**Object Oriented Programming in Python**

Object oriented programming (OOP) is the key to programming Python QT graphical user interfaces. This chapter will cover the basics of OOP in Python.

OOP consists primarily from classes and objects. Objects are instances of classes. Most important features of a class are data and methods. Data are usually just values stored in variables that correspond to that class, whereas methods are functions specified within a class, that give a behavior to objects from that class. In OOP it is common to name classes with the first letter capitalized while objects with names all lower case. All of this is much easier to understand by looking at examples shown below.

#First example

class Calculator: #class definition

#value1 and value2 are the data of this class

value1 = 0

value2 = 5

#add, look\_at\_value1 and look\_at\_value2 are the methods of this class

#the first argument of a method is a self argument, which is replaced with

#object’s name when it is used with an object

def add(self, value3):

self.value1 += value3

self.value2 += value3

return self.value1

def look\_at\_value1(self):

return self.value1

def look\_at\_value2(self):

return self.value2

#my\_calculator is an object created by calling the calculator() class

my\_calculator = Calculator()

my\_calculator.add(result = 111)

my\_calculator.add(result = 10)

print(my\_calculator.look\_at\_value1())

print(my\_calculator.look\_at\_value2())

The next example explains the use of class constructors and destructors. Class constructor is called automatically when an object is created. It can contain input arguments so the class is created with data inputted by the user. The following example shows a simple class declaration using class constructors and destructors.

#Second example

#class declaration

class Car:

value=111

#class constructor \_\_init\_\_ takes in one argument (color) besides self

def \_\_init\_\_(self, color):

self.color = color

print("the object was constructed")

#class destructor \_\_del\_\_

def \_\_del\_\_(self):

print("the object was destructed")

car = Car("blue") #object car is constructed with a value of “blue” for color

print(car.color) #outputs “blue”

#syntax for destructing the object

del car

# print(car.color) ---> Gives a: NameError: name 'car' is not defined

Before we proceed to Python Qt let’s look at the another property of OOP, inheritance. Inheritance is when an object or a class is based on another object (prototypal inheritance) or another class (class-based inheritance). A child object or class inherits parent’s behavior. The following example shows an example of inheritance:

#Example 3: Inheritance

#Parents and children

#Declaring class A

class A:

A\_value = 100

def A\_method(self):

print("Calling parent class A method")

#Declaring class B from class A (argument in class B(A))

class B(A): #B ---> derived CHILD class

def B\_method(self):

print("Calling parent class B method")

#creating an object c from child class B

c = B()

c.B\_method()

c.A\_method()