Taller 5 Python, Punto #2

1) **Sí deseo modificar el mensaje del método ruido al crear un objeto Pájaro sin alterar la clase Animal, ¿Qué debo agregarle al código? (Por ejemplo, al llamar al método ruido imprima, cantar y silbar)**

Es necesario en primer lugar, cambiar el super.__init__() dentro del método __init__(self) de la clase Pájaro, por super().
Lo anterior para permitir que sea posible crear instancias del tipo pájaro, de otro modo solo soltará errores de tipo: "TypeError: descriptor '__init__' requires a 'super' object but received a 'str'" cada vez que se intente inicializar (poniendo el nombre del pájaro, como lo indica el inicializador, como un objeto de tipo string)

En Python es necesario crear un nuevo método con el mismo nombre dentro de la clase Pájaro, de la forma:

```
def ruido(self):
    print("cantar y silbar")
```

De esta manera, al llamar al método ruido(), siempre se imprimirá "cantar y silbar".

2) **Si se crea una nueva clase Pez, y no se definen nuevos métodos, constructor y atributos. ¿Qué constructor tendrá esta clase, qué argumentos recibe? ¿Qué otros métodos y atributos tendrán estos mismos?**

En caso de ser una clase hija de Animal, tendrá todos los métodos y atributos de la clase Animal, para crear una instancia de Pez entonces se usará el constructor (Método __init__()) de la clase Animal. Por tanto, recibirá nombre, edad y raza.
Por lo mismo será posible usar los métodos como setRaza() del método animal desde una instancia de la clase pez.

3) **¿Qué ocurre con el atributo nombre y edad de la clase SerVivo, al momento de definirse en la Clase Animal? ¿Cómo cambiaría el código del constructor para que estos atributos sean el mismo?**

Los atributos nombre y edad son atributos de estancia, por tanto, no pertenecen a la clase como tal, así, aún cuando se redefine el método __init__() en la clase Animal, y se hace uso de self.nombre = nombre y self.edad= edad, realmente no hay ninguna diferencia, puesto que estos atributos solo corresponderían a cualquier instancia que se inicialice, y dado que tienen el mismo nombre, cualquier método de la clase SerVivo que haya sido heredado y que use alguno de estos atributos, podrá ser utilizado sin ningún problema.

Por ejemplo, el método "getNombre()" puede borrarse de la clase Animal y no habrá ningún problema con la clase o con sus clases hijas. El método seguirá existiendo y funcionando para todos (Naturalmente, funciona igual dado a que la redefinición del método getNombre en Animal es igual al método original en SerVivo)

Si quisiéramos que fueran "iguales", lo único sería llamar al método __init__() de la clase SerVivo dentro del método __init__() de la clase Animal, dándole los atributos Nombre y Edad, de la forma super().__init__(nombre, edad).
Lo anterior, haría que el atributo de la clase SerVivo, _totalCreados, aumente cada vez que ser crea una instancia de cualquier clase hija de Animal.

4) **En la clase Animal se sobrescribieron los métodos setNombre y getNombre, ¿Cómo modificaría estos métodos para que su funcionamiento no oculte algún valor de la clase padre? ¿Podría plantearse esta solución usando super()?**

En Python no se ocultan los valores como en JAVA, en su lugar, se sobrescriben totalmente, a menos que desde el código de la clase hija se creé un método que llame que llame al método de la clase padre.
Si no queremos "ocultar" el método de la clase padre, se puede incluir el llamado al método de la clase padre (SerVivo) desde el método de la clase hija (Animal) de la forma:

```
def getNombre(self):
    return [self._nombre, super().getNombre]
```

Dado que ambos son valores de retorno (Aunque son iguales, dado que se está devolviendo un atributo de instancia y no de clase), incluyéndose en un array se podrá trabajar con cada uno independientemente.

Si fuese un atributo de clase el que se está llamando (Y este tiene el mismo nombre en la clase padre e hija), sería posible simplemente definir 2 métodos diferentes para la clase hija, uno para acceder a su valor y otro para acceder al de la clase padre, o incluir un condicional en el método para acceder al valor de la clase padre o hija según un bool incluido en el llamado del método. Por ejemplo:

```
@classmethod
  def getTotalCreados(cls, bool):
    if bool==True:
      return cls._totalCreados
    else:
      return super()._totalCreados;
```

De esta manera, desde una instancia de la clase hija, podemos acceder a los valores del atributo _totalCreados de la clase padre y de la clase hija desde un mismo método, con el uso de super().

5) **El atributo totalCreados de la clase SerVivo ¿es heredado por las demás clases? ¿En este caso ocurre ocultación de los atributos al definirlo de nuevo en las clases hijas?**

El atributo si es heredado, la ocultación si ocurre (Y también debido al llamado en __init__ de "ClaseHija._totalCreados += 1").
Concretamente, se crea el atributo de clase totalCreados para cada una de las clases en las que se redefine, eliminando la referencia al atributo de la clase padre. En caso de que esto no se hiciera, desde todas las clases, cuando se llamase al atributo de clase se obtendría el mismo valor.
En cambio, dado que se redefine, cada clase tiene un atributo con el mismo nombre y por tanto pierden la referencia totalCreados que va al atributo de la clase padre y en su lugar esta se dirige a su propio atributo.
Aún será posible acceder al atributo con super() o con ClasePadre._totalCreados. Aunque esta última no es recomendable.

6) **¿Los métodos getTotalCreados sobrescriben al método de su padre?**

Sí, el método es heredado, y al volver a definirse el método del padre es sobrescrito y solo queda el método de la clase en el que se definió.
Aunque en este caso en particular, el método redefinido hace exactamente lo mismo que el método heredado del padre, y no hay diferencia.

7) **¿Qué métodos hereda una clase que hereda de la clase Persona?**

Todos los métodos de la clase SerVivo (__init__(), setNombre(), getNombre(), setEdad(), getEdad(), getTotalCreados() [Particularmente, el primero y el último son sobrescritos, y por ende no se heredan al final]) y los métodos de la clase persona __init__(), aduenarAnimal() y getTotalCreados().
Los métodos __init__() y getTotalCreados() que queden en la clase que hereda a Persona, dependerán de si fueron o no redefinidos en la clase Persona y por tanto cambiaron respecto a la clase SerVivo.

8) **¿Qué tipo de objetos podrían pasársele al método aduenarAnimal de la clase Persona?¿Se le puede pasar un objeto SerVivo?**

Cualquier tipo de objeto que tenga definido el método ruido().
No se le podría asignar un objeto de la clase SerVivo o Persona, debido a que no tienen el método ruido() definido.

9) **¿Cómo podría obtener la cantidad de seres vivos que se creen (objetos creados)?**

Desde un objeto de la clase SerVivo, solo sería necesario llamar el método getTotalCreados().
Desde un objeto de la clase Persona, es imposible a menos que se cree un método con el atributo super(). (o que llame directamente, SerVivo._totalCreados)
Desde un objeto de la clase Animal, es el mismo caso que en Persona.
Desde un objeto de una clase hija de Animal, si existiese un método en la clase Animal que accede al atributo totalCreados de SerVivo, sería posible, si no, no lo es.

Finalmente, dado que en Python no existe el encapsulamiento, siempre es posible acceder con SerVivo._totalCreados.

Adicionalmente, la clase Animal no llama al método __init__() de la clase ser vivo, por lo que la cantidad de instancias de la clase Animal o de una clase hija de esta, no afectan al atributo totalCreados de SerVivo.
Para arreglar esto, solo sería necesario llamar al __init__() de SerVivo desde Animal (Ya que todas las clases hijas de Animal ya llaman a su __init__()) con super().

10) **Si se define el siguiente método en la clase Perro:**

```
def getRaza(self, tipo):
    return self.raza + " "+ tipo
```

¿Qué ocurre con el método getRaza del a clase Animal? ¿Este método se sobrescribe o se sobrecarga?

El método se sobrescribe. En Python no existe la sobrecarga de métodos. (A menos que se incluyan librerías especiales, pero en este caso particular, no existen)
Por tanto, no sería posible llamar a getRaza(), pues daría un error al no ingresarle el "tipo" al llamado del método.