Taller de Parcial

Juan Pablo Florez Rubio

Respuestas

1) A) y B)

2) False True

3) Falso, Falso, Verdadero

4) abc, el guion bajo del atributo no impone restricciones, y Sub hereda de Base, también tiene acceso

5) 21

6) Da error en la línea c.y=20, ”y” nunca fue definido como atributo en __slots__

7) self._numeromagico = 99

8) (True, False, True), _step puede ser hallado normalmente, __tick no existe, da falso porque fue renombrado a _M__tick, el cual puede ser hallado y da verdadero
9) print(s._S__data)

10) _D__a, porque “__a” fue renombrado a “_D__a”, y “a” no existe por si solo, su nombre es “__a”

11)

```
@property
def saldo(self):
return self._saldo

@saldo.setter
def saldo(self, value):
if (value < 0):
        raise ValueError("Saldo negativo")
self._saldo = value
```

12)

```
@property
  def temperatura_f(self):
        return (self._c *( 9 / 5 )) + 32
```

13)

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

  @nombre.setter
  def nombre(self, value):
     if not isinstance(value, str):
           raise TypeError("El nombre no es string")
     self._nombre = value
```

14) @property

```
def items(self):
    return tuple(self.__items)
```

15)

```
def __init__(self, __velocidad):
self.velocidad = __velocidad
    @property
  def velocidad(self):
    return self._velocidad

  @velocidad.setter
  def velocidad(self, value):
    if (value <0 or value>200):
      raise ValueError("La velocidad no está entre 0 y 200")
    self.__velocidad = value
```

16) _atributo cuando el atributo no debería usarse libremente en la práctica, pero aún es accesible. Y __atributo cuando quiero evitar que el atributo se herede y que fuera de clase sea observable.

17) _data puede ser expuesta y modificada en get_data(self), intercambiar por:

```
@property
def data(self)
        return tuple(self, _data)
```

La tupla es inmutable

18) El codigo falla en get(self), porque self.__x no existe, al usar return podría ser llamado como:

```
def get(self):
        return self._A__x
```

19)

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

20)

```
class ContadorSeguro:
  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")
```

Uso básico:

```
c = ContadorSeguro() # crea una variable y guarda la instancia (la suma de n)
c.inc()  # imprime tick y suma 1
c.inc()  # imprime tick y suma 1, por segunda vez
print(c.n) # imprime 2 (valor final de n)
```