
PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

Clases y objetos

***OOP: Object Oriented
Programming***

PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

- Hasta el momento hemos modelado soluciones de la siguiente forma
 - Buscamos tipos de datos para la situación que estamos tratando: números, listas, diccionarios, etc.
 - Luego construimos en forma independiente funciones u operaciones para usar esos datos y obtener una solución

- Modelado de soluciones con POO: usado desde los principios de la década del 90, plantea un cambio en el planteamiento de la solución de problemas
 - No usa nuevas estructuras de control de programación
 - El enfoque es sobre la modelación de la solución: integra datos y su manipulación
 - Introduce los conceptos de: ***objetos y clases***

OBJETOS

■ Objeto

- Representación de una entidad del mundo real: automóvil, teléfono, estudiante, etc.
- Agrupa o encapsula en una sola unidad:
 - ✓ Características de la entidad: atributos (datos).
 - ✓ Funciones (métodos) para manipular los valores de esas características.
- Para trabajar con objetos primero hay que definir clases

CLASES

- Clase: definición de un objeto en términos de:
 - Atributos (datos)
 - Métodos (funciones)
- Clase: modelo a partir del cual el programador crea objetos de ella:
 - La clase es como el plano de construcción de una casa
 - El plano es único pero a partir de él se pueden crear múltiples casas

-
- Clases son nuevos tipos de datos:
 - TDA (Tipo de Dato Abstracto)
 - En POO, primero se definen las clases y con base en ellas se crean tantos objetos como se necesiten para solucionar el problema
 - Cada objeto creado es único
 - su propio nombre
 - tiene su propio espacio de memoria
 - sus propios valores

DEFINICIÓN DE CLASES EN PYTHON

- Una clase en Python está definida por un encabezado y un cuerpo

```
class Nombre_de_la_clase: # define encabezado
    definición de atributos # define cuerpo
    definición de métodos
```

- Por convención el nombre de la clase empieza con mayúscula
- Definición del cuerpo de la clase
Indentada según la palabra **class** definimos atributos y métodos

Ejemplo de definición de clase: Estudiante

class Estudiante:

 nombre="" # en la definición de atributos

 carnet=0 # asignamos sus valores iniciales

 def asignarDatosEstudiante(self,nom,car): # set

 self.nombre=nom

 self.carnet=car

 def obtenerDatosEstudiante(self): # get

 return self.nombre,self.carnet

-
- Las clases tienen al menos dos métodos básicos para manipular los atributos
 - ✓ Método tipo "set" (asignar)
 - Asigna valores a los atributos
 - ✓ Método tipo "get" (obtener)
 - Obtiene los valores de los atributos
 - A los métodos "set" y "get" se les puede asignar un nombre significativo, acorde con la funcionalidad que deben cumplir

-
- Cualquier programa que use objetos de la clase "Estudiante" debe utilizar:
 - Método "asignarDatosEstudiante" para asignar valores a los atributos (método tipo "set")
 - Método "obtenerDatosEstudiante" para obtener los valores almacenados en los atributos (método tipo "get")

- Las definiciones de las clases pueden aparecer en cualquier parte del programa, pero por convención se ubican al inicio del programa después de los estatutos "import"
- Recomendación de la estructura general de las partes de un programa en Python
 - Comentarios iniciales: objetivos del programa, entradas, salidas, restricciones, autor, fecha
 - Estatutos import
 - Definición de clases
 - Definición de funciones
 - Programa principal

CREACIÓN DE OBJETOS A PARTIR DE CLASES

- Después de definir una clase se pueden crear objetos basados en esa clase
- A los objetos creados se les puede llamar “instancias” de la clase
- ¿ Cómo crear objetos ?
 - Llamar al nombre de la clase como una función sin parámetros
- ¿ Cuántos objetos hay que crear ?
 - Los necesarios para obtener una solución

- Ejemplo de manejo de objetos luego de la definición de la clase

crear un objeto de la clase Estudiante

```
est1 = Estudiante()
```

Crea un objeto al cual se le asigna espacio en la memoria.

Al objeto se le da un nombre válido en Python

Valores que tienen los atributos de este objeto particular cuando se crean

	<u>est1</u>	
<i>nombre</i>		<i>carnet</i>
""		0

¿ Por qué los atributos tienen estos valores ?

EJECUCIÓN DE MÉTODOS

- Para ejecutar un método es necesario enviarle al menos un objeto de su clase
- Notación para la ejecución

`nombre_del_objeto.nombre_del_método(argumentos)`

asignar valores al objeto:

uso de un método tipo "set"

`est1.asignarDatosEstudiante("Juan Pérez",20215555)`

crear otro objeto de la clase Estudiante

est2=Estudiante()

asignar valores al objeto

est2.asignarDatosEstudiante("Alberto Prado",20175560)

■ Ejemplo de uso del método “get”

```
# obtener valores del objeto est1  
print (est1.obtenerDatosEstudiante())  
( 'Juan Pérez', 20215555)
```

```
# obtener valores del objeto est2  
print (est2.obtenerDatosEstudiante())  
( 'Alberto Prado', 20175560)
```

OBJETOS SON MUTABLES

- Los valores de los atributos de un objeto se pueden cambiar

- Ejemplo:

- # cambiar los valores del objeto est1

- est1.asignarDatosEstudiante("Juan Alberto Pérez", 20215555)

- # obtener valores del objeto est1

- print (est1.obtenerDatosEstudiante())

- ('Juan Alberto Pérez', 20215555)

-
- Actualizar o acceder un subconjunto de los atributos:
 - Construir métodos específicos para ello

■ Modificación de la clase para agregar nuevos métodos

```
# definir clase Estudiante
class Estudiante:
    nombre=""
    carnet=0
    def asignarDatosEstudiante(self,nom,car):
        self.nombre=nom
        self.carnet=car
    def asignarNombreEstudiante(self,nom):
        self.nombre=nom
    def asignarCarnetEstudiante(self,car):
        self.carnet=car
    def obtenerDatosEstudiante(self):
        return self.nombre,self.carnet
    def obtenerNombreEstudiante(self):
        return self.nombre
    def obtenerCarnetEstudiante(self):
        return self.carnet
```

```
# cambiar el carnet del objeto est2  
est2.asignarCarnetEstudiante(20153333)
```

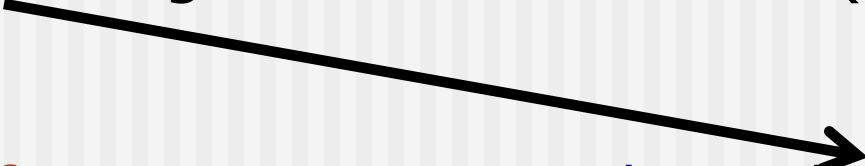
```
# obtener valores del objeto est2  
print (est2.obtenerDatosEstudiante())  
( 'Alberto Prado', 20153333)
```

OBJETO USADO AL LLAMAR LOS MÉTODOS

- Todos los métodos deben tener en su definición al menos un parámetro al cual por convención le llamamos “**self**”
- Este parámetro “**self**” va a ser el primer argumento que definimos en cada método

- Cuando se llama a un método hay que especificar un objeto, automáticamente dicho objeto es insertado como el primer argumento del método

```
est1.asignarDatosEstudiante("Juan Pérez",20155555)
```



```
def asignarDatosEstudiante(self,nom,car):  
    self.nombre=nom  
    self.carnet=car
```

REVISIÓN DE "SELF" EN LA CLASE "ESTUDIANTE"

definir clase Estudiante

class Estudiante:

nombre=""

carnet=0

def asignarDatosEstudiante(self,nombre,carnet):

self.nombre=nombre

self.carnet=carnet

def obtenerDatosEstudiante(self):

return self.nombre,self.carnet

crear un objeto de la clase Estudiante

est=Estudiante()

asignar valores al objeto

est.asignarDatosEstudiante("Juan Pérez",20155555)

obtener valores al objeto

est.obtenerDatosEstudiante()

OBJETOS COMO ARGUMENTOS

- Un objeto (o instancia) se puede pasar como cualquier otro argumento a una función o método

Pasar objetos como argumentos

```
def imprimeEstudiante(x):  
    print ("{0:40s} {1:10d}".format  
          (x.obtenerNombreEstudiante(),  
           x.obtenerCarnetEstudiante()))
```

```
print ("LISTA DE ESTUDIANTES")
```

```
imprimeEstudiante(est1)
```

```
imprimeEstudiante(est2)
```

```
>>>
```

```
LISTA DE ESTUDIANTES
```

```
Juan Pérez
```

```
20155555
```

```
Alberto Prado
```

```
20153333
```

- Si una función o método modifica un objeto, la modificación puede ser vista en cualquier parte del programa
-

Cambio en un objeto recibido

```
def cambiarEstudiante(x):  
    x.asignarCarnetEstudiante(20158888)
```

```
print("Datos antes de llamar a la función que modifica")
```

```
imprimeEstudiante(est2)
```

```
cambiarEstudiante(est2)
```

```
print("Datos despues de llamar a la función que modifica")
```

```
imprimeEstudiante(est2)
```

```
>>>
```

```
Datos antes de llamar a la función que modifica
```

```
Alberto Prado                20153333
```

```
Datos despues de llamar a la función que modifica
```

```
Alberto Prado                20158888
```

OBJETOS COMO VALOR DE RETORNO

- En las funciones y métodos pueden crearse objetos y retornarlos

Retornar instancias

```
def crearEstudiante():
```

```
    nuevoEst=Estudiante()
```

```
    n=input("Nombre del estudiante: ")
```

```
# ← Isabel Solano
```

```
    c=int(input("Carnet: "))
```

```
# ← 20167777
```

```
    nuevoEst.asignarDatosEstudiante(n,c)
```

```
    return nuevoEst
```

```
est3=crearEstudiante()
```

```
imprimeEstudiante(est3)
```

```
>>>
```

Isabel Solano

20167777

COPIA DE OBJETOS

- Al asignar un objeto a otro tenemos la condición de “alias” (como en las listas)
- Copia de objetos (clonar) para manejo independiente de sus valores

```
import copy  
o1=Estudiante()  
o2=copy.deepcopy(o1)
```

Los cambios realizados en un objeto no se reflejan en el otro objeto copiado, sus valores son independientes

MÉTODO CONSTRUCTOR

- Método especial que Python ejecuta en forma automática cuando se crea un objeto
- Funciones del método constructor:
 - Crear los atributos
 - Inicializar los atributos
- Nombre específico: `__init__`

■ Uso del método constructor

```
class Tiempo:
    def __init__(self, h, m, s):
        self.horas=h
        self.minutos=m
        self.segundos=s
    def impTiempo(self):
        print (str(self.horas)+":"+str(self.minutos)+":"+str(self.segundos))
```

Al llamar a la clase se crea el objeto, luego se inicializan sus atributos

```
t1 = Tiempo(9,14,30)
```

```
t1.impTiempo()
```

```
>>>
```

```
9:14:30
```

ELIMINAR OBJETOS

- Función **del**

del t1 # borra objeto t1

LISTAS DE OBJETOS

Crear varios objetos de la clase Estudiante y almacenarlos en una lista

Supuesto: la clase Estudiante tiene varios atributos y métodos

```
estudiantes=[]  
est=Estudiante("Javier",305550555,"Masculino","Cartago",20196234,"Arquitectura")  
estudiantes.append(est)  
est=Estudiante("Martha",309990999,"Femenino","Cartago",20196235,"Electrónica")  
estudiantes.append(est)  
est=Estudiante("Flor",108880888,"Femenino","San José",20196236,"Matemáticas")  
estudiantes.append(est)
```

Recorrer la lista estudiantes e imprimir su información

```
for e in estudiantes:
```

```
    e.muestraNCCEstudiante()
```

```
>>>
```

```
Javier 20196234 Arquitectura
```

```
Martha 20196235 Electrónica
```

```
Flor 20196236 Matemáticas
```


PRÁCTICA

Clase Tiempo

➤ Atributos

- ✓ horas (de 00 a 22)
- ✓ minutos (de 00 a 59)
- ✓ segundos (de 00 a 59)

➤ Métodos

- ✓ `__init__`: inicializa los atributos
- ✓ `getTiempo`: retorna atributos de un objeto en formato hh:mm:ss
- ✓ `sumaTiempos`: recibe dos objetos, uno es el implícito cuando se llama al método (`self`) y el otro es un argumento. Debe retornar otro objeto conteniendo una hora que representa la suma de los tiempos recibidos. Todos los objetos van a ser de la clase

Tiempo. Ejemplo:

Objeto 1 → 13, 20, 30

Objeto 2 → 2, 10, 50

Tiempo retornado → 15, 31, 20

-
- Hacer un programa para:
 - Definir clase Tiempo
 - Crear dos objetos de esa clase
 - Pedir al usuario los valores (input) que va a asignar a los atributos de cada uno de esos objetos usando el método constructor (input de horas, minutos y segundos para cada objeto)
 - Llamar al método sumaTiempos
 - Imprimir el valor retornado usando el método getTiempo

Posible aplicación o uso de este programa:

- calcular la hora final de un proceso

Estudio

Trabajo

Esfuerzo

Honestidad

**¡Lo que tu eres es el regalo de DIOS para ti,
lo que llegues a ser es tu regalo para DIOS!**