

Caderno de Exercícios de Programação

Elementos Básicos

1. Considere a seguinte gramática em notação BNF, cujo símbolo inicial é “palavra”:

```
<palavra> ::= <sílaba> <sílaba>  
<sílaba> ::= <vogal> <consoante> | <consoante> <vogal>  
<vogal> ::= a | e | i | o | u  
<consoante> ::= b | c | d | f | g | h | j | l | m | n | p | q | r  
| s | t | v | x | z
```

- a. Indique os símbolos terminais e os símbolos não terminais da gramática.
- b. Indique, justificando, quais das expressões seguintes pertencem ou não pertencem ao conjunto de palavras da linguagem definida pela gramática.
 - i. asno
 - ii. cria
 - iii. gato
 - iv. leao
 - v. OVOS
 - vi. tu
 - vii. vaca

2. Considere a seguinte gramática em notação BNF, cujo símbolo inicial é “S”:

```
<S> ::= <A> <B>  
<A> ::= <x> | <x> <A>  
<B> ::= <y> | <y> <B>  
<x> ::= A | B | C | D  
<y> ::= 1 | 2 | 3 | 4
```

- a. Diga quais são os símbolos terminais e quais são os símbolos não terminais da gramática.
- b. Quais das seguintes frases pertencem à linguagem definida pela gramática? Justifique a sua resposta.
 - i. ABCD
 - ii. 1CD
 - iii. A123CD
 - iv. AAAAB12
- c. Suponha que a terceira regra desta gramática era definida do seguinte modo:
 $\langle B \rangle ::= \langle y \rangle^+$
Será que as frases definidas pela gramática eram as mesmas?

3. Considere a seguinte gramática em notação BNF em que o símbolo inicial é “Princ”:

$\langle \text{Princ} \rangle ::= a \langle \text{Meio} \rangle a$
 $\langle \text{Meio} \rangle ::= b \langle \text{Fim} \rangle b$
 $\langle \text{Fim} \rangle ::= c \mid c \langle \text{Fim} \rangle$

- Diga quais são os símbolos terminais e os símbolos não-terminais desta gramática.
 - Descreva informalmente as frases definidas pela gramática.
4. Escreva uma gramática em notação BNF para definir números inteiros positivos. Um número inteiro positivo é representado como uma sequência arbitrariamente longa de dígitos de zero a nove. Considere que, à exceção do número inteiro positivo 0, o primeiro dígito de um número inteiro positivo não poderá ser 0. Por exemplo, de acordo com esta gramática 023 não é um número inteiro positivo.
5. Considere a seguinte gramática em notação BNF, em que o símbolo inicial é “operação”:

$\langle \text{operação} \rangle ::= (\langle \text{argumento} \rangle \langle \text{operador} \rangle \langle \text{argumento} \rangle)$
 $\langle \text{operador} \rangle ::= + \mid - \mid * \mid /$
 $\langle \text{argumento} \rangle ::= \langle \text{dígito} \rangle^+$
 $\langle \text{dígito} \rangle ::= 2 \mid 4 \mid 6 \mid 8 \mid 0$

- Indique os símbolos terminais e os símbolos não terminais da gramática.
 - Indique, justificando, quais das expressões seguintes pertencem ou não pertencem ao conjunto de operações da linguagem definida pela gramática.
 - $(1+2)$
 - $(2+-)$
 - $(24*06)$
 - $2*0$
 - $(84+)$
 - $(0/0)$
6. Escreva uma gramática em notação BNF que gera frases constituídas pelos símbolos c, a, r, d . As frases da linguagem começam pelo símbolo c , o qual é seguido por uma ou mais ocorrências dos símbolos a e d , e terminam no símbolo r . Por exemplo $caaddaar$ e $cdrr$ são frases da linguagem, cd e $cdrrr$ não o são.
7. Escreva uma gramática em notação BNF para definir os códigos postais de Portugal. Um código postal de Portugal corresponde a um número inteiro de 4 dígitos, o primeiro dos quais diferente de zero, seguido de um hífen (“-”), seguido de um inteiro de 3 dígitos. Por exemplo:

1049-001
2780-990

8. Considere a linguagem cujas frases são constituídas por um ou mais dos símbolos A, B e C (por qualquer ordem), sendo seguidas por um ou mais dos símbolos 1, 2 e 3 (por qualquer ordem). Por exemplo, A1, ABAAAAC333311 são frases da linguagem e A e 1A1 não o são.
- Escreva uma gramática em notação BNF para a linguagem apresentada no exercício anterior.
 - Diga quais são os símbolos terminais e não terminais da sua linguagem.
9. Escreva um programa em Python que pede ao utilizador que lhe forneça dois números (x e y) e que escreve o valor de $(x + 3 * y) * (x - y)$. O seu programa deve gerar uma interação como a seguinte:

```
Vou pedir-lhe dois numeros
Escreva o primeiro numero, x = 5
Escreva o segundo numero, y = 6
O valor de (x + 3 * y) * (x - y) e: -23
```

10. Escreva um programa em Python que lê valores correspondentes a uma distância percorrida (em Km) e o tempo necessário para a percorrer (em minutos), e calcula a velocidade média em:
- Km / h
 - m/s
11. Escreva um programa em Python que pede ao utilizador que lhe forneça um inteiro correspondente a um número de segundos e que calcula o número de dias correspondentes a esse número de segundos. O seu programa deve permitir a interação:

```
Escreva um número de segundos
? 65432998
O número de dias correspondentes é 757.3263657407407
```

12. Escreva um programa que lê um número inteiro correspondente a um certo número de segundos e que escreve o número de dias, horas, minutos e segundos correspondentes a esse número. Por exemplo:

```
Escreva o número de segundos 345678
dias: 4 horas: 0 mins: 1 segs: 18
```

13. Escreva um programa em Python que lê cinco números reais e calcula a sua média e o seu desvio padrão. A média, \bar{x} , e o desvio padrão, σ , de cinco números x_1, \dots, x_5 são dados respetivamente por:

$$\bar{x} = \frac{\sum_{i=1}^5 x_i}{5}$$

$$\sigma = \sqrt{\frac{1}{4} \sum_{i=1}^5 (x_i - \bar{x})^2}$$

A primeira linha do seu programa deve ser `from math import sqrt`. Esta instrução importa a função `sqrt` que calcula a raiz quadrada. Por exemplo, `sqrt(4)` tem o valor `2.0`.

14. Escreva um programa em Python que lê três números e que diz qual o maior dos números lidos.
15. Escreva um programa em Python que lê o número de horas que um empregado trabalhou numa dada semana e o seu salário/hora e calcula o ordenado semanal tendo em conta as horas extraordinárias. O salário é calculado do seguinte modo: se o número de horas for menor que 40 então salário é dado pelo produto do número de horas pelo salário hora, em caso contrário recebe horas extraordinárias as quais são pagas a dobrar.
16. Escreva um programa em Python que pede ao utilizador que lhe forneça uma sucessão de inteiros correspondentes a valores em segundos e que calcula o número de dias correspondentes a cada um desses inteiros. O programa termina quando o utilizador fornece um número negativo. O seu programa deve possibilitar a seguinte interação:

```
Escreva um número de segundos
(um número negativo para terminar)
? 45
O número de dias correspondente é 0.0005208333333333333
Escreva um número de segundos
(um número negativo para terminar)
? 6654441
O número de dias correspondente é 77.01899305555555
Escreva um número de segundos
(um número negativo para terminar)
? -1
>>>
```

17. Escreva um programa em Python que lê uma sequência de dígitos, sendo cada um dos dígitos fornecido numa linha separada, e calcula o número inteiro composto por esses dígitos, pela ordem fornecida. Para terminar a sequência de dígitos é fornecido ao programa o inteiro `-1`. O seu programa deve permitir a interação:

```
Escreva um dígito
(-1 para terminar)
? 3
Escreva um dígito
(-1 para terminar)
? 2
Escreva um dígito
(-1 para terminar)
? 5
Escreva um dígito
(-1 para terminar)
? 7
Escreva um dígito
(-1 para terminar)
? -1
O número é: 3257
```

18. Escreva um programa em Python que lê um número inteiro positivo e calcula o número obtido do número lido que apenas contém os seus dígitos ímpares. Por exemplo:

```
Escreva um inteiro
? 785554
Resultado: 7555
```

19. Escreva um programa em Python que lê um número inteiro positivo e produz o número correspondente a inverter a ordem dos seus dígitos. Por exemplo:

```
Escreva um inteiro positivo
? 7633256
O número invertido é 6523367
```

20. Escreva um programa em Python que calcula o valor da soma

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

para um dado valor de x e de n . O seu programa deve ter em atenção que o i -ésimo termo da soma pode ser obtido do termo na posição $i - 1$, multiplicando-o por x/i . O seu programa deve permitir a interação:

```
Qual o valor de x
? 2
Qual o valor de n
? 3
O valor da soma é 6.333333333333333
```

21. Escreva um programa em Python que pede ao utilizador que lhe forneça um número e que escreva a tabuada da multiplicação para esse número. O seu programa deve gerar uma interação como a seguinte:

```
Escreva um numero para eu escrever a tabuada da multiplicação
Num -> 8
8 x 1=8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

22. Escreva um programa que lê um inteiro e calcula a soma dos seus dígitos.

23. Escreva um programa que lê uma série de dígitos (terminando com -1) e calcula o inteiro que tem esses dígitos. Por exemplo, lendo os dígitos 1 5 4 5 8 -1, calcula o número inteiro 15458.

24. Escreva um programa que lê um número e cria uma capicua cuja primeira metade é o número lido. Por exemplo:

```
Escreva um número
-> 347
347743
```

25. Dado um conjunto de n inteiros representando notas de alunos, escreva um programa em Python para determinar quantos tiveram nota positiva. Modifique o seu programa de modo a também calcular qual a percentagem de notas positivas.

26. Escreva um programa que lê um número inteiro e determina quantas vezes aparecem dois zeros seguidos. Por exemplo:

```
Escreva um inteiro
? 98007640003
O numero tem 3 zeros seguidos
```

27. Escreva um programa em Python que lê uma quantia em Euros e calcula o número de notas de 50€, 20€, 10€, 5€ e moedas de 2€, 1€, 50 cêntimos, 20 cêntimos, 10 cêntimos, 5 cêntimos, 2 cêntimos e 1 cêntimo, necessário para perfazer, essa quantia, utilizando sempre o máximo número de notas e moedas para cada quantia, da mais elevada, para a mais baixa.

28. Escreva um programa em Python que escreve o seguinte:

```
1x8+1=9
12 x 8 + 2 = 98
123 x 8 + 3 = 987
1234 x 8 + 4 = 9876
12345 x 8 + 5 = 98765
123456 x 8 + 6 = 987654
1234567 x 8 + 7 = 9876543
12345678 x 8 + 8 = 98765432
123456789 x 8 + 9 = 987654321
```

Os valores do primeiro termo da multiplicação e o resultado devem ser calculados pelo seu programa.

Funções, Ciclos Contados e Listas

29. Escreva a função `cinco` que tem o valor `True` se o seu argumento for 5 e `False` no caso contrário. Não pode utilizar uma instrução `if`.

30. Defina a função `horas_dias` que recebe um inteiro correspondente a um certo número de horas e que tem como valor um número real que traduz o número de dias correspondentes ao seu argumento. Por exemplo:

```
>>> horas_dias(48)
2.0
>>> horas_dias(10)
0.4166666666666667
```

31. Defina a função `area_circulo` que recebe o valor do raio de um círculo e tem como valor a área do círculo. Note-se que a área do círculo cujo raio é π é dada por πr^2 . Use o valor 3.14 para o valor de π .

32. Utilizando a função `area_circulo` do exercício anterior, escreva a função `area_coroa` que recebe dois argumentos, `r1` e `r2`, e tem como valor a área da coroa circular de raio interior `r1` e raio exterior `r2`. A sua função deverá gerar um erro de valor (`ValueError`) se o valor de `r1` for maior que o valor de `r2`.

33. Escreva a função `bissexto` que recebe um número inteiro correspondente a um ano e que devolve `True` se o ano for bissexto e `False` em caso contrário. Um ano é bissexto se for divisível por 4 e não for divisível por 100, a não ser que seja também divisível por 400. Por exemplo, 1984 é bissexto, 1100 não é, e 2000 é bissexto. Por exemplo:

```
>>> bissexto(1984)
True
>>> bissexto(1985)
False
>>> bissexto(2000)
True
```

34. Defina a função `dias_mes` que recebe uma cadeia de caracteres, correspondentes às 3 primeiras letras (minúsculas) do nome de um mês e um ano, e tem como valor um número inteiro correspondendo ao número de dias desse mês. No caso de uma cadeia de caracteres inválida, a sua função deverá gerar um erro de valor (`ValueError`). Use a função `bissexto` do exercício anterior. A sua função deve permitir gerar a interação:

```
>>> dias_mes('jan', 2017)
31
>>> dias_mes('fev', 2016)
29
>>> dias_mes('MAR', 2017)
ValueError: Mes não existe
```

35. Quando se efectua um depósito a prazo de uma quantia q com uma taxa de juros j ($0 < j < 1$), o valor do depósito ao fim de n anos é dado por:

$$1 \times (1 + j)^n$$

- a. Escreva a função `valor` que recebe como argumentos um número inteiro positivo `q` correspondente à quantia depositada, um real `j` no intervalo $]0,1[$ correspondente à taxa de juros e um inteiro positivo `n` correspondente ao número de anos que o dinheiro está a render, e, verificando a correcção dos argumentos, devolve um real correspondente ao valor do depósito ao fim desse número de anos. Caso os argumentos não estejam corretos, deverá gerar um erro. Por exemplo:

```
>>> valor(100, 0.03, 4)
112.55088100000002
```

- b. Usando a função da alínea anterior, escreva uma função que calcula ao fim de quantos anos consegue duplicar o seu dinheiro. Não é necessário validar os dados de entrada. Por exemplo:

```
>>> duplicar(100, 0.03) 24
```

36. Um número *primo* é um número inteiro maior do que 1 que apenas é divisível por 1 e por si próprio. Por exemplo, 5 é primo porque apenas é divisível por si próprio e por um, ao passo que 6 não é primo pois é divisível por 1, 2, 3, e 6. Os números primos têm um papel muito importante tanto em Matemática como em Informática. Um método simples, mas pouco eficiente, para determinar se um número, n , é primo consiste em testar se n é múltiplo de algum número entre 2 e \sqrt{n} . Usando este processo, escreva a função `primo` que recebe um número inteiro e tem o valor `True` apenas se o seu argumento for primo.

37. Um número n é o n -ésimo *primo* se for primo e existirem $n - 1$ números primos menores que ele. Usando a função `primo` do exercício anterior, escreva a função `n_esimo_primo` que recebe como argumento um número inteiro, n , e devolve o n -ésimo número primo.

38. A *congruência de Zeller* é um algoritmo inventado pelo matemático alemão Julius Christian Zeller (1822–1899) para calcular o dia da semana para qualquer dia do calendário. Para o nosso calendário, o *calendário Gregoriano*, a congruência de Zeller é dada por:

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \bmod 7$$

em que h é o dia da semana (0 = Sábado, 1 = Domingo, . . .), q é o dia do mês, m é o mês (3 = março, 4 = abril, ..., 14 = fevereiro) – os meses de janeiro e fevereiro são contados como os meses 13 e 14 do ano anterior, K é o ano do século ($ano \bmod 100$), J é o século ($\lfloor ano/100 \rfloor$)¹. Esta expressão utiliza a função matemática, *chão*, denotada por $\lfloor x \rfloor$, a qual converte um número real x no maior número inteiro menor ou igual a x . A definição formal desta função é $\lfloor x \rfloor = \max\{m \in \mathbb{Z} \mid m \leq x\}$. A

¹ Este século é baseado no século zero, nesta definição, os séculos dos anos 1995 e 2000 são 19 e 20, respetivamente. Isto não deve confundido com a numeração comum do século que indica 20 para ambos os casos.

expressão utiliza também a função módulo, em que $a \bmod b$ representa o resto da divisão de a por b .

Escreva uma função em Python, chamada `dia_da_semana`, que recebe três inteiros correspondentes a um dia, um mês e um ano e que devolve o dia da semana em que calha essa data. A sua função deve utilizar outras funções auxiliares a definir por si. Por exemplo:

```
>>> dia_da_semana(18, 1, 2014)
'sabado'
```

39. Considere qualquer inteiro de 3 algarismos, desde que a diferença entre o seu primeiro e último dígito seja superior a 1. Seja este número n . Inverta os algarismos de n , obtendo n_i . Subtraia o maior do menor, obtendo n_s , ou seja $n_s = |n - n_i|$. Inverta os algarismos de n_s , obtendo n_{s_i} . Pode verificar que $n_s + n_{s_i} = 1089$. Por exemplo, suponhamos que $n = 246$, então $n_i = 642$, $n_s = 396$, $n_{s_i} = 693$ e $n_s + n_{s_i} = 396 + 693 = 1089$. Escreva uma função que traduza este mistério. A sua função deve garantir que o argumento é correto. Por exemplo:

```
>>> misterio(246)
1089
>>> misterio(131)
'Condições não verificadas'
```

40. Considere as instruções em Python em baixo. Escreva instruções equivalentes utilizando um ciclo `for`.

```
soma = 0
i = 20
while i > 0:
    soma = soma + 1
    i = i - 2
print('Soma =', soma)
```

41. Escreva a função `explode` que recebe um número inteiro positivo, verificando a correção do seu argumento, e devolve o tuplo contendo os dígitos desse número, pela ordem em que aparecem no número. Por exemplo:

```
>>> explode(34500)
(3, 4, 5, 0, 0)
>>> explode(3.5)
ValueError: explode: argumento não inteiro
```

42. Escreva a função `implode` que recebe um tuplo contendo algarismos, verificando a correção do seu argumento, e devolve o número inteiro contendo os algarismos do tuplo, pela ordem em que aparecem. Escreva duas versões da sua função, uma utilizando um ciclo `while` e outra utilizando um ciclo `for`. Por exemplo:

```
>>> implode((3, 4, 0, 0, 4))
34004
>>> implode((2, 'a', 5))
ValueError: implode: elemento não inteiro
```

43. Escreva a função `filtra_pares` que recebe um tuplo contendo algarismos, verificando a correção do seu argumento, e devolve o tuplo contendo apenas os algarismos pares. Por exemplo:

```
>>> filtra_pares((2, 5, 6, 7, 9, 1, 8, 8))
(2, 6, 8, 8)
```

44. Recorrendo às funções `explode`, `implode` e `filtra_pares`, defina a função `algarismos_pares` que recebe um inteiro e devolve o inteiro que apenas contém os algarismos pares do número original. Por exemplo:

```
>>> algarismos_pares(6643399766641)
6646664
```

45. Escreva a função `num_para_seq_cod` que recebe um número inteiro positivo maior do que zero e que devolve um tuplo contendo os algarismos codificados desse número do seguinte modo: (a) cada algarismo par é substituído pelo número par seguinte, entendendo-se que o número par seguinte a 8 é o 0; (b) cada algarismo ímpar é substituído pelo número ímpar anterior, entendendo-se que o número ímpar anterior a 1 é o 9. Por exemplo:

```
>>> num_para_seq_cod(1234567890)
(9, 4, 1, 6, 3, 8, 5, 0, 7, 2)
```

46. Duas palavras de igual comprimento dizem-se “amigas” se o número de posições em que os respetivos caracteres diferem for inferior a 10%. Escreva a função `amigas` que recebe como argumentos duas cadeias de caracteres e devolve verdadeiro se os seus argumentos corresponderem a palavras amigas e falso em caso contrário. Por exemplo:

```
amigas('amigas', 'amigas')
True
amigas('amigas', 'asigos')
False
```

47. Defina a função `junta_ordenados`, que recebe dois tuplos contendo inteiros, ordenados por ordem crescente, e devolve um tuplo também ordenado com os elementos dos dois tuplos. Por exemplo:

```
>>> junta_ordenados((2, 34, 200, 210), (1, 23))
(1, 2, 23, 34, 200, 210)
```

48. Considere a seguinte gramática em notação BNF, cujo símbolo inicial é “`idt`”:

```
<idt> ::= <letras><números>
<letras> ::= <letra> | <letra> <letras>
<numeros> ::= <num> | <num> <numeros>
<letra> ::= A | B | C | D
<num> ::= 1 | 2 | 3 | 4
```

Escreva a função `reconhece`, que recebe como argumento uma cadeia de caracteres e devolve verdadeiro se o seu argumento corresponde a uma frase da linguagem definida pela gramática e falso em caso contrário. Por exemplo:

```
>>> reconhece('A1')
True
>>> reconhece('ABBBBCDDDD23311')
True
>>> reconhece('ABC12C')
False
```

49. Um método básico para codificar um texto corresponde a isolar os caracteres nas posições pares para um lado e os caracteres nas posições ímpares para outro, juntando depois as duas partes anteriormente obtidas. Por exemplo, o texto `abcde` é codificado por `acebd`.

- a. Defina uma função que codifica uma cadeia de caracteres de acordo com o algoritmo apresentado. Não é necessário validar os dados de entrada. Por exemplo:

```
>>> codifica('abcde')
'acebd'
```

- b. Defina uma função que decodifica uma cadeia de caracteres de acordo com o algoritmo apresentado. Não é necessário validar os dados de entrada. Por exemplo:

```
>>> decodifica('acebd')
'abcde'
```

50. Escreva a função de dois argumentos, `lista_codigos`, que recebe uma cadeia de caracteres e que devolve a lista contendo os códigos “Unicode” de cada um dos caracteres da lista. Por exemplo:

```
>>> lista_codigos('bom dia')
[98, 111, 109, 32, 100, 105, 97]
```

51. Escreva a função de dois argumentos, `remove_multiplos`, que recebe uma lista de inteiros e um inteiro e que devolve a lista que resulta de remover todos os múltiplos do segundo argumento da lista original. Por exemplo:

```
>>> remove_multiplos([2, 3, 5, 9, 12, 33, 34, 45], 3)
[2, 5, 34]
```

52. Escreva a função, `soma_cumulativa`, que recebe uma lista de números e que devolve uma lista que contém a soma cumulativa da lista recebida, ou seja, o elemento na posição `i` da lista devolvida contém a soma de todos os elementos da lista original nas posições de 0 a `i`. Não é necessário validar os dados de entrada. Por exemplo:

```
>>> soma_cumulativa([1, 2, 3, 4, 5])
[1, 3, 6, 10, 15]
```

53. Uma matriz é uma tabela bidimensional em que os seus elementos são referenciados pela linha e pela coluna em que se encontram. Uma matriz pode ser representada como uma lista cujos elementos são listas, cada uma destas sub-listas representa uma linha. Com base nesta representação, escreva a função `elemento_matriz` que recebe três argumentos, uma matriz, uma linha e uma coluna e que devolve o elemento da matriz que se encontra na linha e coluna indicadas. A sua função deve permitir a seguinte interação:

```
>>> m = [[1, 2, 3], [4, 5, 6]]
>>> elemento_matriz(m, 0, 0)
1
>>> elemento_matriz(m, 0, 3)
ValueError: elemento_matriz: indice invalido, coluna 3
```

54. Considere uma matriz como definida no exercício anterior. Escreva uma função em Python que recebe uma matriz e que a escreve sob a forma:

$$\begin{array}{cccc} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{array}$$

55. Considere o conceito de matriz. Escreva a função `soma_mat` que recebe como argumentos duas matrizes e devolve uma matriz correspondente à soma das matrizes que são seus argumentos. Sendo a e b as matrizes a somar, os elementos da matriz produto são dados:

$$s_{ij} = a_{ij} + b_{ij}$$

Por exemplo:

```
m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
m2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
escreve_matriz(soma_mat(m1, m2))
2      4      6
8      10     12
14     16     18
```

56. Considere o conceito de matriz. Escreva a função `multiplica_mat` que recebe como argumentos duas matrizes e devolve uma matriz correspondente ao produto das matrizes que são seus argumentos. Sendo a e b as matrizes a multiplicar, os elementos da matriz produto são dados por:

$$p_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

57. A sequência de Racamán,

0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, ...

é uma sequência de números inteiros não negativos, definida do seguinte modo: (1) o primeiro termo da sequência é zero; (2) para calcular o n -ésimo termo, verifica-se se o termo anterior é maior do que n e se o resultado de subtrair n ao termo anterior ainda não apareceu na sequência, neste caso o n -ésimo termo é dado pela subtração entre o $(n - 1)$ -ésimo termo e n ; em caso contrário o n -ésimo termo é dado pela soma do $(n - 1)$ -ésimo termo com n . Ou seja:

$$r(n) = \begin{cases} 0 & \text{se } n = 0 \\ r(n-1) - n & \text{se } r(n-1) > n \wedge (r(n-1) - n) \notin \{r(i) : i < n\} \\ r(n-1) + n & \text{em caso contrário} \end{cases}$$

Escreva uma função que recebe um inteiro positivo, n , e devolve uma lista contendo os n primeiros elementos da sequência de Racamán. Por exemplo:

```
>>> seq_racaman(15)
[0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9]
```

58. Uma chave do euromilhões é constituída por cinco inteiros ordenados, entre 1 e 50 e **sem repetições** e por dois números **diferentes**, também ordenados, entre 1 e 12. Escreva uma função sem argumentos que devolve aleatoriamente uma lista contendo duas listas, cada uma delas contendo os constituintes de uma chave do euromilhões. Para a geração de números aleatórios utilize a função `random()`, existente na biblioteca `random`, que devolve aleatoriamente um real no intervalo $[0, 1[$. Por exemplo:

```
>>> euromilhoes()
[[8, 9, 30, 32, 37], [5, 6]]
```

Programação Recursiva e Funcional, e Dicionários

59. Escreva a função recursiva `apenas_digitos_impares` que recebe um número inteiro não negativo `n`, e devolve um inteiro composto apenas pelos dígitos ímpares de `n`. Se `n` não tiver dígitos ímpares, a função deve devolver zero. Não pode usar cadeias de caracteres. Por exemplo:

```
>>> apenas_digitos_impares(468)
0
>>> apenas_digitos_impares(12426374856)
1375
```

60. Escreva a função recursiva `junta_ordenadas` que recebe como argumentos duas listas ordenadas por ordem crescente e devolve uma lista também ordenada com os elementos das duas listas. Não é necessário validar os argumentos da sua função. Por exemplo:

```
junta_ordenadas([2, 5, 90], [3, 5, 6, 12])
[2, 3, 5, 5, 6, 12, 90]
```

61. Escreva a função recursiva `sublistas` que recebe uma lista, e tem como valor o número total de sublistas que esta contém. Por exemplo:

```
>>> sublistas([[1], 2, [3]])
2
>>> sublistas([[[[[1]]]]])
4
>>> sublistas(['a', [2, 3, [[[1]]], 6, 7], 'b'])
4
```

62. Escreva a função recursiva `soma_n_vezes` que recebe três números inteiros, `a`, `b` e `n`, e que devolve o valor de somar `n` vezes `a` a `b`, ou seja, $b + a + a + \dots + a$, `n` vezes. A sua função não pode usar a operação de multiplicação. Por exemplo:

```
>>> soma_n_vezes(3, 2, 5)
17
```

63. Escreva a função recursiva `soma_els_atomicos` que recebe como argumento um tuplo, cujos elementos podem ser outros tuplos, e que devolve a soma dos elementos correspondentes a tipos elementares de dados que existem no tuplo original. Não é necessário verificar os dados de entrada. Por exemplo:

```
>>> soma_els_atomicos((3, (((((6, (7, )), ), ), ), ), 2, 1))
19
>>> soma_els_atomicos(((((),),),),))
0
```

64. Escreva a função recursiva `inverte` que recebe uma lista e devolve a lista invertida. Por exemplo:

```
>>> inverte([3, 4, 7, 9])
[9, 7, 4, 3]
```

65. Suponha que a operação `in` não existia em Python. Escreva a função recursiva `pertence` que recebe uma lista e um elemento e devolve `True` se o elemento `pertence` à lista e `False` em caso contrário. Por exemplo:

```
>>> pertence([3, 4, 5], 2)
False
>>> pertence([3, 4, 5], 5)
True
```

66. Escreva a função recursiva `subtrai` que recebe duas listas e devolve a lista que corresponde a remover da primeira lista todos os elementos que pertencem à segunda lista. Por exemplo:

```
>>> subtrai([2, 3, 4, 5], [2, 3])
[4, 5]
>>> subtrai([2, 3, 4, 5], [6, 7])
[2, 3, 4, 5]
```

67. Escreva a função recursiva `parte` que recebe uma lista de números e um número e que devolve uma lista de duas listas, a primeira lista contém os elementos da lista original menores que o número dado (pela mesma ordem) e a segunda lista contém os elementos da lista original maiores ou iguais que o número dado (pela mesma ordem). Não é necessário verificar a correção dos dados de entrada. Sugestão: Use uma função auxiliar. Por exemplo:

```
>>> parte([3, 5, 1, 4, 5, 8, 9], 4)
[[3, 1], [5, 4, 5, 8, 9]]
```

68. Escreva a função recursiva `maior` que recebe uma lista de números e devolve o maior número da lista. Sugestão: Use uma função auxiliar. Por exemplo:

```
>>> maior([5, 3, 8, 1, 9, 2])
9
```

69. A função `somatorio`

```
def somatorio(l_inf, l_sup, calc_termo, prox):
    soma = 0
    while l_inf <= l_sup:
        soma = soma + calc_termo(l_inf)
        l_inf = prox(l_inf)
    return soma
```

é apenas a mais simples de um vasto número de abstrações semelhantes que podem ser capturadas por funções de ordem superior. Por exemplo, podemos usar a função `somatorio` para somar os quadrados dos múltiplos de 3 entre 9 e 21:

```
>>> somatorio(9, 21, lambda x : x * x, lambda x : x + 3)
1215
```

Diga o que fazem as seguintes utilizações da função `somatorio`:

a. `somatorio(4, 500, lambda x: x, lambda x: x + 1)`

```

b. somatorio(5, 500, lambda x: x * x, lambda x: x + 5)
c. somatorio(1,
            5,
            lambda x: somatorio(1,
                                x,
                                lambda x: x,
                                lambda x: x+1),
            lambda x: x+1)

```

70.

- a. Defina a função `piatorio` que calcula o produto dos termos de uma função entre dois limites especificados.
- b. Mostre como definir o `factorial` em termos da utilização da função `piatorio`.

71. Considere a função `soma_fn` que recebe um número inteiro positivo, `n`, e uma função de um argumento inteiro, `fn`, e devolve a soma de todos os valores da função entre 1 e `n`. A função `soma_fn` não verifica a correção do seu argumento nem usa funcionais sobre listas. Por exemplo:

```

>>> soma_fn(4, lambda x: x * x)
30
>>> soma_fn(4, lambda x: x + 1)
14

```

- a. Escreva a função `soma_fn` usando um ciclo `for`.
- b. Escreva a função `soma_fn` usando recursão.

72. Usando recursão, defina os seguintes funcionais sobre listas:

- a. `filtra(lst, tst)` que devolve a lista obtida a partir da lista `lst` que apenas contém os elementos que satisfazem o predicado de um argumento `tst`. Por exemplo:

```

>>> filtra([1, 2, 3, 4, 5], lambda x : x % 2 == 0)
[2, 4]

```

- b. `transforma(lst, fn)` que devolve a lista obtida a partir da lista `lst` cujos elementos correspondem à aplicação da função de um argumento `fn` aos elementos de `lst`. Por exemplo:

```

>>> transforma([1, 2, 3, 4], lambda x : x ** 3)
[1, 8, 27, 64]

```

- c. `acumula(lst, fn)` que devolve o valor obtido da aplicação da função de dois argumentos `fn` a todos os elementos da lista `lst`. Por exemplo:

```

>>> acumula([1, 2, 3, 4], lambda x, y : x + y)
10

```

73. Usando os funcionais sobre listas da pergunta anterior, escreva a função `soma_quadrados_impares`, que recebe uma lista de inteiros e devolve a soma dos quadrados dos seus elementos ímpares. A sua função deve conter apenas uma instrução, a instrução `return`. Não é necessário validar os dados de entrada. Por exemplo:

```

>>> soma_quadrados_impares([1, 2, 3, 4, 5, 6])

```


74. Considere a seguinte definição do predicado `eh_primo` que tem o valor verdadeiro apenas se o seu argumento é um número primo:

```
def eh_primo(n):
    if n == 1:
        return False
    else:
        for i in range(2, n):
            if n % i == 0:
                return False
        return True
```

Escreva a função de ordem superior, `nao_primos` que recebe um número inteiro positivo, `n`, e devolve todos os números inferiores ou iguais a `n` que não são primos. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
nao_primos(10)
[1, 4, 6, 8, 9, 10]
```

75. Considere a seguinte função que recebe como argumentos um número natural, `n`, e um predicado de um argumento, `p`:

```
def misterio(num, p):
    if num == 0:
        return 0
    elif p(num % 10):
        return num % 10 + 10 * misterio(num // 10, p)
    else:
        return misterio(num // 10, p)
```

- Explique o que faz esta função.
- Utilize a função `misterio` para escrever a função `filtra_pares` que recebe um número inteiro e devolve o número obtido a partir dele que apenas contém dígitos pares. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> filtra_pares(5467829)
4682
```

76. Usando funcionais sobre listas, escreva a função `lista_digitos`, que recebe um inteiro positivo `n` e devolve a lista cujos elementos são os dígitos de `n`. A sua função deve conter apenas uma instrução, a instrução `return`. Sugestão: transforme o número numa cadeia de caracteres. Por exemplo:

```
>>> lista_digitos(123)
[1, 2, 3]
```

77. Usando a função `lista_digitos` do exercício 76 e funcionais sobre listas, escreva a função `produto_digitos`, que recebe um inteiro positivo, `n`, um predicado de um argumento, `pred`, e devolve o produto dos dígitos de `n` que satisfazem o predicado `pred`. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> produto_digitos(12345, lambda x : x > 3)
20
```

78. Usando a função `lista_digitos` do exercício 76 e funcionais sobre listas, escreva a função `apenas_digitos_impares`, que recebe um inteiro positivo, n , e devolve o inteiro constituído pelos dígitos ímpares de n . A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> apenas_digitos_impares(12345)
135
```

79. Suponha que as operações de multiplicação ($*$) e potência ($**$) não existiam em Python e que pretende calcular o quadrado de um número natural. O quadrado de um número natural pode ser calculado como a soma de todos os números ímpares inferiores ao dobro do número:

$$n^2 = \sum_{i=1}^n (2i - 1)$$

Note que o dobro de um número também não pode ser calculado recorrendo à operação de multiplicação. Escreva uma função que calcula o quadrado de um número natural utilizando o método descrito.

- a. Usando recursão com operações adiadas.
- b. Usando recursão de cauda.
- c. Usando um processo iterativo.

80. Escreva a função `numero_digitos` que recebe um número inteiro positivo n , e devolve o número de dígitos de n . As suas funções não podem usar cadeias de caracteres. As suas funções devem validar a correção do argumento. Por exemplo:

```
>>> numero_digitos(9)
1
>>> numero_digitos(1012)
4
```

- a. Usando recursão com operações adiadas.
- b. Usando recursão de cauda.
- c. Usando um processo iterativo.

81. Um número é uma capicua se se lê igualmente da esquerda para a direita e vice-versa. Escreva a função recursiva de cauda `eh_capicua`, que recebe um número inteiro positivo n , e devolve verdadeiro se o número for uma capicua e falso caso contrário. A sua função deve utilizar a função `numero_digitos` do exercício anterior. Por exemplo:

```
>>> eh_capicua(12321)
True
>>> eh_capicua(1221)
True
>>> eh_capicua(123210)
False
```

82. O espelho de um número inteiro positivo é o resultado de inverter a ordem de todos os seus algarismos. Escreva a função recursiva de cauda `espelho`, que recebe um número inteiro positivo n , não divisível por 10, e devolve o seu espelho. Por exemplo:

```
>>> espelho(391)
193
>>> espelho(45679)
97654
```

83. Considere a função g , definida para inteiros não negativos do seguinte modo:

$$g(n) = \begin{cases} 0 & \text{se } n = 0 \\ n - g(g(n-1)) & \text{se } n > 0 \end{cases}$$

- Escreva uma função recursiva em Python para calcular o valor de $g(n)$.
- Siga o processo gerado por $g(3)$, indicando todos os cálculos efectuados.
- Que tipo de processo é gerado por esta função?

84. Escreva a função recursiva, `calc_soma`, para calcular o valor da soma

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

para um dado valor de x e de n . A sua função deve ter em atenção que o i -ésimo termo da soma pode ser obtido do termo na posição $i - 1$, multiplicando-o por x/i .

85. Escreva a função recursiva, `maior_inteiro`, que recebe um inteiro positivo, `limite`, e que devolve o maior inteiro (n) tal que $1 + 2 + \dots + n \leq \text{limite}$. Por exemplo:

```
>>> maior_inteiro(6)
3
>>> maior_inteiro(20)
5
```

86. Um número d é divisor de n se o resto da divisão de n por d for 0. Usando recursão de cauda, escreva a função `soma_divisores` que recebe um número inteiro positivo n , e que devolve a soma de todos os divisores de n .

87. Um número diz-se perfeito se for igual à soma dos seus divisores (não contando o próprio número). Por exemplo, 6 é perfeito porque $1 + 2 + 3 = 6$.

- Usando recursão de cauda, escreva a função `perfeito` que recebe como argumento um número inteiro e tem o valor `True` se o seu argumento for um número perfeito e `False` em caso contrário. Não é necessário validar os dados de entrada.
- Usando recursão com operações adiadas e a função `perfeito` da alínea anterior, escreva a função `perfeitos_entre` que recebe dois inteiros positivos e devolve a lista dos números perfeitos entre os seus argumentos, incluindo os seus argumentos. Por exemplo:

```
>>> perfeitos_entre(6, 30)
[6, 28]
```

88. Considere a seguinte lista de dicionários na qual os significados dos campos são óbvios:

```
l_nomes = [{'nome':{'nomep':'Jose', 'apelido':'Silva'},
'morada':{'rua':'R. dos douradores', 'num': 34, 'andar':'6 Esq',
'localidade':'Lisboa', 'estado':'', 'cp':'1100-032',
'pais':'Portugal'}}, {'nome':{'nomep':'John', 'apelido':'Doe'},
'morada':{'rua':'West Hazeltine Ave.', 'num': 57, 'andar':'',
'localidade':'Kenmore', 'estado':'NY', 'cp':'14217', 'pais':'USA'}}]
```

Diga quais são os valores dos seguintes nomes:

- a. `l_nomes[1]`
- b. `l_nomes[1]['nome']`
- c. `l_nomes[1]['nome']['apelido']`
- d. `l_nomes[1]['nome']['apelido'][0]`

89. Escreva a função `agrupa_por_chave` que recebe uma lista de pares, contendo uma chave e um valor, (*k*, *v*), representados por tuplos de dois elementos, devolve um dicionário que a cada chave *k* associa a lista com os valores *v* para essa chave encontrados na lista que é seu argumento. Por exemplo:

```
>>> agrupa_por_chave([('a', 8), ('b', 9), ('a', 3)])
{'a': [8, 3], 'b': [9]}
```

90. Uma carta de jogar é caracterizada por um naipe (espadas, copas, ouros e paus) e por um valor (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K). Uma carta pode ser representada por um dicionário com duas chaves, 'np' e 'vlr', sendo um conjunto de cartas representado por uma lista de cartas.

- a. Escreva uma função em Python que devolve uma lista contendo todas as cartas de um baralho. Por exemplo:

```
>>> baralho()
[{'np': 'esp', 'vlr': 'A'}, {'np': 'esp', 'vlr': '2'}, {'np': 'esp', 'vlr': '3'}, {'np': 'esp', 'vlr': '4'}, {'np': 'esp', 'vlr': '5'}, {'np': 'esp', 'vlr': '6'}, {'np': 'esp', 'vlr': '7'}, {'np': 'esp', 'vlr': '8'}, {'np': 'esp', 'vlr': '9'}, {'np': 'esp', 'vlr': '10'}, {'np': 'esp', 'vlr': 'J'}, {'np': 'esp', 'vlr': 'Q'}, {'np': 'esp', 'vlr': 'K'}, {'np': 'copas', 'vlr': 'A'}, {'np': 'copas', 'vlr': '2'}, {'np': 'copas', 'vlr': '3'}, ... ]
```

- b. Recorrendo à função `random()`, a qual produz um número aleatório no intervalo [0, 1], escreva a função `baralha`, que recebe uma lista correspondente a um baralho de cartas e baralha aleatoriamente essas cartas, devolvendo a lista que corresponde às cartas baralhadas. Sugestão: percorra sucessivamente as cartas do baralho trocando cada uma delas por uma outra carta seleccionada aleatoriamente. Por exemplo:

```
>>> baralha(baralho())
[{'np': 'esp', 'vlr': '3'}, {'np': 'esp', 'vlr': '9'}, {'np':
```

```
'copas', 'vlr': '6'}, {'np': 'esp', 'vlr': 'Q'}, {'np': 'esp',
'vlr': '7'}, {'np': 'copas', 'vlr': '8'}, {'np': 'copas',
'vlr': 'J'}, {'np': 'esp', 'vlr': 'K'}, ... ]
```

- c. Escreva uma função em Python que recebe um baralho de cartas e as distribui por quatro jogadores, devolvendo uma lista que contém as cartas de cada jogador. O seu programa deve garantir que o número de cartas a distribuir é um múltiplo de 4. Por exemplo:

```
distribui(baralha(baralho()))
[[{'np': 'ouros', 'vlr': 'A'}, {'np': 'copas', 'vlr': '7'},
{'np': 'paus', 'vlr': 'A'}, {'np': 'esp', 'vlr': 'J'}, {'np':
'paus', 'vlr': '6'}, {'np': 'esp', 'vlr': '10'}, {'np':
'copas', 'vlr': '5'}, {'np': 'esp', 'vlr': '6'}, {'np':
'copas', 'vlr': '8'}, {'np': 'esp', 'vlr': '3'}, {'np':
'ouros', 'vlr': '5'}, {'np': 'ouros', 'vlr': '8'}, {'np':
'copas', 'vlr': 'K'}], [{'np': 'paus', 'vlr': '2'}, {'np':
'esp', 'vlr': '2'}, ...]]
```

91. Considere um dicionário que contém as notas finais dos alunos da cadeira. O dicionário tem como chave a nota, um número natural entre 0 e 20. Para cada chave do dicionário, o seu valor é uma lista com os números dos alunos com essa nota. Por exemplo, o dicionário poderá ser:

```
notas_dict = {1 : [46592, 49212, 90300, 59312], 15 : [52592, 59212],
20 : [58323]}
```

Escreva a função `resumo_notas` que recebe um dicionário com as notas finais dos alunos da cadeira e devolve um tuplo com dois elementos contendo a média dos alunos aprovados e o número de alunos reprovados. Por exemplo:

```
>>> resumo_notas(notas_dict)
16.666666666666668, 4)
```

92. Escreva a função, `metabolismo`, que recebe um dicionário cujas chaves correspondem a nomes de pessoas e cujos valores correspondem a tuplos, contendo o sexo, a idade, a altura e o peso dessa pessoa. A sua função devolve um dicionário que associa a cada pessoa o seu índice de metabolismo basal. Sendo s o sexo, i a idade, h a altura e p o peso de uma pessoa, o metabolismo basal, m , é definido do seguinte modo:

$$m(s, i, h, p) = \begin{cases} 66 + 6.3 \times p + 12.9 \times h + 6.8 \times i & \text{se } s = M \\ 655 + 4.3 \times p + 4.7 \times h + 4.7 \times i & \text{se } s = F \end{cases}$$

Não é necessário validar os dados de entrada. Por exemplo:

```
>>> d = {'Maria' : ('F', 34, 1.65, 64), 'Pedro': ('M', 34, 1.65, 64),
'Ana': ('F', 54, 1.65, 120), 'Hugo': ('M', 12, 1.82, 75)}
>>> metabolismo(d)
{'Ana': 1458.955, 'Hugo': 675.078, 'Maria': 1109.755, 'Pedro':
736.685}
```

93. Escreva uma função que recebe uma cadeia de caracteres correspondente a um texto e que produz uma lista de todas as palavras que este contém, juntamente com o número de vezes que essa palavra aparece no texto. Sugestão: guarde cada palavra como uma entrada num dicionário contendo o número de vezes que esta apareceu no texto. Por exemplo:

```
>>> cc = 'a aranha arranha a ra a ra arranha a aranha ' \
+ 'nem a aranha arranha a ra nem a ra arranha a aranha'
>>> conta_palavras(cc)
{'aranha': 4, 'arranha': 4, 'ra': 4, 'a': 8, 'nem': 2}
```

94. Usando a ordenação por borbulhamento, escreva a função `mostra_ordenado` que apresenta por ordem alfabética os resultados produzidos pelo exercício anterior. Por exemplo:

```
>>> mostra_ordenado(conta_palavras(cc))
a8
aranha 4
arranha 4
nem 2
ra 4
```

95. Uma matriz é dita esparsa (ou rarefeita) quando a maior parte dos seus elementos é zero. As matrizes esparsas aparecem em grande número de aplicações em engenharia. Uma matriz esparsa pode ser representada por um dicionário cujas chaves correspondem a tuplos que indicam a posição de um elemento na matriz (linha e coluna) e cujo valor é o elemento nessa posição da matriz. Por exemplo, $\{(3, 2): 20, (150, 2): 6, (300, 10): 20\}$ corresponde a uma matriz esparsa com apenas três elementos diferentes de zero.

- a. Escreva uma função em Python que recebe uma matriz esparsa e a escreve sob a forma, na qual os elementos cujo valor é zero são explicitados.

$$\begin{array}{cccc} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{array}$$

```
>>> escreve_esparsa({(1,5): 4, (2, 3): 9, (4, 1): 1})
000000
000004
000900
000000
000000
010000
```

- b. Escreva uma função que recebe duas matrizes esparsas e devolve a sua soma. Por exemplo:

```
e1 = {(1,5): 4, (2, 3): 9, (4, 1): 1}
e2 = {(1, 6): 2, (4, 1): 2, (5,4): 2}
>>> escreve_esparsa(soma_esparsa(e1, e2))
0000000
0000042
```

```
0009000
0000000
0300000
0000200
```

96. Suponha que `bib` é uma lista cujos elementos são dicionários e que contém a informação sobre os livros existentes numa biblioteca. Cada livro é caracterizado pelo seus autores, título, casa editora, cidade de publicação, ano de publicação, número de páginas e ISBN. Por exemplo, a seguinte lista corresponde a uma biblioteca com dois livros:

```
[{'autores': ['G. Arroz', 'J. Monteiro', 'A. Oliveira'], 'titulo': 'Arquitectura de computadores', 'editor': 'IST Press', 'cidade': 'Lisboa', 'ano': 2007, 'numpags': 799, 'isbn': '978-972-8469-54-2'}, {'autores': ['J.P. Martins'], 'titulo': 'Logica e Raciocinio', 'editor': 'College Publications', 'cidade': 'Londres', 'ano': 2014, 'numpags': 438, 'isbn': '978-1-84890-125-4'}]
```

Escreva um programa em Python que recebe a informação de uma biblioteca e devolve o título do livro mais antigo. Por exemplo:

```
>>> mais_antigo(bib)
'Arquitectura de computadores'
```

97. Um número racional na forma canónica é um número da forma n/d em que n e d são inteiros, $d \neq 0$ e n e d são primos entre si. Suponha que o número racional n/d era representado pelo dicionário `{'num': n, 'den': d}`.

- a. Escreva a função `cria_racional` que recebe dois inteiros e devolve o dicionário correspondente ao racional cujo numerador é o primeiro inteiro e cujo denominador é o segundo inteiro. A sua função deve fazer a verificação dos dados de entrada. Por exemplo:

```
>>> cria_racional(4, 6)
{'d': 3, 'n': 2}
>>> cria_racional(4, 0)
ValueError: o denominador não pode ser 0
>>> cria_racional(4.3, 2)
ValueError: os números devem ser inteiros
```

- b. Escreva a função `escreve_racional` que recebe um dicionário correspondente a um racional e escreve o racional sob a forma n/d . Por exemplo:

```
>>> escreve_racional(cria_racional(4, 6))
2/3
```

- c. Escreva a função `soma_racionais` que recebe dois racionais e devolve o número racional correspondente à sua soma. A soma dos racionais a/b e d/e é dada por $(a \times e + b \times d)/(b \times e)$. Por exemplo:

```
>>> escreve_racional(soma_racionais(cria_racional(4, 6),
cria_racional(2, 3)))
4/3
```

Abstração e Programação com Objetos

98. Suponha que desejava criar o tipo *racional*. Um número racional é qualquer número que possa ser expresso como o quociente de dois inteiros: o *numerador* (um inteiro positivo, negativo ou nulo) e o *denominador* (um inteiro positivo). Os racionais a/b e c/d são iguais se e só se $a \times d = b \times c$. As operações básicas para o tipo racional são as seguintes:

- Construtores:
 - $cria_rac : \mathbb{Z} \times \mathbb{N} \rightarrow \text{racional}$
 - $cria_rac(n, d)$ tem como valor o número racional cujo numerador é n e cujo denominador é d ($d > 0$).
- Seletores:
 - $num : \text{racional} \rightarrow \mathbb{Z}$
 - $num(r)$ tem como valor o numerador do racional r .
 - $den : \text{racional} \rightarrow \mathbb{N}$
 - $den(r)$ tem como valor o denominador do racional r .
- Reconhecedores:
 - $eh_racional : universal \rightarrow \text{lógico}$
 - $eh_racional(arg)$ tem o valor verdadeiro se arg é um número racional e tem o valor falso em caso contrário.
 - $eh_rac_zero : \text{racional} \rightarrow \text{lógico}$
 - $eh_rac_zero(r)$ tem o valor verdadeiro se r é o racional 0 e tem o valor falso em caso contrário.
- Testes:
 - $rac_iguais : \text{racional} \times \text{racional} \rightarrow \text{lógico}$
 - $rac_iguais(r1, r2)$ tem o valor verdadeiro se $r1$ e $r2$ correspondem ao mesmo número racional e tem o valor falso em caso contrário.

- Escolha uma representação para o tipo racional usando dicionários.
- Escreva as operações básicas utilizando a representação escolhida.
- Escreva o transformador de saída para o tipo racional. Por exemplo:

```
>>> escreve_rac(cria_rac(1, 3))
'1/3'
```

- Escreva a função `produto_rac` que calcula o produto de dois racionais. Se $r1 = a/b$ e $r2 = c/d$ então $r1 \times r2 = ac/bd$. Por exemplo:

```
>>> escreve_rac(produto_rac(cria_rac(1,3), cria_rac(3,4)))
3/12
```

Note que esta função é uma função de alto nível, pois não pertence às operações básicas e, como tal, não pode usar a representação.

99. Suponha que desejava criar o tipo *relógio* para representar um instante de tempo dentro de um dia. Suponha que um relógio é caracterizado por um triplo de inteiros positivos, correspondentes às horas (entre 0 e 23), aos minutos (entre 0 e 59) e aos segundos (entre 0 e 59). As operações básicas para o tipo racional são as seguintes:

- Construtores:

- $cria_rel : \mathbb{N}_0^3 \rightarrow relógio$
 - $cria_rel(h, m, s)$ tem como valor o relógio cujas horas são h , os minutos são m , e os segundos são s .
- Seletores:
 - $horas : relógio \rightarrow \mathbb{N}_0$
 - $horas(r)$ tem como valor as horas do relógio r .
 - $minutos : relógio \rightarrow \mathbb{N}_0$
 - $minutos(r)$ tem como valor os minutos do relógio r .
 - $secs : relógio \rightarrow \mathbb{N}_0$
 - $secs(r)$ tem como valor os segundos do relógio r .
- Reconhecedores:
 - $eh_relogio : universal \rightarrow lógico$
 - $eh_relogio(arg)$ tem o valor verdadeiro se arg é um relógio e tem o valor falso em caso contrário.
 - $eh_meia_noite : relógio \rightarrow lógico$
 - $eh_meia_noite(r)$ tem o valor verdadeiro se r corresponde à meia noite 00:00:00 e tem o valor falso em caso contrário.
 - $eh_meio_dia : relógio \rightarrow lógico$
 - $eh_meio_dia(r)$ tem o valor verdadeiro se r corresponde ao meio dia 12:00:00 e tem o valor falso em caso contrário.
- Testes:
 - $mesmas_horas : relógio \times relógio \rightarrow lógico$
 - $mesmas_horas(r1, r2)$ tem o valor verdadeiro se $r1$ e $r2$ correspondem às mesmas horas e tem o valor falso em caso contrário.

- a. Escolha uma representação para o tipo relógio usando listas.
- b. Escreva as operações básicas utilizando a representação escolhida.
- c. Suponha que a representação externa para os elementos do tipo relógio é $hh:mm:ss$, em que hh são os dois dígitos que representam as horas, mm são os dois dígitos que identificam os minutos e ss são os dois dígitos que identificam os segundos. Escreva o transformador de saída para o tipo relógio. Por exemplo:

```
>>> escreve_relogio(cria_relogio(9, 2, 34))
'09:02:34'
```

- d. Escreva o predicado `depois_rel` que recebe dois relógios e devolve verdadeiro apenas se o segundo relógio corresponder a um instante de tempo posterior ao primeiro relógio.
- e. Escreva a função `dif_secs` que calcula o número de segundos entre dois instantes, representados por dois relógios. Esta função apenas deve produzir um valor se o segundo instante de tempo for posterior ao primeiro, gerando uma mensagem de erro se essa condição não se verificar. Por exemplo:

```
>>> dif_secs(cria_rel(10, 2, 34), cria_rel(11, 21, 34))
4740
>>> dif_secs(cria_rel(10, 2, 34), cria_rel(9, 21, 34))
ValueError: dif_secs: primeiro arg posterior ao segundo
```

- f. Suponha que altera a representação interna do tipo relógio para um dicionário com as chaves 'horas', 'min' e 'seg'. O que deverá fazer às funções `escreve_relogio` e `dif_segundos` para que estas sejam usadas com esta nova representação? Justifique a sua resposta.

100. Suponha que desejava criar o tipo *data*. Uma *data* é caracterizada por um dia (um inteiro entre 1 e 31), um mês (um inteiro entre 1 e 12) e um ano (um inteiro que pode ser positivo, nulo ou negativo). Para cada data, deve ser respeitado o limite de dias de cada mês, incluindo o caso de Fevereiro nos anos bissextos. Recorde que um ano é bissexto se for divisível por 4 e não for divisível por 100, a não ser que seja também divisível por 400. Por exemplo, 1984 é bissexto, 1100 não é, e 2000 é bissexto. As operações básicas para o tipo racional são as seguintes:

- Construtores:
 - $cria_data : \mathbb{N} \times \mathbb{N} \times \mathbb{Z} \rightarrow data$
 - `cria_data(d, m, a)` tem como valor a data com dia *d*, mês *m*, e ano *a*.
- Seletores:
 - $dia : data \rightarrow \mathbb{N}$
 - `dia(dt)` tem como valor o dia da data *dt*.
 - $mes : data \rightarrow \mathbb{N}$
 - `mes(dt)` tem como valor o mês da data *dt*.
 - $ano : data \rightarrow \mathbb{Z}$
 - `ano(dt)` tem como valor o ano da data *dt*.
- Reconhecedores:
 - $eh_data : universal \rightarrow lógico$
 - `eh_data(arg)` tem o valor verdadeiro se *arg* é uma data e tem o valor falso em caso contrário.
- Testes:
 - $mesma_data : data \times data \rightarrow lógico$
 - `mesma_data(d1, d2)` tem o valor verdadeiro se *d1* e *d2* correspondem à mesma data e tem o valor falso em caso contrário.

- a. Escolha uma representação para o tipo *data* usando dicionários.
- b. Escreva as operações básicas utilizando a representação escolhida.
- c. Suponha que a representação externa para os elementos do tipo *data* é *dd/mm/aaaa ee* (em que *dd* representa o dia, *mm* o mês, *aaaa* o ano e *ee* representa a era, a qual é omitida se o ano for maior ou igual a 0 e é escrita *AC* se o ano for menor que zero), escreva o transformador de saída para o tipo *data*. Por exemplo:

```
>>> escreve_data (cria_data (5, 9, 2018))
'05/09/2018'
>>> escreve_data (cria_data (5, 9, -10))
'05/09/10 AC'
```

- d. Defina o predicado `data_anterior` que recebe como argumentos duas datas e tem o valor verdadeiro apenas se a primeira data é anterior à segunda.

```
>>> data_anterior(cria_data(2, 1, 2003), cria_data(2, 1, 2005))
True
```

- e. Defina a função `idade` que recebe como argumentos a data de nascimento de uma pessoa e outra data posterior e devolve a idade da pessoa, em anos, na segunda data.

```
>>> idade(cria_data(2, 1, 2003), cria_data(1, 1, 2005))
2
>>> idade(cria_data(2, 1, 2003), cria_data(1, 2, 2005))
3
>>> idade(cria_data(2, 1, 2003), cria_data(1, 2, 2000))
ValueError: idade: a pessoa ainda não nasceu
```

101. Defina a classe `estacionamento`, que simula o funcionamento de um parque de estacionamento. A criação de instâncias desta classe recebe um inteiro que determina a lotação do parque e devolve um objeto com os seguintes métodos: `entra()`, corresponde à entrada de um carro; `sai()`, corresponde à saída de um carro; `lugares()` indica o número de lugares livres no estacionamento. Por exemplo:

```
>>> istec = estacionamento(20)
>>> istec.lugares()
20
>>> istec.entra()
>>> istec.entra()
>>> istec.entra()
>>> istec.entra()
>>> istec.sai()
>>> istec.lugares()
17
```

102. Suponha que desejava criar a classe `racional`. Um número racional é qualquer número que possa ser expresso como o quociente de dois inteiros: o *numerador* (um inteiro positivo, negativo ou nulo) e o *denominador* (um inteiro positivo). Os racionais a/b e c/d são iguais se e só se $a \times d = b \times c$. Assuma que a representação externa de um racional é apresentada de modo que o numerador e o denominador são primos entre si. A classe `racional` admite as operações `nume` e `deno` que devolvem, respetivamente o numerador e o denominador.

- a. Defina a classe `racional`, incluindo o transformador de saída.
b. Usando operações polimórficas, escreva métodos para calcular a soma e o produto de racionais. Se $r1 = a/b$ e $r2 = c/d$ então $r1 + r2 = (ad + bc)/bd$ e $r1 \times r2 = ac/bd$. Por exemplo:

```
>>> r1 = racional(2, 4)
>>> r2 = racional(1, 6)
>>> r1
1/2
>>> r2
1/6
```

```
>>> r1 + r2
2/3
>>> r1*r2
1/2
```

103. Os automóveis mais recentes mostram a distância que é possível percorrer até ser necessário um reabastecimento. Pretende-se criar esta funcionalidade em Python através da classe `automovel`. Esta classe é construída indicando a capacidade do depósito, a quantidade de combustível no depósito e o consumo do automóvel em litros aos 100 km. A classe `automovel` apresenta os seguintes métodos:

- `combustivel` devolve a quantidade de combustível no depósito;
- `autonomia` devolve o número de Km que é possível percorrer com o combustível no depósito;
- `abastece(n_litros)` aumenta em `n_litros` o combustível no depósito. Se este abastecimento exceder a capacidade do depósito, gera um erro e não aumenta a quantidade de combustível no depósito;
- `percorre(n_km)` percorre `n_km` Km, desde que a quantidade de combustível no depósito o permita, em caso contrário gera um erro e o trajecto não é efectuado.

Por exemplo:

```
>>> a1 = automovel(60, 10, 15)
>>> a1.combustivel()
10
>>> a1.autonomia()
66
>>> a1.abastece(45)
'366 Km até abastecimento'
>>> a1.percorre(150)
'216 Km até abastecimento'
>>> a1.percorre(250)
ValueError: Não tem autonomia para esta viagem
```

104. Suponha que desejava criar a classe *conjunto*, a qual apresenta métodos correspondentes às seguintes operações básicas:

- Construtores:
 - $conjunto : elemento^n \rightarrow conjunto \ (n \geq 0)$
 - `conjunto(e1, ..., en)` tem como valor conjunto com os elementos `e1, ..., en` ($n \geq 0$).
 - $duplica : conjunto \rightarrow conjunto$
 - `duplica(c)` tem como valor um conjunto igual a `c`.
 - $insere : elemento \times conjunto \rightarrow conjunto$
 - `insere(e, c)` tem como valor o resultado de inserir o elemento `e` no conjunto `c`; se `e` já pertencer a `c`, tem como valor `c`.
- Seletores:
 - $el_conj : conjunto \rightarrow elemento$

- $el_conj(c)$ tem como valor um elemento escolhido aleatoriamente do conjunto c ; se o conjunto for vazio, esta operação é indefinida.
 - $retira_conj : elemento \times conjunto \rightarrow conjunto$
 - $retira_conj(e, c)$ tem como valor o resultado de retirar o elemento e do conjunto c ; se e não pertencer a c , tem como valor c .
 - $cardinal : conjunto \rightarrow inteiro$
 - $cardinal(c)$ tem como valor o número de elementos do conjunto c .
- Reconhecedores:
 - $eh_conjunto_vazio : conjunto \rightarrow lógico$
 - $eh_conjunto_vazio(c)$ tem o valor verdadeiro se c é um conjunto vazio e tem o valor falso em caso contrário.
- Testes:
 - $pertence : elemento \times conjunto \rightarrow lógico$
 - $pertence(e, c)$ tem o valor verdadeiro se e pertence ao conjunto c e tem o valor falso em caso contrário.
- a. Defina a classe `conjunto`, com a qual podemos obter, por exemplo, a seguinte interação:

```
>>> c1 = conjunto(1, 2, 3, 4)
>>> c1
{1,2,3,4}
>>> c1.cardinal()
4
>>> c1.retira(3)
{1,2,4}
>>> c1.el_conj()
2
```

- b. Como parte da classe `conjunto`, defina o método `subconjunto`:
 $subconjunto : conjunto \times conjunto \rightarrow lógico$

$subconjunto(c1, c2)$ tem o valor verdadeiro, se o conjunto $c1$ for um subconjunto do conjunto $c2$, ou seja, se todos os elementos de $c1$ pertencerem a $c2$, e tem o valor falso, em caso contrário. Por exemplo:

```
>>> c1 = conjunto(1, 2, 3, 4)
>>> c1
{1,2,3,4}
>>> c2 = conjunto(2, 3)
>>> c2
{2,3}
>>> c2.subconjunto(c1)
True
>>> c1.subconjunto(c2)
False
```

- c. Como parte da classe `conjunto`, defina o método `uniao`:
 $uniao : conjunto \times conjunto \rightarrow conjunto$

`uniao(c1, c2)` tem como valor o conjunto união de `c1` com `c2`, ou seja, o conjunto formado por todos os elementos que pertencem a `c1` ou a `c2`. Por exemplo:

```
>>> c1 = conjunto(1, 2, 3, 4)
>>> c2 = conjunto(3, 4, 5, 6)
>>> c1.uniao(c2)
{3, 4, 5, 6, 1, 2}
```

- d. Como parte da classe `conjunto`, defina o método `intersecao`:
intersecao : conjunto \times conjunto \rightarrow conjunto

`intersecao(c1, c2)` tem como valor o conjunto interseção de `c1` com `c2`, ou seja, o conjunto formado por todos os elementos que pertencem a `c1` e a `c2`. Por exemplo:

```
>>> c1 = conjunto(1, 2, 3, 4)
>>> c2 = conjunto(3, 4, 5, 6)
>>> c1.intersecao(c2)
{3, 4}
```

- e. Como parte da classe `conjunto`, defina o método `diferenca`:
diferenca : conjunto \times conjunto \rightarrow conjunto

`diferenca(c1, c2)` tem como valor o conjunto diferença de `c1` com `c2`, ou seja, o conjunto formado por todos os elementos que pertencem a `c1` e não pertencem a `c2`. Por exemplo:

```
>>> c1 = conjunto(1, 2, 3, 4)
>>> c2 = conjunto(3, 4, 5, 6)
>>> c1.diferenca(c2)
{1, 2}
```

105. Considere a função de Ackermann:

$$A(m, n) = \begin{cases} n + 1 & \text{se } m = 0 \\ A(m - 1, 1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

esta função pode ser diretamente escrita através da função:

```
def A(m, n):
    if m == 0:
        return n + 1
    elif m > 0 and n == 0:
        return A(m-1, 1)
    else:
        return A(m-1, A(m, n-1))
```

Como pode verificar, esta função calcula várias vezes o mesmo valor. Para evitar este problema, podemos definir uma classe, `mem_A`, cujo estado interno contém informação sobre os valores de `A` já calculados, apenas calculando um novo valor

quando este ainda não é conhecido. Esta classe possui um método `val` que calcula o valor de `A` para os inteiros que são seus argumentos e um método `mem` que mostra os valores memorizados. Por exemplo:

```
>>> a = mem_A()
>>> a.val(2, 3)
9
>>> a.mem()
{(0, 1): 2,
 (0, 2): 3,
 (0, 3): 4,
 (0, 4): 5,
 (0, 5): 6,
 (0, 6): 7,
 (0, 7): 8,
 (0, 8): 9,
 (1, 0): 2,
 (1, 1): 3,
 (1, 2): 4,
 (1, 3): 5,
 (1, 4): 6,
 (1, 5): 7,
 (1, 6): 8,
 (1, 7): 9,
 (2, 0): 3,
 (2, 1): 5,
 (2, 2): 7,
 (2, 3): 9}
```

Defina a classe `mem_A`.