Capítulo 4

Tuplos e Ciclos Contados

1. Diga o que é escrito por cada uma das seguintes instruções. Execute primeiro os ciclos manualmente e só depois use o computador para verificar os resultados.

   (a)
   ```
   for i in range(1,6):
       for j in range (1,4):
           if (i % j) == 1:
               print(i+j)
   ```

   (b)
   ```
   i = 20
   while i > 5:
       print(4 * i)
       i = i - 2
   ```

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

2. Considere o seguinte código em Python:

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

   Escreva código equivalente utilizando um ciclo `for`.

3. Escreva em Python 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
```

4. Escreva em Python 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. Por exemplo

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

   Escreva duas versões da sua função, uma utilizando um ciclo `while` e outra utilizando um ciclo `for`.

5. Escreva em Python a função `digitos` que, dado um número inteiro positivo n, e verificando a correcção do argumento, devolve um tuplo de inteiros com os dígitos de n pela ordem que aparecem no mesmo. Caso o argumento não seja um inteiro positivo, deverá ser emitido um erro.

6. Considere a gramática em notação BNF:

   ⟨idt⟩ ::= ⟨letras⟩ ⟨numeros⟩

   ⟨letras⟩ ::= ⟨letra⟩ |
   ⟨letra⟩ ⟨letras⟩

   ⟨numeros⟩ ::= ⟨num⟩ |
   ⟨num⟩ ⟨numeros⟩

   ⟨letra⟩ ::= `A` | `B` | `C` | `D`

   ⟨num⟩ ::= `1` | `2` | `3` | `4`

   Escreva uma função em Python, chamada `reconhece`, que recebe como argumento uma cadeia de carateres 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
   ```

7. Escreva uma função em Python 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)
```

8. Escreva em Python 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)
```

9. Duas palavras de igual comprimento dizem-se "amigas"se o número de posições em que os respetivos caracteres diferem for inferior a 10%. Escreva uma função em Python 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
```

10. Escreva em Python uma função, `cc_para_int`, que converte uma cadeia de carateres num inteiro usando o código ASCII. Use o facto que a representação de um carácter tem três algarismos. Por exemplo

```
>>> cc_para_int('bom dia')
98111109032100105097
```

11. Considere a seguinte gramática em notação BNF, na qual o símbolo inicial é ⟨prim⟩:

    ⟨prim⟩ ::= a ⟨seg⟩ a | a ⟨prim⟩ a

    ⟨seg⟩ ::= b ⟨seg⟩ b | b ⟨ter⟩ b

    ⟨ter⟩ ::= c | c ⟨ter⟩

    Escreva uma função em Python, chamada `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('abccccba')
True
>>> reconhece('abcccccb')
False
>>>> reconhece('abc')
False
```

12. Escreva uma função em Python que recebe como parâmetros duas cadeias de caracteres de comprimento 1 (`C1` e `C2`) e escreve todas as possíveis combinações de três letras contendo os caracteres compreendidos entre `C1` e `C2`. Por exemplo, se `C1` for `'a'` e `C2` for `'c'` então algumas das combinações possíveis são: `aaa aab aac aba abb abc aca acb acc baa ...`

13. A sequência de Racamán,

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

    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,

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

    Escreva uma função em Python que recebe um inteiro positivo, n, e devolve um tuplo 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)
    ```

14. Escreva em Python uma função, `int_para_cc`, que converte um inteiro numa cadeia de carateres usando o código ASCII. Use o facto que a representação de um carácter tem três algarismos. Por exemplo

    ```
    >>> int_para_cc(97098)
    'ab'
    >>> int_para_cc(cc_para_int('bom dia'))
    'bom dia'
    ```

15. Considere a seguinte gramática em notação BNF:

    ⟨frase⟩ ::= `c` ⟨meio⟩ `r`

⟨meio⟩ ::= ⟨letra⟩$^+$

⟨letra⟩ ::= `a` | `d`

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

```
reconhece('cdr')
True
reconhece('cdaaaaaaddddddr')
True
reconhece('cdaaaaaaddddddra')
False
```

16. 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 descodifica uma cadeia de caracteres de acordo com o algoritmo apresentado. Não é necessário validar os dados de entrada. Por exemplo,

    ```
    >>> descodifica('acebd')
    'abcde'
    ```