Taller

José Luis Cancelado Castro

Respuestas:

1) Selección múltiple
   ¿Cuáles de los siguientes nombres existen como atributos accesibles directamente desde
   a?
   A) a.x
   B) a._y
   C) a.__z
   D) a._A__z

```python
class A:
    x = 1
    _y = 2
    __z = 3
a = A()
print(a.x)#este lo toma
print(a._y)#este lo toma
#print(a.__z)#este no lo va a tomar
print(a._A__z)#este lo toma
```

2) Salida del programa
   ¿Qué imprime?

```python
class A:
    def __init__(self):
        self.__secret = 42
a = A()
print(hasattr(a, '__secret'), hasattr(a, '_A__secret'))
#RTA: El primero va a devolver false y el segundo true
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C:/Use
rs/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/punto2.py"
False True
PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial>
```

3) Verdadero/Falso (explica por qué)

```python
#3) Verdadero/Falso (explica por qué)
#a) El prefijo _ impide el acceso desde fuera de la clase.
#b) El prefijo __ hace imposible acceder al atributo.
#c) El name mangling depende del nombre de la clase.
#RTA:
#a)Falso, el prefijo solo se usa por convencion para el uso dentro de la clase y subclases.
#b)Falso, el prefijo solo le cambia el nombre(name mangling).
#c)Verdadero, el name mangling depende del nombre de la clase, por ejemplo para accerder a miembros privados
```

4) Lectura de código
   ¿Qué se imprime y por qué no hay error de acceso?

```python
# punto4.py > ...
# You, 1 second ago | 1 author (You)
class Base:
    def __init__(self):
        self._token = "abc"
# You, 1 second ago | 1 author (You)
class Sub(Base):
    def reveal(self):
        return self._token
print(Sub().reveal())
#¿Qué se imprime y por qué no hay error de acceso?
#RTA: abc, no hay error ya la convención _ es precisamente para el uso en la misma clase o las subclases
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C:/Users/Jhose/AppData/Local
rs/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/punto4.py"
abc
```

5) Name mangling en herencia
¿Cuál es la salida?

```python
# punto5.py > ...
# You, 23 hours ago | 1 author (You)
class Base:
    def __init__(self):
        self.__v = 1
# You, 23 hours ago | 1 author (You)
class Sub(Base):
    def __init__(self):
        super().__init__()
        self.__v = 2
    def show(self):
        return (self.__v, self._Base__v)
print(Sub().show())       You, 23 hours ago • Uncommitted changes
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C
rs/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/punto5.py"
(2, 1)
```

6) Identifica el error
¿Qué ocurre y por qué?

```python
# punto6.py > ...
class Caja:
    __slots__ = ('x',)
c = Caja()
c.x = 10
c.y = 20
#RTA: Esto ya que se estan limitando los atributos
#AttributeError: 'Caja' object has no attribute 'y'
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C:/Use
rs/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/punto6.py"
Traceback (most recent call last):
  File "c:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial\punt
    c.y = 20
    ^^^
AttributeError: 'Caja' object has no attribute 'y'
PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial>
```

7) Rellenar espacios
Completa para que b tenga un atributo “protegido por convención”.

```python
class B:
    def __init__(self):
        self._numero = 99
```

8) Lectura de métodos “privados”
¿Qué imprime y por qué?

```python
class M:
    def __init__(self):
        self._state = 0
    def _step(self):
        self._state += 1
        return self._state
    def __tick(self):
        return self._step()
m = M()
print(hasattr(m, '_step'), hasattr(m, '__tick'), hasattr(m,
'_M__tick'))
#La funcion hasattr es una funcion que devuelve si encuentra atributos y metodos y e
#al estar protegido "privado", no lo va a encontrar y va a devolver false
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C:
n.exe "c:/Users/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/P
True False True
PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial>
```

9) Acceso a atributos privados
Escribe la línea solicitada.

```python
class S:
    def __init__(self):
        self.__data = [1, 2]
    def size(self):
        return len(self.__data)
s = S()
# Accede a __data (solo para comprobar), sin modificar el código de la clase:
# Escribe una línea que obtenga la lista usando name mangling y laimprima.
#RTA:
print(s._S__data)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C:/Users/Jhos
n.exe "c:/Users/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/PUNTOS/punto
[1, 2]
PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial>
```

10) Comprensión de dir y mangling
¿Cuál de estos nombres es más probable que aparezca en la lista: __a, _D__a o a?
Explica.

```
PUNTOS > punto10.py > ...
1   class D:
2       def __init__(self):
3           self.__a = 1
4           self._b = 2
5           self.c = 3
6   d = D()
7   names = [n for n in dir(d) if 'a' in n]
8   print(names)
9   #RTA: Es mas probable que aparezca _D__a ya que el dir me devuelve
10  #el nombre con el name mangling
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial> & C:/Users/Jhose/AppData/Local/P
n.exe "c:/Users/Jhose/OneDrive/Documentos/UNAL/SEMESTRES/PRIMER SEMESTRE/POO/Taller-del-parcial/PUNTOS/punto10.py"
['_D__a', '__class__', '__delattr__', '__format__', '__getattribute__', '__getstate__', '__hash__', '__init_subclass__', '__s
 '__weakref__']
PS C:\Users\Jhose\OneDrive\Documentos\UNAL\SEMESTRES\PRIMER SEMESTRE\POO\Taller-del-parcial>
```

11) Completar propiedad con validación
Completa para que saldo nunca sea negativo.

```python
class Cuenta:
    def __init__(self, saldo):
        self._saldo = 0
        self.saldo = saldo
    @property
    def saldo(self):
        return self._saldo
    @saldo.setter
    def saldo(self, saldonuevo):
        if saldonuevo < 0:
            print("el saldo no puede ser negativo")
        else:
            self._saldo = saldonuevo

saldonuevo = int(input("Ingrese un saldo"))
saldonue=Cuenta(saldonuevo)
print(saldonue.saldo)        You, 2 hours ago • Un
```

12) Propiedad de solo lectura
Convierte temperatura_f en un atributo de solo lectura que se calcula desde temperatura_c.

```
PUNTOS > punto12.py > ...
1   class Termometro:
2       def __init__(self, temperatura_c):
3           self._c = float(temperatura_c)
4   # Define aquí la propiedad temperatura_f: F = C * 9/5 + 32
5   #RTA:
6       @property
7       def temperatura_f(self):
8           return self._c*9/5 +32
9
```

13) Invariante con tipo
Haz que nombre sea siempre str. Si asignan algo que no sea str, lanza TypeError.

```python
class Usuario:
    def __init__(self, nombre):
        self._nombre = None
        self.nombre = nombre

    @property
    def nombre(self):
        return self._nombre

    @nombre.setter
    def nombre(self, valor):
        if not isinstance(valor, str):
            raise TypeError("el valor debe ser str")
        else:
            self._nombre = valor
```

14) Encapsulación de colección

Expón una vista de solo lectura de una lista interna.

```python
PUNTOS > punto14.py > ...
class Registro:
    def __init__(self):
        self.__items = []
    def add(self, x):
        self.__items.append(x)
    @property
    def listar_items(self):
        return tuple(self.__items)
# Crea una propiedad 'items' que retorne una tupla inmutable con
#RTA:
R=Registro()
R.add("x")
R.add("t")
R.add("b")
print(R.listar_items)

```

15) Refactor a encapsulación

Refactoriza para evitar acceso directo al atributo y validar que velocidad sea entre 0 y 200.

```python
class Motor:
    def __init__(self, velocidad):
        self._velocidad = 0
        self.velocidad = velocidad
    @property
    def velocidad(self):
        return self._velocidad
    @velocidad.setter
    def velocidad(self, velo):
        if self.velo < 0:
            print("la velocidad no puede ser menor que cero")
        elif self.velo > 200:
            print("la velocidad no puede ser mayor a 200")
velocidad = int(input("ingrese la velocidad"))
carro=Motor(velo)
carro.velocidad
```

16) Elección de convención

Explica con tus palabras cuándo usarías _atributo frente a __atributo en una API pública de una librería.
RTA: Lo haría para evitar el sniffing de datos, y para evitar choques con las subclases, además de una encapsulación mas fuerte

PUNTOS > punto16.py

```python
#Lo haría para evitar el sniffing de datos,
# y para evitar choques con las subclases,
# además de una encapsulación mas fuerte
```

17) Detección de fuga de encapsulación
¿Qué problema hay aquí?

```python
#¿Qué problema hay aquí?
#RTA: El get data rompe la encapsulacion podiendola modificar desde fuera
#mi solucion fue usar el property con una tupla
class Buffer:
    def __init__(self, data):
        self._data = list(data)
    @property
    def data(self):
        return tuple(self._data)
```

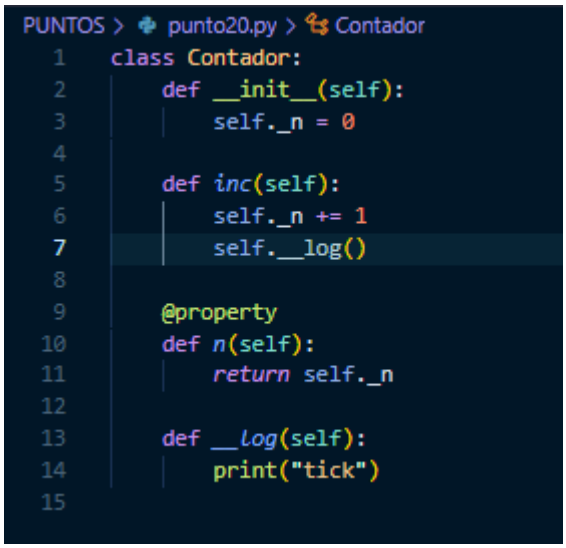
18) Diseño con herencia y mangling
¿Dónde fallará esto y cómo lo arreglas?

```python
#¿Dónde fallará esto y cómo lo arreglas?
#RTA: Fallara en la funcion get ya que __x tiene name mangling
#lo arregle poniendo el name mangling
class A:
    def __init__(self):
        self.__x = 1

class B(A):
    def get(self):
        #return self.__x # esta no va a funcionar
        return self._a__x
```

19) Composición y fachada
Completa para exponer solo un método seguro de un objeto interno.

```python
class _Repositorio:
    def __init__(self):
        self._datos = {}
    def guardar(self, k, v):
        self._datos[k] = v
    def _dump(self):
        return dict(self._datos)
class Servicio:
    def __init__(self):
        self.__repo = _Repositorio()

    def guardar(self, k, v):
        self.__repo.guardar(k, v)
```

20) Mini-kata
Escribe una clase ContadorSeguro con:


PUNTOS > punto20.py > Contador
class Contador:
def __init__(self):
self._n = 0
def inc(self):
self._n += 1
self.__log()
@property
def n(self):
return self._n
def __log(self):
print("tick")