Guião 2 – Encapsulamento

Objectivos:

- Atributos privados
- Métodos privados
- Getter/Setter
- @Property

1. Considere a seguinte classe que armazena um valor de temperatura em Celsius.
Instancie um objecto e aceda (ler e escrever) no atributo `temperatura`.
Consulte também o valor da temperatura armazenada em Fahrenheit.
Implemente o método em falta.

```
class Celsius:
    def __init__(self, valor=0):
        self.temperatura = valor

    def get_fahrenheit(self):
        return (self.temperatura * 1.8) + 32

    def get_kelvin(self):...
```

2. Considere ainda a seguinte função soma (está fora da classe). A função tem como argumento de entrada uma lista (tipo list []) de objectos Celsius e retorna um novo objecto Celsius que representa o somatório das temperaturas dos elementos na lista.

```
def soma(lista):
    sum=Celsius(0)
    for t in lista:
        sum.temperatura += t.temperatura
    return sum
```

Com "type annotations" (opcional):

```
def soma(lista:list[Celsius])->Celsius:
    sum=Celsius(0)
    for t in lista:
        sum.temperatura += t.temperatura
    return sum
```

Crie uma função `def media(lista)` que calcule a média dos valores de temperatura de uma lista de objetos `Celsius`. Deve utilizar a função `soma` no cálculo e retornar a média.

3. Já temos algum código que envolve utilização da classe `Celsius` e que depende directamente do atributo `temperatura`. Suponha que esta classe é mantida por si e que já tem dezenas de utilizadores com código como o anterior a utilizar o `.temperatura`.

Sabendo que a temperatura não pode ser inferior a -273.15 ºC, avalie a possibilidade incluir esta verificação.

4. Transforme o atributo `temperatura` em atributo privado. Crie os getters e setters correspondentes de forma a poder manipular a variável privada.

- Adicione ainda a validação anterior no setter. Se menor que -273.15 a temperatura deve ficar nos -273.15 ºC.
- E no `__init__`, que fazer?

5. Agora, enquanto cliente e utilizador da classe o código anterior (`soma` e `media`) deixou de funcionar. Altere-o para respeitar as alterações para a nova classe. Qual das versões lhe parece mais legível/limpa (Pythonic).

Obs: Guarde o código anterior, vamos voltar a ele.

6. O atributo `temperatura` (agora `__temperatura`) deixou de estar acessível de "fora". Crie um `property` (da forma clássica) com o nome `temperatura` poder manipular o `__temperatura` através do setter e getter anterior. Deverá conseguir executar o código original, feito no ponto 2.

7. A utilização do `property` tem uma versão mais simplificada, utilizando as anotações `@property` e `@temperatura.setter`. Modifique a classe de forma a incluir estas anotações.

7. Adicione à classe um método de objecto (com o `self`) que devolva uma string com o valor da temperatura por extenso.

Ex: Celsius(-1).extenso() -> "Menos um grau Celsius"

Celsius(0).extenso() -> "Zero graus Celsius"

Celsius(1).extenso() -> "Um grau Celsius"

Celsius(2).extenso() -> "Dois graus Celsius"

Deve existir um método auxiliar **privado** que apenas retorne a string "grau" ou "graus" de acordo com o valor da temperatura.

Dica:

```
from num2words import num2words
print(num2words(1, lang='pt'))
```

Dica:

Procure utilizar funções (métodos) auxiliares, possivelmente privados.

8. Mais alguns testes. Instancie um objecto Celcius() e:

- Tente aceder ao atributo (agora property) temperatura.
- Tente aceder ao atributo __temperatura.
- Tente outra vez... (consulte o campo __dict__ do objecto)

O que pode concluir quanto à utilização de atributos/métodos privados

9. As funções `soma` e `media` estão fora da classe. No entanto, estas operam exclusivamente objetos da classe Celsius. Faz sentido passar esta função para contexto da classe? Qual a melhor abordagem?
Não se esqueça de atualizar as chamadas a estas funções.

10. Finalize o guião e envie via moodle.