TÉCNICO LISBOA

Fundamentos da Programação
Ano lectivo 2025-26
Primeiro Projecto
26 de Setembro de 2025

Scrabble IST 1

Neste primeiro projeto de Fundamentos da Programação, os alunos irão escrever um programa em Python que permita jogar uma adaptação do jogo Scrabble[1]

1 Descrição do jogo

O Scrabble é um jogo de tabuleiro de dois a quatro jogadores em que os jogadores colocam peças por turnos para obter pontos, cada peça com uma única letra e uma pontuação, num tabuleiro de 15×15 casas. As peças devem formar palavras que, à semelhança de um jogo de palavras cruzadas, são lidas da esquerda para a direita nas linhas horizontais ou de cima para baixo nas verticais. A pontuação duma palavra é a soma dos pontos das peças que a formam.

Antes do início do jogo, todas as peças com **letras** do jogo são colocadas num **saco**. A seguir, cada **jogador** retira sete **letras** do saco, formando o seu **conjunto de letras**. Começando pelo primeiro jogador e em turnos consecutivos, cada jogador pode realizar uma das seguintes ações:

1. **Passar**, perdendo a vez e obtendo zero pontos.

2. **Trocar** uma ou mais letras do seu conjunto de letras por um número igual de letras retiradas aleatoriamente do saco, obtendo zero pontos. Esta opção só está disponível se restarem pelo menos 7 letras no saco.

3. **Jogar** letras do seu conjunto nas casas livres duma única linha vertical ou horizontal do tabuleiro, para formar uma palavra do vocabulário. Nesta versão adaptada do jogo, consideramos as seguintes condições para jogar letras:

   (a) A primeira jogada da partida deve incluir pelo menos duas letras e cobrir a casa central do tabuleiro.

   (b) As jogadas seguintes devem colocar no tabuleiro pelo menos uma letra do jogador, todas as letras devem ser jogadas numa mesma linha (horizontal ou vertical), e a palavra formada deve conter pelo menos uma letra já presente no tabuleiro.

   Após jogar, o jogador soma o valor da palavra formada ao seu total de pontos e retira do saco o mesmo número de letras jogadas. Se não houver letras suficientes no saco, retira apenas as que restarem.

[1] `https://pt.wikipedia.org/wiki/Scrabble`

O jogo termina quando um jogador joga todas as letras do seu conjunto e não restam letras no saco (*esgotar letras*); ou quando todos os jogadores passam de forma consecutiva. No fim do jogo, o jogador com mais pontos é o vencedor.

2 Permutação e geração de números pseudoaleatórios

Neste projeto, implementaremos um algoritmo de permutação e um algoritmo de geração de números pseudoaleatórios para baralhar as letras do saco.

2.1 Algoritmo de permutação

Dada uma sequência de letras, pretendemos obter uma nova sequência dessas letras com as suas posições trocadas. Para isto vamos usar o algoritmo de Fisher–Yates[2] que produz uma permutação imparcial, isto é, todas as permutações são igualmente prováveis.

Este algoritmo recebe uma sequência A com n elementos e determina o próximo elemento na sequência baralhada selecionando aleatoriamente um elemento ainda não processado. Formalmente:

1. Para $i \leftarrow n-1$ decrescendo até 1
2. $\quad j \leftarrow$ inteiro aleatório tal que $0 \leq j \leq i$
3. $\quad$ trocar $A[j]$ e $A[i]$

2.2 Gerador de números pseudoaleatórios

Os geradores de números pseudoaleatórios tipicamente obtêm sequências de números transformando sucessivamente o seu *estado*, isto é, o número anterior gerado. O primeiro número utilizado para inicializar o estado do gerador é normalmente conhecido como *seed*. Neste projeto, o gerador a implementar será do tipo *xorshift* [3]. Estes geradores têm um ciclo de repetição que depende da dimensão do seu estado (número de bits) e de alguns parâmetros. Para transformar o seu estado, o gerador aplica uma sequência de 3 operações bit a bit sobre o seu estado conhecidas como `xorshift`. A operação `xorshift` consiste num deslocamento de bits ou *shift* (à esquerda ou à direita), seguido de um ou-exclusivo (`xor`) bit a bit do resultado do `shift` com o próprio estado. O primeiro e o último `xorshift` são à esquerda, e o segundo à direita. Neste projeto, consideraremos um gerador de 32 bits. Este gerador utiliza valores $(13, 17, 5)$ para os deslocamentos. Em Python, para as operações de deslocamento são usados os operadores `<<` e `>>`, para o ou-exclusivo bit a bit o símbolo `^`, e para o `and` bit a bit `&`. Como o Python oferece precisão ilimitada para representar inteiros, de modo a restringir o comportamento a 32 bits, é preciso aplicar um `and` bit-a-bit com uma máscara binária de 32 `1`s (`0xFFFFFFFF`) após cada deslocamento (o prefixo `0x` indica que os dígitos que definem o inteiro estão na base hexadecimal – `F` corresponde a 4 bits a `1`). A sequência de operações para a geração de um novo valor "aleatório" será portanto:

[2] `https://en.wikipedia.org/wiki/Fisher-Yates_shuffle`
[3] `https://pt.wikipedia.org/wiki/Xorshift`

$$
\begin{aligned}
s \ \hat{}= \ (s \ << \ 13) \ \& \ 0xFFFFFFFF \\
s \ \hat{}= \ (s \ >> \ 17) \ \& \ 0xFFFFFFFF \\
s \ \hat{}= \ (s \ << \ 5) \ \ \& \ 0xFFFFFFFF
\end{aligned}
$$

3 Trabalho a realizar

O objetivo do primeiro projeto é escrever um programa em Python, correspondendo às funções apresentadas nesta secção, que permita jogar um jogo de Scrabble de dois a quatro jogadores, conforme descrito anteriormente. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso seja necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida o devem fazer, para as restantes assume-se que os argumentos estão corretos.

3.1 Representação e funções das letras

Considere que as *letras* do jogo são representadas internamente (ou seja, no seu programa) como uma cadeia de carateres de comprimento unitário e valor igual a uma das letras maiúsculas do abecedário português:

```
letras = 'A','B','C','Ç','D','E','F','G','H','I','J','L','M','N','O',
         'P','Q','R','S','T','U','V','X','Z'
```

As *palavras* são representadas como cadeias de carateres formadas por duas ou mais *letras*, por exemplo `'ISTO'`. Um *conjunto de letras* é representado internamente por um dicionário (potencialmente vazio) com o número de ocorrências das letras do abecedário português.

3.1.1 cria_conjunto: tuplo × tuplo → conjunto letras (0,5 valores)

cria_conjunto(let, occ) recebe dois tuplos de igual tamanho (potencialmente vazios), contendo o primeiro tuplo letras únicas do abecedário e o segundo tuplo inteiros positivos correspondentes ao número de ocorrências de cada uma das letras do primeiro tuplo, e devolve o conjunto de letras definido por um dicionário, como descrito anteriormente. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'cria_conjunto: argumentos inválidos'`.

```
>>> cria_conjunto(('A','B'), (14, 3))
{'A': 14, 'B':3}
>>> cria_conjunto(('A','B'), (14, -3))
Traceback (most recent call last): <...>
ValueError: cria_conjunto: argumentos inválidos
```

3.1.2 gera_numero_aleatorio: inteiro → inteiro (0,5 valores)

gera_numero_aleatorio(estado) recebe um inteiro positivo e devolve um número pseudo-aleatório utilizando o argumento fornecido como o estado inicial do gerador descrito na Secção 2.2 .

```
>>> gera_numero_aleatorio(1)
270369
>>> gera_numero_aleatorio(25)
6759192
```

3.1.3 permuta_letras: lista × inteiro → {} (1 valor)

permuta_letras(letras, estado) recebe uma lista de letras (potencialmente vazia) e um inteiro positivo representando o estado inicial do gerador pseudo-aleatório da Secção 2.2, e modifica destrutivamente a lista com os seus elementos permutados de acordo com o algoritmo descrito na Secção 2.1. Na geração dos índices de 0 até i (inclusive) deste algoritmo, deverá: 1) invocar a função `gera_numero_aleatorio` para obter um número aleatório e atualizar o estado para a seguinte chamada; 2) obter um número entre 0 e i aplicando a operação módulo (de $i + 1$) ao número aleatório obtido.

```
>>> letras = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
>>> permuta_letras(letras, 1)
>>> letras
['D', 'B', 'A', 'C', 'G', 'H', 'I', 'F', 'E', 'J']
>>> letras = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
>>> permuta_letras(letras, 2)
>>> letras
['H', 'A', 'J', 'F', 'G', 'E', 'B', 'C', 'D', 'I']
```

3.1.4 baralha_conjunto: conjunto letras × inteiro → lista (1 valor)

baralha_conjunto(conj, estado) recebe um conjunto de letras e um inteiro positivo representando o estado inicial do gerador pseudo-aleatório da Secção 2.2, e devolve uma lista *baralhada* com todas as letras contidas no conjunto de letras. Para baralhar as letras a sua função deverá: 1) construir uma lista com todas as letras contidas no conjunto considerando a ordem indicada na Secção 3.1; e 2) permutar as letras. A função não deve alterar o conjunto de letras fornecido como argumento.

```
>>> conj_letras = cria_conjunto(('A','B','C','D'), (4,2,3,3))
>>> baralha_conjunto(conj_letras, 1)
['C', 'A', 'A', 'C', 'A', 'B', 'B', 'D', 'C', 'D', 'A', 'D']
>>> baralha_conjunto(conj_letras, 3)
['A', 'B', 'A', 'D', 'B', 'C', 'A', 'C', 'D', 'C', 'D', 'A']
```

3.1.5 testa_palavra_padrao: cad. carateres × cad. carateres × conjunto letras → booleano (1 valor)

testa_palavra_padrao(palavra, padrao, conj) recebe uma *palavra*, uma cadeia de carateres padrão formada por letras e pelo carater `'.'` e um conjunto de letras, e devolve `True` caso seja possível formar a palavra fornecida substituindo os carateres `'.'` do padrão por letras do conjunto, e `False` caso contrário. Esta função não modifica os seus argumentos.

```
>>> conj_letras = cria_conjunto(('A','C','S','V'), (1,2,1,1))
>>> testa_palavra_padrao('VACA', '....', conj_letras)
False
>>> testa_palavra_padrao('VACA', '.A.', conj_letras)
False
>>> testa_palavra_padrao('VACA', '.A..', conj_letras)
True
```

3.2 Representação e funções do tabuleiro

Considere que o *tabuleiro* de dimensão 15×15 é representado internamente (ou seja, no seu programa) por uma lista com 15 listas. Cada uma das 15 listas corresponde a uma linha horizontal do tabuleiro contendo 15 elementos do tipo cadeia de carateres. Cada elemento representa uma casa do tabuleiro, podendo tomar o valor de uma das *letras* do abecedário português, assim como o carater ponto (`'.'`) para representar casas livres. Considere também que a localização de cada uma das *casas* num tabuleiro é representada internamente por um tuplo de dois números inteiros positivos entre 1 e 15 correspondendo respetivamente à linha horizontal e vertical que ocupa no tabuleiro. A casa central do tabuleiro é representada pelo tuplo `(8,8)`.

3.2.1 cria_tabuleiro: {} → tabuleiro (0,5 valores)

cria_tabuleiro() devolve um tabuleiro vazio. Considere que um tabuleiro é definido por uma lista de listas, como descrito anteriormente.

```
>>> tab = cria_tabuleiro()
>>> len(tab), len(tab[3])
15, 15
>>> tab[3][:5]
['.', '.', '.', '.', '.']
```

3.2.2 cria_casa: inteiro × inteiro → casa (0,5 valores)

cria_casa(l, c) recebe dois inteiros correspondentes a uma linha e uma coluna dum tabuleiro de Scrabble, e devolve a casa do tabuleiro. Considere que as casas no tabuleiro são definidas por um tuplo, como descrito anteriormente. Se algum dos argumentos dado for

inválido, a função deve gerar um erro com a mensagem `'cria_casa: argumentos inválidos'`.

```
>>> cria_casa(8,8)
(8, 8)
>>> cria_casa(1, 20)
Traceback (most recent call last): <...>
ValueError: cria_casa: argumentos inválidos
```

3.2.3 obtem_valor: tabuleiro × casa → cad. carateres (0,5 valores)

obtem_valor(tab, casa) recebe um tabuleiro e uma casa do tabuleiro, e devolve o valor contido nessa casa.

3.2.4 insere_letra: tabuleiro × casa × cad. carateres → tabuleiro (0,5 valores)

insere_letra(tab, casa, letra) recebe um tabuleiro, uma casa do tabuleiro e uma letra, e insere a letra na casa indicada modificando destrutivamente o tabuleiro.

```
>>> tab = cria_tabuleiro()
>>> c1, c2 = cria_casa(8,6), cria_casa(6,8)
>>> tab = insere_letra(tab, c1, 'A')
>>> tab = insere_letra(tab, c2, 'B')
>>> obtem_valor(tab, c1),  obtem_valor(tab, c2)
('A', 'B')
```

3.2.5 obtem_sequencia: tabuleiro × casa × cad. carateres × inteiro → cad. carateres (1 valor)

obtem_sequencia(tab, casa, direcao, tamanho) recebe um tabuleiro, uma casa do tabuleiro, uma direção (`'H'` para horizontal ou `'V'` para vertical) e um inteiro positivo, e devolve a cadeia de carateres de tamanho igual ao argumento inteiro fornecido formada por todos os valores nas casas do tabuleiro a partir da casa e na direção indicadas.

3.2.6 insere_palavra: tabuleiro × casa × cad. carateres × cad. carateres → tabuleiro (1 valor)

insere_palavra(tab, casa, direcao, palavra) recebe um tabuleiro, uma casa do tabuleiro, uma direção (`'H'` para horizontal ou `'V'` para vertical) e uma cadeia de carateres, e insere a palavra fornecida na casa e direção indicada modificando destrutivamente o tabuleiro.

```
>>> tab = cria_tabuleiro()
>>> _ = insere_palavra(tab, cria_casa(8,3), 'H', 'COMPUTADOR')
>>> obtem_sequencia(tab, cria_casa(8,10), 'H', 3)
'DOR'
>>> obtem_sequencia(tab, cria_casa(7,5), 'V', 4)
'.M..'
```

3.2.7 tabuleiro_para_str: tabuleiro → cad. carateres (1 valor)

tabuleiro_para_str(tab) recebe um tabuleiro e devolve a cadeia de caracteres que o representa (a representação externa ou representação "para os nossos olhos"), de acordo com o exemplo na seguinte interação.

```
>>> tab = cria_tabuleiro()
>>> _ = insere_palavra(tab, cria_casa(8,1), 'H', 'FUNDAMENTOS')
>>> _ = insere_palavra(tab, cria_casa(8,4), 'V', 'DA')
>>> _ = insere_palavra(tab, cria_casa(2,6), 'V', 'PROGRAMAÇAO')
>>> print(tabuleiro_para_str(tab))
                      1 1 1 1 1 1
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . P . . . . . . . . . |
 3 | . . . . . R . . . . . . . . . |
 4 | . . . . . O . . . . . . . . . |
 5 | . . . . . G . . . . . . . . . |
 6 | . . . . . R . . . . . . . . . |
 7 | . . . . . A . . . . . . . . . |
 8 | F U N D A M E N T O S . . . . |
 9 | . . . A . A . . . . . . . . . |
10 | . . . . . Ç . . . . . . . . . |
11 | . . . . . A . . . . . . . . . |
12 | . . . . . O . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
```

3.3 Representação e funções do jogador

Considere que um *jogador* de Scrabble é representado internamente (ou seja, no seu programa) por um dicionário com três chaves do tipo cadeia de carateres: `'id'` de valor inteiro positivo a representar a ordem do jogador; `'pontos'` de valor inteiro a representar

os pontos acumulados pelo jogador; e `'letras'` um *conjunto de letras* a representar as letras do jogador.

3.3.1 cria_jogador: inteiro × inteiro × conjunto letras → jogador (0,75 valores)

cria_jogador(ordem, pontos, conj_letras) recebe dois inteiros representando respetivamente a ordem do jogador e os pontos iniciais, e um conjunto de letras representando as letras do jogador de Scrabble, e devolve um jogador de Scrabble como descrito anteriormente. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'cria_jogador: argumentos inválidos'`.

```
>>> cria_jogador(2, 0, cria_conjunto((),()))
{'id':2, 'pontos':0, 'letras':{}}
>>> cria_jogador(5, 0, cria_conjunto((),()))
Traceback (most recent call last): <...>
ValueError: cria_jogador: argumentos inválidos
```

3.3.2 jogador_para_str: jogador → cad. carateres (0,75 valores)

jogador_para_str(jog) recebe um jogador e devolve a cadeia de caracteres que o representa (a representação externa ou representação "para os nossos olhos"), de acordo com os exemplos na seguinte interação (as letras são mostradas pela ordem indicada na Secção 3.1).

```
>>> conj1 = cria_conjunto(('C', 'A', 'O', 'N'),(2,1,3,1))
>>> conj2 = cria_conjunto(('A', 'B', 'E', 'J'),(2,1,3,1))
>>> jog1, jog2 = cria_jogador(1, 0, conj1), cria_jogador(2, 13, conj2)
>>> jogador_para_str(jog1)
'#1 (  0): A C C N O O O'
>>> jogador_para_str(jog2)
'#2 ( 13): A A B E E E J'
```

3.3.3 distribui_letra: lista × jogador → booleano (1 valor)

distribui_letra(letras, jogador) recebe uma lista de letras (potencialmente vazia) e um jogador, retira a última letra da lista acrescentando-a ao conjunto de letras do jogador e devolve `True`, ou não faz nada e devolve `False` caso a lista estiver vazia. A função modifica destrutivamente a lista de letras e o jogador passados como argumento.

```
>>> jog = cria_jogador(1, 0, cria_conjunto((),()))
>>> letras = ['A', 'B', 'C', 'A']
>>> distribui_letra(letras, jog)
```

```
True
>>> letras, jog
(['A', 'B', 'C'], {'id': 1, 'pontos': 0, 'letras': {'A': 1}})
>>> while letras: distribui_letra(letras, jog)
>>> letras, jog
([], {'id': 1, 'pontos': 0, 'letras': {'A': 2, 'C': 1, 'B': 1}})
>>> distribui_letra(letras, jog)
False
```

3.4 Funções de jogo

3.4.1 joga_palavra: tabuleiro × palavra × casa × cad. carateres × conjunto letras × booleano → tuplo (1,5 valores)

joga_palavra(tab, palavra, casa, direcao, conj_letras, primeira) recebe um tabuleiro, uma palavra, uma casa do tabuleiro, uma direção, um conjunto de letras e um booleano a identificar a primeira jogada (ver regras na Secção 1). Caso seja possível formar a palavra no tabuleiro (sem sair dos limites do tabuleiro) com as letras do conjunto fornecido e a jogada for válida de acordo com regras da Secção 1, a função insere a palavra no tabuleiro e devolve o tuplo formado pelas letras utilizadas pela ordem indicada na Secção 3.1, sem alterar o conjunto de letras. Caso contrário, a função devolve um tuplo vazio, sem alterar o tabuleiro.

```
>>> tab = cria_tabuleiro()
>>> conj_let1 = cria_conjunto(('C', 'A', 'O', 'N'),(2,1,3,1))
>>> joga_palavra(tab, 'VACA', cria_casa(8,8), 'H', conj_let1, True)
()
>>> joga_palavra(tab, 'CAO', cria_casa(7,8), 'H', conj_let1, True)
()
>>> joga_palavra(tab, 'CAO', cria_casa(8,8), 'H', conj_let1, False)
()
>>> joga_palavra(tab, 'CAO', cria_casa(8,8), 'H', conj_let1, True)
('A', 'C', 'O')
>>> conj_let2 = cria_conjunto(('P', 'E', 'O', 'R'),(2,1,3,1))
>>> joga_palavra(tab, 'PERO', cria_casa(2,2), 'V', conj_let2, False)
()
>>> joga_palavra(tab, 'PERO', cria_casa(5,10), 'V', conj_let2, False)
('E', 'P', 'R')
>>> print(tabuleiro_para_str(tab))
                      1 1 1 1 1 1
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +------------------------------+
 1 | . . . . . . . . . . . . . . . |
```

```
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . P . . . . . . |
 6 | . . . . . . . . E . . . . . . |
 7 | . . . . . . . . R . . . . . . |
 8 | . . . . . . . C A O . . . . . |
 9 | . . . . . . . . . . . . . . . |
10 | . . . . . . . . . . . . . . . |
11 | . . . . . . . . . . . . . . . |
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
```

3.4.2 processa_jogada: tabuleiro × jogador × lista × dicionário × booleano → booleano (1,5 valores)

processa_jogada(tab, jog, pilha, pontos, primeira) recebe um tabuleiro, um jogador, uma lista de letras, um dicionário com os pontos de cada uma das letras do abecedário Português e um booleano a identificar a primeira jogada (ver regras na Secção 1). A função processa o turno completo do jogador devendo apresentar a mensagem do exemplo a seguir, repetindo a mensagem até o jogador introduzir uma jogada válida num formato dependente da ação pretendida:

- **Passar**: `'P'`. Neste caso, a função devolve `False` sem alterar nenhum dos argumentos.
- **Trocar**: `'T <seq_letras>'`, sendo `<seq_letras>` a sequência de uma ou mais letras separada por espaços do conjunto de letras do jogador para trocar. Caso a jogada seja válida, a função devolve `True`, e modifica o conjunto de letras do jogador retirando novas letras do final da lista de letras (que é também modificada).
- **Jogar**: `'J <linha> <coluna> <dir> <palavra>'`. Caso a jogada seja válida, a função devolve `True` e modifica o tabuleiro, atualiza a pontuação do jogador, e atualiza o seu conjunto de letras retirando do final da lista de letras (que é também modificada).

```
>>> pontos = {
            'A': 1, 'B': 3, 'C': 2, 'Ç': 3, 'D': 2, 'E': 1,
            'F': 4, 'G': 4, 'H': 4, 'I': 1, 'J': 5, 'L': 2,
            'M': 1, 'N': 3, 'O': 1, 'P': 2, 'Q': 6, 'R': 1,
            'S': 1, 'T': 1, 'U': 1, 'V': 4, 'X': 8, 'Z': 8 }
```

```
>>> tab = cria_tabuleiro()
>>> pilha = ['S', 'B', 'P', 'E', 'C', 'E', 'E', 'S', 'J', 'D', 'I']
>>> conj1 = cria_conjunto(('A','U','O','T','X','F'),(2,1,1,1,1,1))
>>> jog1 = cria_jogador(1, 0, conj1)
>>> processa_jogada(tab, jog1, pilha, pontos, True)
Jogada J1: olà
Jogada J1: P
False
>>> processa_jogada(tab, jog1, pilha, pontos, True)
Jogada J1: T X A
True
>>> print(jogador_para_str(jog1))
#1 (  0): A D F I O T U
>>> pilha
['S', 'B', 'P', 'E', 'C', 'E', 'E', 'S', 'J']
>>> processa_jogada(tab, jog1, pilha, pontos, True)
Jogada J1: J 7 8 V LUTA
Jogada J1: J 7 8 V TOFU
True
>>> print(jogador_para_str(jog1))
#1 (  7): A D E E I J S
>>> pilha
['S', 'B', 'P', 'E', 'C']
>>> print(tabuleiro_para_str(tab))
                      1 1 1 1 1 1
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . . . . . . . . . |
 7 | . . . . . . . T . . . . . . . |
 8 | . . . . . . . O . . . . . . . |
 9 | . . . . . . . F . . . . . . . |
10 | . . . . . . . U . . . . . . . |
11 | . . . . . . . . . . . . . . . |
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
```

3.4.3 scrabble: inteiro × conjunto letras × dicionário × inteiro→ tuplo (1,5 valores)

scrabble(jogadores, saco, pontos, seed) é a função principal que permite jogar um jogo completo de Scrabble de dois a quatro jogadores. A função recebe o número de jogadores, o conjunto de letras do saco, o dicionário com a pontuação de cada letra e um inteiro positivo representando o estado inicial do gerador pseudo-aleatório da Secção 2.2, e devolve o tuplo com a pontuação final obtida pelos jogadores. O jogo começa baralhando o saco de letras e distribuindo o conjunto de 7 letras a cada um dos jogadores em ordem. O jogo desenrola-se depois conforme as regras descritas na Secção 1 e como mostrado no exemplo a seguir. O jogo termina quando todos os jogadores passam ou quando um jogador fica sem letras e o saco estiver esgotado. A função deve verificar a validade dos seus argumentos, gerando um erro com a mensagem `'scrabble: argumentos inválidos'`. Considere para este fim que o saco de letras não pode estar vazio e que o dicionário de pontos contém a pontuação de todas as letras do abecedário Português.

Exemplo

```
>>> pontos = {
            'A': 1, 'B': 3, 'C': 2, 'Ç': 3, 'D': 2, 'E': 1,
            'F': 4, 'G': 4, 'H': 4, 'I': 1, 'J': 5, 'L': 2,
            'M': 1, 'N': 3, 'O': 1, 'P': 2, 'Q': 6, 'R': 1,
            'S': 1, 'T': 1, 'U': 1, 'V': 4, 'X': 8, 'Z': 8 }
>>> saco = {
            'A': 14, 'B': 3, 'C': 4, 'Ç': 2, 'D': 5, 'E': 11,
            'F': 2, 'G': 2, 'H': 2, 'I': 10, 'J': 2, 'L': 5,
            'M': 6, 'N': 4, 'O': 10, 'P': 4, 'Q': 1, 'R': 6,
            'S': 8, 'T': 5, 'U': 7, 'V': 2, 'X': 1, 'Z': 1 }
>>> scrabble(2, saco, pontos, 32)
Bem-vindo ao SCRABBLE.
                      1 1 1 1 1 1
    1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . . . . . . . . . |
 7 | . . . . . . . . . . . . . . . |
 8 | . . . . . . . . . . . . . . . |
 9 | . . . . . . . . . . . . . . . |
10 | . . . . . . . . . . . . . . . |
11 | . . . . . . . . . . . . . . . |
12 | . . . . . . . . . . . . . . . |
```

```
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
#1 (  0): B Ç D E M O U
#2 (  0): A B I N O O V
Jogada J1: J 6 8 V DEMO
                       1 1 1 1 1 1
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . D . . . . . . . |
 7 | . . . . . . . E . . . . . . . |
 8 | . . . . . . . M . . . . . . . |
 9 | . . . . . . . O . . . . . . . |
10 | . . . . . . . . . . . . . . . |
11 | . . . . . . . . . . . . . . . |
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
#1 (  5): A B Ç E I T U
#2 (  0): A B I N O O V
Jogada J2: J 9 4 H NOIVO
                       1 1 1 1 1 1
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . D . . . . . . . |
 7 | . . . . . . . E . . . . . . . |
 8 | . . . . . . . M . . . . . . . |
 9 | . . . N O I V O . . . . . . . |
10 | . . . . . . . . . . . . . . . |
11 | . . . . . . . . . . . . . . . |
```

```
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
#1 (  5): A B Ç E I T U
#2 ( 10): A A B B C O U
Jogada J1: J 6 8 H DEITA
                      1 1 1 1 1 1
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . D E I T A . . . |
 7 | . . . . . . . E . . . . . . . |
 8 | . . . . . . . M . . . . . . . |
 9 | . . . N O I V O . . . . . . . |
10 | . . . . . . . . . . . . . . . |
11 | . . . . . . . . . . . . . . . |
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-------------------------------+
#1 ( 11): A B Ç O R S U
#2 ( 10): A A B B C O U
Jogada J2: J 6 12 V ACABOU
                      1 1 1 1 1 1
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-------------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . D E I T A . . . |
 7 | . . . . . . . E . . . C . . . |
 8 | . . . . . . . M . . . A . . . |
 9 | . . . N O I V O . . . B . . . |
10 | . . . . . . . . . . . O . . . |
```

```
11 | . . . . . . . . . . . U . . . |
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-----------------------------+
#1 ( 11): A B Ç O R S U
#2 ( 19): A A A B E M O
Jogada J1: P
                       1 1 1 1 1 1
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +-----------------------------+
 1 | . . . . . . . . . . . . . . . |
 2 | . . . . . . . . . . . . . . . |
 3 | . . . . . . . . . . . . . . . |
 4 | . . . . . . . . . . . . . . . |
 5 | . . . . . . . . . . . . . . . |
 6 | . . . . . . . D E I T A . . . |
 7 | . . . . . . . E . . . C . . . |
 8 | . . . . . . . M . . . A . . . |
 9 | . . . N O I V O . . . B . . . |
10 | . . . . . . . . . . . O . . . |
11 | . . . . . . . . . . . U . . . |
12 | . . . . . . . . . . . . . . . |
13 | . . . . . . . . . . . . . . . |
14 | . . . . . . . . . . . . . . . |
15 | . . . . . . . . . . . . . . . |
   +-----------------------------+
#1 ( 11): A B Ç O R S U
#2 ( 19): A A A B E M O
Jogada J2: P
(11, 19)
>>> scrabble(2, saco, {'A': 1}, 32)
Traceback (most recent call last): <...>
ValueError: scrabble: argumentos inválidos
```

4 Condições de Realização e Prazos

- A entrega do 1º projeto será efetuada exclusivamente por via eletrónica. Para submeter o seu projeto deverá realizar pelo menos uma atualização do repositório remoto GitLab fornecido pelo corpo docente, até às **17:00 do dia 10 de Outubro de 2025**. Depois desta hora, qualquer atualização do repositório será ignorada.

Não serão aceites submissões de projetos por outras vias sob pretexto algum.

- A solução do projeto deverá consistir apenas num único ficheiro com extensão *.py* contendo todo o código do seu projeto. Os alunos não devem alterar a estrutura do seu repositório, nem acrescentar novos ficheiros, apenas precisam editar o ficheiro com extensão *.py*.
- Cada aluno tem direito a **15 submissões sem penalização**. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.
- Será considerada para avaliação a **última** submissão (mesmo que tenha pontuação inferior a submissões anteriores). Deverá, portanto, verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada.
- Submissões que não corram nenhum dos testes automáticos por causa de pequenos erros de sintaxe ou de codificação, poderão ser corrigidos pelo corpo docente, incorrendo numa penalização de três valores.
- Não é permitida a utilização de qualquer módulo ou função não disponível *built-in* no Python 3.
- Pode, ou não, haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).
- Lembre-se de que no Técnico, a fraude académica é levada muito a sério e que a cópia em qualquer momento de avaliação (projetos incluídos) leva à reprovação na disciplina e eventualmente a um processo disciplinar. Os projetos serão submetidos a um sistema automático de deteção de cópias[4], o corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.
- A submissão do projeto por parte dos alunos, é interpretada pelo corpo docente como uma declaração de honra conforme cada aluno (ou grupo) é o autor único de todo o trabalho apresentado.

5 Submissão

A submissão do projeto de FP é realizada atualizando o repositório remoto GitLab privado fornecido pelo corpo docente a cada aluno (ou grupo de projeto). O endereço web do repositório do projeto dos alunos é `https://gitlab.rnl.tecnico.ulisboa.pt/ist-fp/fp25/prj1/(curso)/(grupo)`, onde:

- `(curso)` pode ser `leic-a`, `leic-t`, `leti` ou `leme`;
- `(grupo)` pode ser:

[4] `https://theory.stanford.edu/~aiken/moss`

– o ist-id para os alunos da LEIC-A, LEIC-T e LETI (ex. `ist190000`);

– `g` seguido dos dois dígitos que identificam o número de grupo de projeto dos alunos da LEME (ex. `g00`).

Sempre que é realizada uma nova atualização do repositório remoto é desencadeado o processo de avaliação automática do projeto e é contabilizada uma nova submissão. Quando a submissão tiver sido processada, poderá visualizar um relatório de execução com os detalhes da avaliação automática do seu projeto em `http://fp.rnl.tecnico.ulisboa.pt/fp25p1/reports/(grupo)/`. Adicionalmente, receberá no seu email o mesmo relatório. Se não receber o email ou o relatório web aparentar não ter sido atualizado, contacte com o corpo docente. Note que o sistema de submissão e avaliação não limita o número de submissões simultâneas. Um número elevado de submissões num determinado momento, poderá ocasionar a rejeição de alguns pedidos de avaliação. Para evitar problemas de último momento, **recomenda-se que submeta o seu projeto atempadamente**.

Detalhes sobre como aceder ao GitLab, configurar o par de chaves SSH, executar os comandos de Git e recomendações sobre ferramentas, encontram-se na página da disciplina na secção "Material de Apoio - Ambiente de Desenvolvimento"[5].

6 Classificação

A nota do projeto será baseada nos seguintes aspetos:

1. **Avaliação automática (80%).** A avaliação da correta execução será feita com um conjunto de testes unitários utilizando o módulo de Python pytest[6].
   Serão usados um conjunto de testes públicos (disponibilizados na página da disciplina) e um conjunto de testes privados. Como a avaliação automática vale 80% (equivalente a 16 valores) da nota do projeto, uma submissão obtém a nota máxima de 1600 pontos.
   O facto de um projeto completar com sucesso os testes públicos fornecidos não implica que esse projeto esteja totalmente correto, pois estes não são exaustivos. É da responsabilidade de cada aluno garantir que o código produzido está de acordo com a especificação do enunciado usando testes próprios adicionais, de forma a completar com sucesso os testes privados.

2. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura. Em particular, serão consideradas as seguintes componentes:

   - Boas práticas (1,5 valores): serão considerados entre outros a clareza do código, a integração de conhecimento adquirido durante a UC e a criatividade das soluções propostas.

[5] `https://fenix.tecnico.ulisboa.pt/disciplinas/FProg3112/2025-2026/1-semestre/ambiente-de-desenvolvimento`

[6] `https://docs.pytest.org/en/7.4.x/`

- Comentários (1 valor): deverão incluir a assinatura das funções definidas, comentários para o utilizador (*docstring*) e comentários para o programador.
- Tamanho de funções, duplicação de código e abstração procedimental (1 valor).
- Escolha de nomes (0,5 valores).

7 Recomendações e aspetos a evitar

As seguintes recomendações e aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

- Leia todo o enunciado, procurando perceber o objetivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer as suas questões.
- No processo de desenvolvimento do projeto, comece por implementar as várias funções pela ordem apresentada no enunciado, seguindo as metodologias estudadas na disciplina.
- Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes *outputs*, conforme ilustrado nos vários exemplos.
- Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá sentir a Lei de Murphy em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).
- Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
- Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.
- A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
- Quando o programa gerar um erro, preocupe-se em descobrir qual a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.