5.12 Implemente a função teste(), que aceita como entrada um inteiro e exibe 'Negativo', 'Zero' ou 'Positivo' dependendo do seu valor.

```
>>> teste(-3)
Negativo
>>> teste(0)
Zero
>>> teste(3)
Positivo
```

5.13 Leia cada Exercício de 5.14 a 5.22 e decida qual padrão de laço deverá ser usado em cada um.

5.14 Escreva a função mult3(), que aceite como entrada uma lista de inteiros e exiba somente os múltiplos de 3, um por linha.

```
>>> mult3([3, 1, 6, 2, 3, 9, 7, 9, 5, 4, 5])
3
6
3
9
9
```

5.15 Implemente a função vogais(), que aceite como entrada uma string e exiba os índices de todas as vogais na string. Dica: uma vogal pode ser definida como qualquer caractere na string 'aeiouAEIOU'.

```
>>> vogais('Hello WORLD')
1
4
7
```

5.16 Implemente a função índices(), que aceite como entrada uma palavra (como uma string) e uma letra de um caractere (como uma string) e retorne uma lista de índices em que a letra ocorre na palavra.

```
>>> índices('mississippi', 's')
[2, 3, 5, 6]
>>> índices('mississippi', 'i')
[1, 4, 7, 10]
>>> índices('mississippi', 'a')
[]
```

5.17 Escreva a função dobros(), que aceite como entrada uma lista de inteiros e resulte nos inteiros na lista que são exatamente o dobro do inteiro anterior na lista, um por linha.

```
>>> dobros([3, 0, 1, 2, 3, 6, 2, 4, 5, 6, 5])
2
6
4
```

5.18 Implemente a função quatro_letras(), que aceite como entrada uma lista de palavras (ou seja, strings) e retorne a sublista de todas as palavras de quatro letras na lista.

```
>>> quatro_letras(['cão', 'letra', 'pare', 'tela', 'bom', 'dica'])
['pare', 'tela', 'dica']
```

5.19 Escreva uma função emAmbas(), que aceite duas listas e retorna True se houver um item que seja comum às duas listas, e False caso contrário.

```
>>> emAmbas([3, 2, 5, 4, 7], [9, 0, 1, 3])
True
```

5.20 Escreva uma função interseção(), que aceite duas listas, cada uma não contendo valores duplicados, e retorne uma lista contendo valores que estão presentes nas duas listas (ou seja, a interseção das duas listas de entrada).

```
>>> interseção([3, 5, 1, 7, 9], [4, 2, 6, 3, 9])
[3, 9]
```

5.21 Implemente a função par(), que aceite como entrada duas listas de inteiros e um inteiro n e exiba os pares de inteiros, um da primeira lista de entrada e o outro da segunda lista de entrada, que somem n. Cada par deverá ser exibido na tela.

```
>>> par([2, 3, 4], [5, 7, 9, 12], 9)
2 7
4 5
```

5.22 Implemente a função parSoma(), que aceite como entrada uma lista de inteiros distintos lst e um inteiro n, e exiba os índices de todos os pares de valores em lst que somam n.

```
>>> parSoma([7, 8, 5, 3, 4, 6], 11)
0 4
1 3
2 5
```

Problemas

5.23 Escreva a função pagar(), que aceite como entrada um salário horário e o número de horas que um empregado trabalhou na última semana. A função deverá calcular e retornar o pagamento do empregado. O trabalho em hora extra deverá ser pago da seguinte forma: qualquer total de horas além de 40, porém menor ou igual a 60, deverá ser pago a 1,5 vez o salário horário normal. Qualquer total além de 60 deverá ser pago a duas vezes o salário horário normal.

```
>>> pagar(10, 35)
350
>>> pagar(10, 45)
475
>>> pagar(10, 61)
720
```

5.24 Escreva a função case(), que aceite uma string como entrada e retorne 'inicial maiúscula', 'inicial minúscula' ou 'desconhecido', dependendo se a string começa com uma letra maiúscula, uma letra minúscula ou algo diferente de uma letra do nosso alfabeto, respectivamente.

```
>>> case('Android')
'capitalized'
>>> case ('3M')
'unknown'
```

5.25 Implemente a função bissexto(), que aceite um argumento de entrada — um ano — e retorne True se o ano for um ano bissexto e False caso contrário. (Um ano é ano bissexto se for divisível por 4, mas não por 100, a menos que seja divisível por 400, quando será um ano bissexto. Por exemplo, 1700, 1800 e 1900 não são anos bissextos, mas 1600 e 2000 são.)

```
>>> bissexto(2008)
True
>>> bissexto(1900)
False
>>> bissexto(2000)
True
```

5.26 Pedra, Papel, Tesoura é um jogo de dois jogadores no qual cada jogador escolhe um de três itens. Se os dois jogadores escolherem o mesmo item, o jogo fica empatado. Caso contrário, as regras que determinam o vencedor são:
(a)Pedra sempre vence Tesoura (Pedra esmaga Tesoura).
(b)Tesoura sempre vence Papel (Tesoura corta Papel).
(c)Papel sempre vence Pedra (Papel cobre Pedra).
Implemente a função dpt(), que aceita a escolha ('D', 'P' ou 'T') do jogador 1 e a escolha do jogador 2, e retorna –1 se o jogador 1 vencer, 1 se o jogador 2 vencer ou 0 se houver um empate.

```
>>> dpt('D', 'P')
1
>>> dpt('D', 'T')
-1
>>> dpt('T', 'T')
0
```

5.27 Escreva a função letra2número(), que aceite como entrada uma nota de letra (A, B, C, D, F, possivelmente com um – ou um +) e retorne a nota numérica correspondente. Os valores numéricos para A, B, C, D e F são 4, 3, 2, 1, 0. Um + aumenta o valor da nota numérica em 0,3 e um – a diminui em 0,3.

```
>>> letra2número('A-')
3.7
>>> letra2número('B+')
3.3
>>> letra2número('D')
1.0
```

5.28 Escreva a função geométrica(), que aceite uma lista de inteiros como entrada e retorne True se os inteiros na lista formarem uma sequência geométrica. Uma sequência a0, a1, a2, a3, a4, …, an – 2, an – 1 é uma sequência geométrica se as razões a1/a0, a2/a1, a3/a2, a4/a3, …, an-1/an-2 forem todas iguais.

```
>>> geométrica([2, 4, 8, 16, 32, 64, 128, 256])
True
>>> geométrica([2, 4, 6, 8])
False
```

5.29 Escreva a função últimoprimeiro(), que aceite um argumento — uma lista de strings no formato <Sobrenome, Nome> — e retorne uma lista consistindo em duas listas:
(a)Uma lista de todos os nomes.
(b)Uma lista de todos os sobrenomes.

```
>>> últimoprimeiro(['Gerber, Len', 'Fox, Kate', 'Dunn, Bob'])
[['Len', 'Kate', 'Bob'], ['Gerber', 'Fox', 'Dunn']]
```

5.30 Desenvolva a função muitos(), que aceite como entrada o nome de um arquivo no diretório atual (como uma string) e gere o número de palavras de tamanho 1, 2, 3 e 4. Teste sua função no arquivo sample.txt.

```
>>> muitos('sample.txt')
Palavras de tamanho 1 : 2
Palavras de tamanho 2 : 5
Palavras de tamanho 3 : 1
Palavras de tamanho 4 : 10
```

5.31 Escreva uma função subSoma(), que aceite como entrada uma lista de números positivos e um número positivo alvo. Sua função deverá retornar True se houver três números na lista que, somados, resultem no alvo. Por exemplo, se a lista de entrada for [5, 4, 10, 20, 15, 19] e o alvo for 38, então True deve ser retornado, pois 4 + 15 + 19 = 38. Porém, se a lista de entrada for a mesma, mas o valor de alvo for 10, então o valor retornado deverá ser False, pois 10 não é a soma de três números quaisquer na lista informada.

```
>>> subSoma([5, 4, 10, 20, 15, 19], 38)
True
>>> subSoma([5, 4, 10, 20, 15, 19], 10)
False
```

5.32 Implemente a função fib(), que aceite um inteiro não negativo n como entrada e retorne o n-ésimo número de Fibonacci.

```
>>> fib(0)
1
>>> fib(4)
5
>>> fib(8)
34
```

5.33 Implemente uma função mistério(), que aceite como entrada um inteiro positivo n e respostas a esta pergunta: Quantas vezes n pode ser dividido ao meio (usando a divisão de inteiros) antes de alcançar 1? Esse valor deverá ser retornado.

```
>>> mistério(4)
2
>>> mistério(11)
3
>>> mistério(25)
4
```

5.34 Escreva uma função extrato(), que aceite como entrada uma lista de números de ponto flutuante, com números positivos representando depósitos e números negativos representando retiradas de uma conta bancária. Sua função deverá retornar uma lista de dois números de ponto flutuante; o primeiro será a soma dos depósitos, e o segundo (um número negativo) será a soma das retiradas.

```
>>> extrato([30.95, -15.67, 45.56, -55.00, 43.78])
[-70.67, 120.29]
```

5.35 Implemente a função pixels(), que aceite como entrada uma lista bidimensional de entradas de inteiros não negativos (representando os valores de pixels de uma imagem) e retorne o número de entradas que são positivas (ou seja, o número de pixels que não são escuros). Sua função deverá funcionar com listas bidimensionais de qualquer tamanho.

```
l = [[0, 156, 0, 0], [34, 0, 0, 0], [23, 123, 0, 34]]
>>> pixels(l)
5
>>> l = [[123, 56, 255], [34, 0, 0], [23, 123, 0], [3, 0, 0]]
>>> pixels(l)
7
```

5.36 Implemente a função primo(), que aceite um inteiro positivo como entrada e retorne True se ele for um número primo e False caso contrário.

```
>>> primo(2)
True
>>> primo(17)
True
>>> primo(21)
False
```

5.37 Escreva a função mssl() (sublista de soma máxima), que aceite como entrada uma lista de inteiros. Depois, ela calcula e retorna a soma da sublista de soma máxima da lista de entrada. A sublista de soma máxima é a sublista (pedaço) da lista de entrada cuja soma de entradas seja a maior. A sublista vazia é definida como tendo soma 0. Por exemplo, a sublista de soma máxima da lista

```
  [4, -2, -8, 5, -2, 7, 7, 2, -6, 5]
```

é [5, -2, 7, 7, 2] e a soma de suas entradas é 19.

```
  >>> l = [4, -2, -8, 5, -2, 7, 7, 2, -6, 5]
  >>> mssl(l)
  19
  >>> mssl([3,4,5])
  12
  >>> mssl([-2,-3,-5])
  0
```

No último exemplo, a sublista de soma máxima é a sublista vazia, pois todos os itens da lista são negativos.

5.38 Escreva a função collatz(), que aceite um inteiro positivo x como entrada e exibe a sequência de Collatz começando em x. Uma sequência de Collatz é obtida aplicando repetidamente essa regra ao número anterior x na sequência:

$$x = \begin{cases} x/2 & \text{se } x \text{ for par} \\ 3x+1 & \text{se } x \text{ for ímpar} \end{cases}$$

Sua função deverá parar quando a sequência chegar ao número 1. Nota: é uma questão ainda não resolvida se a sequência de Collatz de cada inteiro positivo sempre termina em 1.

```
>>> collatz(10)
10
5
16
8
4
2
1
```

5.39 Escreva a função exclamação(), que aceite como entrada uma string e a retorne com esta modificação: cada vogal é substituída por quatro cópias consecutivas de si mesmo e um ponto de exclamação (!) é acrescentado no final.

```
>>> exclamação('argh')
'aaaargh!'
exclamação('hello')
'heeeelloooo!'
```

5.40 A constante π é um número irracional com valor aproximado de 3,1415928… O valor exato de π é igual a esta soma infinita:

$$\pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + \ldots$$

Podemos ter uma boa aproximação de π calculando a soma dos primeiros poucos termos. Escreva uma função aproxPi(), que aceite como entrada um erro como valor de ponto flutuante e aproxime a constante n dentro do erro, calculando a soma indicada, termo por termo, até que a diferença entre a soma atual e a anterior (com um termo a menos) não seja maior do que o erro. A função deverá retornar a nova soma.

```
>>> aproxPi(0.01)
3.1611986129870506
>>> aproxPi(0.0000001)
3.1415928535897395
```

5.41 Um polinômio de grau n com coeficientes a0, a1, a2, a3, …, an é a função

$$p(x) = a0 + a1x + a2x2 + a3 * x3 + \ldots + an * xn$$

Essa função pode ser avaliada em diferentes valores de x. Por exemplo, se p(x) = 1 + 2x + x2, então p(2) = 1 + 2*2 + 22 = 9. Se p(x) = 1 + x2 + x4, então p(2) = 21 e p(3) = 91.
Escreva uma função poli() que aceite como entrada uma lista de coeficientes a0, a1, a2, a3, …, an de um polinômio p(x) e um valor x. A função retornará p(x), que é o valor do polinômio quando avaliado em x. Observe que o uso a seguir é para os três exemplos mostrados.

```
>>> poli([1, 2, 1], 2)
9
>>> poli([1, 0, 1, 0, 1], 2)
21
>>> poli([1, 0, 1, 0, 1], 3)
91
```

5.42 Implemente a função fatPrimo(), que aceite como entrada um inteiro positivo n e retorne uma lista contendo todos os números nos fatores primos de n. (Os fatores primos de um inteiro positivo n é a lista exclusiva de números primos cujo produto é n.)

```
>>> fatPrimo(5)
[5]
>>> fatPrimo(72)
[2, 2, 2, 3, 3]
```

5.43 Implemente a função linhaPar(), que aceite uma lista bidimensional de inteiros e retorne True se cada linha da tabela totalizar um número par e False caso contrário (isto é, se alguma linha totalizar um número ímpar).

```
>>> linhaPar([[1, 3], [2, 4], [0, 6]])
True
>>> linhaPar([[1, 3], [3, 4], [0, 5]])
False
>>> linhaPar([[1, 3, 2], [3, 4, 7], [0, 6, 2]])
```

True
>>> linhaPar([[1, 3, 2], [3, 4, 7], [0, 5, 2]])
False

5.44 Uma cifra de substituição para os dígitos 0, 1, 2, 3, …, 9 substitui cada dígito em 0, 1, 2, 3, …, 9 por outro dígito em 0, 1, 2, 3, …, 9. Ele pode ser representado como uma string de 10 dígitos, especificando como cada dígito em 0, 1, 2, 3, …, 9 é substituído. Por exemplo, a string de 10 dígitos '3941068257' especifica uma cifra de substituição em que o dígito 0 é substituído pelo dígito 3, 1 por 9, 2 por 4 e assim por diante. Para criptografar um inteiro não negativo, substitua cada um de seus dígitos pelo dígito especificado na chave de criptografia.
Implemente a função cifra(), que aceite como entrada uma chave de string de 10 dígitos e uma string de dígitos (ou seja, o texto claro a ser criptografado) e retorne a criptografia do texto claro.
>>> cifra('3941068257', '132')
'914'
>>> cifra('3941068257', '111')
'999'

5.45 A função médiamédia() aceita como entrada uma lista cujos itens são listas de três números. Cada lista de três números representa as três notas que determinado estudante recebeu para um curso. Por exemplo, aqui está uma lista de entrada para uma turma de quatro estudantes:
[[95,92,86], [66,75,54],[89, 72,100],[34,0,0]]
A função médiamédia() deverá exibir duas linhas na tela. A primeira linha terá uma lista contendo a nota média de cada aluno. A segunda linha terá apenas um número: a nota média da turma, definida como a média das notas médias de todos os estudantes.
>>> médiamédia([[95, 92, 86], [66, 75, 54],[89, 72, 100], [34, 0, 0]])
[91.0, 65.0, 87.0, 11.333333333333334]
63.5833333333

5.46 Uma inversão em uma sequência é um par de entradas que estão fora da ordem. Por exemplo, os caracteres F e D formam uma inversão na string 'ABBFHDL', pois F aparece antes de D; o mesmo ocorre com os caracteres H e D. O número total de inversões em uma sequência (ou seja, o número de pares que estão fora da ordem) é uma medida de como a sequência está desordenada. O número total de inversões em 'ABBFHDL' é 2. Implemente a função inversões(), que aceite uma sequência (ou seja, uma string) de caracteres maiúsculos de A até Z e retorne o número de inversões na sequência.
>>> inversões('ABBFHDL')
2
>>> inversões('ABCD')
0
>>> inversões('DCBA')
6

5.47 Escreva a função d2x(), que aceite como entrada um inteiro não negativo n (na representação decimal padrão) e um inteiro x entre 2 e 9, e retorne uma string de dígitos que corresponda à representação de n na base x.
>>> d2x(10, 2)
'1010'
>>> d2x(10, 3)

'101'
>>> d2x(10, 8)
'12'

5.48 Considere que lista1 e lista2 sejam duas listas de inteiros. Dizemos que lista1 é uma sublista de lista2 se os elementos em lista1 aparecerem em lista2 na mesma ordem em que aparecem em lista1, mas não necessariamente de forma consecutiva. Por exemplo, se lista1 for definida como
[15, 1, 100]
e lista2 for definida como
[20, 15, 30, 50, 1, 100]
então, lista1 é uma sublista de lista2, porque os números em lista1 (15, 1 e 100) aparecem na lista2 na mesma ordem. Porém, a lista
[15, 50, 20]
não é uma sublista de lista2.
Implemente a função sublista(), que aceite como entrada as listas lista1 e lista2 e retorne True se a lista1 for uma sublista de lista2, e False caso contrário.
>>> sublista([15, 1, 100], [20, 15, 30, 50, 1, 100])
True
>>> sublista([15, 50, 20], [20, 15, 30, 50, 1, 100])
False

5.49 O método de Heron é um método dos gregos antigos usado para calcular a raiz quadrada de um número n. O método gera uma sequência de números que representa aproximações cada vez melhores para . O primeiro número na sequência é uma escolha qualquer; cada outro número na sequência é obtido a partir do número anterior ant usando a fórmula

$$\frac{1}{2}(\mathtt{ant} + \frac{n}{\mathtt{ant}})$$

Escreva a função heron(), que aceite como entrada dois números: n e erro. A função deverá começar com uma escolha inicial de 1.0 para e depois gerar repetidamente aproximações melhores até que a diferença (mais precisamente, o valor absoluto da diferença) entre as aproximações sucessivas seja, no máximo, erro.
>>> heron(4.0, 0.5)
2.05
>>> heron(4.0, 0.1)
2.000609756097561