Aula 1 - Listas

Cada variável do programa armazena um único valor (inteiro, double, string, *etc*), que pode ser acessado ou modificado ao longo do código. Uma **lista** é uma estrutura que permite trabalhar com um conjunto de dados e permite o acesso a cada um deles.

Como exemplo, podemos ter uma lista de convidados para uma festa. Em Python, uma lista pode ser criada da seguinte forma:

```
1 # Criando uma lista de strings:
2 convidados = ["Enzo", "Valentina", "Pedro", "Maria"]
```

Os elementos da lista são indexados

Em uma lista, os elementos são **indexados** (começando pela posição zero). Na lista acima, os índices de cada item são:

Índice:	0	1	2	3
Elemento:	"Enzo"	"Valentina"	"Pedro"	"Maria"

Tabela 1: Índices dos elementos em uma lista

Os índices permitem que cada elemento seja acessado individualmente:

```
1 # Imprimindo o terceiro elemento,
2 # que se encontra na posicao 2:
3 print(convidados[2])
```

A saída da impressão acima seria:

```
Pedro
```

Percorrendo os elementos de uma lista

Quando for necessário percorrer todos os elementos de uma lista, podemos criar uma repetição que vai iterar sobre todos os elementos, através de seus índices. O comando `len(l)` retorna o tamanho de uma lista *l*:

```
1 # Percorrendo todos os elementos da lista
2 # atraves de seus indices:
3 i = 0
4
5 while i < len(convidados):
6     print( convidados[i] )
7     i += 1
```

A saída da impressão acima seria:

```
Enzo
Valentina
Pedro
Maria
```

Existe ainda o comando **for** para iterar sobre os elementos de uma lista:

```
1 for pessoa in convidados:
2     print(pessoa)
```

A saída da impressão acima também seria:

```
Enzo
Valentina
Pedro
Maria
```

Listas são mutáveis

Cada elemento da lista pode ser alterado individualmente, através de seu índice. Também é possível adicionar um novo elemento ao final da lista, através do comando append:

```
1  # Alterando o valor do terceiro elemento,
2  # que se encontra na posicao 2:
3  convidados[2] = "Ana"
4
5  # Adicionando um novo elemento ao final da lista,
6  #sem modificar os demais:
7  convidados.append("Rita")
8
9  # Imprimindo a lista:
10 for pessoa in convidados:
11     print(pessoa)
```

A saída da impressão acima seria:

```
Enzo
Valentina
Ana
Maria
Rita
```

Note que, na linha 3, alteramos o terceiro elemento da lista, que deixou de ter o valor "Pedro" e passou a ter o valor "Ana". Depois, na linha 6, adicionamos um novo elemento ao final da lista, sem modificar os demais. Por fim, na linha 9, iteramos sobre os elementos da lista. A cada iteração, escolhemos identificar cada elemento através da variável chamada *pessoa*, mas poderíamos ter utilizado qualquer outro nome de variável.

Exercícios

Crie funções e procedimentos para manipular listas já criadas, que serão recebidas como parâmetro:

- **Pertinência**: Dada uma lista e um elemento, verifique se aquele elemento pertence ou não à lista:

```
1 def pertence(x, l):
2     for elem in l:
3         if x == elem:
4             return True
5     return False
```

A função poderia ser usada da seguinte forma:

```
1 nome = input("Digite seu nome: ")
2 if pertence(nome, convidados):
3     print("Seja bem-vindo.")
4 else:
5     print("Desculpe, voce nao foi convidado.")
```

Na verdade, o python já disponibiliza o comando **in** para verificar se um elemento pertence a uma lista:

```
1 nome = input("Digite seu nome: ")
2 if nome in convidados:
3     print("Seja bem-vindo.")
4 else:
5     print("Desculpe, voce nao foi convidado.")
```

- **Sem repetição**: Dada uma lista l, crie e retorne uma nova lista que contenha todos os elementos de l sem repetição:

```
1 def semRepeticao(l):
2     l2 = []
3 
4     for elem in l:
5         if elem not in l2:
```

```
6               l2.append(elem)
7
8       return l2
```

- **Positivos**: Crie uma função que receba uma lista de inteiros, e retorna o número de elementos maiores que zero na lista:

```
1  def positivos(l):
2      n = 0
3
4      for elem in l:
5          if elem > 0
6              n += 1
7
8      return n
```

- **Torneio**: Na primeira fase de um certo torneio de futsal, todos os times jogam entre si exatamente uma vez. Crie um procedimento que receba uma lista com o nome dos times e imprima a listagem dos jogos (um jogo por linha). Exemplo de saída para `l = ["Flamengo", "Fluminense", "Vasco", "Botafogo"]`:

```
Flamengo x Fluminense
Flamengo x Vasco
Flamengo x Botafogo
Fluminense x Vasco
Fluminense x Botafogo
Vasco x Botafogo
```

```
1  def torneio(l):
2      i = 0
3
4      while i < len(l):
5          j = i+1
6          while j < len(l):
7              print(l[i], "x", l[j])
8              j = j+1
9          i = i+1
```

Aula 2 - Listas

Cuidado ao tentar alterar dois valores da lista

Se em algum momento for necessário trocar o valor de dois elementos da lista entre si, tenha cuidado. O código a seguir é incorreto:

```
1 l = [1, 2, 3, 4, 5, 6]
2
3 l[0] = l[2]
4 l[2] = l[0]
```

A ideia do código seria trocar os elementos nas posições 0 e 2 entre si. Ou seja, transformar a lista $[1, 2, 3, 4, 5, 6]$ em $[3, 2, 1, 4, 5, 6]$. Porém, na linha 3, o valor de `l[2]` é salvo por cima do valor de `l[0]` e a lista passa a ser $[3, 2, 3, 4, 5, 6]$. Com isso, o valor original de `l[0]` foi perdido, e não é mais possível salvá-lo em `l[2]`. Para fazermos isso, precisamos de uma variável auxiliar:

```
1 l = [1, 2, 3, 4, 5, 6]
2
3 aux = l[0]
4 l[0] = l[2]
5 l[2] = aux
```

Operadores e métodos das listas

A linguagem Python dispõe de vários métodos e operadores para auxiliar na manipulação de listas. Alguns deles já foram mostrados: `len(l)` para calcular a quantidade de elementos de uma lista l, `l.append(x)` para adicionar o elemento x ao final da lista l, e `x in l` para verificar se o elemento x pertence ao conjunto de elementos em l. Outros operadores comuns são:

- Somar listas, que é o mesmo que concatenar uma lista ao final da outra:

```
1 l1 = [1,2,3,4,5]
2 l2 = [6,7,8,9,10]
3
4 l = l1 + l2
```

  No trecho de código acima, a lista l será formada por todos os elementos de $l1$ e seguidos pelos elementos da lista $l2$. Ou seja, $l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$.

- Multiplicar listas: Na matemática, multiplicação $x \times y$ é o mesmo que somar $x vezes$ o valor de y. Em Python, vale o mesmo para listas:

```
1 l1 = [1,2,3]
2 l2 = 3*l1
```

No trecho acima, `3*l1` equivale ao mesmo que `l1 + l1 + l1`. Portanto, o valor final da lista $l2$ é $[1, 2, 3, 1, 2, 3, 1, 2, 3]$.

- Valores mínimos, máximos e soma:

  l = [1,2,3,4,5]

```
1 print(min(l))
2 print(max(l))
3 print(soma(l))
```

A saída do código acima é:

```
1
5
15
```

- Inserir um elemento em uma posição específica da lista: O método `append()` sempre insere o elemento ao final da lista. Já o método `l.insert(i, x)` insere o elemento x na i-ésima posição de l.

- Remover um elemento da lista: o método `l.pop(i)` remove e retorna o i-ésimo elemento da lista l. Se a posição não for passada como parâmetro (`l.pop()`), será removido o último elemento. Se não soubermos a posição do elemento a ser removido, podemos utilizar `l.remove(x)`, que remove a primeira ocorrência do elemento x na lista l. Exemplos:

```
1  l = ["Maria", "Joao", "Jose", "Aline", "Carlos"]
2
3  x = l.pop(1)
4  print("Removido", x)
5  print(l)
6
7  x = l.pop()
8  print("Removido", x)
9  print(l)
10
11 l.remove("Jose")
12 print(l)
```

A saída do código acima será:

```
Removido: Joao
["Maria", "Jose", "Aline", "Carlos"]

Removido: Carlos
["Maria", "Jose", "Aline"]

["Maria", "Aline"]
```

O programa é abortado com erro caso se tente utilizar o método `l.remove(x)` para algum x que não esteja presenta na lista l.

- Colocar os elementos da lista em ordem crescente: `l.sort()`
- Colocar os elementos da lista em ordem reversa: `l.reverse()`

```
1 l = [1, 4, 2, 9, 8]
2 l.sort()
3 l.reverse()
4
5 print(l)
```

A saída do código acima seria os itens ordenados em ordem decrescente: $[9, 8, 4, 2, 1]$

- Por fim, o método `l.count(x)` conta o número de vezes que o elemento x aparece em l.

Exercícios

- **Múltiplos**: Dada uma lista l e um número x como parâmetros, retorne uma nova lista com todos os elementos de l que forem múltiplos de x

```
1 def buscarMultiplos(x, l):
2     l2 = []
3     for y in l:
4         if y%x == 0:
5             l2.append(y)
6
7     return l2
```

- **Fatoriais**: Dado n, crie e retorne uma lista com o fatorial de todos os números até n:

```
1 from fat import fat
2
3 def fatoriais(n):
4     l = []
5     i = 1
6     while i <= n:
7         l.append(fat(i))
8         i += 1
9     return l
```

Aula 3 - Listas

Índices negativos

Também é possível acessar os elementos através de índices negativos. Neste caso, contamos a partir de -1, da direita para a esquerda:

Índice:	-5	-4	-3	-2	-1
Elemento:	"Enzo"	"Valentina"	"Ana"	"Maria"	"Rita"

Tabela 2: Índices negativos para os elementos em uma lista

Fatias de listas

Também podemos acessar um intervalo de índices na lista. Se acessarmos o valor de *convidados*$[1:4]$, por exemplo, estaremos criando uma sublista com os elementos de *convidados* do índice 1 ao índice 4 (exceto ele). Exemplo:

```
1 print(convidados[1:4])
```

A saída da impressão acima seria:

```
['Valentina', 'Ana', 'Maria']
```

Se omitirmos o primeiro índice do intevalo, os elementos serão acessados a partir do início da lista. Se o último índice for omitido, os elementos serão acessados até o final da lista:

```
1 print(convidados[:4])
2 print(convidados[1:])
```

A saída da impressão acima seria:

```
['Enzo', 'Valentina', 'Ana', 'Maria']
['Valentina', 'Ana', 'Maria', 'Rita']
```

Por fim, podemos definir um terceiro parâmetro que indica de quantos em quantos elementos queremos acessar. Exemplo:

```
1 numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
2 print(numeros[2:8:2])
3 print(numeros[::2])
```

A saída do trecho acima é:

```
[3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
```

A função range()

A função **range()** gera uma sequência de números e é usada quando o usuário precisa repetir uma ação por uma quantidade específica de vezes. É muito comum utilizá-la para iterar sobre listas quando se deseja saber o índice do elemento analisado em cada iteração. Parâmetros:

- **início**: Inteiro indicando a partir de qual valor a sequência irá começar.
- **final**: Inteiro indicando até qual valor a sequência irá se repetir (o último número gerado será o elemento anterior a ele).
- **passo**: De quantos em quantos elementos serão utilizados.

Se a função for chamado com apenas um parâmetro, consideramos que **início = 0** e o parâmetro indicao o **final** da sequência e que o **passo = 1**. Com dois parâmetros, consideramos que **passo = 1**. Exemplos:

```
1  for i in range(10):
2      print(i, end =" ")
3  print()
4
5  l = [10, 20, 30, 40]
6  for i in range(len(l)):
7      print(l[i], end =" ")
8  print()
9
10 sum = 0
```

```
11 for i in range(1, 11):
12     sum = sum + i
13 print("Soma dos 10 primeiros numeros naturais", sum)
```

A saída do código acima é:

```
0 1 2 3 4 5 6 7 8 9
10 20 30 40
Soma dos 10 primeiros numeros naturais: 55
```

Exercícios

- **Esquerda, Volver!**: *[Maratona de Programação 2006]* Este ano o sargento está tendo mais trabalho do que de costume para treinar os recrutas. Um deles é muito atrapalhado, e de vez em quando faz tudo errado – por exemplo, ao invés de virar à direita quando comandado, vira à esquerda, causando grande confusão no batalhão. O sargento tem fama de durão e não vai deixar o recruta em paz enquanto este não aprender a executar corretamente os comandos.

  No sábado à tarde, enquanto todos os outros recrutas estão de folga, ele obrigou o recruta a fazer um treinamento extra. Com o recruta marchando parado no mesmo lugar, o sargento emitiu uma série de comandos "esquerda volver!" e "direita volver!". A cada comando, o recruta deve girar sobre o mesmo ponto e dar um quarto de volta na direção correspondente ao comando. Por exemplo, se o recruta está inicialmente com o rosto voltado para a direção norte, após um comando de "esquerda volver!" ele deve ficar com o rosto voltado para a direção oeste. Se o recruta está inicialmente com o rosto voltado para o leste, após um comando "direita, volver!" ele deve ter o rosto voltado para o sul. No entanto, durante o treinamento, em que o recruta tinha inicialmente o rosto voltado para o norte, o sargento emitiu uma série tão extensa de comandos, e tão rapidamente, que até ele ficou confuso, e não sabe mais para qual direção o recruta deve ter seu rosto voltado após executar todos os comandos. Você pode ajudar o sargento?

  A função recebe como parâmetro uma lista com os comandos emitidos pelo sargento. Cada comando é representado por uma letra: 'E' (para "esquerda, volver!") e 'D' (para "direita, volver!"). A função deve imprimir uma única linha da saída, indicando a direção para a qual o recruta deve ter sua face voltada após executar a série de comandos, considerando que no início o recruta tem a face voltada para o norte. A linha deve conter uma letra entre 'N', 'L', 'S' e 'O', representando respectivamente as direções norte, leste, sul e oeste. Exemplos:

Entrada ***Comandos***	***Saída***
$['D','D','E']$	**L**
$['E','E']$	**S**

```
1  def direita(x):
2      x = x+1
3      if x == 4:
4          x = 0
5      return x
6
7  def esquerda(x):
8      x = x-1
9      if x == -1:
10         x = 3
11     return x
12
13 def recruta(l):
14     direcoes = ['Norte', 'Leste', 'Sul', 'Oeste']
15     d = 0
16
17     for comando in l:
18         if comando == 'D':
19             d = direita(d)
20         else:
21             d = esquerda(d)
22
23     print(direcoes[d])
```

Aula 4 - Matrizes

Uma lista pode conter elementos de quaiquer tipos, inclusive outras listas. Portanto, podemos utilizar listas de listas para implementarmos uma matriz em Python. Um elemento da lista contém uma linha da matriz, que por sua vez corresponde a uma lista com os elementos da coluna da matriz. Matrizes são estruturas bidimensionais (tabelas) com m linhas por n colunas muito importantes na matemática (utilizadas por exemplo para a resolução de sistemas de equações e transformações lineares) e para finanças.

Exemplo: Venda de sanduíches em uma lanchonete no primeiro trimestre do ano:

Lanche / Mês	**Janeiro**	**Fevereiro**	**Março**
Hamburguer	300	330	280
X-burguer	480	565	423

Os dados da tabela podem ser sintetizados na matriz a seguir:

$$\mathbf{M} = \begin{bmatrix} 330 & 300 & 280 \\ 480 & 565 & 423 \end{bmatrix}$$

Esta tabela pode ser armazenada em Python como uma matriz 2×3, ou seja, como uma lista que contém 2 listas, sendo que cada uma delas contém 3 elementos:

```
1 lanches = [ [330, 300, 280], [480, 565, 423] ]
```

Para acessarmos o elemento $a_{3,1}$ de uma matriz $A_{5\times6}$, por exemplo, usamos $A[2][0]$ (lembrando que os elementos de uma lista são indexados a partir de zero). O elemento $A[2]$ é a terceira linha da matriz, e portanto o elemento $A[2][0]$ é o primeiro elemento da terceira linha da matriz.

Podemos criar uma matriz utilizando um *loop*, no qual cada iteração irá criar uma linha da matriz:

```
1 M = []
2 for i in range(5):
3     linha = []
4     for j in range(7):
5         linha.append(0)
6     M.append( linha )
```

No exemplo acima, temos uma matriz $M_{5\times7}$ nula (todos os elementos iguais a zero). O loop interno pode ser simplificado (o externo, não):

```
1 M = []
2 for i in range(5):
3     M.append( 7*[0] )
```

A seguir, criamos uma função que cria uma matriz nula de m linhas e n colunas, e outra função para impressão da matriz:

```
1 def cria_matriz(m, n):
2     M = []
3     for i in range(m):
4         M.append( n*[0] )
5     return M
6
7 def imprime_matriz(M):
8     for linha in M:
9         for elemento in linha:
```

```
10              print(M[i][j], end='\t')
11          print()
```

A função a seguir recebe como parâmetros duas matrizes A e B de mesmo tamanho e retorna uma terceira matriz com o resultado de $A + B$:

```
1  def soma_matrizes(A, B):
2      nLinhas = len(A)
3      nColunas = len(A[0])
4      C = cria_matriz(nLinhas, nColunas)
5
6      for i in range(nLinhas):
7          for j in range(nColunas):
8              C[i][j] = A[i][j] + B[i][j]
9
10     return C
```

Exercícios

- **Notas**: Um professor deseja calcular o média de notas de uma turma. Faça um procedimento que leia uma matriz contendo as notas dos alunos. O procedimento começa perguntado o número m de alunos e o número n de notas, e cria uma matriz $m \times n$ que armazena as n notas de cada um dos m alunos (os valores de m, n e das notas serão lidos do teclado). A nota final de cada aluno é a média simples das suas n notas. O procedimento deve imprimir a nota de cada aluno, e no final a média geral da turma. Exemplo:

```
Aluno 1: 10.0
Aluno 2: 8.0
Aluno 3: 6.0
Média da turna: 8.0
```

```
1  def lerNotas(m, n):
2      alunos = []
3      for i in range(m):
4          notas = []
5
6          for j in range(n):
7              nota = int(input("Digite a nota {} do aluno {}: "
8                      .format(i+1,j+1)))
9              notas.append(nota)
10
```

```
11          alunos.append(notas)
12
13      return alunos
14
15  def medias():
16      m = int(input("Digite o numero de alunos: "))
17      n = int(input("Digite o numero de avaliacoes: "))
18
19      alunos = lerNotas(m, n)
20      somaTotal = 0
21      for i in range(m):
22          somaAluno = 0
23          for nota in alunos[i]:
24              somaAluno += nota
25
26          mediaAluno = somaAluno / len(alunos[i])
27          somaTotal += mediaAluno
28          print("Aluno {}: {}".format(i+1, mediaAluno))
29
30      mediaTotal = somaTotal / len(alunos)
31      print("Media da turma: {}".format(mediaTotal))
```

- **Matriz identidade**: Dada uma matriz, verifique se ela é uma matriz identidade:

```
1  def verifica_identidade(M):
2      nLinhas = len(M)
3      i = 0
4      while i < nLinhas:
5          nElem = len(M[i])
6          if nLinhas != nElem: return False
7          j = 0
8          while j < nElem:
9              if i == j and M[i][j] != 1: return False
10             if i != j and M[i][j] != 0: return False
11             j += 1
12         i += 1
13     return True
```

- **Determinante** Dada uma matriz $M_{3\times3}$, calcule o determinante de M:

```
1  def det(M):
2      diagPrincipal1 = M[0][0] * M[1][1] * M[2][2]
3      diagPrincipal2 = M[0][1] * M[1][2] * M[2][0]
```

```
    diagPrincipal3 = M[0][2] * M[1][0] * M[2][1]
    soma1 = diagPrincipal1 + diagPrincipal2 + diagPrincipal3

    diagSecundaria1 = M[0][2] * M[1][1] * M[2][0]
    diagSecundaria2 = M[0][0] * M[1][2] * M[2][1]
    diagSecundaria3 = M[0][1] * M[1][0] * M[2][2]
    soma2 = diagSecundaria1 + diagSecundaria2 + diagSecundaria3

    return soma1 - soma2
```

- **Triangular inferior da transposta de uma matriz**: Defina um procedimento que receba uma matriz quadrada M como parâmetro. A função deve transformar a matriz M na matriz transposta de M, em seguida transformar essa matriz transposta em uma matriz triangular inferior e, por fim, imprimir a matriz resultante. Observações: (1) A função não pode criar listas auxiliares. (2) Uma matriz é triangular inferior quando todos os seus elementos acima da diagonal principal são iguais a 0.

```
def troca(M, x1, y1, x2, y2):
    aux = M[x1][y1]
    M[x1][y1] = M[x2][y2]
    M[x2][y2] = aux

def transposta(M):
    i = 0
    while i < len(M):
        j = i+1
        while j < len(M):
            troca(M, i,j, j,i)
            j += 1
        i += 1

def triangInferior(M):
    i = 0
    while i < len(M):
        j = i+1
        while j < len(M):
            M[i][j] = 0
            j += 1
        i += 1

def printMatriz(M):
    for linha in M:
```

```
26          for celula in linha:
27              print(celula, end="\t")
28          print()
29      print()
30
31  def matriz_transp_inf(M):
32      transposta(M)
33      triangInferior(M)
34      printMatriz(M)
```

- **Cavalo**: Considere um jogo de xadrez onde peças são movimentadas em um tabuleiro dividido em 8 linhas e 8 colunas. Considere ainda os movimentos do cavalo: a partir de uma dada posição, conforme diagrama a seguir (onde cada possível movimento é designado por * e o cavalo é representado na célula em destaque). No esquema, o cavalo localizado na posição $(4, 3)$ pode fazer oito movimentos, sendo que um deles o levaria para a posição $(6, 4)$. Defina um procedimento que recebe uma matriz $M_{8\times8}$ como parâmetro, na qual todos os elementos são nulos exceto a posição em que o cavalo se encontra (representado pelo número 1). Encontre esta posição e imprima a quantidade de movimentos que este cavalo pode fazer, considerando que ele não pode se movimentar para uma posição fora do tabuleiro.

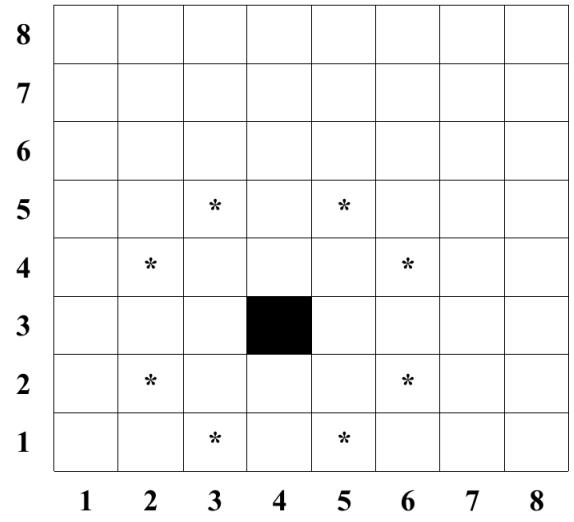


Figura 1: Movimentos possíveis do cavalo em um tabuleiro de xadrez.

- **Robô Colecionador**: *[Maratona de Programação 2010]* Um dos esportes favoritos na Robolândia é o Rali dos Robôs. Este rali é praticado em uma arena retangular gigante de N linhas por M colunas de células quadradas. Algumas das células estão vazias, algumas contêm figurinhas da Copa (muito apreciadas pelas inteligências artificiais da Robolândia) e algumas são ocupadas por pilastras que sustentam o teto da arena. Em seu percurso os robôs podem ocupar qualquer célula da arena, exceto as que contém pilastras, que bloqueiam o seu movimento. O percurso do

robô na arena durante o rali é determinado por uma sequência de instruções. Cada instrução é representada por um dos seguintes caracteres: 'D', 'E' e 'F', significando, respectivamente, "gire 90 graus para a direita", "gire 90 graus para a esquerda" e "ande uma célula para a frente". O robô começa o rali em uma posição inicial na arena e segue fielmente a sequência de instruções dada (afinal, eles são robôs!). Sempre que o robô ocupa uma célula que contém uma figurinha da Copa ele a coleta. As figurinhas da Copa não são repostas, ou seja, cada figurinha pode ser coletada uma única vez. Quando um robô tenta andar para uma célula onde existe uma pilastra ele patina, permanecendo na célula onde estava, com a mesma orientação. O mesmo também acontece quando um robô tenta sair da arena. Dados o mapa da arena, descrevendo a posição de pilastras e figurinhas, e a sequência de instruções de um robô, você deve escrever um programa para imprimir o número de figurinhas coletadas pelo robô.

A entrada contém uma matriz representando a arena e uma lista com as instruções do robô. Cada elemento da matriz pode conter um dos seguintes caracteres:

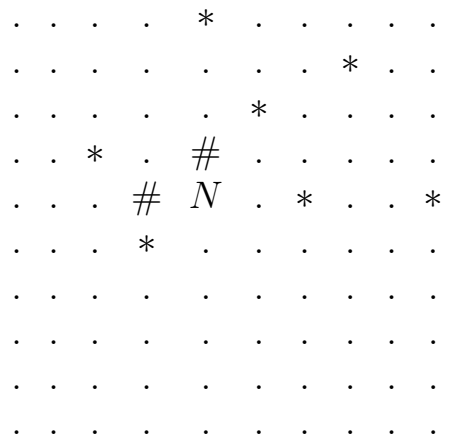
- '.' - célula normal;
- '*' - célula que contém uma figurinha da Copa;
- '#' - célula que contém uma pilastra;
- 'N', 'S', 'L', 'O' - célula onde o robô inicia o percurso (única na arena). A letra representa a orientação inicial do robô (Norte, Sul, Leste e Oeste, respectivamente). A lista de entrada contém uma sequência de S caracteres dentre 'D', 'E' e 'F', representando as instruções do robô. Exemplos:
  - Para a arena a seguir,

    $$\begin{matrix} * & * & * \\ * & N & * \\ * & * & * \end{matrix}$$

    e a sequência de instruções "DE", o robô coleciona 0 figurinhas.
  - Para a arena a seguir,

    $$\begin{matrix} . & . & . & \# \\ * & \# & O & . \\ * & . & * & . \\ * & . & \# & . \end{matrix}$$

    e a sequência de instruções "FFEFF", o robô coleciona 1 figurinha.
  - Por fim, para a arena a seguir,



e a sequência de instruções “FDFFFFFFEEFFFFFFEFDF”, o robô coleciona 3 figurinha.