present on the form.

Property Editor - to view and change the properties of the form and widgets.

Resource Browser – used to manage resources like images, audio, video, etc.

Widget Box - displays a categorized list of widgets and other objects that you can use for designing a user interface.

Signal/Slot Editor - displays the signal/slot connections between objects.

Action Editor - used to manage the actions of your applications.

5. The Widgets

Found in the Widget window, the widgets are the objects used to design GUIs. The following are the different categories of widgets in Python and a brief description of the features and/or uses;

Layouts are used for arranging and sizing the widgets in a desired manner.

Spacers are used for inserting spaces between widgets or groups of widgets.

Buttons are used to initiate an action. Most of the coding in the applications are placed in the buttons so that the program can react to users when they click the buttons.

Item Views widgets are used for displaying large volumes of data. Item views may be model based or item based.

Container widgets are used to control a collection of objects on a form.

Input Widgets are for used for interacting with the user. These are used to collect input from the user into the program for processing.

Display widgets are used for displaying information or messages to the user.

Phonon is a multimedia (application programming interface) API that provides an abstraction layer for capturing, mixing, processing, and playing audio and video.

6. Toolbar

The toolbar has icons used to pass commands to the IDE.

7. Event Handling in PyQt

An event in this context occurs when something happens in the program in execution (run) state, for example the user may click a button or press enter or enter text or a certain time or date may be reached. The program would be designed to handle (react and do something) when the appropriate event occurs, depending on the purpose of the program.

In PyQt, the event-handling mechanism is also known as **signals and slots.** Every widget emits signals when its state changes.

To perform a task in response to a signal, the signal has to be connected to a slot, which is the method containing the code that you want to be executed on occurrence of a signal. Some objects have predefined slots and these can be connected to signals in QT designer. To do this, the signals and connections mode of Qt should be selected.

Custom slots can be developed in Python by writing a method and calling it when a widget sends a signal, e.g.

```
QtCore.QObject.connect(self.ui.ClickMeButton, QtCore.SIGNAL('clicked()'),self.dispmessage)
```

Connects the signal sent by ClickMeButton to the method dispmessage. See the textbook page 231 to 232 for the complete program.

8. Converting Data Types

The widgets that collect input from users, e.g. text boxes collect string data. You may want to use this data as numerical data in your program.

The methods below can be used to convert the data to numerical (int()) data for calculations in the program and back to string data (int()) for displaying on the UI.

int(): Converts the passed argument into integer data type.
str(): Converts the passed argument into string data type.

9. Defining Buddies

Buddies are used to make your program user friendly, so that users may use shortcut keys such as ctrl + I for example to navigate to a particular widget in the program.

Label widgets can create short cut keys if & is placed before a letter in the text of the label. In the Buddy editing mode, other widgets, e.g. a text box can then be connected to the label widget, allowing the user to use the label's shortcut to go to the text box.

10. Setting Tab Order

The tab order is the order in which the widgets in the program will get focus (the cursor) as the user type and presses the tab button. If a user is entering First Name, Surname and Date of Birth on a form, if the tab order is not set up correctly, after typing the First Name, the cursor may go to Date of Birth before the user can type Surname. The default tab order is the order in which the widgets are added to the form.

To set the tab order in Python, select Tab Order editing mode on the toolbar in Qt and change the numbers next to the widgets so that the sequence is what you want the order to be when the user is running the program.

Chapter 8 Basic Widgets

1. Using Radio Buttons

Radio buttons are used to allow a user to select only one option from two or more options.

Radio buttons provide the following methods;
isChecked() - returns true if the button is in selected state.

setIcon() - used to display an icon with the radio button.

setText() - used to set the text of the radio button. To specify a shortcut key for the radio button, precede the preferred character in the text with an ampersand (&).

setChecked() - pass the Boolean value true to this method to make the radio button the default.
Radio buttons emit the following signals;
toggled() - emitted whenever button changes its state from checked to unchecked or vice versa.

clicked() - emitted when a button is activated (i.e., pressed and released) or when its shortcut key is pressed.

stateChanged() - emitted when a radio button changes its state from checked to unchecked or vice versa.

2. Using Checkboxes

Check boxes allow the selection of more than one option.

Checkboxes provide the following methods;

isChecked() - returns true if the checkbox is checked; otherwise returns false.

setTristate() - pass Boolean value true to this method to use the “no change” state of the checkbox. With this state, you give the user the option of neither checking nor unchecking a checkbox.
setIcon() - used to display an icon with the checkbox.

setText() - used to set the text of the checkbox. To specify a shortcut key for the checkbox, precede the preferred character with an ampersand in the text.

setChecked() - pass Boolean value true to this method to make the checkbox checked by default.

Checkboxes emit the following signals;
toggled() - the signal is emitted whenever a checkbox changes its state from checked to unchecked or vice versa.

clicked() - the signal is emitted when a checkbox is activated (i.e. pressed and released) or when its shortcut key is typed.
stateChanged() - the signal is emitted whenever a checkbox changes its state from checked to unchecked or vice versa.

3. ScrollBars

Scrollbars are used for viewing documents or images that are larger than the view area.
There are two types of scrollbars, **HorizontalScrollBar** and **VerticalScrollBar**.

A ScrollBar has the following controls:

Slider handle: Used to move to any part of the document or image quickly.

Scroll arrows: arrows on either side of the scrollbars that are used to accurately navigate to a particular place in a document or image.

Page control: The page control is the background of the scrollbar over which the slider handle is dragged.

Scrollbars provide the following methods;

value() - retrieves a value that indicates the distance of the slider handle from the start of the scrollbar.

setValue() - sets the value of the scrollbar and hence the location of the slider handle in the scrollbar.

minimum() - returns the minimum value of the scrollbar.

maximum() - returns the maximum value of the scrollbar.

setMinimum() - sets the minimum value of the scrollbar.

setMaximum() - sets the maximum value of the scrollbar.

setSingleStep() - sets the single step value.

setPageStep() - sets the page step value.

Scrollbars emit the following signals

valueChanged() - emitted when the scrollbar's value is changed.

sliderPressed() - emitted when the user starts to drag the slider handle.

sliderMoved() - emitted when the user drags the slider handle.

sliderReleased() - emitted when the user releases the slider handle.

actionTriggered()Emitted when the scrollbar is changed by user interaction.

4. Sliders

Sliders are used to represent some integer value.
There are two types of Sliders, HorizontalSlider and VerticalSlider.

Sliders emit the following signals

valueChanged() - emitted when the scrollbar's value is changed.

sliderPressed() - emitted when the user starts to drag the slider handle.

sliderMoved() - emitted when the user drags the slider handle.

sliderReleased() - emitted when the user releases the slider handle.

The following methods are used to set positions of the sliders;

setTickPosition() - sets the position of tickmarks.

setTickInterval() - specifies the number of ticks desired.

tickPosition() - returns the current tick position.

tickInterval() - returns the current tick interval.

5. Working with a List Widget

The List widget is used to list items, the items can then be viewed, added to and removed from the list.

The list widget provides the following methods;

insertItem() - inserts a new item with the specified text into the List widget at the specified row.

insertItems() - inserts multiple items from a list of supplied labels, starting at the specified row.

count() - returns the count of the number of items in the list.

takeItem() - removes and returns item from the specified row in the List widget.

currentItem() - returns the current item in the list.

setCurrentItem() - replaces the current item in the list with the specified item.

addItem() - inserts an item with the specified text at the end of the List widget.

addItems() - inserts items with the specified text at the end of the List widget.

clear() - removes all items and selections in the view permanently.

currentRow() - returns the row number of the current item. If there is no current item, it returns the value -1.

setCurrentRow() - selects the specified row in the List widget.

item() - returns the item at the specified row.

List widget emits the following signals
currentRowChanged() - emitted whenever the row of the current item changes.

currentTextChanged() - emitted whenever the text in the current item is changed.

currentItemChanged() - emitted when the focus of the current item is changed.