Modelado y Programación 2015-2 Documentation

Publicación 1.0

Marco N

05 de March de 2015

Índice general

Contenido:

CAPÍTULO 1

Introducción a Python

Este texto es un conjunto de información general sobre Python, se pretende que sea una referencia rápida para prácticas dejadas en clase. En ningún momento llega a sustituir la documentación oficial ni los libros sugeridos en la bibliografía del curso.

1.1 ¿Qué es Python?

Es un lenguaje de programación fuertemente tipado [1] y de tipado dinámico(las variables pueden cambiar de tipo en asignaciones durante la ejecución del programa).

Es un lenguaje "Orientado a objetos" en el sentido en que todo es tratado como uno, incluso las funciones aunque Python no impone la orientación a objetos como el paradigma principal.

1.2 Interprete

Para poder ejecutar nuestros archivo de python necesitamos de un interprete. Python nos ofrece uno que funciona de 2 maneras. Pasando parámetros y modo interctivo.

1.2.1 Pasando parámetros

De esta forma invocamos al interprete pasándole como parámetro nuestro arhivo a ejecutar

```
pyhton mi_programa.py
```

```
python3 mi_programa.py
```

La diferencia es que en nuestro sistema podemos tener asociado python a la versión 2 o 3 por lo que puede hacer falta especificar la versión del interprete con la cual queremos ejecutar nuestro código

1.2.2 Modo interactivo

El módo interactivo nos permite escribir código y ejecutarlo al momento sin necesidad de escribir en un archivo. Es muy útil cuando se necesita probar código.

[1] QtGui.Application.exec_

```
pyhton
```

```
python3
```

La diferencia es que en nuestro sistema podemos tener asociado python a la versión 2 o 3 por lo que puede hacer falta especificar la versión del interprete con la cual queremos ejecutar nuestro código

1.3 Hola Mundo

Aquí el ejemplo clásico del programa **Hola Mundo!**. No sé necesita más que una sola línea de código

```
print("Hola Mundo!")
Hola Mundo!
```

1.4 ¿Dónde está el main?

Los programas escritos en Python **no** se empiezan a ejecutar desde álgún metodo main, en cambio se empiezan a ejecutar desde la primera línea del archivo.

Aunque no exista un método main, para programas algo más complejos, se recomienda declarar uno.

```
def main():
  codigo...

main()
```

1.5 Tipos de Datos

1.5.1 Cadenas

```
variable = "Esta es una 'cadena' en Python"
variable = 'Esta es una "cadena" en Python'
variable = """Esta es una cadena de
varias
líneas"""
variable = str(234)
```

1.5.2 Números Enteros

```
variable = 234
variable = int(234.0)
variable = int("234")
```

1.5.3 Números Flotantes

```
variable = 234.0
variable = float(234)
variable = float("234.0")
```

1.5.4 Números Complejos

```
variable = 234j
variable = 234 + 456j
variable = complex(234,456)
```

1.5.5 Booleanos

```
variable = True
variable = False
```

1.6 Entrada y Salida

1.6.1 Entrada

```
variable = input()
```

1.6.2 Salida

```
print(objeto_a_imprimir)
```

1.7 Control de Flujo

1.7.1 For

```
for elemento in objeto_iterable:
  codigo...
else:
  codigo...
```

Si queremos iterar y no necesitamos de la variable podemos usar un guión bajo en la seccion de la variable del for

```
for _ in iterable:
  codigo...
```

1.7.2 While

```
while condicion:
  codigo...
else:
  codigo...
```

1.7.3 If

```
if condicion1:
  codigo...
elif condicion2:
  codigo...
else:
  codigo...
```

1.8 Selector

```
"cadena"[3] # 4to caracter
'e'
"cadena"[-1] # Último caracter
'a'
"cadena"[2:4] # Subcadena desde el 2do hasta el 4to
'de'
"cadena"[:3] # Todos los caracteres hasta el 3ro
'cad'
"cadena"[3:] # Todos los caracteres apartir del 4to
'ena'
"cadena"[::2] # Todos los caracteres impares
'cdn'
```

1.9 Módulos

Un módulo es un archivo que es importado usando alguna de las formas siguientes:

```
from modulo import *

from modulo import sqrt

import modulo
```

1.10 Paquete

Un paquete es una colección de módulos en directorios en cierta jerarquía.

```
from my_paquete.timing.danger.internets import modulo
```

1.11 Estructuras de Datos

1.11.1 Listas

```
lista = list()  # Constructor de lista vacía
lista = []  # Constructor de lista vacía
lista = [1, 2, 3, "a", "b", "c"]  # Constructor de lista con elementos
```

Se debe notar que las listas pueden tener elementos de varios tipos

1.12 Clases

```
class MiClase:
  def __init__(self, parametro):
    self.variable = parametro

  def mi_metodo(self):
    print(self.variable)
```

1.12.1 Métodos

```
def metodo(self, parametro1, parametro2=default):
  codigo...
```

Cuando se manda a llamar a un método se necesita pasar el objeto mismo. Por eso es necesario usar el parámetro *self*.

1.13 Herencia

Python2

```
class A(object):
  def __init__(self, variable):
    self.var1 = 1
    self.var2 = variable

class B(A):
  def __init__(self, param1, param2):
    super(A, self).__init__(param1)
    self.var3 = param2
```

Python3

```
class A:
  def __init__(self, variable):
    self.var1 = 1
    self.var2 = variable

class B(A):
  def __init__(self, param1, param2):
    super().__init__(param1)
    self.var3 = param2
```

1.14 Excepciones

1.14.1 Try

```
try:  # Código a ser monitoreado
  pass
except Exception as ex:  # Cacha la excepción y la da un nombre
  raise ex
else:  # Se ejecuta sólo si no se lanza una excepción en el bloque *try*
```

```
  pass
finally:  # Se ejecuta haya o no haya una excepción
  pass
```

1.14.2 Raise

Sirve para lanzar una excepción

```
raise Exception("Algo salió mal :(")
```

1.15 Preguntas frecuentes y cosas Interesantes

1.15.1 Retorno de las funciones

Las funciones siempre regresan algo. Si no se especifica el valor de retorno la función regresa el objeto **None**

1.16 Errores comunes

1.16.1 No se modifican las variables globales

Manera incorrecta de hacerlo

```
variable = 56

def mi_funcion():
  variable = 0

mi_funcion()
print(variable)
56
```

Se debe usar la palabra reservada *global* para modificar variables globales

```
variable = 56

def mi_funcion():
  global variable
  variable = 0

mi_funcion()
print(variable)
0
```

1.16.2 Argumentos mutables por defecto

Lo que escribiste

```
def encola(elemento, lista=[]):
  lista.append(elemento)
  return lista
```

Lo que esperas

```
mi_lista = encola(12)
print(mi_lista)
[12]

mi_otra_lista = encola(42)
print(mi_otra_lista)
[42]
```

Lo que pasó

```
mi_lista = encola(12)
print(mi_lista)
[12]

mi_otra_lista = encola(42)
print(mi_otra_lista)
[12, 42]
```

Solución

```
def encola(elemento, lista=None):
  if lista is None:
    lista = list()
  lista.append(elemento)
  return lista
```

1.16.3 Las variables de clase no se modifican

Lo que escribiste

```
class MiClase:
  var = 100

a = MiClase()
b = MiClase()
```

Lo que esperaste

```
a.var = 20
print(b.var)
20
```

Lo que pasó

```
a.var = 20
print(b.var)
100
```

Solución

```
MiClase.var = 20
print(b.var)
20
```

1.17 Pendientes de clase

1.17.1 ¿Qué ordenamiento usa Python?

Timsort [2], derivado de merge sort e insertion sort

1.17.2 Operador Ternario

```
a if condicion else b
```

1.17.3 Interprete necesario para usar PyQt

PyQt funciona tanto con Python2 como Python3 [3]

1.18 Práctica 1

- Lista los comandos de linux que puedan realizar las siguientes acciones:
  - Crear un archivo
  - item Borrar un archivo
  - item Crear un directorio
  - item Borrar un directorio
  - item Listar archivos y directorios
  - item Cambiar de directorio
  - item Imprimir el directorio actual
- Lista al menos 5 funciones que Python ofrece por defecto y escribe una pequeña descripción de lo que hacen.
- Hacer un programa que reciba números de uno por uno de la línea de comandos y cumpla con lo siguiente:
  - Si el número actual es el primer número que se lee entonces se guarda
  - Si el número actual es menor que los recibidos anteriormente se debe ignorar e imprimir los números anteriores que sí fueron agregados. Es decir, sólo se aceptan números en orden no decreciente.
  - Si se recibe la letra m se debe calcular la media aritmética de los números que sí fueron agregados y se debe de imprimir
  - Si se recibe una cadena vacía el programa debe terminar

[2] Wikipedia
[3] PyQt

CAPÍTULO 2

Estilo de código

2.1 Código explícito

2.2 Una declaración por línea

Mal

```
print ("uno"); print ("dos")
if x == 1: print("uno")

if comparacion_compleja and otra_comparacion_compleja:
  pass
```

Bien

```
print ("uno")
print ("dos")

if x == 1:
  print ("uno")

cond1 = comparacion_compleja
cond2 = otra_comparacion_compleja
if cond1 and cond2:
  pass
```

2.3 Propiedades privadas

Se utiliza un underscore

```
_variable_privada = "No me toques"
```

En realidad esto no es diferente de declararla sin el guión bajo. Es una convención entre programadores del lenguaje

2.4 Multiple retornos en una función

Intentar usar sólo uno para el regreso si funcinó correctamente

No Recomendado

```
if algo:
  return algo
elif algo2:
  return algo2
elif error1:
  return error1
else:
  raise Except("error2")
```

Recomendado

```
regreso = None
if algo:
  regreso = algo
elif algo2:
  regreso = algo2
elif error1:
  return error1
else:
  raise Except("error2")
return regreso
```

Se pueden usar varios para casos de errores o se tenga que avisar lo más pronto posible

2.5 Crear lista de la misma cosa

Se crea una lista donde cada entrada es el mismo objeto, esto significa que si el elemento es un objeto mutable, como una lista, si se modifica uno entonces se modificaran todos los demás.

No son copias. Es el mismo objeto en cada entrada de la lista

```
cuatro_nones = [None] * 4
cuatro_listas = [[]] * 4
```

2.6 Crear listas de tamaño N

Se crea una lista donde cada entrada es una copia diferente, así si modificas un objeto mutable o no mutable esto no repercute en los demás.

```
cuatro_listas = [[] for __ in range(4)]
numeros = [x for x in range(20)]
numeros_impares = [x for x in range(20) if x%2 != 0]
```

CAPÍTULO 3

Patrones de Diseño

Los patrones de diseño consisten en dar soluciones a problemas generales.

Un patrón es la descripción de la solución a un problema en cierto contexto

3.1 Factory(Fábrica)

El patrón **Factory** es uno de los patrones más usados en Java. Este patrón de diseño cabe dentro de la categoria de **creational pattern** ya que nos proporciona una de las mejores formas de crear un objeto

En este patrón nosotros creamos un objeto sin exponer la lógica al cliente y rifiriendonos al nuevo objeto usando una interfaz común.

3.1.1 Implementación

Vamos a crear una interfaz Shape y clases concretas que la implementen. A continuación definiremos una clase ShapeFactory.

Nuestra clase *FactoryPatternDemo* usará *ShapeFactory* para obtener objetos de tipo Shape. Le pasará información(CIRCLE / RECTANGLE / SQUARE) a ShapeFactory para obtener los objetos que necesita. s Paso 1 ++++++ Creamos nuestra interfaz

```
public interface Shape {
   void draw();
}
```

3.1.2 Paso 2

Creamos las clases concretas implementando la interfaz

Rectangle.java

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Square.java

```
public class Square implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Circle.java

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

3.1.3 Paso 3

Creamos nuestro objeto Fábrica(Factory) que genera nuestros objetos concretos basados en información proporcionada.

ShapeFactory.java

```
public class ShapeFactory {

   //Usa el método getShape para obtener objetos de tipo Shape
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

      } else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }

      return null;
   }
}
```

3.1.4 Paso 4

Usamos la Fábrica(Factory) para obtener los objetos de las clases concretas utilizando algún tipo de información como el tipo

FactoryPatternDemo.java

```
public class FactoryPatternDemo {
```

```
   public static void main(String[] args) {
      ShapeFactory shapeFactory = new ShapeFactory();

      //obtenemos un objeto tipo Circle.
      Shape shape1 = shapeFactory.getShape("CIRCLE");

      //llamamos al método draw de Circle
      shape1.draw();

      //obtenemos un objeto tipo Rectangle.
      Shape shape2 = shapeFactory.getShape("RECTANGLE");

      //llamamos al método draw de Rectangle
      shape2.draw();

      //obtenemos un objeto tipo Square.
      Shape shape3 = shapeFactory.getShape("SQUARE");

      //llamamos al método draw de circle
      shape3.draw();
   }
}
```

3.1.5 Paso 5

Verificamos la salida

```
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
```

3.2 Abstract Factory(Fábrica abstracta)

Los patrones Abstract Factory trabajan alrededor de una super-fábrica que crea otras fábricas. Esta fábrica también es llamada fábrica de fábricas. Este tipo de diseño cae en la categoría de **creational patter** ya que nos proporciona una de las mejores formas de crear un objeto

En el patrón Abstract Factory una interfaz es la responsable de crear una fábrica de objetos relacionados sin especificarle sus clases. Cada fábrica generada puede proporcionar los objetos de acuerdo con el patrón Factory.

3.2.1 Implementación

Vamos a crear las interfaces Shape y Color y las clases concretas que las implementan. Creamos una clase de fábricas abstracta *AbstractFactory* en el siguiente paso. Definimos las clases Factory ShapeFactory y ColorFactory las cuales extienden Abstractactory. Y creamos una clase constructora/generadora FactoryProducer

AbstractFactoryPatternDemo usa FactoryProducer para obtener uno objeto de tipo AbstractFactory. Le pasará información (CIRCLE / RECTANGLE / SQUARE para Shape) a AbstractFactory para obtener el objeto que necesita. También le pasará información (RED / GREEN / BLUE para Color) a AbstractFactory para obtener el objeto que necesita.

3.2.2 Paso 1

Creamos la interfaz Shape

Shape.java

```
public interface Shape {
   void draw();
}
```

3.2.3 Paso 2

Creamos las clases concretas implementando la interfaz.

Rectangle.java

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Square.java

```
public class Square implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Circle.java

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

3.2.4 Paso 3

Creamos la interfaz Color

Color.java

```
public interface Color {
  void fill();
}
```

3.2.5 Paso 4

Creamos las clases concretas que implementan la interfaz.

Red.java

```
public class Red implements Color {

   @Override
   public void fill() {
      System.out.println("Inside Red::fill() method.");
   }
}
```

Green.java

```
public class Green implements Color {

   @Override
   public void fill() {
      System.out.println("Inside Green::fill() method.");
   }
}
```

Blue.java

```
public class Blue implements Color {

   @Override
   public void fill() {
      System.out.println("Inside Blue::fill() method.");
   }
}
```

3.2.6 Paso 5

Creamos una clase abstracta para obtener las fábricas de objetos tipo Color y Shape.

AbstractFactory.java

```
public abstract class AbstractFactory {
   abstract Color getColor(String color);
   abstract Shape getShape(String shape) ;
}
```

3.2.7 Paso 6

Creamos las clases Factory extendiendo de AbstractFactory para generar objetos de las clases concretas basados en información dada.

ShapeFactory.java

```
public class ShapeFactory extends AbstractFactory {

   @Override
   public Shape getShape(String shapeType){
```

```
      if(shapeType == null){
         return null;
      }

      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      }else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

      }else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }

      return null;
   }

   @Override
   Color getColor(String color) {
      return null;
   }
}
```

ColorFactory.java

```
public class ColorFactory extends AbstractFactory {

   @Override
   public Shape getShape(String shapeType){
      return null;
   }

   @Override
   Color getColor(String color) {

      if(color == null){
         return null;
      }

      if(color.equalsIgnoreCase("RED")){
         return new Red();

      }else if(color.equalsIgnoreCase("GREEN")){
         return new Green();

      }else if(color.equalsIgnoreCase("BLUE")){
         return new Blue();
      }

      return null;
   }
}
```

3.2.8 Paso 7

Creamos una clase Fábrica generadora para obtener fábricas pasandole información como Color o Shape

FactoryProducer.java

```
public class FactoryProducer {
   public static AbstractFactory getFactory(String choice){

      if(choice.equalsIgnoreCase("SHAPE")){
         return new ShapeFactory();

      }else if(choice.equalsIgnoreCase("COLOR")){
         return new ColorFactory();
      }

      return null;
   }
}
```

3.2.9 Paso 8

Usamos FactoryProducer para obtener AbstractFactory para obtener fábricas de clases concretas pasandoles información como el tipo

AbstractFactoryPatternDemo.java

```
public class AbstractFactoryPatternDemo {
   public static void main(String[] args) {

      //get shape factory
      AbstractFactory shapeFactory = FactoryProducer.getFactory("SHAPE");

      //get an object of Shape Circle
      Shape shape1 = shapeFactory.getShape("CIRCLE");

      //call draw method of Shape Circle
      shape1.draw();

      //get an object of Shape Rectangle
      Shape shape2 = shapeFactory.getShape("RECTANGLE");

      //call draw method of Shape Rectangle
      shape2.draw();

      //get an object of Shape Square
      Shape shape3 = shapeFactory.getShape("SQUARE");

      //call draw method of Shape Square
      shape3.draw();

      //get color factory
      AbstractFactory colorFactory = FactoryProducer.getFactory("COLOR");

      //get an object of Color Red
      Color color1 = colorFactory.getColor("RED");

      //call fill method of Red
      color1.fill();

      //get an object of Color Green
      Color color2 = colorFactory.getColor("Green");
```

```
      //call fill method of Green
      color2.fill();

      //get an object of Color Blue
      Color color3 = colorFactory.getColor("BLUE");

      //call fill method of Color Blue
      color3.fill();
   }
}
```

3.2.10 Paso 9

Verificamos la salida

```
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
Inside Red::fill() method.
Inside Green::fill() method.
Inside Blue::fill() method.
```

3.3 Composite

El patrón Composite es usado cuando necesitamos tratar un grupo de objetos en una manera similar a uno solo. El patrón compone objetos en términos de un árbol partes como la jerarquía completa. Este patrón de diseño cabe dentro de la categoria de **structural pattern** ya que crea una estructura de árbol a partir de un grupo de objetos.

Este patrón crea una clase que contiene grupos de sus mismos objetos. Esta clase proporciona maneras de modificar su grupo de mismos objetos.

Demostraremos el uso de este patrón con el siguiente ejemplo en el cual mostraremos la jerarquía de empleados en una organización.

3.3.1 Implementación

Tenemos la clase Employee que tiene el rol de la clase actor en el patrón composite. CompositePatternDemo, usará la clase Employee para añadir jerarquía entre los departamentos e imprimir todos los empleados.

3.3.2 Paso 1

Creamos la clase Employee con una lista de objetos tipo Employee.

Employee.java

```
import java.util.ArrayList;
import java.util.List;

public class Employee {
   private String name;
   private String dept;
   private int salary;
   private List<Employee> subordinates;
```

```
   // constructor
   public Employee(String name,String dept, int sal) {
      this.name = name;
      this.dept = dept;
      this.salary = sal;
      subordinates = new ArrayList<Employee>();
   }

   public void add(Employee e) {
      subordinates.add(e);
   }

   public void remove(Employee e) {
      subordinates.remove(e);
   }

   public List<Employee> getSubordinates(){
     return subordinates;
   }

   public String toString(){
      return ("Employee :[ Name : " + name + ", dept : " + dept + ", salary :" + salary+" ]");
   }
}
```

3.3.3 Paso 2

Usamos la clase Employee para crear e imprimir la jerarquía de empleados.

CompositePatternDemo

```
public class CompositePatternDemo {
   public static void main(String[] args) {

      Employee CEO = new Employee("John","CEO", 30000);

      Employee headSales = new Employee("Robert","Head Sales", 20000);

      Employee headMarketing = new Employee("Michel","Head Marketing", 20000);

      Employee clerk1 = new Employee("Laura","Marketing", 10000);
      Employee clerk2 = new Employee("Bob","Marketing", 10000);

      Employee salesExecutive1 = new Employee("Richard","Sales", 10000);
      Employee salesExecutive2 = new Employee("Rob","Sales", 10000);

      CEO.add(headSales);
      CEO.add(headMarketing);

      headSales.add(salesExecutive1);
      headSales.add(salesExecutive2);

      headMarketing.add(clerk1);
      headMarketing.add(clerk2);

      //print all employees of the organization
      System.out.println(CEO);
```

```
      for (Employee headEmployee : CEO.getSubordinates()) {
         System.out.println(headEmployee);

         for (Employee employee : headEmployee.getSubordinates()) {
            System.out.println(employee);
         }
      }
   }
}
```

3.3.4 Paso 3

Verificamos la salida

```
Employee :[ Name : John, dept : CEO, salary :30000 ]
Employee :[ Name : Robert, dept : Head Sales, salary :20000 ]
Employee :[ Name : Richard, dept : Sales, salary :10000 ]
Employee :[ Name : Rob, dept : Sales, salary :10000 ]
Employee :[ Name : Michel, dept : Head Marketing, salary :20000 ]
Employee :[ Name : Laura, dept : Marketing, salary :10000 ]
Employee :[ Name : Bob, dept : Marketing, salary :10000 ]
```

3.4 State

En el patrón el comportamiento de una clase cambia dependiendo de su estado. Este tipo de patrón de diseño es de tipo behavior.

Creamos objetos que representan varios estados y un objeto de contexto cuyo comportamiento varia cuando su objeto de estado varia.

3.4.1 Implementación

Vamos a crear la interfaz State definiendo una action y clase concretas implementando State. Context es una clase que contiene un State.

Usaremos Context y objetos de estado para demostrar el cambio de comportamiento en Context basado en el tipo de estado en el que se encuentra.

3.4.2 Paso 1

Creamos una interfaz

State.java

```
public interface State {
   public void doAction(Context context);
}
```

3.4.3 Paso 2

Creamos la clases concretas implementando la misma interfaz.

StartState.java

```
public class StartState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in start state");
      context.setState(this);
   }

   public String toString(){
      return "Start State";
   }
}
```

StopState.java

```
public class StopState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in stop state");
      context.setState(this);
   }

   public String toString(){
      return "Stop State";
   }
}
```

3.4.4 Paso 3

Creamos la clase Context.

Context.java

```
public class Context {
   private State state;

   public Context(){
      state = null;
   }

   public void setState(State state){
      this.state = state;
   }

   public State getState(){
      return state;
   }
}
```

3.4.5 Paso 4

Usa el contexto para ver el cambio de comportamiente cuando State cambias

StatePatternDemo.java

```
public class StatePatternDemo {
   public static void main(String[] args) {
      Context context = new Context();

      StartState startState = new StartState();
      startState.doAction(context);

      System.out.println(context.getState().toString());

      StopState stopState = new StopState();
      stopState.doAction(context);

      System.out.println(context.getState().toString());
   }
}
```

3.4.6 Paso 5

Verifica la salida

```
Player is in start state
Start State
Player is in stop state
Stop State
```

3.5 MVC

3.6 Fuentes

tutorialspoint

wikipediaState

CAPÍTULO 4

GUI (Graphic User Interface)

Es una interfaz gráfica que permite la comunicación entre un humano y un sistema

4.1 Callbacks, Signals y Slots

Los callback son apuntadores a funciones. Así que si queremos que una función en ejecución nos notifique acerca de algún evento le debemos pasar un apuntador a otra función(callback). La función en ejecución llama al callback cuando se requiera.

Con Signal y Slots una señal(Signal) es emitida cuando un evento en particular ocurre. Un Slot es una función que es llamada en respuesta a una señal determinada.

Una clase que emite una señal no sabe ni le importa quien recibe la señal, ni siquiera le importa si es recibida por alguien.

4.2 Qt

La instalación depende mucho del sistema operativo. Para windows y linux se recomienda usar la siguiente liga http://download.qt.io/archive/qt/

4.3 Instalación

La instalación depende mucho del sistema operativo. Para windows y linux se recomienda usar la siguiente liga http://download.qt.io/archive/

4.4 Main event loop ó mainloop

En escencia no es nada más que un loop infinito que se ve más o menos así:

```
def main_loop():
  while True:
    event = wait_for_event()
    event.process()

    if main_window_has_been_destroyed():
      break
```

4.5 GENERAL NOTES

1. Qt4 and Qt5 developer tools are co-installable thanks to the qtchooser tool. See 'man qtchooser' for more information.

CAPÍTULO 5

PyQt4

5.1 Introducción

PyQt4 es un conjunto de herramientas para crear aplicaciones con interfaz gráfica. Es una mezcla entre Python y la biblioteca Qt. Qt library is one of the most powerful GUI libraries. Es desarrollado por la compañía Riverbank Computing Limited.

PyQt4 está implementado como un conjunto de módulos de Python. Cuenta con 440 clases y 6000 funciones. Es un conjunto de herramientas multiplataforma que corre en los sistemas operativos más importantes, incluyendo Unix, Windows, y Mac OS.

5.1.1 Clases

Las clases de PyQt4 están divididas en varios módulos:

- QtCore
- QtGui
- QtNetwork
- QtXml
- QtSvg
- QtOpenGL
- QtSql

QtCore

Contiene las las funcionalidades principales que no tienen que ver con la interfaz gráfica. Se usa para trabajar con tiempo, archivos y directorios, varios tipos de datos, flujos, urls.

QtGui

Contiene los componentes gráficos y sus clases relacionadas. Por ejemplo botones, ventanas, barras de estado, barras de herramienta, deslizadores, mapas de bit, colores y fuentes.

QtNetwork

Contiene las clases para programación en redes. Estas clses facilitan la programanción de clientes y servidores de tipo TCP/IP y UDPT haciendo que sea más fácil y más portable nuestor código.

QtXml

Contiene clases para trabajar con archivos XML.

QtSvg

Nos provee de claes para desplegar contenido de archivos SVG. Scalable Vector Graphics (SVG) es un lenguage para describir gráficos bidimensionales y aplicaciones gráficas en XML.

QtOpenGL

Es usado para mostrar gráficos en 3D y 2D usando la biblioteca OpenGL.

QtSql

Nos provee de clases para trabajar con bases de datos.

5.2 Requisitos

Para poder usar PyQt4 se necesitan al menos lo siguiente: - Qt versión 4 - PyQt4 para la versión de python que estén usado

http://www.riverbankcomputing.com/software/pyqt/download

5.3 Hola Mundo!

```
import sys
from PyQt4 import QtGui

def main():
  app = QtGui.QApplication(sys.argv)

  etiqueta_hola = QtGui.QLabel("Hola Mundo!")
  etiqueta_hola.show()

  return app.exec_()

if __name__ == '__main__':
  sys.exit(main())
```

Surgen algunos pensamientos:

- ¿Qué hace *app.exec_()*? [1]

[1] QtGui.Application.exec_

- No veo ningún ciclo. ¿Por qué la aplicación no termina? [2]
- ¿No necesitamos una ventana para mostrar cosas? [3] [4]

5.4 Ejemplo sencillo

```
import sys
from PyQt4 import QtGui

def main():
  app = QtGui.QApplication(sys.argv)

  widget = QtGui.QWidget()
  widget.resize(250, 150)
  widget.move(300, 300)
  widget.setWindowTitle('Simple')
  widget.show()

  return app.exec_()

if __name__ == '__main__':
  sys.exit(main())
```

5.5 Botones

QtGui.QPushButton

```
import sys
from PyQt4 import QtGui


class Example(QtGui.QWidget):

  def __init__(self):
    super().__init__()
    self.initUI()

  def initUI(self):
    boton1 = QtGui.QPushButton('Botón1', self)
    boton2 = QtGui.QPushButton('Botón2', self)
    boton1.move(50, 50)
    self.setGeometry(300, 300, 200, 150)
    self.setWindowTitle('Boton')
    self.show()

def main():
  app = QtGui.QApplication(sys.argv)
  ex = Example()
  sys.exit(app.exec_())
```

[2] main event loop
[3] QWidget PyQt
[4] QWidget Qt

```
if __name__ == '__main__':
  main()
```

5.6 Cerrando una ventana

```
import sys
from PyQt4 import QtGui, QtCore


class Example(QtGui.QWidget):

  def __init__(self):
    super().__init__()
    self.initUI()

  def initUI(self):
    boton1 = QtGui.QPushButton('Quit button', self)
    boton2 = QtGui.QPushButton('Botón2', self)
    boton1.clicked.connect(QtCore.QCoreApplication.instance().quit)
    boton1.move(50, 50)
    self.setGeometry(300, 300, 200, 150)
    self.setWindowTitle('Boton')
    self.show()

def main():
  app = QtGui.QApplication(sys.argv)
  ex = Example()
  sys.exit(app.exec_())


if __name__ == '__main__':
  main()
```

5.7 Entrada de texto y etiquetas

QtGui.QLineEdit

QtGui.QLabel

```
import sys
from PyQt4 import QtGui, QtCore


class Example(QtGui.QWidget):

  def __init__(self):
    super().__init__()
    self.initUI()

  def initUI(self):
    self.entrada_texto = QtGui.QLineEdit(self)
    self.resultado_lbl = QtGui.QLabel('Resultado:', self)

    self.entrada_texto.move(0, 0)
```

```
        self.resultado_lbl.move(0,40)

        self.setGeometry(200, 200, 400, 300)
        self.setWindowTitle('Mostrar y recibir texto')
        self.show()

def main():
  app = QtGui.QApplication(sys.argv)
  ex = Example()
  sys.exit(app.exec_())


if __name__ == '__main__':
  main()
```

5.8 Widgets

5.8.1 Mostrar texto

QtGui.QLabel

5.8.2 Recibir texto

QtGui.QLineEdit

5.9 Fuentes

```
fuente.setPixelSize(20)
fuente.setUnderline(True)
fuente.setBold(True)
fuente.setItalic(True)

fuente = QtGui.QFont()
fuente.setPixelSize(20)
QLabel("Hola").setFont(fuente)

etiqueta = QLabel("Hola")
fuente = etiqueta.font()
fuente.setPixelSize(20)
fuente.setFamily("Courier New")
etiqueta.setFont(fuente)
```

5.10 Layouts

5.10.1 Posición absoluta

Se espicifica a mano la posición y el tamaño de cada widget en pixeles. Cuando usamos posición absoluta debemos tener en cuenta las siguientes limitaciones:

- El tamaño y posición del widget no cambia si cambiamos el tamaño de la ventana.
- La aplicación puede cambiar de apariencia dependiendo de la plataforma
- Cmabiar la fuente la aplicación puede echar a perder la disposición de los elementos

```
widget.move(x, y)
widget.setGeomtry(x, y, anchura, altura)
```

5.10.2 Box layout

QtGui.QVBoxLayout y QtGui.QHBoxLayout

```
import sys
from PyQt4 iport QtGui

class Ejemplo(QtGui.QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()

    def initUI(self):

        boton_ok = QtGui.QPushButton("OK")
        boton_cancelar = QtGui.QPushButton("Cancelar")

        fuente = QtGui.QFont()
        fuente.setPixelSize(40)
        boton_ok.setFont(fuente)

        hbox = QtGui.QHBoxLayout()
        hbox.addWidget(boton_ok)
        hbox.addStretch(1)
        hbox.addWidget(boton_cancelar)

        vbox = QtGui.QVBoxLayout()
        vbox.addStretch(1)
        vbox.addLayout(hbox)

        self.setLayout(vbox)

        self.setGeometry(300, 300, 300, 150)
        self.setWindowTitle('Buttons')
        self.show()

def main():
    app = QtGui.QApplication(sys.argv)
    ex = Ejemplo()
    sys.exit(app.exec_())
```

5.10.3 QtGui.QGridLayout

```
grid = QtGui.QGridLayout()
self.setLayout(grid)
grid.addWidget(widget, fila, columna)
```

Cosas importantes sobre los layouts y los padres:

- Cuando se usen layouts, no se necesita pasar el padre como parámetro al construir un widget. El layout va a asignarle como padre(usando QWidget.setParent()) el widget al que pertenece.
- Los Widgets en un layout son hijos del widget al que pertenece el layout, no son hijos del layout. Los widgets sólo pueden tener como padres otros widgets, no layouts.
- Se pueden anidar layous usando addLayout() en un layout. El layout interior se vuelve hijo del layout en el que fue insertado.

5.11 Signals y Slots

Para que una clase pueda contener Signals y Slots debe de heredar de *QObject* o una de sus subclases. Cuando se emite una señal los slots conectados a esta usualmente se ejecutan inmediatamente, como si se hubiera mandado a llamar a cada una en vez de emitir la señal.

La ejecución del código que sigue a la emisión de una señal se realiza después de que todos los slots hayan hecho su *return*, es decir al emitir una señal esta bloquea la ejecución del método hasta que las funciones que estén escuchando esta señal hayan terminado de ejecutarse.

Si muchos slots están conectados a la misma señal se ejecutaran en el orden en el que fueron conectadas.

```
mi_senial = QtCore.pyqtSignal(int, str)
mi_senial.emit()
mi_senial.connect(objeto.funcion)
```

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
  mi_senial = QtCore.pyqtSignal(Objeto)
  def __init__(self):
    super().__init__()
    self.initUI()

  def initUI(self):
    self.entrada_texto = QtGui.QLineEdit(self)
    self.resultado_lbl = QtGui.QLabel('Resultado:', self)

    self.entrada_texto.move(0, 0)
    self.resultado_lbl.move(0,40)

    self.entrada_texto.textChanged.connect(self.calcula_suma)

    self.setGeometry(200, 200, 200, 150)
    self.setWindowTitle('Boton')
    self.show()

  def calcula_suma(self, entrada):
    resultado = "Error"
    print(entrada)
    try:
```

```
      resultado = "Resultado: " + str(eval(entrada))
    except Exception as e:
      print(e)
    self.resultado_lbl.setText(resultado)
    self.resultado_lbl.adjustSize()

def main():
  app = QtGui.QApplication(sys.argv)
  ex = Example()
  sys.exit(app.exec_())
if __name__ == '__main__':
  main()

import sys
from PyQt4 import QtGui, QtCore

class Objeto:
  val = 99

class Ejemplo(QtGui.QWidget):
  mi_senial = QtCore.pyqtSignal(Objeto)
  def __init__(self):
      super().__init__()

      self.initUI()

  def initUI(self):

      boton_ok = QtGui.QPushButton("OK")
      boton_cancelar = QtGui.QPushButton("Cancelar")

      fuente = QtGui.QFont()
      fuente.setPixelSize(40)
      boton_ok.setFont(fuente)

      hbox = QtGui.QHBoxLayout()
      hbox.addWidget(boton_ok,0)
      hbox.addStretch(1)
      hbox.addWidget(boton_cancelar)

      vbox = QtGui.QVBoxLayout()
      vbox.addStretch(1)
      vbox.addLayout(hbox)

      self.setLayout(vbox)

      boton_ok.clicked.connect(self.emite)
      self.mi_senial.connect(self.procesa_senial)
      self.algo = Objeto()
      self.setGeometry(300, 300, 300, 150)
      self.setWindowTitle('Buttons')
      self.show()

  def emite(self):
    self.mi_senial.emit(self.algo)
  def procesa_senial(self, algo):
    print(algo, type(algo), algo.val)
```

```
def main():

    app = QtGui.QApplication(sys.argv)
    ex = Ejemplo()
    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

5.12 Ejemplo más complicado

5.13 Recomendaciones

- Recuerda que PyQt4 usa Qt4 por lo que si la documentación de PyQt4 no te satisface puedes ir a la documentación de Qt4
- Salvo tu Widget principal intenta que todos los widgets que construyas tengan un padre para evitar errores

CAPÍTULO 6

QML

6.1 Elementos

Lista extensiva de los elementos de QML

```
import sys
from PyQt4.QtCore import QUrl
from PyQt4.QtGui import QApplication
from PyQt4.QtDeclarative import QDeclarativeView

# Creamos una aplicación Qt y una vista QDeclarative
app = QApplication(sys.argv)
view = QDeclarativeView()
# Create an URL to the QML file
url = QUrl('calculadora.qml')
# Set the QML file and show
view.setSource(url)
view.setResizeMode(QDeclarativeView.SizeRootObjectToView)
view.setGeometry(100, 100, 400, 300)
view.show()
# Enter Qt main loop
sys.exit(app.exec_())
```

6.2 Hola mundo

Vamos a crear el tradicional *Hello World!*

```
import QtQuick 1.0

Rectangle {
    id: page
    width: 500; height: 200
    color: "lightgray"

   Text {
       id: helloText
       text: "Hello world!"
       y: 30
       anchors.horizontalCenter: page.horizontalCenter
       font.pointSize: 24; font.bold: true
```

```
    }
}
```

6.2.1 Explicación

Import

Necesitamos importar los tipos que necesitamos para el ejemplo. La mayoría de los archivos QML suelen hacer el import de los tipos que vienen por defecto en Qt(Rectangle, Image, ...) usando:

```
import QtQuick 1.0
```

Rectangle

```
Rectangle {
    id: page
    width: 500; height: 200
    color: "lightgray"
```

Declaramos el elemento raíz del tipo Rectangle. Es uno de los bloques de construcción válidos que puedes usar para crear aplicaciones en QML. Le damps un id para que podamos refirirnos a él más tarde. En este caso lo nombramos "page". También le añadimos un ancho, alto y prodiedades de sus colores. El Rectangle contiene muchas más propiedades (como 'x' y 'y'), pero estos se dejan con sus valores por defecto.

Text

```
Text {
    id: helloText
    text: "Hello world!"
    y: 30
    anchors.horizontalCenter: page.horizontalCenter
    font.pointSize: 24; font.bold: true
}
```

Añadimos un elemento Text como hijo del elemento raíz Rectangle y despliega el texto 'Hello world!'.

La propiedad y es usada para posicionar el texto verticalmente a 30 pixeles del tope de su padre.

La propiedad `anchors.horizontalCenter` se refiere al centro horizontal de un elemento. En este caso especificamos que nuestro elemento de texto debe de estar centrado horizontalmente con respecto al elemento page (Layouts basadas en anclas).

Las propiedades `font.pointSize` y `font.bold` están relacionadas con las fuentes y usan la notación de punto.

6.3 Eco

```
import QtQuick 1.0

  Rectangle {
    id: page
    width: 500; height: 200
    color: "lightgray"
```

```
    Text {
      id: texto
      text: input.text
      anchors.top: parent.top
      anchors.horizontalCenter: page.horizontalCenter
      font.pointSize: 24; font.bold: true
    }

    Rectangle{
      id: rect_input
      color: "white"
      anchors.top: texto.bottom
      anchors.bottom: parent.bottom
      anchors.left: parent.left
      anchors.right: parent.right
      TextInput{
        anchors.fill: parent
        id: input
        focus: true
      }
    }
  }
```

6.4 Componentes

Vamos a crear un programa para cambiar el color de un texto.

```
import QtQuick 1.0

Item {
   id: container
   property alias cellColor: rectangle.color
   signal clicked(color cellColor)

   width: 40; height: 25

   Rectangle {
       id: rectangle
       border.color: "white"
       anchors.fill: parent
   }

   MouseArea {
       anchors.fill: parent
       onClicked: container.clicked(container.cellColor)
   }
}
```

6.5 Calculadora

```
import QtQuick 1.0

Item {
```

```
    id: boton
    property alias texto: boton_texto.text
    signal clicked(string algo)

    width: 60; height: 60

    Rectangle {
        id: rectangle
        color: "#aaf"
        anchors.fill: parent
        smooth: true
        radius: 30
        Text {
         id: boton_texto
         text : ""
         anchors.centerIn: parent
         font.pixelSize: 40
        }
    }

    MouseArea {
        anchors.fill: parent
        onClicked: boton.clicked(boton.texto)
    }
}

import QtQuick 1.0

Rectangle {
  id: ventana
  signal calcula_resultado(string cadena)
  property string name: pantalla.text
  onNameChanged: console.log("Name has changed to:", name)

  color: "lightgray"
  anchors.fill: parent

  function muestra_resultado(text) {
      pantalla.text = text
  }

  Rectangle{
    id: rect_input
    height: 40
    color: "black"
    anchors.top: parent.top
    anchors.left: parent.left
    anchors.right: parent.right
    Text{
      id: pantalla
      anchors.centerIn: parent
      focus: true
      color: "#ccc"
      font.pixelSize : 40
    }
  }

  Grid {
```

```
        id: botones
        anchors.top: rect_input.bottom
        anchors.bottom: parent.bottom
        anchors.horizontalCenter: parent.horizontalCenter
        rows: 4; columns: 4; spacing: 3

        Boton { texto: "1"; onClicked: pantalla.text += texto }
        Boton { texto: "2"; onClicked: pantalla.text += texto }
        Boton { texto: "3"; onClicked: pantalla.text += texto }
        Boton { texto: "+"; onClicked: pantalla.text += texto }
        Boton { texto: "4"; onClicked: pantalla.text += texto }
        Boton { texto: "5"; onClicked: pantalla.text += texto }
        Boton { texto: "6"; onClicked: pantalla.text += texto }
        Boton { texto: "-"; onClicked: pantalla.text += texto }
        Boton { texto: "7"; onClicked: pantalla.text += texto }
        Boton { texto: "8"; onClicked: pantalla.text += texto }
        Boton { texto: "9"; onClicked: pantalla.text += texto }
        Boton { texto: "*"; onClicked: pantalla.text += texto }
        Boton { texto: "0"; onClicked: pantalla.text += texto }
        Boton { texto: "clr"; onClicked: pantalla.text = "" }
        Boton { texto: "="; onClicked: ventana.calcula_resultado(pantalla.text) }
        Boton { texto: "/"; onClicked: pantalla.text += texto }
    }

}

import sys
from PyQt4.QtCore import QUrl, QObject, pyqtSignal
from PyQt4.QtGui import QApplication
from PyQt4.QtDeclarative import QDeclarativeView

class Evaluador(QObject):
    evaluado = pyqtSignal(str)
    def evalua(self, expresion):
        resultado = "Error"
        try:
            resultado = str(eval(expresion))
        except Exception:
            print("'" + expresion + "'")
        self.evaluado.emit(resultado)

evaluador = Evaluador()

# Create Qt application and the QDeclarative view
app = QApplication(sys.argv)
view = QDeclarativeView()
# Create an URL to the QML file
url = QUrl('calculadora.qml')
# Set the QML file and show
view.setSource(url)
view.setResizeMode(QDeclarativeView.SizeRootObjectToView)
view.setGeometry(100, 100, 400, 300)

rootObject = view.rootObject()
evaluador.evaluado.connect(rootObject.muestra_resultado)
rootObject.calcula_resultado.connect(evaluador.evalua)

view.show()
```

```
# Enter Qt main loop
sys.exit(app.exec_())
```

CAPÍTULO 7

Indices y tablas

- *genindex*
- *modindex*
- *search*