

GUÍA 2

COLABORACIÓN DE CLASES

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.

Plantearemos un problema separando las actividades en dos clases.

Problema 1:

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositado.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco.

Luego debemos definir los atributos y los métodos de cada clase:

Cliente

atributos

nombre

monto

métodos

__init__

depositar

extraer

retornar_monto

Banco

atributos

3 Cliente (3 objetos de la clase Cliente)

métodos

__init__

operar

depositos_totales

CÓDIGO

```
class Cliente:

    def __init__(self,nombre):
        self.nombre=nombre
        self.monto=0

    def depositar(self,monto):
        self.monto=self.monto+monto

    def extraer(self,monto):
        self.monto=self.monto-monto

    def retornar_monto(self):
        return self.monto

    def imprimir(self):
        print(self.nombre,"tiene depositado la suma de",self.monto)

class Banco:

    def __init__(self):
        self.cliente1=Cliente("Juan")
        self.cliente2=Cliente("Ana")
        self.cliente3=Cliente("Diego")

    def operar(self):
        self.cliente1.depositar(100)
```

```
self.cliente2.depositar(150)

self.cliente3.depositar(200)

self.cliente3.extraer(150)


def depositos_totales(self):

total=self.cliente1.retornar_monto()+self.cliente2.retornar_monto()+self.cliente3.retornar_monto()

print("El total de dinero del banco es:",total)

self.cliente1.imprimir()

self.cliente2.imprimir()

self.cliente3.imprimir()


# bloque principal

banco1=Banco()

banco1.operar()

banco1.depositos_totales()
```

Primero hacemos la declaración de la clase Cliente, en el método `__init__` inicializamos los atributos nombre con el valor que llega como parámetro y el atributo monto con el valor cero:

```
class Cliente:

    def __init__(self,nombre):

        self.nombre=nombre

        self.monto=0
```

Recordemos que en Python para diferenciar un atributo de una variable local o un parámetro le antecedemos la palabra clave `self` (es decir nombre es el parámetro y `self.nombre` es el atributo):

```
self.nombre=nombre
```

El método que aumenta el atributo monto es:

```
def depositar(self, monto):  
    self.monto=self.monto+monto
```

Y el método que reduce el atributo monto del cliente es:

```
def extraer(self, monto):  
    self.monto=self.monto-monto
```

Lo más común para que otro objeto conozca el monto depositado por un cliente es la implementación de un método que lo retorne:

```
def retornar_monto(self):  
    return self.monto
```

Para mostrar los datos del cliente tenemos el método:

```
def imprimir(self):  
    print(self.nombre, "tiene depositado la suma de", self.monto)
```

La segunda clase de nuestro problema es el Banco. Esta clase define tres atributos de la clase Cliente (la clase Cliente colabora con la clase Banco):

```
class Banco:  
  
    def __init__(self):  
        self.cliente1=Cliente("Juan")  
        self.cliente2=Cliente("Ana")  
        self.cliente3=Cliente("Diego")
```

El método operar realiza una serie de depósitos y extracciones de los clientes:

```
def operar(self):  
    self.cliente1.depositar(100)  
    self.cliente2.depositar(150)  
    self.cliente3.depositar(200)  
    self.cliente3.extraer(150)
```

El método que muestra cuanto dinero tiene depositado el banco se resuelve pidiendo a cada cliente que retorne el monto que tiene:

```
def depositos_totales(self):
```

```
total=self.cliente1.retornar_monto()+self.cliente2.retornar_monto()+self.cliente3.retornar_monto()

print("El total de dinero del banco es:",total)

self.cliente1.imprimir()

self.cliente2.imprimir()

self.cliente3.imprimir()
```

En el bloque principal de nuestro programa en Python procedemos a crear un objeto de la clase Banco y llamar a los dos métodos:

```
# bloque principal

banco1=Banco()

banco1.operar()

banco1.depositos_totales()
```

En el bloque principal no se requiere crear objetos de la clase Cliente, esto debido que los clientes son atributos del Banco.

Problema 2:

Plantear un programa que permita jugar a los dados. Las reglas de juego son: se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "gano", sino "perdió".

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Dado y la clase JuegoDeDados.

Luego los atributos y los métodos de cada clase:

```
Dado

atributos

    valor

métodos

    tirar

    imprimir

    retornar_valor
```

JuegoDeDados

atributos

3 Dado (3 objetos de la clase Dado)

métodos

__init__

jugar

CÓDIGO

```
import random

class Dado:

    def tirar(self):
        self.valor=random.randint(1,6)

    def imprimir(self):
        print("Valor del dado:",self.valor)

    def retornar_valor(self):
        return self.valor

class JuegoDeDados:

    def __init__(self):
        self.dado1=Dado()
        self.dado2=Dado()
        self.dado3=Dado()
```

```
def jugar(self):

    self.dado1.tirar()

    self.dado1.imprimir()

    self.dado2.tirar()

    self.dado2.imprimir()

    self.dado3.tirar()

    self.dado3.imprimir()

    if self.dado1.retornar_valor()==self.dado2.retornar_valor() and
self.dado1.retornar_valor()==self.dado3.retornar_valor():

        print("Gano")

    else:

        print("Perdio")

# bloque principal del programa

juego_dados=JuegoDeDados()

juego_dados.jugar()
```

Importamos el módulo "random" de la biblioteca estándar de Python ya que requerimos utilizar la función randint:

```
import random
```

La clase Dado define un método tirar que almacena en el atributo valor un número aleatorio comprendido entre 1 y 6:

```
class Dado:

    def tirar(self):

        self.valor=random.randint(1,6)
```

Los otros dos métodos de la clase Dado tienen por objetivo mostrar el valor del dado y retornar dicho valor a otra clase que lo requiera:

```
def imprimir(self):
```

```
print("Valor del dado:",self.valor)
```

```
def retornar_valor(self):  
    return self.valor
```

La clase JuegoDeDados define tres atributos de la clase Dado, en el método `__init__` crea dichos objetos:

```
class JuegoDeDados:  
  
    def __init__(self):  
        self.dado1=Dado()  
        self.dado2=Dado()  
        self.dado3=Dado()
```

En el método `jugar` de la clase `JuegoDeDados` procedemos a pedir a cada dado que se tire, imprima y verificamos si los tres valores son iguales:

```
def jugar(self):  
    self.dado1.tirar()  
    self.dado1.imprimir()  
    self.dado2.tirar()  
    self.dado2.imprimir()  
    self.dado3.tirar()  
    self.dado3.imprimir()  
    if self.dado1.retornar_valor()==self.dado2.retornar_valor() and  
self.dado1.retornar_valor()==self.dado3.retornar_valor():  
        print("Gano")  
    else:  
        print("Perdio")
```

En el bloque principal de nuestro programa creamos un objeto de la clase `JuegoDeDados`:

```
# bloque principal del programa  
  
juego_dados=JuegoDeDados()
```


juego_dados.jugar()

Acotación

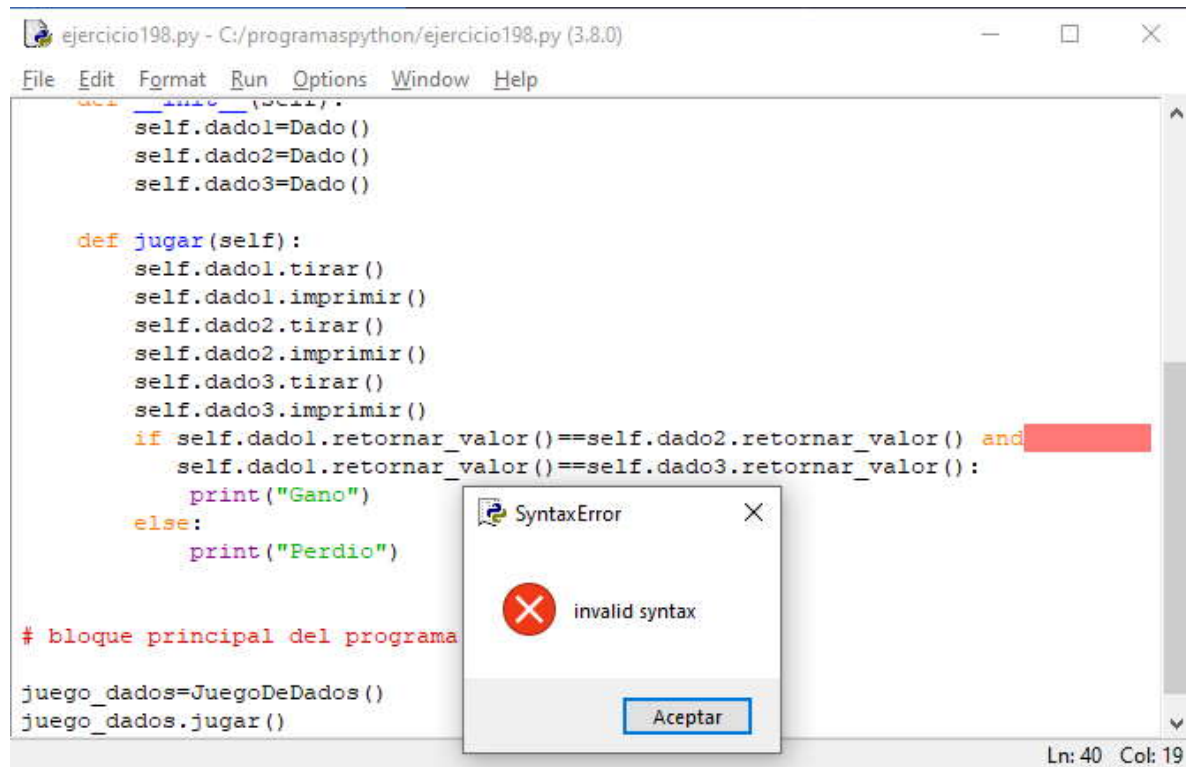
Para cortar una línea en varias líneas en Python podemos encerrar entre paréntesis la condición:

```
if (self.dado1.retornar_valor()==self.dado2.retornar_valor()
    and self.dado1.retornar_valor()==self.dado3.retornar_valor()):
```

O agregar una barra al final:

```
if self.dado1.retornar_valor()==self.dado2.retornar_valor() and \
    self.dado1.retornar_valor()==self.dado3.retornar_valor():
```

Si no utilizamos los paréntesis o la barra al final y tratamos de disponer el if en dos líneas se produce un error sintáctico:



Problema propuesto

Plantear una clase Club y otra clase Socio.

La clase Socio debe tener los siguientes atributos: nombre y la antigüedad en el club (en años).

En el método __init__ de la clase Socio pedir la carga por teclado del nombre y su antigüedad.

La clase Club debe tener como atributos 3 objetos de la clase Socio.

Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.