1 Paradigmas, I/O, Operações, Condicionais, Repetições e Sub-rotinas

1. **Divisor**: Dados dois inteiros positivos x e y, verifique se x é divisor de y. O retorno deve ser booliano (verdadeiro ou falso).

```
1 def divisor(x, y):
2     if y%x == 0:
3         return True
4     else: return False
```

2. **Divisores**: Dado um número k, imprima todos os divisores de k.

```
1 from divisor import divisor
2
3 def divisores(k):
4     x = 1
5     while x <= k:
6         if divisor(x, k):
7             print(x, end=" ")
8         x = x+1
9     print()
```

3. **Máximo Divisor Comum (MDC)**: Dados dois números m e n, imprima o máximo divisor comum entre m e n.

```
1 from divisor import divisor
2
3 def maior(m, n):
4     if m > n:
5         return m
6     else:
7         return n
8
9 def mdc(m, n):
10    i = 1
11    maximo = 1
12    while i <= maior(m, n):
13        if divisor(i, m) and divisor(i, n):
14            maximo = i
15        i = i+1
16    print("MDC({},{}) = {}".format(m, n, maximo))
```

4. **Primo**: Dado um número x, verifique se ele é primo.

```
1  from divisor import divisor
2
3  def primo(x):
4      i = 2
5      divisores = 0
6      while i < x:
7          if divisor(i, x):
8              divisores = divisores + 1
9          i = i+1
10     # x eh primo se ele tem zero divisores alem de 1 e ele mesmo
11     return divisores == 0
```

5. **Primos**: Dado um número k, imprima todos os números primos até k.

```
1  from divisor import divisor
2  from primo import primo
3
4  def primos(k):
5      i = 2
6      while i <= k:
7          if primo(i):
8              print(i, end=" ")
9          i = i+1
10     print()
```

6. **Dama**: *[Maratona de Programação 2008]* O jogo de xadrez possui várias peças com movimentos curiosos. Uma delas é a dama, que pode se mover qualquer quantidade de casas na mesma linha, na mesma coluna, ou em uma das duas diagonais, conforme exemplifica a Figura 1.

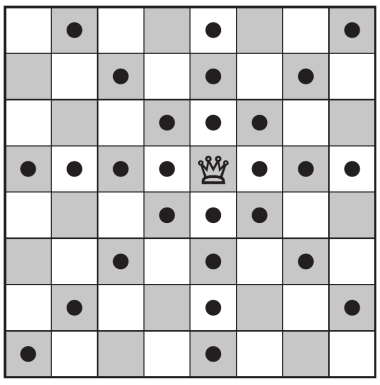

Figura 1: Movimentos possíveis da dama em um tabuleiro de xadrez.

Dada duas posições (x, y) e (m, n) em um tabuleiro de xadrez vazio (ou seja, um tabuleiro 8×8, com 64 casas), calcule e imprima a quantidade mínima de movimentos que a dama precisa fazer para ir da posição (x, y) para a posição (m, n).Exemplos:

Entrada		***Saída***
(x, y)	(m, n)	
$(4, 4)$	$(6, 2)$	1
$(3, 5)$	$(3, 5)$	0
$(5, 5)$	$(4, 3)$	2

```
1  def abs(x):
2      if x < 0: return -x
3      else: return x
4
5  def mesmaPos(x1, y1, x2, y2):
6      return x1 == x2 and y1 == y2
7
8  def alinhados(x1, y1, x2, y2):
9      return x1 == x2 or y1 == y2 or (abs(x1-x2) == abs(y1-y2))
10
11 def dama(x, y, m, n):
12     if (mesmaPos (x, y, m, n)): print(0)
13     elif (alinhados (x, y, m, n)): print(1)
14     else: print(2)
```

7. **Acerola**: *[Maratona de Programação 2008]* Natural das Antilhas, a acerola (Malpighia glabra Linn, também conhecida como cereja das Antilhas) já era apreciada pelos nativos das Américas há muitos séculos. Mas o grande interesse por essa fruta surgiu na década de 1940, quando cientistas porto-riquenhos descobriram que a acerola contém grande quantidade de ácido ascórbico (vitamina C). A acerola apresenta, em uma mesma quantidade de polpa, até 100 vezes mais vitamina C do que a laranja e o limão, 20 vezes mais do que a goiaba e 10 vezes mais do que o caju e a amora.

   Um grupo de amigos está visitando o Sítio do Picapau Amarelo, renomado produtor de acerola. Com a permissão de Dona Benta, dona do sítio, colheram uma boa quantidade de frutas, e pretendem agora fazer suco de acerola, que será dividido igualmente entre os amigos durante o lanche da tarde.

   Conhecendo o número de amigos, a quantidade de frutas colhidas, e sabendo que cada unidade da fruta é suficiente para produzir 50 ml de suco, escreva uma função que receba como parâmetros o número N de amigos e a quantidade F de frutas

colhidas, e imprima com precisão de duas casas decimais qual o volume, em litros, que cada amigo poderá tomar. Exemplos:

Entrada		***Saída***
N	*F*	
1	1	0.05
5	431	4.31
101	330	0.16

```
1 def acerola(N, F):
2     q = F*0.05
3     print("%.2f"%(q/N))
```

8. **Alarme Despertador**: *[Maratona de Programação 2009]* Daniela é enfermeira em um grande hospital e tem os horários de trabalho muito variáveis. Para piorar, ela tem sono pesado, e uma grande dificuldade para acordar com relógios despertadores. Recentemente ela ganhou de presente um relógio digital, com alarme com vários tons, e tem esperança que isso resolva o seu problema. No entanto, ela anda muito cansada e quer aproveitar cada momento de descanso. Por isso, carrega seu relógio digital despertador para todos os lugares, e sempre que tem um tempo de descanso procura dormir, programando o alarme despertador para a hora em que tem que acordar.

   No entanto, com tanta ansiedade para dormir, acaba tendo dificuldades para adormecer e aproveitar o descanso. Um problema que a tem atormentado na hora de dormir é saber quantos minutos ela teria de sono se adormecesse imediatamente e acordasse somente quando o despertador tocasse. Mas ela realmente não é muito boa com números, e pediu sua ajuda para escrever uma função que, dada a hora corrente e a hora do alarme, determine o número de minutos que ela poderia dormir. Exemplos:

Entrada				***Saída***
Hora atual	***Minuto atual***	***Hora alarme***	***Minuto alarme***	
1	5	3	5	120
23	59	0	34	35
21	33	21	10	1417

```
1 def alarme(hora_atual, minuto_atual, hora_alarme, minuto_alarme)
2     t1 = hora_atual*60 + minuto_atual
3     t2 = hora_alarme*60 + minuto_alarme
4
5     if (t1 < t2):
```

```
6           # Dorme e acorda no mesmo dia:
7           print(t2-t1)
8       else:
9           # Dorme e acorda em dias diferentes
10          print (24*60 + t2-t1)
```

2 Listas

1. **Torneio**: Na primeira fase de um certo torneio de futsal, todos os times jogam entre si exatamente uma vez. Crie uma funcao que receba uma lista com o nome dos times e imprima a listagem dos jogos (um jogo por linha).

```
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
```

2. **Troca de Cartas**: *[Maratona de Programação 2009]* Alice e Beatriz colecionam cartas de Pokémon. As cartas são produzidas para um jogo que reproduz a batalha introduzida em um dos mais bem sucedidos jogos de videogame da história, mas Alice e Beatriz são muito pequenas para jogar, e estão interessadas apenas nas cartas propriamente ditas. Para facilitar, vamos considerar que cada carta possui um identificador único, que é um número inteiro.

   Cada uma das duas meninas possui um conjunto de cartas e, como a maioria das garotas de sua idade, gostam de trocar entre si as cartas que têm. Elas obviamente não têm interesse em trocar cartas idênticas, que ambas possuem, e não querem receber cartas repetidas na troca. Além disso, as cartas serão trocadas em uma única operação de troca: Alice dá para Beatriz um sub-conjunto com N cartas distintas e recebe de volta um outro sub-conjunto com N cartas distintas.

   As meninas querem saber qual é o número máximo de cartas que podem ser trocadas. Por exemplo, se Alice tem o conjunto de cartas $[1, 1, 2, 3, 5, 7, 8, 8, 9, 15]$ e Beatriz o conjunto $[2, 2, 2, 3, 4, 6, 10, 11, 11]$, elas podem trocar entre si no máximo quatro cartas. Escreva uma função que receba como parâmetros a lista de cartas que Alice tem e a lista de cartas que Beatriz possui, e imprima o número máximo de cartas que podem ser trocadas. As cartas de Alice e Beatriz são apresentadas em ordem não decrescente. Exemplos:

Entrada		***Saída***
Cartas de Alice	***Cartas de Beatriz***	
[1000]	[1000]	0
[1, 3, 5]	[2, 4, 6, 8]	3
[1, 1, 2, 3, 5, 7, 8, 8, 9, 15]	[2, 2, 2, 3, 4, 6, 10, 11, 11]	4

```
1  def naoPertence(x, l):
2      ''' Verifica se o elemento x NAO pertence a lista l.
3      Entrada: x
4      Saida: Verdadeiro ou Falso (boolean)'''
5      for item in l:
6          if item == x:
7              return False
8
9      return True
10
11 def semRepeticao(l):
12     '''Dada uma lista ordenada l, retorna os elementos
13     de l sem repeticao.
14     Entrada: Lista
15     Saida: Lista'''
16     l2 = [l[0]]
17
18     for item in l:
19         if item != l2[-1]:
20             l2.append(item)
21
22     return l2
23
24 def menor(x, y):
25     '''Retorna o menor dentre dois valores.
26     Entrada: Dois numeros
27     Saida: O menor numero entre eles'''
28     if x < y:
29         return x
30     else:
31         return y
32
33
34 def troca(l1, l2):
35     '''
36     Entrada: Duas listas contendo o identificador das cartas
```

```
37      de cada jogador.
38      Saida: O numero de trocas que podem ser realizadas.
39      '''
40      l1 = semRepeticao(l1)
41      l2 = semRepeticao(l2)
42
43      trocasA = 0
44      for carta in l1:
45          if naoPertence(carta, l2):
46              trocasA += 1
47
48      trocasB = 0
49      for carta in l2:
50          if naoPertence(carta, l1):
51              trocasB += 1
52
53      return min(trocasA, trocasB)
```

3. **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 execu-

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

4. **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.

```
1  def troca(M, p1, p2):
```

```
2       (x1, y1) = p1
3       (x2, y2) = p2
4       aux = M[x1][y1]
5       M[x1][y1] = M[x2][y2]
6       M[x2][y2] = aux
7
8   def transposta(M):
9       i = 0
10      while i < len(M):
11          j = i+1
12          while j < len(M):
13              troca(M, (i,j), (j,i))
14              j += 1
15          i += 1
16
17  def diagInferior(M):
18      i = 0
19      while i < len(M):
20          j = i+1
21          while j < len(M):
22              M[i][j] = 0
23              j += 1
24          i += 1
25
26  def printMatriz(M):
27      for linha in M:
28          for celula in linha:
29              print(celula, end="\t")
30          print()
31      print()
32
33  def matriz_transp_inf(M):
34      transposta(M)
35      diagInferior(M)
36      printMatriz(M)
```