# Programação II

# Exercícios 4
# Testes baseados na partição do espaço de entrada

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Tecnologias da Informação

2020/2021

**Nota prévia.** Para testar os nossos programas vamos utilizar o módulo `doctest`. Consulte a documentação.

1. Considere a função `conta_positivos` indicada abaixo.

```
def conta_positivos (v):
    """número de elementos positivos numa lista

    Args:
        v (list): lista de números

    Returns:
        int: número de valores positivos em v
    """
    conta = 0
    for x in v:
        if x >= 0:
            conta = conta + 1
    return conta
```

   (a) Encontre a falha (o defeito) na implementação.

   (b) Encontre um teste que detete e um que não detete a falha.

   (c) Repare a falha.

2. Repita os passos do exercício 1 para a função `encontra_ultimo` indicada abaixo.

```
def encontra_ultimo (lista, x):
    """índice do último elemento numa lista que é
    igual a um dado elemento.

    Args:
        lista (list): lista
        x (any): valor a procurar numa lista
    Returns:
        índice da última ocorrência de x em lista,
        ou None se x não ocorrer na lista
    """
    for i in range(len(lista)-1, 0, -1):
        if lista[i] == x:
            return i
    return None
```

3. Escreva três testes para a função `fibonacci` indicada abaixo.

```
def fibonacci(n, a=1, b=1):
    """termo n da sequência de Fibonacci

    Args:
        n (int): índice do termo a calcular
        a (int, optional): guarda o termo atual da
                           sequência. Defaults to 1.
        b (int, optional): guarda o termo seguinte da
                           sequência. Defaults to 1.
    Returns:
        int: termo de índice n
    """
    return a if n==0 else fibonacci(n-1, b, a+b)
```

4. Escreva quatro testes para o predicado `e_bissexto` indicado abaixo.

```
def e_bissexto(ano):
    """Verifica se um dado ano é bissexto

    Args:
        ano (int): ano a verificar
    Returns:
        bool: True se ano for bissexto, False c.c.
    """
    return ano % 400 == 0 if ano % 100 == 0 else ano
       % 4 == 0
```

5. Considere a função `media` indicada abaixo.

```
def media(lista):
    """média dos valores de uma lista

    Args:
        lista (list): lista de números

    Returns:
        média dos valores na lista se esta for não
        vazia, ou None caso contrário.
    """
    return sum(lista)/len(lista) if len(lista)>0 else
        None
```

Considere as seguintes características:

- Número de elementos na `lista`.
- A `lista` contém números negativos?

(a) Para cada característica identifique blocos adequados.

(b) Combine todos os blocos, eliminando os casos inviáveis. Apresente os resultados numa tabela desta forma:

| Características e blocos | | Testes | |
|---|---|---|---|
| Elementos | Negativos | `lista` | Resultado |
| 0 | `False` | `[]` | `None` |
| ... | ... | ... | ... |

(c) Apresente os testes em formato `doctest`.

6. Considere a função `substituir` indicada abaixo.

```
def substituir(lista, antigo, novo):
    """cria uma cópia da lista dada onde ocorrências
    de um dado valor são substituídas por outro valor

    Args:
        lista (list): lista de valores
        antigo (any): valor a substituir
        novo (any): novo valor

    Returns:
        list: lista após a substituição
    """
    return [novo if x==antigo else x for x in lista]
```

Escreva uma bateria de testes para esta função, seguindo os passos do exercício 5. Considere as seguintes características:

- Número de elementos na `lista`.
- Número de vezes que `antigo` ocorre em `lista`.

7. Considere o predicado `e_palindromo` indicado abaixo.

```
def e_palindromo(string):
    """verifica se uma dada string é um palíndromo
    (igual leitura em ambos os sentidos)

    Args:
        string (str): string a verificar

    Returns:
        bool: True se string é palíndromo, False c.c.
    """
    if len(string) == 0:
        return True
    elif string[0] != string[-1]:
        return False
    else:
        return e_palindromo(string[1:-1])
```

Escreva uma bateria de testes para esta função, seguindo os passos do exercício 5. Considere as seguintes características:

- Paridade (zero, par, ímpar) do tamanho da `string`.
- A `string` é um palíndromo?

8. Considere a função `intersecao` indicada abaixo:

```
def intersecao (lista1, lista2):
    """lista interseção de duas listas.

    Pre:
        ambas as listas são sem duplicados

    Post:
        a lista resultante é sem duplicados

    Args:
        lista1 (list): uma lista
        lista2 (list): outra lista

    Returns:
        list: a lista com os elementos que ocorrem em
              ambas as listas
    """
```

Escreva uma bateria de testes para esta função, seguindo os passos do exercício 5. Considere as seguintes características:

- A `lista1` está vazia?
- A `lista2` está vazia?
- Relação entre `lista1` e `lista2`: as listas têm os mesmos elementos, `lista1` é subconjunto de `lista2`, `lista2` é subconjunto de `lista1`, as listas não têm elementos em comum, nenhuma das anteriores.

9. Considere a função `potencia_natural(base, expoente)`. A `base` é um número qualquer; o `expoente` é um número inteiro não negativo. Escreva uma bateria de testes para esta função, seguindo os passos do exercício 5. Considere as seguintes características:

   - Sinal (negativo, zero, positivo) da `base`;
   - Sinal (zero, positivo) do `expoente`;
   - A `base` é um número inteiro?

10. Considere a função `inverter(lista)` que inverte uma lista qualquer. A função não devolve coisa nenhuma; em vez disso altera a lista passada como parâmetro. Escreva uma bateria de testes para esta função, seguindo os passos do exercício 5. Considere as seguintes características:

    - Número de elementos na lista;
    - A lista tem elementos repetidos?

11. Considere a função `maximo(x, y)` que devolve o máximo de dois números. Escreva uma bateria de testes para esta função, seguindo os passos do exercício 5. Considere as seguintes características:

    - Sinal de `x`.
    - Sinal de `y`.
    - Relação entre `x` e `y`.

12. Escreva uma bateria de testes para a função `conta_positivos` (exercício 1) utilizando a técnica da partição de espaço de entrada.

    (a) Modele o domínio da função, identificando uma ou mais características.

    (b) Para cada característica identifique blocos adequados.

    (c) Combine todos os blocos, eliminando os pares inviáveis. Apresente os resultados na forma de uma tabela.

    (d) Apresente os testes em formato `doctest`.

13. Escreva uma bateria de testes para a função `encontra_ultimo` (exercício 2) utilizando a técnica da partição de espaço de entrada. Siga os passos do exercício 12.

14. Considere a função `diferenca` indicada abaixo. Escreva uma bateria de testes para esta função utilizando a técnica da partição de espaço de entrada. Siga os passos do exercício 12.

```
def diferenca(l1, l2):
    """lista que contém os elementos de l1 que não
    estão em l2
    Args:
        l1 (list): lista
        l2 (list): lista
    Returns:
        list: lista l1 \ l2
    """
    resultado = []
    for x in l1:
        if x not in l2:
            resultado.append(x)
    return resultado
```

15. Considere a função `soma_produtos` indicada abaixo. No caso de a função ser chamada com listas de tamanho diferente, levanta a exceção `ValueError`. Observe que a documentação inclui um teste para o caso em que a primeira lista tem dimensão inferior à segunda lista.

```
def soma_produtos(xs, ys):
    """soma dos produtos dos elementos de duas listas
    dadas, elemento a elemento
    Args:
        xs (list[float]): lista de floats
        ys (list[float]): lista de floats
    Raises:
        ValueError: caso as listas tenham
           comprimentos diferentes
    Returns:
        float: produto interno entre xs e ys

    >>> soma_produtos([1.0], [1.0, 2.0])  # (x1,y2)
    Traceback (most recent call last):
    ...
    ValueError
    """
    if len(xs) != len(ys):
        raise ValueError
```

```
    resultado = 0.0
    for i in range(len(xs)):
        resultado += xs[i] * ys[i]
    return resultado
```

Inclua testes adicionais para esta função, utilizando a técnica da partição de espaço de entrada. Siga os passos do exercício 12.

16. Considere a função `maximo_lista` que devolve o máximo de uma lista de números inteiros. No caso de a função ser chamada com uma lista vazia, levanta a exceção `IndexError`.

    Nota: a lista vazia faz parte do espaço de entrada da função `maximo_lista`, mesmo que não faça parte do seu domínio. Como temos informação sobre o comportamento excecional (fora do domínio) podemos testar a função para este caso.

    Escreva uma bateria de testes para esta função utilizando a técnica da partição de espaço de entrada. Siga os passos do exercício 12.

17. Considere a função `obtem` indicada abaixo:

```
def obtem(lista, indice):
    """elemento da lista na posição dada pelo índice
    Args:
        lista (list): uma lista
        indice (int): número entre 0 e len(lista) -
            1.
    Raises:
        IndexError: caso indice < 0 ou indice >= len(
            lista)
        TypeError: caso indice não seja um número
            inteiro
    Returns:
        any: elemento na posição dada pelo índice
    """
```

    Leia a nota do exercício 16. Escreva uma bateria de testes para esta função utilizando a técnica da partição de espaço de entrada. Siga os passos do exercício 12.

18. Considere a seguinte especificação da função `unicos`.

    > A função recebe uma lista e devolve uma outra contendo exactamente uma ocorrência de cada elemento da lista original. A lista original está ordenada; a lista resultado deverá estar também ordenada. A lista original não deve ser alterada.

    Escreva uma bateria de testes para esta função utilizando a técnica da partição de espaço de entrada. Siga os passos do exercício 12.