Exercícios para Fundamentos da Programação

Fausto Almeida, Cláudia Antunes, Ana Cardoso-Cachopo,
Pedro Amaro de Matos, Francisco Saraiva de Melo

Departamento de Engenharia Informática
Instituto Superior Técnico
Universidade Técnica de Lisboa

Preâmbulo

O objectivo desta publicação é disponibilizar um conjunto de exercícios que permitam aos alunos da disciplina de Fundamentos de Programação das Licenciaturas em Engenharia Informática e de Computadores e em Engenharia de Redes de Comunicações do Instituto Superior Técnico consolidarem os conhecimentos adquiridos ao longo do semestre.

Os exercícios estão organizados de acordo com os capítulos do livro adoptado na disciplina.

Chama-se a atenção dos alunos para a indicação do *nível de dificuldade* de 1 a 4, correspondendo o nível 1 a exercícios muito fáceis e o nível 4 a exercícios francamente difíceis. Esta informação é apresentada dentro de um quadrado cinzento antes do enunciado do exercício a que diz respeito.

Conteúdo

Capítulo 1

Computadores, algoritmos e programas

1. **(1)** Escreva uma gramática em notação BNF para definir números binários. Um número binário é representado como uma sequência arbitrariamente longa de `1`'s e `0`'s. Considere que, à excepção do número binário `0`, o primeiro dígito de um número binário deverá ser `1`. Alguns exemplos de números binários são:

   ```
   0
   1
   100
   101
   1010111001011
   ```

2. **(1)** 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
   ```

3. **(1)** Considere a seguinte gramática em notação BNF:

   ```
   <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.

```
asno
cria
gato
leao
ovos
tu
vaca
```

4. **(1)** Considere a seguinte gramática em notação BNF:

```
<operação>  ::= (<argumento> <operador> <argumento>)
<operador>  ::= + | - | * | /
<argumento> ::= <dígito>+
<dígito>    ::= 2 | 4 | 6 | 8 | 0
```

   (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 operações da linguagem definida pela gramática.

```
( 1 + 2 )
( 2 + - )
( 24 * 06 )
  2 * 0
( 84 + )
( 0 / 0 )
```

5. **(1)** (*Teste de 27/10/2012*) 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 `cdr` são frases da linguagem, `cd` e `cdrr` não o são.

Capítulo 2

Elementos básicos de programação

1. **(1)** Explique por palavras suas o que aconteceu na seguinte interação com o Python.

```
Python 3.2.3 (v3.2.3:3d0686d90f55, Apr 10 2012, 11:25:50)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
Type "help", "copyright", "credits" or "license" for more information.
>>> 5 + 3
8
>>> 5 + 6.0
11.0
>>> not(True)
False
>>> true and false
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.NameError: name 'true' is not defined
```

2. **(1)** Diga qual o resultado de avaliar sequencialmente os seguintes comandos no interpretador de Python.

   (a) `(3 + 4 * 5 - 2) / 7`
   (b) `9 // 4 == 7 % 5`
   (c) `8 // 2 == 8 / 2.0`
   (d) `round(3.5)`
   (e) `round(4.5)`
   (f) `int(5.78)`
   (g) `float(2)`
   (h) `float(2.0)`
   (i) `3 > 2.0 and 7 > 8.5`
   (j) `3.0 > 2 or 7.5 > 8`
   (k) `a, b = 2, 3`

(l) `a, b = b, a`

(m) `print('a = ', a, '\nb = ', b)`

3. **(1)** Escreva um programa em Python que pede ao utilizador que lhe forneça um inteiro e que escreve o quadrado do triplo do inteiro. O seu programa deve gerar uma interação como a seguinte:

```
Escreva um inteiro -> 5
O quadrado do triplo de 5 resulta em 225
```

4. **(1)** 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) fica em -23
```

5. **(1)** Escreva um programa em Python que pede ao utilizador que lhe forneça um inteiro correspondente a um certo número de horas e que escreve um número real que traduz o número de dias correspondentes ao inteiro lido. O seu programa deve gerar uma interação como a seguinte:

```
Escreva as horas para saber a quantos dias corresponde.
Horas -> 45678
45678 horas correspondem a 1903.25 dias.
```

6. **(1)** Escreva um programa em Python que pede ao utilizador que lhe forneça um número correspondente ao raio de um círculo e que escreve a área do círculo. A área de um círculo de raio r é dada pela fórmula πr^2. Use o valor 3.14 para a constante π. O seu programa deve gerar uma interação como a seguinte:

```
Escreva o raio do circulo para eu calcular a area.
Raio -> 7
Um circulo de raio 7 tem area de 153.86 .
```

7. **(1)** Escreva um programa em Python que pede ao utilizador que lhe forneça um número e que escreve *positivo*, *negativo* ou *zero*, caso o número seja, respectivamente, maior, menor ou igual a zero. O seu programa deve gerar uma interação como a seguinte:

```
Escreva um numero para eu dizer o seu sinal.
Num -> -78
O numero -78 e
negativo
```

8. **(2)** Escreva um programa em Python que pede ao utilizador que lhe forneça um número correspondente a um ano e que indica se o ano é bissexto. 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. O seu programa deve gerar uma interação como a seguinte:

```
Escreva um ano para eu dizer se e bissexto.
Ano -> 1984
O ano 1984 e bissexto.
```

9. **(2)** Escreva um programa em Python que pede ao utilizador que lhe forneça um número e que escreve a tabuada da multiplicação para esse número. O seu programa deve gerar uma interação como a seguinte. Tenha em conta que o programa deve ser fácil de alterar se quisermos obter, por exemplo, os quinhentos primeiros valores das multiplicações.

```
Escreva um numero para eu escrever a tabuada da multiplicacao.
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
fim
```

10. **(1)** (*Teste de 27/10/2012*) Considere o seguinte programa em Python:

```
numero_1 = 5
numero_2 = 10
while numero_1 > 0:
    numero_2 = numero_2 - numero_1
```

Será que este programa pode ser considerado um algoritmo? Justifique a sua resposta.

11. Escreva um programa em Python que pede ao utilizador que lhe forneça um valor em Euros e troca esse valor no menor número de moedas possível (2EUR, 1EUR, 50c., 20c., 10c., 5c., 2c. e 1c.). O programa deve começar por calcular quantas moedas de 2EUR são necessárias, seguidamente calcula quantas de 1EUR são necessárias e assim sucessivamente. No seu programa pode assumir que o utilizador introduz um número real correspondente aos euros e cêntimos. Deverá depois separar os euros dos cêntimos. O seu programa deve gerar uma interação como a seguinte:

```
Introduza uma quantia em EUROS: 12.5
12.5 EUR correspondem a:
- 6 moedas de 2EUR
- 0 moedas de 1EUR
- 1 moedas de 50c
- 0 moedas de 20c
- 0 moedas de 10c
- 0 moedas de 5c
- 0 moedas de 2c
- 0 moedas de 1c
```

12. Escreva um programa em Python que converte temperaturas entre graus centígrados e Fahrenheit. O seu programa deve começar por ler um caracter seguido de um real. Se o primeiro for ‘‘`f`’’ ou ‘‘`F`’’, o programa converte a temperatura de graus Centígrados para Fahrenheit; se for ‘‘`c`’’ ou ‘‘`C`’’ converte de Fahrenheit para Centígrados; caso contrário escreve uma mensagem de aviso e termina antes de ler o valor numérico.

    As fórmulas de conversão são as seguintes:

$$F = C \times 1.8 + 32$$
$$C = \frac{F - 32}{1.8}$$

Capítulo 3

Funções

1. **(2)** Considere a seguinte interacção em Python:

```
>>> def f1(x):
...     return x * x
...
>>> f1(5)
25
>>> f2 = f1
>>>
```

   (a) Qual o valor retornado pela chamada `f2(5)`? Justifique a sua resposta.

   (b) Suponha que agora definia o procedimento `f2`, do seguinte modo:

```
>>> def f2(x):
...     return x + 10
...
>>>
```

   Quais os valores de `f1(5)` e de `f2(5)`? Justifique a sua resposta.

2. **(2)** Considere a seguinte interacção em Python:

```
>>> x = 5
>>> def soma(x, y):
...     return x + y
...
>>> soma(1, 2)
3
>>> x
5
>>> y
NameError: name 'y' is not defined
>>>
```

   Explique a interacção observada, representando graficamente os ambientes envolvidos.

3. **(1)** Defina uma função com o nome `dobro` que tem como valor o dobro do seu argumento.

4. **(1)** Defina uma função com o nome `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.

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

5. **(2)** Defina uma função com o nome `sao_iguais` que recebe dois argumentos arbitrários e tem o valor verdadeiro se os seus argumentos forem iguais e falso no caso contrário. Não utilize `if` nem valores lógicos.

6. **(1)** Defina uma função com o nome `area_circulo` que tem como valor a área de um círculo cujo raio é r. Note-se que esta área é dada pela fórmula πr^2. Use o valor 3.14 para a constante π.

7. **(2)** Utilizando a função `area_circulo` do exercício anterior, escreva uma função com o nome `area_coroa` que recebe dois argumentos, `r1` e `r2`, e tem como valor a área de uma 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`.

8. **(2)** Defina uma função com o nome `dias_mes` que recebe uma cadeia de caracteres, correspondentes às 3 primeiras letras (minúsculas) do nome de um mês, e tem como valor um número inteiro correspondendo a número de dias desse mês. No caso de uma cadeia de caracteres inválida, a sua função deverá gerar um erro de valor (`ValueError`). Assuma que o mês de Fevereiro tem sempre 28 dias.

Mês	Número de dias
`jan, mar, mai, jul, ago, out, dez`	31
`abr, jun, set, nov`	30
`fev`	28

9. **(2)** Escreva uma função com o nome `soma_quadrados` que recebe um número inteiro positivo, `n`, e tem como valor a soma do quadrado de todos os números até `n`.

```
>>> soma_quadrados(3)
14
>>> soma_quadrados(5)
55
```

10. Escreva uma função com o nome `potencia` que recebe como argumentos dois números inteiros não negativos, `b` e `n`, e tem como valor a potência n de b, *i.e.*, b^n. Neste exercício não pode usar a função `pow` nem o operador `**`.

```
>>> potencia(3, 2)
9
>>> potencia(2, 4)
16
```

11. **(2)** Um número d é *divisor* de n se o resto da divisão de n por d for 0. Escreva uma função com o nome `num_divisores` que recebe um número inteiro positivo `n`, e tem como valor o número de divisores de `n`. No caso de `n` ser 0 deverá devolver 0.

```
>>> num_divisores(20)
6
>>> num_divisores(13)
2
```

12. **(2)** Escreva uma função com o nome `soma_divisores` que recebe um número inteiro positivo `n`, e tem como valor a soma de todos os divisores de `n`. No caso de `n` ser 0 deverá devolver 0.

```
>>> soma_divisores(20)
42
>>> soma_divisores(13)
14
```

13. **(3)** *Pedra-papel-tesoura* é um jogo envolvendo dois jogadores em que cada jogador escolhe uma de 3 jogadas possíveis: "pedra", "papel" ou "tesoura". O resultado do jogo é determinado com base nas seguintes regras:

    - Um jogador que escolha "pedra" perde o jogo se o outro escolher "papel" e ganha caso o outro escolha "tesoura".
    - Um jogador que escolha "papel" perde o jogo se o outro escolher "tesoura" e ganha caso o outro escolha "pedra".
    - Um jogador que escolha "tesoura" perde o jogo se o outro escolher "pedra" e ganha caso o outro escolha "papel".

    Caso ambos escolham a mesma jogada, o jogo é considerado um empate.

    (a) Escreva uma função com o nome `pedra_papel_tesoura` que recebe como argumentos duas cadeias de caracteres, correspondendo às jogadas dos dois jogadores, e anuncia qual dos dois ganhou, escrevendo uma mensagem no ecrã. A sua função tem o valor lógico `False` se algum dos argumentos for inválido (isto é, não corresponder a uma das 3 cadeias de caracteres `'pedra'`, `'papel'` ou `'tesoura'`) e `True` caso contrário.

    ```
    >>> pedra_papel_tesoura('pedra', 'papel')
    Parabens, jogador 2, ganhou o jogo!
    True
    >>> pedra_papel_tesoura('tesoura', 'papel')
    Parabens, jogador 1, ganhou o jogo!
    ```

```
True
>>> pedra_papel_tesoura('papel', 'papel')
O jogo foi um empate.
True
>>> pedra_papel_tesoura('papel', 'jhg')
False
```

(b) Utilizando a função da alínea anterior, escreva um programa que vá pedindo sucessivamente aos jogadores 1 e 2 para introduzirem as jogadas respectivas, anunciando de seguida o resultado do jogo. O seu programa deve terminar se algum dos jogadores introduzir uma jogada inválida. Um exemplo possível de interacção é:

```
Jogador 1, por favor introduza a sua jogada: pedra
Jogador 2, por favor introduza a sua jogada: papel
Parabens, jogador 2, ganhou o jogo!
Jogador 1, por favor introduza a sua jogada: tesoura
Jogador 2, por favor introduza a sua jogada: papel
Parabens, jogador 1, ganhou o jogo!
Jogador 1, por favor introduza a sua jogada: wejkrlw
Jogador 2, por favor introduza a sua jogada: tesoura
Jogo terminado.
```

14. **(2)** A função *arctg* pode ser aproximada através da fórmula

$$arctg(z) = \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{2n+1} = z - \frac{z^3}{3} + \frac{z^5}{5} - \frac{z^7}{7} + \dots$$

Escreva uma função com o nome `arctg` que tem como valor o *arctg*, calculado de acordo com a fórmula acima. A sua função deverá receber o número z para o qual se quer calcular o *arctg*, bem como o número de termos da expressão a calcular.

15. **(3)** A constante e é um dos números mais importantes em Matemática, a par com os elementos neutros da adição (0) e da multiplicação (1), com a constante π e com a unidade imaginária i.

O valor de e corresponde ao número real que é a soma da seguinte série infinita

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots,$$

em que $n!$ corresponde ao factorial de n,

$$n! = 1 \times 2 \times \dots \times n.$$

Escreva uma função com o nome `e` para calcular uma aproximação do número e utilizando a série apresentada. A condição de paragem deve ser determinada por si e devidamente justificada.

16. **(2)** (*Teste de 27/10/2012*) Escreva uma função em Python com o nome `numero_algarismos` que recebe um inteiro positivo, `n`, e devolve o número de algarismos de `n`. Por exemplo:

```
>>> numero_algarismos(0)
1
>>> numero_algarismos(23464321)
8
```

17. **(2)** (*Teste de 27/10/2012*) Suponha que a operação de multiplicação não existia em Python, existindo as operações de adição, $+$, de subtracção, $-$, e o operador relacional $>$. Escreva em Python uma função com o nome `vezes`, que efectua a multiplicação de dois inteiros positivos. A sua função não tem que verificar se os inteiros são positivos. Pode usar instruções correspondentes a ciclos.

Capítulo 4

Tuplos e ciclos contados

1. **(2)** Lembre-se do exercício 8 do capítulo anterior. Defina uma função com o nome `dias_mes_tuplos` que recebe uma cadeia de caracteres, correspondentes às 3 primeiras letras (minúsculas) do nome de um mês, e tem como valor um número inteiro correspondendo a número de dias desse mês. No caso de uma cadeia de caracteres inválida, a sua função deverá gerar um erro de valor (`ValueError`). Assuma que o mês de Fevereiro tem sempre 28 dias.

Mês	Número de dias
`jan, mar, mai, jul, ago, out, dez`	31
`abr, jun, set, nov`	30
`fev`	28

2. **(2)** Escreva uma função `produto_elementos_tuplo` que recebe um tuplo de números e devolve o produto de todos os elementos do tuplo.

```
>>> produto_elementos_tuplo((4, 5, 6))
120
```

3. **(2)** Escreva uma função `conta_pares_tuplo` que recebe um tuplo de inteiros e devolve o número de elementos pares no tuplo.

```
>>> conta_pares_tuplo((4, 5, 6))
2
>>> conta_pares_tuplo((3, 5, 7))
0
```

4. **(2)** Escreva uma função `posicao_tuplo` que recebe um tuplo de números e um número, e devolve a posição da primeira ocorrência desse número no tuplo. Caso o número não ocorra no tuplo deverá devolver *falso*.

```
>>> posicao_tuplo((4, 3, 2, 2, 1, 4), 4)
0
>>> posicao_tuplo((4, 3, 2, 2, 1, 4), 1)
```

```
4
>>> posicao_tuplo((4, 3, 2, 2, 1, 4), 6)
False
```

5. **(2)** Escreva uma função `todos_pares_tuplo` que recebe um tuplo de inteiros e devolve *verdadeiro* se o tuplo for constituído exclusivamente por números pares e *falso* em caso contrário.

```
>>> todos_pares_tuplo((2, 4, 6))
True
>>> todos_pares_tuplo((2, 3, 6))
False
```

6. **(2)** Escreva uma função `numero_ocorrencias_tuplo_numeros` que recebe um tuplo de números e um número, e devolve o número de vezes que o número ocorre no tuplo.

```
>>> numero_ocorrencias_tuplo_numeros((4, 5, 6),  5)
1
>>> numero_ocorrencias_tuplo_numeros((3, 5, 7),  2)
0
>>> numero_ocorrencias_tuplo_numeros((3, 5, 3),  3)
2
```

7. **(3)** Escreva uma função `quadrados_tuplo` que recebe um tuplo de números, e devolve outro tuplo em que cada elemento corresponde ao quadrado do elemento na mesma posição no tuplo original.

```
>>> quadrados_tuplo((3, 2, 4))
(9, 4, 16)
```

8. **(3)** Escreva uma função `substitui_tuplo` que recebe um tuplo de números, um número `v` e um número `n`, e devolve um tuplo idêntico ao primeiro mas substituindo todas as ocorrências de `v` por `n`.

```
>>> substitui_tuplo((4, 3, 2, 4), 4, 5)
(5, 3, 2, 5)
```

9. **(3)** Escreva uma função `posicoes_tuplo` que recebe um tuplo de números e um número, e devolve o tuplo de todas as posições em que o número ocorre no tuplo.

```
>>> posicoes_tuplo((4, 3, 2, 2, 1, 4), 4)
(0, 5)
>>> posicoes_tuplo((4, 3, 2, 2, 1, 4), 6)
()
```

10. **(2)** Escreva uma função `tuplo_sem_repeticoes` que recebe um tuplo de números, e devolve *verdadeiro* se o tuplo não contém elementos repetidos e *falso* em caso contrário.

```
>>> tuplo_sem_repeticoes((1, 2, 3, 4))
True
>>> tuplo_sem_repeticoes((1, 2, 4, 2))
False
```

11. **(3)** Escreva uma função `seleciona_menores_tuplo` que recebe um tuplo de números, e devolve o tuplo com todos os números do tuplo original menores que `n`.

```
>>> seleciona_menores_tuplo((2, 3, 5, 4, 3, 5, 6), 5)
(2, 3, 4, 3)
>>> seleciona_menores_tuplo((1, 3, 5), 1)
()
```

12. **(3)** Escreva uma função `tuplo_capicua` que recebe um tuplo de números, e devolve *verdadeiro* se o tuplo for uma capicua e *falso* em caso contrário.

```
>>> tuplo_capicua((1,2,3,2,1))
True
>>> tuplo_capicua((1,2,2,1))
True
>>> tuplo_capicua((1,2,3,1))
False
```

13. **(3)** Escreva uma função `cria_capicua` que recebe um tuplo de números, e devolve um novo tuplo que resulta de anexar o inverso do tuplo ao tuplo original.

```
>>> cria_capicua((1, 2, 3))
(1, 2, 3, 3, 2, 1)
```

14. **(2)** (*Teste de 27/10/2012*) Diga o que é escrito pela seguinte instrução:

```
for i in range(2):
    for j in range(3, 5):
        for k in range(4, 1, -1):
            if (i + j) % 2 == 0:
                print(i, j, k)
```

15. (*Teste de 27/10/2012*)

    (a) **(3)** Utilizando a representação de caracteres, escreva funções em Python, com os nomes `codifica` e `descodifica`, que recebem uma mensagem (uma cadeia de caracteres) e, respectivamente, codificam e descodificam essa mensagem, utilizando a cifra de Atbash. Uma mensagem é codificada através da cifra de Atbash, substituindo cada uma das suas letras pela letra que se encontra à mesma distância do fim do alfabeto que a distância da letra original ao início do alfabeto. Com a cifra de Atbash originamos a seguinte correspondência entre as letras do alfabeto:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

A sua cifra apenas considera letras maiúsculas, sendo qualquer símbolo que não corresponda a uma letra maiúscula ignorado. Os espaços entre as palavras mantêm-se como espaços. A seguinte interacção ilustra o funcionamento destas funções:

```
>>> codifica('ESTA E UMA PERGUNTA BOA PARA O TESTE DE FP')
    'VHGZ V FNZ KVITFMGZ YLZ KZIZ L GVHGV WV UK'
>>> descodifica('VHGZ V FNZ KVITFMGZ YLZ KZIZ L GVHGV WV UK')
    'ESTA E UMA PERGUNTA BOA PARA O TESTE DE FP'
```

(b) **(3)** Escreva um programa em Python que utiliza as funções definidas na alínea anterior, interaccionando com o utilizador através da solicitação de frases para codificar ou descodificar. Parta do princípio que as funções `codifica` e `descodifica` da alínea anterior já estão definidas. A seguinte interacção ilustra o funcionamento do seu programa:

```
>>> codificador()
Número:
Introduza uma mensagem
Use a palavra FIM para terminar
-> AQUI VAI UM EXEMPLO
C(odificar), D(escodificar)?
-> C
A mensagem codificada é:
 ZJFR EZR FN VCVNKOL
Introduza uma mensagem
Use a palavra FIM para terminar
-> FIM
```

Capítulo 5

Listas

1. **(2)** Escreva uma função `concatena_strings_lista` que recebe uma lista de cadeias de caracteres, e devolve uma cadeia de caracteres que resulta de concatenar todos os elementos da lista. A sua função deve testar se o seu argumento é uma lista de cadeias de caracteres e dar uma mensagem de erro adequada em caso contrário.

```
>>> concatena_strings_lista(('ola', 'bom'))
Traceback (most recent call last):
[...]
builtins.ValueError: concatena_strings_lista: arg devia ser uma lista
>>> concatena_strings_lista(['ola', 22])
Traceback (most recent call last):
[...]
builtins.ValueError: concatena_strings_lista: elemento devia ser uma string
>>> concatena_strings_lista([])
''
>>> concatena_strings_lista(['ola', ' ', 'bom', ' ', 'dia!'])
'ola bom dia!'
```

2. **(2)** Escreva uma função `cria_lista_multiplos` que recebe um número inteiro positivo, e devolve uma lista com os dez primeiros múltiplos desse número. A sua função deve testar se o seu argumento é um número inteiro positivo e dar uma mensagem de erro adequada em caso contrário. Considere que zero é múltiplo de todos os números.

```
>>> cria_lista_multiplos(6)
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54]
>>> cria_lista_multiplos(6.0)
Traceback (most recent call last):
[...]
builtins.ValueError: cria_lista_multiplos: arg devia ser numero inteiro positivo
```

3. **(2)** Escreva uma função `e_lista_numeros` que recebe um argumento e tem o valor *verdadeiro* se o argumento corresponder a uma lista não vazia de números e *falso* em caso contrário.

```
>>> e_lista_numeros([1, 2.0, 3])
True
>>> e_lista_numeros([])
False
>>> e_lista_numeros([1, 'ola', 3])
False
```

4. **(3)** Escreva uma função `remove_pares` que recebe uma lista não vazia de números, e altera a lista de modo a remover todos os números pares que ela contém. A sua função deve testar se o seu argumento é uma lista não vazia de números e dar uma mensagem de erro adequada em caso contrário.

```
>>> remove_pares([])
Traceback (most recent call last):
[...]
builtins.ValueError: remove_pares: arg devia ser lista nao vazia de numeros
>>> a = [0, 1, -2, 3, 34, -45.0, 76.0, 0]
remove_pares(a)
>>> a
[1, 3, -45.0]
```

5. **(2)** Escreva uma função `cria_lista_tuplos` que recebe uma lista, e devolve uma outra lista que contém os tuplos da lista inicial. A sua função deve testar se o seu argumento é uma lista e dar uma mensagem de erro adequada em caso contrário.

```
>>> cria_lista_tuplos((1, 2, (3), 'ola'))
Traceback (most recent call last):
[...]
builtins.ValueError: cria_lista_tuplos: arg devia ser uma lista
>>> cria_lista_tuplos([1, 2, (3, True), 'ola', ('qwerty', 23, 2.0), 56])
[(3, True), ('qwerty', 23, 2.0)]
>>> cria_lista_tuplos([])
[]
```

6. Uma matriz é uma tabela bidimensional em que os seus elementos são referenciados pela linha e pela coluna em que se encontram. Neste exercício não vamos considerar as matrizes de 0x0. Uma matriz pode ser representada como uma lista não vazia cujos elementos são listas não vazias. Com base nesta representação, escreva as seguintes funções.

   (a) **(2)** Uma função `e_matriz` que recebe um argumento e tem o valor *verdadeiro* se o seu argumento corresponder a uma matriz e *falso* em caso contrário. Uma matriz é uma lista não vazia em que os seus elementos são listas não vazias, todas com o mesmo comprimento.

   (b) **(2)** Uma função chamada `dimensoes_matriz` que recebe como argumento uma matriz e devolve um tuplo com dois inteiros, o primeiro correspondendo ao número

de linhas e o segundo ao número de colunas da matriz. Nota: funções que determinem apenas o número de linhas e de colunas de uma matriz serão úteis nos exercícios seguintes.

(c) **(2)** Uma função `escreve_matriz` que recebe como argumento uma matriz e a escreve na forma que estamos habituados a usar em álgebra.

(d) **(2)** Uma função chamada `elemento_matriz` que recebe como argumentos uma matriz e dois inteiros, correspondendo à linha e à coluna e que devolve o elemento da matriz que se encontra na linha e coluna indicadas. A sua função deve verificar se os inteiros recebidos estão dentro das dimensões da matriz e dar uma mensagem de erro adequada em caso contrário.

(e) **(3)** Uma função chamada `soma_matrizes` que recebe como argumentos duas matrizes, verifica se elas têm dimensões iguais, e devolve uma nova matriz que corresponde à soma das matrizes recebidas como argumento.

(f) **(2)** Uma função chamada `multiplica_matriz` que recebe como argumentos uma matriz e um número, e altera destrutivamente a matriz original de modo a que cada elemento corresponda ao elemento da matriz original multiplicado pelo número recebido. A sua função deve devolver a matriz alterada.

(g) **(3)** Uma função chamada `multiplica_matrizes` que recebe como argumentos duas matrizes, verifica se elas têm dimensões compatíveis para serem multiplicadas, e devolve uma nova matriz que corresponde à multiplicação das matrizes recebidas como argumento. Recorde que, dadas duas matrizes A e B, o elemento ij da matriz AB é dado por:

$$[AB]_{ij} = \sum_k A_{ik} B_{kj}.$$

Com estas funções, deve ser possível obter a seguinte interacção:

```
>>> a = [[2, 33.0], [44, 5], [6, 7]]
>>> b = [[2, 3, 4, 5], [4, 5, 6, 7]]
>>> c = [[2, 3, 4, 5], [1, 2], [4, 5, 6, 7]]
>>> e_matriz(a)
True
>>> e_matriz(c)
False
>>> dimensoes_matriz(a)
(3, 2)
>>> dimensoes_matriz(b)
(2, 4)
>>> escreve_matriz(a)
2       33.0
44      5
6       7
>>> escreve_matriz(b)
2       3       4       5
4       5       6       7
>>> elemento_matriz(a, 2, 1)
```

```
44
>>> elemento_matriz(a, 2, 8)
Traceback (most recent call last):
  File "/Applications/WingIDE/WingIDE.app/Contents/MacOS/src/debug/tserver/_sandbox.py", l
    # Used internally for debug sandbox under external interpreter
  File "/Applications/WingIDE/WingIDE.app/Contents/MacOS/src/debug/tserver/_sandbox.py", l
builtins.ValueError: elemento_matriz: linha ou coluna inválida
>>> soma_matrizes(a, b)
Traceback (most recent call last):
  File "/Applications/WingIDE/WingIDE.app/Contents/MacOS/src/debug/tserver/_sandbox.py", l
    # Used internally for debug sandbox under external interpreter
  File "/Applications/WingIDE/WingIDE.app/Contents/MacOS/src/debug/tserver/_sandbox.py", l
builtins.ValueError: soma_matrizes: matrizes com dimensoes diferentes
>>> soma_matrizes(a, a)
[[4, 66.0], [88, 10], [12, 14]]
>>> escreve_matriz(soma_matrizes(a, a))
4       66.0
88      10
12      14
>>> multiplica_matriz(a, 2)
[[4, 66.0], [88, 10], [12, 14]]
>>> multiplica_matrizes(a, b)
[[272.0, 342.0, 412.0, 482.0], [216, 314, 412, 510], [80, 106, 132, 158]]
>>> escreve_matriz(multiplica_matrizes(a, b))
272.0   342.0   412.0   482.0
216     314     412     510
80      106     132     158
```

7. **(3)** Defina uma função `serie_factorial` que recebe um número n e devolve a lista com o factorial dos números entre *0* e n inclusive.

```
>>> serie_factorial(4)
[1, 1, 2, 6, 24]
```

8. **(3)** A série de Fibonacci é definida pela expressão seguinte:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases} \tag{5.1}$$

Defina uma função `serie_fibonacci` que recebe um número inteiro n maior ou igual a 2 e devolve a lista com os n primeiros elementos da série de fibonacci. A sua função deve testar se o seu argumento é um inteiro maior ou igual a 2.

```
>>> serie_fibonnaci(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

9. **(3)** O número de combinações de m objectos n a n, com m e n inteiros positivos, pode ser dado pela seguinte fórmula:

$$C(m,n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } m = n \\ C(m-1,n) + C(m-1,n-1) & \text{if } m > n \end{cases} \tag{5.2}$$

Defina uma função `combinacoes` que recebe um número m, e devolve a matriz quadrada com as combinações de *0* a *0* até às combinações de m, m a m. Considere que a matriz tem o elemento a *0* sempre que m é menor do que n.

```
>>> escreve_matriz(combinacoes(4))
1       1       1       1       1
0       1       2       3       4
0       0       1       3       6
0       0       0       1       4
0       0       0       0       1
```

10. **(4)** O triângulo de Pascal é um triângulo numérico infinito que pode ser determinado recorrendo ao cálculo das combinações de m objectos n a n. Usando a função anterior, defina uma função `triangulo_pascal` que recebe um número n, e escreve o triângulo de Pascal com *n+1* linhas.

```
>>> triangulo_pascal(4)
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
```

Capítulo 6

Ficheiros

1. **(2)** Escreva a função `concatena` que recebe uma lista de cadeias de caracteres, cada uma correspondendo ao nome de um ficheiro, e uma cadeia de caracteres, correspondendo ao nome do ficheiro de saída, e concatena o conteúdo dos primeiros ficheiros no ficheiro de saída. Por exemplo, se o ficheiro `fich1` contiver o texto:

   ```
   Um ficheiro para fazer uns testes.
   Este ficheiro contem duas linhas.
   ```

   e o ficheiro `fich2` contiver o texto:

   ```
   Outro ficheiro para fazer os mesmos testes.
   Este ficheiro contem mais uma linha
   alem desta.
   ```

   é produzida a seguinte interacção:

   ```
   >>> concatena(['fich1', 'fich2'], 'saida')
   >>>
   ```

   em que o ficheiro `saida` contém o texto:

   ```
   Um ficheiro para fazer uns testes.
   Este ficheiro contem duas linhas.
   Outro ficheiro para fazer os mesmos testes.
   Este ficheiro contem mais uma linha
   alem desta.
   ```

2. **(2)** Escreva a função `procura` que recebe duas cadeias de caracteres, em que a primeira corresponde à palavra a procurar e a segunda contém o nome de um ficheiro. A sua função deve escrever no ecrã as linhas do ficheiro que contêm a palavra a procurar. Por exemplo, se o ficheiro `fich` contiver o texto:

   ```
   Outro ficheiro para fazer os mesmos testes.
   Este ficheiro contem mais uma linha
   alem desta.
   ```

é produzida a seguinte interacção:

```
>>> procura('ficheiro', 'fich')
Outro ficheiro para fazer os mesmos testes.
Este ficheiro contem mais uma linha

>>>
```

3. **(2)** Escreva a função `corta` que recebe duas cadeias de caracteres, uma contendo o nome de um ficheiro de entrada, outra contendo o nome do ficheiro de saída, e um número inteiro não negativo `n`, e escreve os `n` primeiros caracteres do ficheiro de entrada no ficheiro de saída, no caso de o ficheiro conter mais que `n` caracteres, ou todo o conteúdo do ficheiro de entrada no ficheiro de saída, no caso contrário. Por exemplo, se o ficheiro `fich` contiver o texto:

   ```
   Um ficheiro para fazer uns testes.
   Este ficheiro contem duas linhas.
   ```

   é produzida a seguinte interacção:

   ```
   >>> corta('teste', 'saida', 20)
   >>>
   ```

   em que o ficheiro `saida` contém o texto:

   ```
   Um ficheiro para faz
   ```

4. **(2)** Escreva a função `recorta` que recebe uma cadeia de caracteres contendo o nome de um ficheiro de entrada e um número inteiro não negativo `n`, e divide o ficheiro de entrada em vários ficheiros de saída de `n` caracteres (o último poderá ter menos caracteres), cujos nomes se constroem com o nome do ficheiro de entrada seguido de números inteiros positivos, de forma a que a concatenação dos ficheiros com nomes sucessivos produz um ficheiro com o conteúdo do ficheiro original. Por exemplo, se o ficheiro `fich` contiver o texto:

   ```
   Um ficheiro para fazer uns testes.
   ```

   é produzida a seguinte interacção:

   ```
   >>> recorta('fich', 20)
   >>>
   ```

   em que o ficheiro `fich1` contém o texto:

   ```
   Um ficheiro para faz
   ```

e o ficheiro `fich2` contém o texto:

```
er uns testes.
```

5. **(3)** Escreva a função `ficheiro_ordenado` que recebe uma cadeia de caracteres, que contém o nome do ficheiro de entrada, e devolve *verdadeiro* se as linhas do ficheiro estiverem ordenadas alfabeticamente de forma estritamente crescente e *falso* no caso contrário. Por exemplo, se o ficheiro `fich` contiver o texto:

```
A primeira linha do ficheiro.
Seguida de uma segunda.
Seguida de uma ultima linha.
```

é produzida a seguinte interacção:

```
>>> ficheiro_ordenado('fich')
True
>>>
```

6. **(3)** Escreva a função `ordena_ficheiro` que recebe como argumento uma cadeia de caracteres correspondendo ao nome de um ficheiro e escreve no ecrã as linhas do ficheiro ordenadas alfabeticamente. Por exemplo, se o ficheiro `fich` contiver o texto:

```
Outro ficheiro para fazer os mesmos testes.
Este ficheiro contem mais uma linha
alem desta, que comeca com letra minuscula.
```

é produzida a seguinte interacção:

```
>>> ordena('fich')
Este ficheiro contem mais uma linha
Outro ficheiro para fazer os mesmos testes.
alem desta, que comeca com letra minuscula.
>>>
```

7. **(3)** Escreva a função `junta_ficheiros_ordenados` que recebe três cadeias de caracteres correspondendo a dois ficheiros de entrada e um de saída. Cada um dos ficheiros de entrada contém números ordenados por ordem crescente, contendo cada linha apenas um número. A sua função deve produzir um ficheiro ordenado de números (contendo um número por linha) correspondente à junção dos números existentes nos dois ficheiros de entrada. Para cada um dos ficheiros de entrada, o seu programa só pode ler uma linha de cada vez. Por exemplo, se o ficheiro `fich1` contiver:

```
5
123.0
789
1200
2345
```

e o ficheiro `fich2` contiver:

```
1
2
123
456.0
```

é produzida a seguinte interacção:

```
>>> junta_ficheiros_ordenados('fich1', 'fich2', 'fich3')
>>>
```

e o ficheiro `fich3` contém:

```
1
2
5
123.0
123
456.0
789
1200
2345
```

8. **(3)** Escreva a função `divide` que recebe uma cadeia de caracteres, que contém o nome do ficheiro de entrada, e um inteiro `n`. Esta função divide o ficheiro de entrada em dois ficheiros, um primeiro, cujo nome é o nome do ficheiro de entrada seguido de 0, em que cada linha contém os `n` primeiros caracteres da linha correspondente do ficheiro original e outro, cujo nome é o nome do ficheiro de entrada seguido de 1, em que cada linha contém os restantes caracteres da linha correspondente no ficheiro original. Por exemplo, se o ficheiro `fich` contiver o texto:

```
Um ficheiro para fazer uns testes.
Este ficheiro contem duas linhas.
```

é produzida a seguinte interacção:

```
>>> divide('fich', 20)
>>>
```

em que o ficheiro `fich0` contém o texto:

```
Um ficheiro para faz
Este ficheiro contem
```

e o ficheiro `fich1` contém o texto:

```
er uns testes.
 duas linhas.
```

9. **(3)** Escreva a função `separa` que recebe uma cadeia de caracteres, que contém o nome do ficheiro de entrada, outra cadeira de caracteres com apenas um caracter e um inteiro `n`. Esta função divide o ficheiro de entrada em dois ficheiros, um primeiro, cujo nome é o nome do ficheiro de entrada seguido de 0, e outro, cujo nome é o nome do ficheiro de entrada seguido de 1. O conteúdo do primeiro ficheiro de saída é o mesmo do ficheiro de entrada, ao qual foi retirado o texto de cada linha entre a `n`-ésima ocorrência do caracter, incluindo o caracter, e a `n`+1-ésima ocorrência do caracter, excluindo o caracter. O segundo ficheiro de saída tem em cada linha o texto que foi retirado ao ficheiro de entrada para produzir o primeiro ficheiro de saída. Por exemplo, se o ficheiro `fich` contiver o texto:

```
Outro ficheiro para fazer os mesmos testes.
alem desta.
Este ficheiro contem mais uma linha
```

é produzida a seguinte interacção:

```
>>> separa('fich', 'h', 1)
>>>
```

em que o ficheiro `fich0` contém o texto:

```
Outro fic
alem desta.
Este ficha
```

e o ficheiro `fich1` contém o texto:

```
heiro para fazer os mesmos testes.

heiro contem mais uma lin
```

Capítulo 7

Dicionários

1. **(1)** Considere o seguinte dicionário, associado ao nome `pt_es`:

   ```
   pt_es = {'um'     : 'uno',
            'dois'   : 'dos',
            'tres'   : 'tres',
            'quatro' : 'cuatro',
            'cinco'  : 'cinco'}
   ```

   Para cada uma das seguintes expressões escreva o resultado retornado pelo interpretador de Python. No caso de a expressão originar um erro explique a razão do erro.

   (a) `type(pt_es)`

   (b) `pt_es(1)`

   (c) `pt_es[1]`

   (d) `1 in pt_es`

   (e) `pt_es('um')`

   (f) `pt_es['um']`

   (g) `'um' in pt_es`

   (h) `'uno' in pt_es`

   (i) `pt_es['oito']`

   (j) `len(pt_es)`

   (k) `pt_es.keys()`

   (l) `pt_es.values()`

   (m)
   ```
   for k in pt_es:
       print(pt_es[k])
   ```

2. **(1)** Escreva uma função `escreve_dicionario` que recebe como argumento um dicionário e escreve para o ecrã as associações contidas no mesmo.

   ```
   >>> misturado = {1            : 'uno',
                    'dois'       : [1, 'dois', 3],
   ```

```
                355.5        : -1,
                ('a', 'b')   : 123,
                2.0          : -1,
                200          : (45, 56, 78),
                'cinco'      : {1:'a', 2:'b', 'c':56}}
>>> escreve_dicionario(misturado)
1 : uno
2.0 : -1
355.5 : -1
200 : (45, 56, 78)
cinco : {1: 'a', 2: 'b', 'c': 56}
dois : [1, 'dois', 3]
('a', 'b') : 123
```

3. **(2)** (*Teste 18/01/2013*) Considere a seguinte variável:

```
teste = {'Portugal' : {'Lisboa' : [('Leonor', '1700-097'),
                                   ('Nuno', '1050-100')],
                        'Porto'  : [('Ana', '4150-036')]},
         'Estados Unidos' : {'Miami'   : [('Nancy', '33136'),
                                          ('Fred', '34136')],
                             'Chicago' : [('Cesar', '60661')]},
         'Reino Unido' : {'London' : [('Stuart', 'SW1H 0BD')]}}
```

Qual o valor de cada um dos seguintes nomes? Se algum dos nomes originar um erro, explique a razão do erro.

(a) `teste['Portugal']['Porto']`

(b) `teste['Portugal']['Porto'][0][0]`

(c) `teste['Estados Unidos']['Miami'][1]`

(d) `teste['Estados Unidos']['Miami'][1][0][0]`

(e) `teste['Estados Unidos']['Miami'][1][1][1]`

4. (a) **(2)** Escreva uma função `traduz` que recebe como argumentos uma lista de palavras e um dicionário de sinónimos e devolve uma nova lista de palavras traduzidas usando os sinónimos no dicionário. Caso a lista de palavras contenha uma palavra que não exista no dicionário, essa palavra deverá ficar por traduzir.

```
>>> pt_en = {'hoje'     : 'today',
             'esta'     : 'is',
             'enevoado' : 'cloudy'}
>>> traduz(['hoje', 'esta', 'muito', 'enevoado'], pt_en)
['today', 'is', 'muito', 'cloudy']
```

(b) **(3)** Usando a função `traduz` da alínea anterior, escreva uma função `traduz_texto` que recebe como argumentos uma cadeia de caracteres (correspondendo ao texto a traduzir) e um dicionário de sinónimos, e devolve uma nova cadeia de caracteres,

correspondendo à tradução do texto original usando os sinónimos no dicionário. Caso o texto a traduzir contenha palavras que não existem no dicionário, essas palavras deverão ficar por traduzir.

```
>>> pt_en = {'hoje'     : 'today',
             'esta'     : 'is',
             'enevoado' : 'cloudy'}
>>> traduz_texto('hoje esta muito enevoado', pt_en)
'today is muito cloudy'
```

Sugestão: Escreva uma função auxiliar `separa_string`, que recebe como argumentos duas cadeias de caracteres, e devolve uma lista de cadeias de caracteres, que correspondem à separação da primeira cadeia de caracteres pelos caracteres especificados na segunda cadeia.

```
>>> a = '    joao joao quer          ser cowboy'
>>> separa_string(a, ' ')
['joao', 'joao', 'quer', 'ser', 'cowboy']
>>> separa_string(a, 'o')
['    j', 'a', ' j', 'a', ' quer          ser c', 'wb', 'y']
>>> separa_string(a, 'o ')
['j', 'a', 'j', 'a', 'quer', 'ser', 'c', 'wb', 'y']
>>> separa_string(a, '$')
['    joao joao quer          ser cowboy']
```

5. **(3)** Escreva uma função `inverte_dic` que recebe como argumento um dicionário e devolve um novo dicionário, correspondente ao inverso do seu argumento. Por outras palavras, a sua função deverá transformar chaves em valores e valores em chaves.

   Nota: Num dicionário, pode haver várias chaves às quais está associado o mesmo valor. Por esta razão, a sua função deverá associar a cada valor *uma lista* com todas as chaves correspondentes.

   ```
   >>> en_pt = {'corner' : 'canto',
                'sail'   : 'vela',
                'candle' : 'vela',
                'leg'    : 'perna'}
   >>> inverte_dic(en_pt)
   {'canto': ['corner'], 'vela': ['sail', 'candle'], 'perna': ['leg']}
   ```

6. **(3)** Escreva uma função `histograma_palavras` que recebe como argumento uma cadeia de caracteres correspondente ao nome de um ficheiro de texto a ler, e devolve um dicionário correspondente ao histograma das palavras que aparecem no ficheiro. Por outras palavras, cada palavra que aparece no ficheiro de texto corresponderá a uma chave do dicionário, à qual estará associado o número de ocorrências dessa palavra no texto. Por exemplo, dado o ficheiro `exemplo.txt` contendo

   ```
   o tempo perguntou ao tempo quanto tempo o tempo tem
   ```

   a interacção:

```
>>> histograma_palavras('exemplo.txt')
```

daria origem ao dicionário:

```
{'tem': 1, 'tempo': 4, 'ao': 1, 'perguntou': 1, 'o': 2, 'quanto': 1}
```

Sugestão: Use a função `separa_string` sugerida no Exercício 4b.

7. **(3)** (*Teste 18/01/2013*) Escreva uma função em Python que que recebe uma cadeia de caracteres, que contém o nome de um ficheiro, lê esse ficheiro, linha a linha, e calcula quantas vezes aparece cada uma das vogais. A sua função deve devolver um dicionário cujas chaves são as vogais e os valores associados correspondem ao número de vezes que a vogal aparece no ficheiro. Apenas conte as vogais que são letras minúsculas. Por exemplo,

```
>>> conta_vogais('testevogais.txt')
{'a': 36, 'u': 19, 'e': 45, 'i': 16, 'o': 28}
```

Capítulo 8

Abstração de dados

8.1 Tipo carta

1. **(2)** (*Teste de 18/01/2013*) Especifique as operações básicas do tipo abstracto de informação *carta* o qual é caracterizado 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).

2. **(2)** Escolha uma representação para o tipo *carta*, e implemente as suas operações básicas e o transformador de saída.

3. **(2)** Usando o tipo *carta*, defina uma função em Python que devolve uma lista em que cada elemento corresponde a uma carta de um baralho.

4. **(3)** Usando o tipo *carta* e recorrendo à função `random()`, a qual produz um número aleatório no intervalo $[0, 1[$, escreva uma 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.

8.2 Tipo racional

Suponha que desejava criar o tipo *racional* em Python. Um número racional é qualquer número que possa ser expresso como o quociente de dois inteiros: o numerador, um número inteiro positivo, negativo ou zero, e o denominador, um número inteiro positivo. Dois racionais a/b e c/d são iguais se $ad = bc$.

1. **(2)** Especifique as operações básicas para o tipo *racional*.

2. **(1)** Escolha uma representação interna para o tipo *racional* usando tuplos.

3. **(2)** Escreva em Python as operações básicas, de acordo com a representação escolhida.

4. **(1)** Suponha que a representação externa para os elementos do tipo *racional* é *num/den*, em que *num* representa o numerador, e *den* representa o denominador. Escreva o transformador de saída `escreve_racional` para o tipo racional. Por exemplo,

```
>>> escreve_racional(cria_racional(1, 3))
    1/3
```

5. **(1)** Escreva o procedimento `produto_racionais` que calcula o produto de dois racionais. Se $r1 = a/b$ e $r2 = c/d$ então $r1 \times r2 = ac/bd$.

   ```
   >>> escreve_racional(produto_racionais(cria_racional(1,3),cria_racional(3,4)))
       3/12
   ```

6. **(1)** Escreva o procedimento `soma_racionais` que calcula a soma de dois racionais. Se $r1 = a/b$ e $r2 = c/d$ então $r1 + r2 = (ad + bc)/bd$.

   ```
   >>> escreve_racional(soma_racionais(cria_racional(1,3),cria_racional(3,4)))
       13/12
   ```

8.3 Tipo relogio

Suponha que desejava criar o tipo *relogio* em Python. Suponha que o tipo é caracterizado por um número que representa as horas (um inteiro entre 0 e 23), um segundo número que representa os minutos (um inteiro entre 0 e 59) e por último um número que representa os segundos (um inteiro entre 0 e 59).

1. **(2)** Especifique as operações básicas para o tipo *relogio*.

2. **(1)** Escolha uma representação interna para o tipo *relogio* usando tuplos.

3. **(2)** Escreva em Python as operações básicas, de acordo com a representação escolhida.

4. **(1)** Suponha que a representação externa para os elementos do tipo *relogio* é *hh:mm:ss*, em que *hh* são os dois dígitos que representam as horas, *mm* são dois dígitos que identificam os minutos e *ss* são dois dígitos que identificam os segundos. Escreva o transformador de saída `escreve_relogio` para o tipo *relogio*. Por exemplo,

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

5. **(1)** Escreva o procedimento `diferenca_segundos` que calcula o número de segundos entre dois instantes de tempo, representados por dois relógios. Este procedimento apenas deve produzir um valor se o segundo instante de tempo for maior do que o primeiro, gerando uma mensagem de erro se essa condição não se verificar.

   ```
   >>> diferenca_segundos(cria_relogio(10, 2, 34), cria_relogio(11, 2, 34))
       3600
   ```

6. **(3)** Suponha agora que pretende representar os elementos do tipo relógio através de inteiros positivos com a forma *hhmmss*, em que *hh* é um inteiro entre 0 e 23 que representa o número de horas, *mm* e *ss* são dois inteiros entre 0 e 59 cada um e que representam, respectivamente, os minutos e os segundos do tempo. Escreva em Python as operações básicas, de acordo com esta nova representação.

7. **(1)** Seria possível utilizar as operações `escreve_relogio` e `diferenca_segundos` com esta nova representação? Justifique.

8.4 Tipo data

Suponha que desejava criar o tipo *data* em Python. Suponha que 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 não negativo). Para cada data, deve ser respeitado o limite de dias de cada mês, incluindo o caso de Fevereiro nos anos bissextos.

1. **(2)** Especifique as operações básicas para o tipo *data*.

2. **(2)** Escolha uma representação interna para o tipo *data* usando dicionários.

3. **(2)** Escreva em Python as operações básicas, de acordo com a representação escolhida.

4. **(1)** Supondo que a representação externa para um elemento do tipo *data* é *DD-MM-AAAA* (em que *DD* representa o dia, *MM* o mês e *AAAA* o ano), escreva o transformador de saída para o tipo *data*. Por exemplo

```
>>> escreve_data (cria_data (5, 9, 987))
    05/09/987
```

5. **(2)** Tendo em conta as operações básicas do tipo *data*, defina um procedimento `data_anterior` que recebe como argumentos duas datas e tem o valor verdadeiro se a primeira data é anterior à segunda e falso caso contrário.

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

6. **(2)** Tendo em conta as operações básicas do tipo data, defina um procedimento `idade` que recebe como argumentos a data de nascimento de uma pessoa e outra data posterior e devolve a idade da pessoa na segunda data.

```
>>> idade(cria_data(2, 1, 2003), cria_data(1, 1, 2005))
    1
>>> idade(cria_data(2, 1, 2003), cria_data(2, 1, 2005))
    2
```

8.5 Tipo timestamp

1. **(2)** Defina e implemente o tipo *timestamp*, para representar um instante de tempo. Um instante de tempo é um par *data* - *relogio*.

Capítulo 9

Funções revisitadas

1. **(2)** Escreva uma função recursiva `numero_digitos` que recebe um número inteiro não negativo `n`, e devolve o número de dígitos de `n`.

```
>>> numero_digitos(0)
    1
>>> numero_digitos(12426374856)
    11
```

2. **(2)** Escreva uma função recursiva `soma_digitos_pares` que recebe um número inteiro não negativo `n`, e devolve a soma dos dígitos pares de `n`.

```
>>> soma_digitos_pares(0)
    0
>>> soma_digitos_pares(12426374856)
    32
```

3. **(2)** Escreva uma 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.

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

4. **(2)** Escreva uma função recursiva `muda_digito` que recebe três números inteiros positivos `n`, `p` e `d` e devolve o inteiro que resulta de substituir o dígito na posição `p` de `n` por `d`. Note que esta função considera que as posições começam em 1 e contam-se da direita para a esquerda.

```
>>> muda_digito(12345, 2, 8)
    12385
>>> muda_digito(12345, 9, 6)
    600012345
```

5. **(2)** Escreva uma função recursiva `numero_ocorrencias_lista` que recebe uma lista e um número, e devolve o número de vezes que o número ocorre na lista e nas suas sublistas, se existirem.

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

6. **(2)** Escreva uma função recursiva `sublistas` que recebe uma lista, e tem como valor o número total de sublistas que esta contém.

```
>>> sublistas([1, 2, 3])
    0
>>> sublistas([[1], 2, [3]])
    2
>>> sublistas([[[[[1]]]]])
    4
>>> sublistas(['a', [2, 3, [[[1]], 6, 7], 'b']])
    4
```

7. **(3)** (*Teste de 18/01/2013*) Escreva uma função recursiva `junta_ordenadas` que recebe como argumentos duas listas ordenadas contendo números, em que cada lista não contém repetições, e que devolve uma lista ordenada, sem elementos repetidos, correspondente à junção das duas listas. Por exemplo:

```
>>> junta_ordenadas([1, 2, 5, 19], [1, 3, 5, 6, 7, 8])
    [1, 2, 3, 5, 6, 7, 8, 19]
```

8. A função `somatorio` apresentada no livro

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

é apenas a mais simples de um vasto número de abstracções 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(lambda x: x ** 2, 9, lambda x: x + 3, 21)
```

   (a) **(2)** Diga o que fazem as seguintes utilizações dessa função:

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

ii. `somatorio(lambda x: x * x, 5, lambda x: x + 5, 500)`

iii. `somatorio(lambda x: somatorio(lambda x: x, 1, lambda x: x + 1, x), 1, lambda x: x + 1, 5)`

(b) **(2)** Defina uma função `piatorio` que calcula o produto dos termos de uma função entre dois limites especificados.

(c) **(2)** Mostre como definir o factorial em termos da utilização da função `piatorio`.

9. A conversão de valores é uma operação comum em programação. Por exemplo, convertem-se temperaturas em graus Farenheit para graus Centígrados, horas locais em Lisboa para horas locais em Nova Iorque, etc.

   (a) **(2)** Escreva uma função `converte` que recebe a função correspondente à função de conversão e o valor a converter e devolve o valor convertido. Por exemplo, se `far_cent` for a função correspondente à função de conversão de graus Farenheit em Centígrados definida como

   ```
   def far_cent(f):
       return  ((f - 32) * 5) / 9
   ```

   a avaliação de `converte(far_cent, 32)` tem o valor `0.0`.

   (b) **(2)** Escreva uma função `lis_ny` para converter uma hora local em Lisboa (um número inteiro entre 0 e 23) para a hora local em Nova Iorque (onde são menos cinco horas do que em Lisboa). Tenha cuidado com a passagem da meia noite. Utilize a função converte da alínea anterior e a função `lis_ny` para mostrar qual a hora em Nova Iorque quando são 3 da manhã em Lisboa.

10. **(2)** Escreva uma função `nenhum_p` que recebe um número inteiro positivo `n` e um predicado unário `p`, e devolve *verdadeiro* se nenhum inteiro positivo menor ou igual a `n` satisfaz `p` e *falso* no caso contrário.

    ```
    >>> nenhum_p(87, lambda x: x % 100 == 0)
        True
    >>> nenhum_p(187, lambda x: x % 100 == 0)
        False
    ```

11. **(2)** Escreva uma função `conta_p` que recebe um número inteiro positivo `n` e um predicado unário `p`, e devolve o número de inteiros positivos menores ou iguais a `n` que satisfazem `p`.

    ```
    >>> conta_p(87, lambda x: x % 100 == 0)
        0
    >>> conta_p(487, lambda x: x % 100 == 0)
        4
    ```

12. **(2)** Escreva uma função `todos_lista` que recebe uma lista e um predicado unário, e devolve *verdadeiro* caso todos os elementos da lista satisfaçam o predicado e *falso* no caso contrário.

```
>>> todos_lista([4, 5, 6], lambda x: x > 5)
    False
>>> todos_lista([4, 5, 6], lambda x: x >= 4)
    True
```

13. **(1)** A composição de duas funções de um argumento, $f(x)$ e $g(x)$, é a função $f(g(x))$. Escreva uma função `fn_composta` que recebe como argumentos duas funções de um argumento e que devolve a função correspondente à composição dos seus argumentos. Por exemplo, com a sua função, seria gerada a seguinte interacção:

```
>>> fn_composta(lambda x: x * x, lambda x: x + 6)(3)
    81
```

14. **(1)** A soma de duas funções de um argumento, $f(x)$ e $g(x)$, é a função $f(x) + g(x)$. Escreva uma função `fn_soma` que recebe como argumentos duas funções de um argumento e que devolve a função correspondente à soma de f com g. Por exemplo, com a sua função seria gerada a seguinte interacção:

```
>>> fn_soma(lambda x: x + 3, lambda y: y * 10)(12)
    135
```

15. **(2)** Defina uma função `funcao_i` que recebe funções para calcular as funções reais de variável real f, g e h e devolve uma função que se comporta como a seguinte função matemática:

$$i(x) = (2f(x))^3 + 4g(x)^5 - h(x^6)$$

Capítulo 10

Recursão e iteração

1. **(1)** Considere a seguinte função:

```
def misterio(a):
    def misterio_aux(b, c):
        if b == 0:
            return True
        elif c == 0:
            return False
        else:
            return misterio_aux(b - 2, c - 2)

    if not(isinstance(a, int)) or a < 0:
        raise ValueError ('misterio: arg devia ser inteiro nao negativo')
    else:
        return misterio_aux(a, a + 1)
```

   (a) Explique o que é calculado pela função `misterio`.

   (b) A função `misterio` é recursiva? E a função `misterio_aux`? Justifique a resposta.

   (c) De que tipo é o processo gerado pela função `misterio`? Justifique a sua resposta.

2. **(1)** Considere a seguinte função:

```
def misterio(n):
    def misterio_aux(n, ac):
        if n < 10:
            return 10 * ac + n
        else:
            return misterio_aux(n // 10, 10 * ac + n % 10)

    if not(isinstance(n, int)) or n < 0:
        raise ValueError ('misterio: arg devia ser inteiro nao negativo')
    else:
        return misterio_aux(n, 0)
```

(a) Entre as funções apresentadas, `misterio` e `misterio_aux`, existe alguma que seja recursiva? Justifique a sua resposta.

(b) Mostre a evolução do processo gerado pela avaliação de `misterio(149)`.

(c) De que tipo é o processo gerado pela função `misterio`? Justifique a sua resposta.

(d) Explique porque é que o teste de validade do argumento deve ser feito na função `misterio` e não na função `misterio_aux`.

3. **(2)** Considere a seguinte função:

```
def misterio(x, n):
    if n == 0:
        return 0
    else:
        return x * n + misterio(x, n - 1)
```

(a) Mostre a evolução do processo gerado pela avaliação de `misterio(2, 3)`.

(b) A função apresentada é uma função recursiva? Justifique.

(c) De que tipo é o processo gerado pela função apresentada? Justifique.

(d) Se a função apresentada é uma função recursiva de cauda, defina uma nova função recursiva por transformação da primeira de modo a deixar operações adiadas. Se é uma função recursiva com operações adiadas, defina uma função recursiva de cauda.

4. **(2)** Suponha que as operações de multiplicação (`*`) e potência (`**` e `pow`) 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. Com efeito,

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

n	$1 + \ldots + (2n-1)$	n^2
1	1 =	1
2	1+3 =	4
3	1+3+5 =	9
4	1+3+5+7 =	16
5	1+3+5+7+9 =	25
6	1+3+5+7+9+11 =	36
...	...	...

Note que o dobro de um número também não pode ser calculado recorrendo à operação de multiplicação. Escreva uma função que calcule 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 e um ciclo `while`.

   (d) Usando um processo iterativo e um ciclo `for`.

5. **(2)** Escreva uma função que recebe três argumentos, a, b e n, e que devolve o valor de somar n vezes a a b, isto é, $b + a + a + \cdots + a$, n vezes.

   (a) Usando recursão com operações adiadas;

   (b) Usando recursão de cauda;

   (c) Usando um processo iterativo e um ciclo `while`.

   (d) Usando um processo iterativo e um ciclo `for`.

6. **(2)** (Exerc. 11 do Cap. 3) Escreva uma função `num_divisores` que recebe um número inteiro positivo `n`, e devolve o número de divisores de `n`. No caso de `n` ser 0 deverá devolver 0.

   (a) Usando recursão com operações adiadas;

   (b) Usando recursão de cauda;

   (c) Usando um processo iterativo e um ciclo `while`.

   (d) Usando um processo iterativo e um ciclo `for`.

```
>>> num_divisores(20)
    6
>>> num_divisores(13)
    2
```

7. **(2)** (Exerc. 12 do Cap. 3) Escreva uma função `soma_divisores` que recebe um número inteiro positivo `n`, e devolve a soma de todos os divisores de `n`. No caso de `n` ser 0 deverá devolver 0.

   (a) Usando recursão com operações adiadas;

   (b) Usando recursão de cauda;

   (c) Usando um processo iterativo e um ciclo `while`.

   (d) Usando um processo iterativo e um ciclo `for`.

```
>>> soma_divisores(20)
    42
>>> soma-divisores(13)
    14
```

8. **(2)** Escreva uma função `conta_pares_tuplo` que recebe um tuplo de inteiros e devolve o número de elementos pares no tuplo.

   (a) Usando recursão com operações adiadas;

   (b) Usando recursão de cauda;

(c) Usando um processo iterativo.

```
>>> conta_pares_tuplo((4, 5, 6))
    2
>>> conta_pares_tuplo((3, 5, 7))
    0
>>> conta_pares_tuplo((3, ))
    0
```

9. **(3)** Escreva uma função `troca_ocorrencias_lista` que recebe uma lista e dois valores arbitrários, `a` e `b`, e devolve uma nova lista, obtida a partir da original substituindo todas as ocorrências de `a` por `b`.

   (a) Usando recursão com operações adiadas;

   (b) Usando recursão de cauda;

   (c) Usando um processo iterativo.

```
>>> troca_ocorrencias_lista([(2, 3), 'a', 3, True, [7, 3]],  3, 'b')
    [(2, 3), 'a', 'b', True, [7, 3]]
>>> troca_ocorrencias_lista([(1, 3), True, [1, 3]],  [1, 3], 'b')
    [(1, 3), True, 'b']
>>> troca_ocorrencias_lista(['a', [1, 2], [False, [1, 2]]], [1, 2], 4)
    ['a', 4, [False, [1, 2]]]
>>> troca_ocorrencias_lista([], 2, 4)
    []
```

Nota: Atenção à ordem dos elementos.

10. **(4)** Escreva uma função `troca_ocorrencias_lista_sublistas` que recebe uma lista e dois valores arbitrários, `a` e `b`, e devolve uma nova lista, obtida a partir da original substituindo todas as ocorrências de `a` por `b` na lista e nas suas sublistas.

    (a) Usando recursão com operações adiadas;

    (b) Usando recursão de cauda;

    (c) Usando um ciclo `for`.

    (d) Indique, justificando, qual o tipo de processo gerado pela função definida na alínea anterior.

```
>>> troca_ocorrencias_lista_sublistas_a([(2, 3), 'a', 3, True, [7, 3]],  3, 'b')
    [(2, 3), 'a', 'b', True, [7, 'b']]
>>> troca_ocorrencias_lista_sublistas_a([(1, 3), True, [1, 3]],  [1, 3], 'b')
    [(1, 3), True, 'b']
>>> troca_ocorrencias_lista_sublistas_a(['a', [1, 2], [False, [1, 2]]], [1, 2], 4)
    ['a', 4, [False, 4]]
>>> troca_ocorrencias_lista_sublistas_a([], 2, 4)
    []
```

Nota: Atenção à ordem dos elementos.

11. **(3)** Escreva uma função `numero_digitos` que recebe um número inteiro não negativo `n`, e devolve o número de dígitos de `n`.

    (a) Usando recursão com operações adiadas;

    (b) Usando recursão de cauda;

    (c) Usando um processo iterativo.

```
>>> numero_digitos(3)
    1
>>> numero_digitos(0)
    1
>>> numero_digitos(1012)
    4
```

12. **(3)** Escreva uma função recursiva `digito_pos`, que recebe dois números inteiros positivos `n` e `d`, e devolve o dígito de ordem `d` de `n`.

```
>>> digito_pos(12345, 1)
    5
>>> digito_pos(12345, 4)
    2
>>> digito_pos(12345, 8)
    0
```

13. **(4)** Um número é uma *capicua* se se lê igualmente da esquerda para a direita e vice-versa. Escreva uma função recursiva `e_capicua`, que recebe um número inteiro positivo `n`, e devolve *verdadeiro* se o número for uma capicua e *falso* caso contrário.

    Sugestão: Use as funções definidas nos exercícios anteriores.

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

14. **(3)** O *espelho* de um número inteiro positivo é o resultado de inverter a ordem de todos os seus algarismos. Escreva uma função recursiva de cauda `espelho`, que recebe um número inteiro positivo `n`, não divisível por 10, e devolve o seu espelho.

```
>>> espelho(391)
    139
>>> espelho(45679)
    97654
```

15. **(3)** Utilizando as funções `e_capicua` e `espelho`, escreva uma função recursiva `gera_capicua` que gera uma capicua usando o seguinte algoritmo: dado um número inteiro positivo, gera-se em primeiro lugar um outro inteiro com os mesmos dígitos mas colocados por ordem inversa e somam-se os dois. Se o inteiro correspondente a esta soma for uma capicua, devolve-se o seu valor; se ainda não for uma capicua, devolve-se o resultado de repetir o procedimento inicial com esta soma.

    Exemplo:
    Inteiro dado: 329
    Inteiro obtido por inversão: 923
    Soma: 1252 (ainda não é capicua)

    Inteiro dado: 1252
    Inteiro obtido por inversão: 2521
    Soma: 3773 (já é capicua)

    ```
    >>> gera_capicua(329)
        3773
    ```

16. **(2)** (*Teste de 18/01/2013*) Considere a função g, definida para inteiros não negativos do seguinte modo:

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

    (a) Escreva uma função recursiva em Python para calcular o valor de $g(n)$.

    (b) Siga o processo gerado por $g(3)$, indicando todos os cálculos efectuados.

    (c) Que tipo de processo é gerado por esta função?

Capítulo 12

Programação com objectos

Nos exercícios deste capítulo, deve validar os argumentos dos construtores, mas não deve validar os argumentos dos outros métodos, a não ser que isso seja pedido explicitamente.

1. **(2)** Defina a classe `contador_limitado` cujo construtor recebe dois números inteiros, correspondendo ao limite inferior e superior do contador. O contador quando é criado tem como valor inicial o limite inferior. Os outros métodos suportados pela classe são:

   - `consulta`, que devolve o valor do contador.
   - `inc`, que permite incrementar de uma unidade o valor do contador e devolve o valor do contador no final. Se se tentar incrementar o valor do contador para cima do limite superior este não é alterado.
   - `dec`, que permite decrementar de uma unidade o valor do contador e devolve o valor do contador no final. Se se tentar decrementar o valor do contador para baixo do limite inferior este não é alterado.

   Mostra-se a seguir um exemplo de interacção:

   ```
   >>> c1 = contador_limitado(3, 5)
   >>> c1.inc()
       4
   >>> c1.consulta()
       4
   >>> c1.inc()
       5
   >>> c1.inc()
       5
   >>> c1.dec()
       4
   >>> c1.dec()
       3
   >>> c1.dec()
       3
   ```

2. **(2)** Crie a classe `garrafa` cujo construtor recebe a capacidade da garrafa em litros. A garrafa inicialmente será criada vazia. Os outros métodos suportados pela classe são:

- `capacidade`, que devolve a capacidade total da garrafa.
- `nivel`, que devolve o volume de líquido presente na garrafa.
- `despeja`, que recebe a quantidade de líquido a remover da garrafa, em litros. Se a quantidade exceder o volume presente na garrafa, o volume presente na garrafa passa a ser 0.
- `enche`, que recebe a quantidade de líquido a colocar na garrafa. Se a quantidade de líquido a colocar na garrafa fizer com que esta ultrapasse a sua capacidade, o volume presente na garrafa passa a ser igual à sua capacidade.

Mostra-se a seguir um exemplo de interacção:

```
>>> g1 = garrafa(1.5)
>>> g1.nivel()
    0
>>> g1.capacidade()
    1.5
>>> g1.despeja(2)
>>> g1.nivel()
    0
>>> g1.enche(2)
>>> g1.nivel()
    1.5
>>> g1.despeja(1)
>>> g1.nivel()
    0.5
>>> g1.despeja(1)
>>> g1.nivel()
    0
```

3. **(2)** Considere a classe `Point` e a classe `FigGeometrica` que implementam os TAIs `Ponto` e `Figura Geometrica`, definidas somo se segue:

   `TAI Ponto`

   - *Construtor*:
     - `ponto` : $\mathbb{N}_0 \times \mathbb{N}_0 \mapsto$ *ponto*
       `ponto`(x, y) tem como valor o ponto de coordenadas (x,y).
   - *Selectores*:
     - `coord_x` : *ponto* $\mapsto \mathbb{N}_0$
       `coord_x`(p) tem como valor a coordenada x de p.
     - `coord_y` : *ponto* $\mapsto \mathbb{N}_0$
       `coord_y`(p) tem como valor a coordenada y de p.
   - *Reconhecedor*:
     - `e_ponto` : *universal* $\rightarrow$ *lógico*
       `e_ponto`(arg) tem o valor *verdadeiro* se o *arg* for do tipo ponto e *falso* caso contrário.

- *Testes*:
  - `pontos_iguais` : *ponto* × *ponto* → *lógico*
    `pontos_iguais`(p_1, p_2), devolve o valor *verdadeiro* se os pontos p_1 e p_2 são iguais e *falso* caso contrário.
  - `pontos_colineares` : *ponto* × *ponto* × *ponto* → *lógico*
    `pontos_colineares`(p_1, p_2, p_3), devolve o valor *verdadeiro* se os pontos p_1, p_2 e p_3 são colineares e *falso* caso contrário.

`TAI Figura Geometrica`

- *Construtor*:
  - `figura_geometrica` : *string* × *lista_de_pontos* ↦ *figura_geometrica*
    `figura_geometrica`(c, lst) tem como valor a figura geométrica com a cor c e vértices nos pontos definidos na lista lst.
- *Selectores*:
  - `vertice` : *figura_geometrica* × *int* ↦ *ponto*
    `vertice`(f, i) tem como valor o iésimo vértice da figura f.
  - `numero_vertices` : *figura_geometrica* ↦ N_0
    `numero_vertices`(f) tem como valor o número de vértices da figura f.
  - `cor` : *figura_geometrica* ↦ *string*
    `cor`(f) tem como valor a cor da figura f.
- *Reconhecedor*:
  - `e_figura_geometrica` : *universal* → *lógico*
    `e_figura_geometrica`(arg) tem o valor *verdadeiro* se o *arg* for do tipo figura geometrica e *falso* caso contrário.
- *Testes*:
  - `figuras_iguais` : *figura_geometrica* × *figura_geometrica* → *lógico*
    `figuras_iguais`(f_1, f_2), devolve o valor *verdadeiro*
    se as figuras são iguais e *falso* caso contrário.
- *Transformadores*:
  - `desenha` : *figura_geometrica* ↦
    `desenha`(f) tem como resultado a apresentação gráfica da figura f.
  - `para_string` : *figura_geometrica* ↦ *string*
    `para_string`(f) tem como valor uma string que representa a figura f.

Tendo em conta as classes anteriores, implemente as classes `triangulo` e `retangulo` por extensão da classe `FigGeometrica`, de acordo com a especificação seguinte:

`TAI Triangulo`

- *Construtor*:
  - `triangulo` : *string* × *ponto* × *ponto* × *ponto* ↦ *triangulo*
    `triangulo`(c, p_1, p_2, p_3) tem como valor o triangulo com a cor c e vértices nos pontos p_1, p_2 e p_3, se estes não forem colineares.
- *Reconhecedor*:

  - `e_triangulo` : *universal* → *lógico*
    `e_triangulo`(arg) tem o valor *verdadeiro* se o *arg* for do tipo triangulo e *falso* caso contrário.
  - `e_triangulo_retangulo` : *universal* → *lógico*
    `e_triangulo_retangulo`(t) tem o valor *verdadeiro* se o triângulo t for retângulo e *falso* caso contrário. Um triângulo é retângulo se h2=a2+b2, de acordo com o Teorema de Pitágoras.

- *Testes*:
  - `triangulos_iguais` : *triangulo* × *triangulo* → *lógico*
    `triangulos_iguais`(t_1, t_2), devolve o valor *verdadeiro* se os triangulos são iguais e *falso* caso contrário.

`TAI Retangulo`

- *Construtor*:
  - `retangulo` : *string* × *ponto* × *ponto* ↦ *retangulo*
    `triangulo`(c, p_1, p_2) tem como valor o retangulo com a cor c e vértices opostos nos pontos p_1 e p_2.
- *Reconhecedor*:
  - `e_retangulo` : *universal* → *lógico*
    `e_retangulo`(arg) tem o valor *verdadeiro* se o *arg* for do tipo retangulo e *falso* caso contrário.
- *Testes*:
  - `retangulos_iguais` : *retangulo* × *retangulo* → *lógico*
    `retangulos_iguais`(r_1, r_2), devolve o valor *verdadeiro* se os retangulos são iguais e *falso* caso contrário.

Mostra-se a seguir um exemplo de interacção:

```
>>> j = janela(400, 200)
>>> t = Triangulo('red', Ponto(10, 10), Ponto(50, 30), Ponto(20, 70))
>>> r = Retangulo('blue', Ponto(100, 100), Ponto(300, 150))
>>> j.mostra([t, r])
>>> j.fecha()
```

4. **(2)** Considere a classe `numero` implementada a seguir, que representa um número inteiro e as operações de igualdade, soma, multiplicação e conversão para cadeia de caracteres.

```
class numero:
    def __init__(self, n):
        if (isinstance(n, int)):
            self.num = n
        else:
            raise ValueError('numero: argumento deve ser um inteiro')

    def get_num(self):
        return self.num
```

```
    def __eq__(self, outro):
        return self.num == outro.num

    def __str__(self):
        return str(self.num)

    def __repr__(self):
        return str(self.num)

    def __add__(self, outro):
        return numero(self.num + outro.num)

    def __mul__(self, outro):
        return numero(self.num * outro.num)
```

Implemente a classe `racional`, por extensão da classe `numero`, tendo em conta as definições apresentadas no capítulo de Tipos Abstratos de Informação.

5. **(3)** Tendo em conta as classes `numero` e `racional` implemente a classe `calculadora` que aplica as operações de adição e multiplicação a dois argumentos. Considere apenas as operações seguintes:

   - `calculadora` : *numero* × *numero* × *string* ↦ *numero*
     `calculadora`($n1, n2, op$) inicia a calculadora com os argumentos n1 e n2, e operação op.
   - `aplica_operacao` :↦ *numero*
     `aplica_operacao`() devolve o resultado de aplicar a operação aos dois argumentos.

   Mostra-se a seguir um exemplo de interacção:

```
>>> calc = calculadora(numero(3), numero(4), "+")
>>> print(calc.aplica_operacao())
    7
>>> calc = calculadora(racional(1,3), racional(3,4), "*")
>>> print(calc.aplica_operacao())
    3/12
>>> calc = calculadora(numero(3), racional(3,4), "+")
>>> print(calc.aplica_operacao())
    15/4
```

6. **(3)** Pretende-se definir a classe `cartao_telefonico` cujo construtor recebe o tarifáio em vigor. O `cartao_telefonico` deve registar o número de minutos consumidos em cada uma das tarifas do tarifário, para além do custo total das chamadas efectuadas. O tarifário é representado por um dicionário, em que cada tipo de chamada é representado por uma cadeia de carateres a que está associado o custo por minuto de conversação, por exemplo:

```
{'local':2, 'nacional':12, 'movel':20, 'internacional':41}
```

   Os outros métodos suportados pela classe são:

- `consulta`, escreve para o ecrã e devolve o custo total das chamadas efetuadas.
- `consulta-tarifa`, escreve para o ecrã e devolve o número de minutos das chamadas efetuadas na tarifa que recebe como argumento.
- `chamada`, recebe a tarifa e a duração da chamada em minutos e efetua uma chamada, atualizando o número de minutos gastos nessa tarifa e o valor do custo total das chamadas.

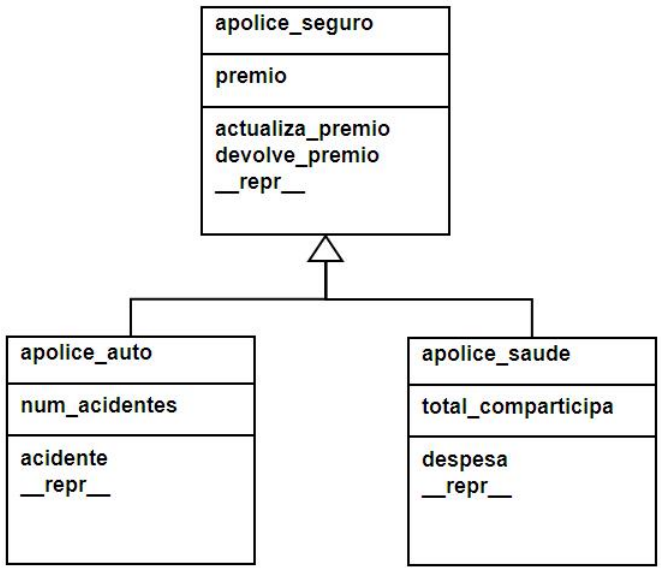
Mostra-se a seguir um exemplo de interacção:

```
>>> tarifario = {'local':2, 'nacional':12, 'movel':20, 'internacional':41}
>>> c1 = cartao_telefonico(tarifario)
>>> c1.consulta()
    custos = 0
>>> c1.chamada('local', 5)
>>> c1.consulta()
    custos = 10
>>> c1.consulta_tarifa('local')
    minutos = 5
```

7. (4) Pretende-se definir a classe `cartao_telefonico_pre_pago` por extensão da classe `cartao_telefonico`. Neste caso, para além do tarifário, o construtor recebe também o valor do saldo inicial do cartão. Implemente a nova classe, redefinindo apenas os métodos necessários e o método para carregamento do cartão - `carrega`, que recebe o montante a carregar. Note que só se pode efetuar uma chamada caso o cartão tenha saldo disponível. Mostra-se a seguir um exemplo de interacção:

```
>>> tarifario = {'local':2, 'nacional':12, 'movel':20, 'internacional':41}
>>> c2 = cartao_telefonico_pre_pago(tarifario, 100)
>>> c2.chamada('local', 5)
>>> c2.consulta()
    custos = 10
>>> c2.chamada('movel', 5)
    chamada: nao pode efetuar chamada por falta de saldo
    saldo = 90
>>> c2.consulta()
    custos = 10
>>> c2.chamada('nacional', 2)
>>> c2.consulta()
    custos = 34
```

8. (4) Considere a seguinte hierarquia de classes que indica que há dois tipos de apólices de seguro - as apólices do ramo automóvel e as apólices de saúde.



(a) Implemente a classe `apolice_seguro` de acordo com o diagrama apresentado e a descrição dos seus métodos:

- `apolice_seguro(valor)`, cria uma nova apólice com o valor indicado para o prémio.
- `actualiza_premio(valor)`, atribui o novo valor indicado para o prémio.
- `devolve_premio()`, devolve o valor do prémio da apólice.
- Método que o Python usa para apresentar a representação externa. Por exemplo,

```
>>> apolice_seguro(1000)
    premio: 1000.0
```

(b) Implemente a classe `apolice_auto` de acordo com o diagrama apresentado e a descrição dos seus métodos:

- `acidente()`, permite a comunicação de um acidente, o que incrementa o número de acidentes e faz aumentar o prémio em 20 por cento.
- Método que o Python usa para apresentar a representação externa. Por exemplo,

```
>>> apolice_auto(1000)
    premio: 1000.0   num de acidentes: 0
```

(c) Implemente a classe `apolice_saude` de acordo com o diagrama apresentado e a descrição dos seus métodos:

- `despesa(quantia)`, se *quantia* > 10, adiciona a `total_comparticipa` metade de (*quantia* $- 10$). Se *quantia* ≤ 10 não há qualquer comparticipação. Considera-se que 10 é o valor da franquia e 50 por cento a percentagem de comparticipação. Por exemplo, para uma despesa de 50, a comparticipação será 20.
- Método que o Python usa para apresentar a representação externa. Por exemplo,

```
>>> apolice_saude(1000)
    premio: 1000.0   total de comparticipacoes: 0.0
```

Considere o seguinte exemplo de interacção:

```
>>> ap_001 = apolice_auto(1000)
>>> ap_001
    premio: 1000.0   num de acidentes: 0
>>> ap_002 = apolice_saude(200)
>>> ap_002
    premio: 200   total de comparticipacoes: 0
>>> ap_001.acidente()
>>> ap_001
    premio: 1200.0   num de acidentes: 1
>>> ap_001.acidente()
>>> ap_001
    premio: 1440.0   num de acidentes: 2
>>> ap_002.despesa(50)
>>> ap_002
    premio: 200.0   total de comparticipacoes: 20.0
>>> ap_002.actualiza_premio(235)
>>> ap_002.devolve_premio()
    235
```

9. **(4)** Considere a função g, definida para inteiros não negativos do seguinte modo:

```
def g(n):
    if n == 0:
        return 0
    else:
        return n - g(g(n-1))
```

Como pode verificar, a função calcula várias vezes o mesmo valor quando chamada com diferentes argumentos. Para evitar este problema, podemos definir uma classe, `mem_g`, cujo estado interno contém informação sobre os valores de g já calculados, apenas calculando um novo valor quando este ainda não é conhecido. Esta classe possui um método `calcula` que calcula o valor de g para o inteiro que é seu argumento. Por exemplo,

```
>>> g = mem_g()
>>> g.calcula(0)
    0
>>> g.calcula(12)
    8
```

Defina a classe *mem_ g*.

Capítulo 13

Estruturas Lineares

Considere o TAI `Fila` definido nas aulas teóricas e implementado pela classe fila, como se apresenta em seguida:

- *Construtores:*
  - *nova_fila* : $\{\} \mapsto fila$
    nova_fila() tem como valor uma fila sem elementos.
- *Selectores:*
  - *inicio* : *fila* $\mapsto$ *elemento*
    inicio(*fila*) tem como valor o elemento que se encontra no início da fila *fila*. Se a fila não tiver elementos, o valor desta operação é indefinido.
  - *comprimento* : *fila* $\mapsto \mathbb{N}_0$
    comprimento(*fila*) tem como valor o número de elementos da fila.
- *Modificadores:*
  - *coloca* : *fila* $\times$ *elemento* $\mapsto$ *fila*
    coloca(*fila*, elm) altera de forma permanente a $fila$ para a fila que resulta em inserir $elem$ no fim da $fila$. Tem como valor a fila que resulta de inserir o elemento elm no fim da fila *fila*.
  - *retira* : *fila* $\mapsto$ *fila*
    retira(*fila*) altera de forma permanente a $fila$ para a fila que resulta em remover o elemento no início da $fila$. Tem como valor a fila que resulta de remover o elemento que se encontra no início da fila *fila*. Se a fila não contiver elementos, o valor desta operação é indefinido.
- *Transformadores:*
  - *fila_para_lista* : *fila* $\mapsto$ *lista*
    fila_para_lista(*fila*) devolve a lista com o mesmos elementos que $fila$, e na mesma ordem, estando o primeiro elemento da fila na primeira posição da lista.
- *Reconhecedores:*

  - *fila* : *universal* $\mapsto$ *logico*
    fila(*arg*) tem o valor *verdadeiro*, se *arg* é uma fila, e tem o valor *falso*, em caso contrário.
  - *fila_vazia* : *fila* $\mapsto$ *logico*
    fila_vazia(*fila*) tem o valor *verdadeiro*, se *fila* é a fila vazia, e tem o valor *falso*, em caso contrário.

- *Testes:*
  - *filas_iguais* : *fila* $\times$ *fila* $\mapsto$ *logico*
    filas_iguais($fila_1, fila_2$) tem o valor *verdadeiro*, se $fila_1$ é igual a $fila_2$, e tem o valor *falso*, em caso contrário.

```
class fila:

    def __init__(self):
        self.elems = []

    def inicio(self):
        if self.elems != []:
            return self.elems[0]
        else:
            raise IndexError('inicio: a fila nao tem elementos')

    def comprimento(self):
        return len(self.elems)

    def coloca(self, elemento):
        self.elems.insert(len(self.elems), elemento)
        return self

    def retira(self):
        if self.elems != []:
            del(self.elems[0])
            return self
        else:
            raise IndexError('retira: a fila nao tem elementos')

    def fila_para_lista(self):
        # tem que fazer uma copia da lista, usando elems[:]
        return self.elems[:]

    def fila_vazia(self):
        return self.elems == []

    def __eq__(self, outra):
        return self.elems == outra.elems

    def __repr__(self):
        res = '< '
        for e in self.elems:
            res = res + str(e) + ' '
        res = res + '<'
        return res
```

```
print('================Fila')
f = fila()
print(f.coloca('a'))
print(f.coloca(2))
print(f.coloca(3.2))
print(f.fila_para_lista())
print(f.fila_para_lista()[2])
print('antes:', f)
f.fila_para_lista()[1] = 50
print('depois:', f)
print('comprimento:', f.comprimento())
print('fila_vazia:', f.fila_vazia())
print(f.retira())
print(f.retira())
print(f.retira())
#print(f.retira())
print('comprimento:', f.comprimento())
print(f.coloca('ola'))
print('inicio:', f.inicio())
f2 = fila()
print('fila_vazia f2:', f2.fila_vazia())
print('filas_iguais:', f == f2)
print(f2.coloca('ola'))
print('filas_iguais:', f == f2)
```

1. A estrutura de informação *DEQUE* (do Inglês "Double Ended Queue") corresponde a uma fila à qual novos elementos podem ser adicionados a qualquer das extremidades e elementos podem ser removidos de qualquer das extremidades.

   (a) **(2)** Defina as operações básicas do TAI *DEQUE*

   (b) **(3)** Implemente a classe `deque` como extensão da classe `fila`.

2. Uma *fila de prioridades* é uma estrutura de informação composta por um certo número de filas, cada uma das quais associada a uma determinada prioridade.

   Suponha que desejava criar uma fila de prioridades com duas prioridades, *urgente* e *normal*. Nesta fila de prioridades, os novos elementos são adicionados à fila, indicando a sua prioridade, e são colocados no fim da fila respectiva. Os elementos são removidos da fila através da remoção do elemento mais antigo da fila urgente. Se a fila urgente não tiver elementos, a operação de remoção remove o elemento mais antigo da fila normal. Existe uma operação para aumentar a prioridade, a qual remove o elemento mais antigo da fila normal e coloca-o como último elemento da fila urgente.

   (a) **(2)** Especifique as operações básicas para o tipo *fila de prioridades* (com prioridades urgente e normal).

   (b) **(4)** Implemente a classe `fila de prioridades` (com prioridades urgente e normal) usando a classe `fila`.

Capítulo 14

Árvores Binárias

14.1 Tipo Árvore

Considere o TAI `Árvore`, definido e implementado como se apresenta em seguida:

- *Construtores*:
  - $nova\_arv : \{\} \mapsto árvore$
    $arvore()$ tem como valor uma árvore vazia.
  - $cria\_arv : elemento \times árvore \times árvore \mapsto árvore$
    $arvore(raiz, a_{esq}, a_{dir})$ tem como valor a árvore com raiz $raiz$, com árvore esquerda a_{esq} e com árvore direita a_{dir}.
- *Selectores*:
  - $raiz : árvore \mapsto elemento$
    $raiz(árv)$ recebe uma árvore, $árv$, e tem como valor a sua raiz. Se a árvore for vazia, o valor desta operação é indefinido.
  - $arv\_esq : árvore \mapsto árvore$
    $arv\_esq(árv)$ recebe uma árvore, $árv$, e tem como valor a sua árvore esquerda. Se a árvore for vazia, o valor desta operação é indefinido.
  - $arv\_dir : árvore \mapsto árvore$
    $arv\_dir(árv)$ recebe uma árvore, $árv$, e tem como valor a sua árvore direita. Se a árvore for vazia, o valor desta operação é indefinido.
- *Reconhecedor*:
  - $arv : universal \mapsto lógico$
    $arv(arg)$ tem o valor $verdadeiro$ se arg é uma árvore e tem o valor $falso$, em caso contrário.
  - $arv\_vazia : árvore \mapsto lógico$
    $arv\_vazia(árv)$ tem o valor $verdadeiro$ se $árv$ é uma árvore vazia e tem o valor $falso$, em caso contrário.
- *Testes*:

– $arv\_iguais : árvore \times árvore \mapsto lógico$
$arv\_iguais(árv_1, árv_2)$ tem o valor *verdadeiro* se $árv_1$ e $árv_2$ são árvores iguais e tem o valor $falso$, em caso contrário.

```
def nova_arv():
    return []

def cria_arv(r, a_e, a_d):
    if arv(a_e) and arv(a_d):
        return [r, a_e, a_d]
    else:
        raise ValueError ('cria_arv: o segundo e terceiro \
        argumentos devem ser arvores')

def raiz(a):
    if a == []:
        raise ValueError ('raiz: a arvore e vazia')
    else:
        return a[0]

def arv_esq(a):
    if a == []:
        raise ValueError ('arv_esq: a arvore e vazia')
    else:
        return a[1]

def arv_dir(a):
    if a == []:
        raise ValueError ('arv_dir: a arvore e vazia')
    else:
        return a[2]

def arv(x):
    if isinstance(x, list):
        return x == [] or (len(x) == 3 and arv(x[1]) and arv(x[2]))
    else:
        return False

def arv_vazia(a):
    return a == []

def arv_iguais(a1, a2):
    if arv_vazia(a1):
        return arv_vazia(a2)
    elif arv_vazia(a2):
        return False
    elif raiz(a1) == raiz(a2):
        return (arv_iguais(arv_esq(a1), arv_esq(a2)) and
                arv_iguais(arv_dir(a1), arv_dir(a2)))
    else:
        return False

def escreve_arv(a):

    def escreve_aux(a, indent):
        if arv_vazia(a):
            print(' ' * indent, '-')
```

```
        else:
            print(' ' * indent, raiz(a))
            escreve_aux(arv_esq(a), indent + 2)
            escreve_aux(arv_dir(a), indent + 2)

    escreve_aux(a, 0)



print('================Testes do tai arvore')
a1 = cria_arv(1, nova_arv(), nova_arv())
a2 = cria_arv('ola', nova_arv(), nova_arv())
a3 = cria_arv(3.4, a1, a2)
print('arvore a3:')
escreve_arv(a3)
print('raiz da arv_esq de a3:', raiz(arv_esq(a3)))
print('arv_dir de a2:')
escreve_arv(arv_dir(a2))
print('a2 e vazia?:', arv_vazia(a2))
print('arv_dir de a2 e vazia?:', arv_vazia(arv_dir(a2)))
a4 = cria_arv(3.4, cria_arv(1, nova_arv(), nova_arv()), cria_arv('ola',
    nova_arv(), nova_arv()))
print('a3 e a4 sao arvores iguais?:', arv_iguais(a3, a4))
print('a2 e a1 sao arvores iguais?:', arv_iguais(a2, a1))
```

1. **(2)** Escreva uma função `arv_incrementa` que recebe como argumentos uma árvore de números e um número e devolve uma árvore de números cujos elementos correspondem à soma do número aos elementos da árvore na posição correspondente. Por exemplo,

```
>>> arv1 = cria_arv(1, nova_arv(), cria_arv(2, nova_arv(), nova_arv()))
>>> escreve_arv(arv_incrementa(arv1, 2))
 3
   -
   4
     -
     -
```

2. **(2)** Escreva uma função `arv_para_lista` que recebe como argumento uma árvore e devolve a lista dos elementos representados na árvore. Por exemplo,

```
>>> arv1 = cria_arv(1, nova_arv(), cria_arv(2, nova_arv(), nova_arv()))
>>> arv_para_lista(arv1)
    [1, 2]
```

3. **(2)** Escreva uma função `arv_soma_elementos` que recebe como argumento uma árvore de números e devolve a soma dos números representados na árvore. Por exemplo,

```
>>> arv1 = cria_arv(1, nova_arv(), cria_arv(2, nova_arv(), nova_arv()))
>>> arv_soma_elementos(arv1)
    3
```

4. **(2)** Escreva uma função `arv_espelha` que recebe uma árvore e devolve uma nova árvore idêntica à recebida, mas em que em cada nível da árvore se trocou a árvore esquerda com a direita, mantendo-se a raiz. Por exemplo,

```
>>> arv1 = cria_arv(1, nova_arv(), cria_arv(2, nova_arv(), nova_arv()))
>>> escreve_arv(arv_espelha(arv1))
 1
   2
     -
     -
   -
```

5. Considere uma função `arv_transforma` que recebe como argumentos uma árvore e uma função aplicável aos elementos da árvore e devolve uma nova árvore cujos elementos correspondem à aplicação da função aos elementos da árvore na posição correspondente. Por exemplo,

```
>>> def quad(x): return x * x
>>> arv1 = cria_arv(1, nova_arv(), cria_arv(2, nova_arv(), nova_arv()))
>>> escreve_arv(arv_transforma(arv1, quad))
 1
   -
   4
     -
     -
```

   (a) **(2)** Escreva a função `arv_transforma`.

   (b) **(2)** Reescreva a função `arv_incrementa` utilizando a função `arv_transforma`.

6. Escreva uma função `arv_acumula` que recebe como argumentos uma árvore, uma operação de dois argumentos aplicável aos elementos da árvore e o elemento neutro dessa operação, e devolve o resultado de aplicar a operação aos elementos da árvore. Por exemplo,

```
>>> arv1 = cria_arv(1, nova_arv(), cria_arv(2, nova_arv(), nova_arv()))
>>> arv_acumula(arv1, lambda x,y: x*y, 1)
    2
```

   (a) **(2)** Escreva a função `arv_acumula`.

   (b) **(2)** Reescreva a função `arv_soma_elementos` utilizando a função `arv_acumula`.

7. Considere uma função `aplica` que recebe como argumentos uma árvore, uma função f de três argumentos e um valor. Se a árvore recebida for vazia, `aplica` devolve o valor recebido como argumento. Se a árvore recebida não for vazia, devolve o resultado de aplicar f a três argumentos: a raiz da árvore; o resultado de `aplica` aplicado à árvore esquerda da árvore, f e ao valor; e o resultado de `aplica` aplicado à árvore direita da árvore, f e ao valor.

(a) **(3)** Escreva a função `aplica`.

(b) **(2)** Reescreva a função `arv_para_lista` utilizando a função `aplica`.

(c) **(2)** Reescreva a função `arv_soma_elementos` utilizando a função `aplica`.