Iteração: instruções `while`, `for`, `break`

Exercícios

1. Analise os excertos de código abaixo. Para cada excerto, tente prever quantas iterações vão ser executadas e que valores vão ser impressos. Depois visualize a execução de cada excerto usando o PythonTutor.com.

```
n = 4
while n > 0:
    print(n)
    n -= 1
#□
```

```
n = 1
while n < 1000:
    print(n)
    n *= 2
#□
```

```
for n in (1, 2, 5, 10, 20,
50):
    print(n)
#□
```

```
for c in "abracadabra":
    print(c)
#□
```

```
for n in range(10):
    print(n)
#□
```

```
for n in range(10, 0, -2):
    print(n)
#□
```

2. O programa `table.py` mostra uma tabela dos quadrados de quatro números naturais. Experimente-o. Modifique o programa para mostrar a tabela para números entre 1 e 20. Use a função `range`. Acrescente uma coluna para mostrar 2^n. Ajuste a largura das colunas e o alinhamento do cabeçalho para obter um resultado semelhante ao abaixo.

```
 n  n²     2**n
 1   1        2
 2   4        4
 3   9        8
   ...
19 361   524288
20 400  1048576
```

3. Considere a sequência real $(U_0, U_1, \ldots)$ onde o primeiro termo é $U_0 = 100$ e os seguintes são dados por $U_n = 1.01 \cdot U_{n-1} - 1.01$. O programa `sequenceUn.py` gera os primeiros 20 termos dessa sequência. Modifique o programa para mostrar todos os termos, enquanto forem positivos. Note que terá que usar uma instrução `while`. No fim, o programa deve dizer quantos termos mostrou.

4. Escreva uma função `factorial(n)` que calcule o fatorial de n, definido por $n! = 1 \times 2 \times 3 \times \cdots \times n$. Teste a função com diversos valores de n.

5. O jogo HiLo consiste em tentar adivinhar um número (inteiro) entre 1 e 100. No início, o programa escolhe um número aleatoriamente. Depois, o utilizador introduz um número e o programa indica se é demasiado alto (High), ou demasiado baixo (Low). Isto é repetido até o utilizador acertar no número. Nessa altura o programa indica quantas tentativas foram feitas e termina. O programa `hilo.py` já tem um instrução para gerar um número aleatório com a função `randrange` do módulo `random`. Complete o programa para fazer o resto do jogo.

6. O número π pode ser aproximado por uma versão truncada da série de Leibniz: $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$ ou, equivalentemente, $\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = \frac{\pi}{4}$.

   Escreva uma função, `leibnizPi4(n)`, que devolva a soma dos `n` primeiros termos dessa série. Teste esta função num programa que pede o valor `n` ao utilizador.

7. ** Escreva um programa que peça ao utilizador uma sequência de números reais. Para terminar a sequência, o utilizador pressiona ENTER, introduzindo uma linha vazia[1]. Nessa altura, o programa deve mostrar o valor máximo, o valor mínimo e a média dos números introduzidos. (Confira `examples/ex4sentinelTotal.py`).

8. ** A sequência de Fibonacci[2] é uma sequência de inteiros na qual cada elemento é igual à soma dos dois anteriores: 0, 1, 1, 2, 3, 5, 8, 13, …, ou seja, cada termo obtém-se como $F_n = F_{n-1} + F_{n-2}$. Os primeiros valores são definidos como $F_0 = 0$ e $F_1 = 1$. Escreva uma função `Fibonacci(n)` para calcular o n-ésimo número de Fibonacci. *Sugestão: em cada iteração atualize e guarde os dois últimos valores da sequência.*

9. ** Escreva uma função `isPrime(n)` que devolva `True` se o número n é primo e `False`, caso contrário. Um número N é primo se não tiver divisores além de 1 e de N. *Sugestão: tente dividir o número por 2, por 3, etc. Se encontrar um divisor exato, então o número não é primo.* Teste a função fazendo um programa que percorre todos os números entre 1 e 100 e indique para cada um se é primo ou não.

10. ** Escreva um programa que leia do teclado um número inteiro positivo, `N`, e imprima no ecrã a lista de todos os seus divisores próprios (todos os números naturais que dividem `N`, exceto o próprio `N`). O programa deve ainda indicar se `N` é um número *deficiente*, *perfeito* ou *abundante*. Tenha em conta as definições seguintes:

    a. *Número deficiente*: diz-se do número inteiro cuja soma dos seus divisores próprios é menor do que o próprio número. Por exemplo, 16 é um número deficiente porque 1+2+4+8 < 16

    b. *Número perfeito*: diz-se do número inteiro cuja soma dos seus divisores próprios iguala o próprio número. Por exemplo, 6 é um número perfeito porque 1+2+3 = 6

    c. *Número abundante*: diz-se do número inteiro cuja soma dos seus divisores próprios é superior ao próprio número. Por exemplo, 18 é um número abundante porque 1+2+3+6+9 > 18