PROGRAMACIÓN AVANZADA CON PYTHON

(CEFIRE CTEM)

POO en Python (parte 2)
Ejercicios voluntarios

Autora: María Paz Segura Valero (segura_marval@gva.es)

CONTENIDO

1. Introducción

En este documento puedes encontrar una serie de **ejercicios voluntarios** para que pongas en práctica lo aprendido durante esta unidad.

En el aula virtual dispondrás de las soluciones pero te recomiendo que intentes solucionarlos por ti mismo/a porque, aunque no sean obligatorios para superar el curso, sí pueden ayudarte a enfrentarte al *ejercicio obligatorio* del final.

Puedes utilizar el **foro** del curso para consultar tus dudas.

2. Ejercicios voluntarios

En el aula virtual dispones del fichero **clases_base.py** donde está la definición de tres clases que utilizarás como base en los ejercicios que se explican a continuación.

Te recomiendo que las estudies antes de seguir leyendo este documento.

2.1. Ejercicio 1: Herencia simple

Vamos a crear dos clases derivadas, **Prenda** y **Zapatos**, que heredan de la clase base **Articulo**. Estas clases son una especialización de la clase base. De esta forma podremos crear objetos que representen piezas de ropa y objetos que representen pares de zapatos.

Ten en cuenta los siguientes requisitos:

- Deberás importar la clase `Articulo` del módulo `clases_base`.
- Clase `Prenda`:
  - Atributos: `codigo`, `modelo` y `temporada`. Recuerda que los dos primeros son heredados de la clase base `Articulo`.
  - Métodos: `__init__()` para construir un objeto de la clase y `__str__()` para construir una cadena de texto con los valores de los atributos del objeto.
    - En los dos métodos, utiliza el método correspondiente de la *clase base* y luego añade el código necesario de la *clase derivada*.
- Clase `Zapatos`:
  - Atributos: `codigo`, `modelo` y `suela`. Recuerda que los dos primeros son heredados de la clase base `Articulo`.
  - Métodos: `__init__()` para construir un objeto de la clase y `__str__()` para construir una cadena de texto con los valores de los atributos del objeto.

- En los dos métodos, utiliza el método correspondiente de la *clase base* y luego añade el código necesario de la *clase derivada*.

No es necesario que contemples el encapsulamiento de objetos.

Ejemplo de uso de las clases:

```
print("\nCREAMOS OBJETOS DE PRENDA: ")
p1 = Prenda("P1", "pantalón vaquero corto", "verano"); print(p1)
p2 = Prenda("P2", "blusa ligera", "primavera"); print(p2)
p3 = Prenda("P3", "falda larga", "invierno"); print(p3)

print("\nCREAMOS ZAPATOS: ")
z1 = Zapatos("Z1", "salón", "tacón 7cm"); print(z1)
z2 = Zapatos("Z2", "sandalias", "cuero bajo"); print(z2)
```



```
CREAMOS OBJETOS DE PRENDA:
<P1> es <pantalón vaquero corto> de verano
<P2> es <blusa ligera> de primavera
<P3> es <falda larga> de invierno

CREAMOS ZAPATOS:
<Z1> es <salón> de tacón 7cm
<Z2> es <sandalias> de cuero bajo
```

En el aula virtual dispones de la solución del ejercicio en el fichero **ud4_ejer1_herencia_simple.py**

2.2. Ejercicio 2: Herencia múltiple

Vamos a crear la clase `Socio-Musico` que hereda de las clases `Socio` y `Musico` definidas en el fichero `clases_base.py`. La idea es que una persona afiliada a una sociedad musical pueda ser solo socio, solo músico o las dos cosas a la vez, en cuyo caso podrá disfrutar de descuentos especiales en las actividades de pago.

Ten en cuenta los siguientes requisitos:

- Deberás importar las clases `Socio` y `Musico` del módulo `clases_base`.
- Atributos: `num_socio`, `nombre`, `cuota`, `dni`, `especialidad`, `descuento`. Recuerda que algunos atributos son heredados de las clases base.
- Métodos:
  - `__init__()` para construir un objeto de la clase.
  - `__str__()` para construir una cadena de texto con los valores de los atributos del objeto.

- En los dos métodos anteriores, utiliza el método correspondiente de la *clase base* y luego añade el código necesario de la *clase derivada*.
- `modificar_descuento()` que recibirá como parámetro un número entero y cambiará el valor del descuento del objeto.

No es necesario que contemples el encapsulamiento de objetos.

Ejemplo de uso de la clase:

```
sm1 = Socio_Musico('1', 'Ana', 10, '11111111H', 'violín', 5)
print(sm1, '\n')

sm2 = Socio_Musico('2', 'Pep', 15, '22222222J', 'flauta', 6)
print(sm2, '\n')
```



```
Ana (socio nº 1) paga 10€ de cuota.
Ana (dni: 11111111H) practica violín.
Tiene un descuento de 5€ en actividades.

Pep (socio nº 2) paga 15€ de cuota.
Pep (dni: 22222222J) practica flauta.
Tiene un descuento de 6€ en actividades.
```

En el aula virtual dispones de la solución del ejercicio en el fichero **ud4_ejer2_herencia_múltiple.py**.

2.3. Ejercicio 3: Personal Shopper

Vamos a crear un programa que ayude al departamento de *Personal Shopper* de una empresa de moda. Para ello, mostrará un menú que permitirá crear un conjunto de ropa compuesto por varias prendas y un par de zapatos.

El programa mostrará el siguiente menú:

```
================================
       PERSONAL SHOPPER
================================
 1 - Añadir prenda de ropa
 2 - Elegir zapatos
 3 - Mostrar conjunto
 0 - SALIR
--------------------------
Dame la opción:
```

Deberás crear una clase llamada `Conjunto`, que no heredará de ninguna otra pero sí utilizará el mecanismo de delegación. Tendrá estas características:

- Atributos de objeto:
  - `prendas`, se corresponderá con una **lista de objetos** de la clase `Prenda`.
  - `zapatos`, se corresponderá con un **objeto** de la clase `Zapatos`.

- Métodos de instancia:
  - `__init__()` para construir un objeto de la clase.
  - `__str__()` para construir una cadena de texto con los valores de los atributos del objeto.
  - En los dos métodos anteriores, utiliza el método correspondiente de la *clase base* y luego añade el código necesario de la *clase derivada*.
  - `anyadir_prenda()` que recibirá como parámetro un objeto de la clase `Prenda` y lo añadirá al final de la lista `prendas` del conjunto.

Retomando el menú del programa, cada opción seleccionada ejecutará una nueva función que responderá a la tarea correspondiente.

Opción	Función ejecutada

1	**elegir_prenda()**
	Esta función pedirá al usuario los datos necesarios para crear un nuevo objeto de la clase `Prenda` y lo añadirá a la lista de prendas pasada por parámetro (utiliza el método `anyadir_prenda()`). Además, imprimirá la nueva lista de prendas.
	
	Ejemplo:
	`AÑADIR PRENDA:`
	`--> Dame el código: p3`
	`--> Dame el modelo: Valencia`
	`--> Dame la temporada: primavera`
	
	`Prendas de ropa:`
	`<p1> es <Morocco> de verano`
	`<p2> es <Thorn> de otoño`
	`<p3> es <Valencia> de primavera`
	
	A la función se le pasará como parámetro de entrada el objeto `Conjunto` del programa principal y devolverá el mismo objeto con la lista de prendas modificada.

2	**elegir_zapatos()** Esta función pedirá al usuario los datos necesarios para crear un nuevo objeto de la clase `Zapatos` y lo imprimirá por pantalla.

	Ejemplo:
	`AÑADIR ZAPATOS:`
	`--> Dame el código: z1`
	`--> Dame el modelo: deportivas`
	`--> Dame la suela: goma`
	
	`Zapatos:  <z1> es <deportivas> de goma`
	
	A la función se le pasará como parámetro de entrada el objeto `Conjunto` del programa principal y devolverá el mismo objeto con los zapatos modificados.

3	**mostrar_conjunto()**
	Esta función mostrará por pantalla la información del conjunto creado con las opciones anteriores.
	
	Ejemplo 1: No se han elegido aún prendas ni zapatos.
	`CONJUNTO ELEGIDO:`
	`--> Prendas de ropa: (no seleccionadas)`
	`--> Zapatos: (no seleccionados)`
	
	Ejemplo 2: Se han seleccionado varias prendas y un par de zapatos.
	`CONJUNTO ELEGIDO:`
	`--> Prendas de ropa:`
	`  <p1> es <Morocco> de verano`
	`  <p2> es <Thorn> de otoño`
	`  <p3> es <Valencia> de primavera`
	`--> Zapatos:`
	`  <z1> es <deportivas> de goma`
	
	A la función se le pasará como parámetro de entrada el objeto `Conjunto` y no devolverá ningún parámetro de salida.
	
	Si tienes bien construido el método `__str__()` solo hará falta que hagas `print(objeto_conjunto)` para que funcione.

0	El programa acabará.

Cuando se acabe de ejecutar una acción, el programa volverá a mostrar el menú al usuario, excepto si se ha seleccionado la *opción 0*.

Te recomiendo que al principio del programa crees un objeto `Conjunto` vacío que sea el que pases como parámetro a las funciones. Puedes crearlo así:

```
c = Conjunto([], None)
```

Donde `[]` es la lista de prendas vacía y `None` es el objeto `Zapatos` sin instanciar.

No es necesario que contemples el encapsulamiento de objetos.

Puedes crear el programa desde cero o basarte en el esquema que tienes en el fichero **ud4_ejer3_personal_shopper (ESQUEMA).py** del aula virtual. En este caso, deberás cambiar las instrucciones *pass* que encuentres en el código.

En el aula virtual dispones de la solución del ejercicio en el fichero **ud4_ejer3_personal_shopper.py**.