

Funções: definição, parâmetros, variáveis locais

Exercícios

1. **Antes da aula:** Execute no PythonTutor este programa passo-a-passo: [happy](#).
2. **Antes da aula:** Responda a estes [exercícios sobre utilização e definição de funções](#).
3. O programa `bmi.py` serve para calcular o índice de massa corporal, mas está incompleto. O programa inclui três funções. Analise o seu funcionamento.
 - a. Complete a definição da função `bodyMassIndex` para calcular o índice pela razão $bmi = \frac{weight}{height^2}$. Complete os argumentos na invocação da função, dentro da função principal. Teste o programa.
 - b. Complete a função `bmiCategory` para devolver uma string com a categoria correspondente ao índice de massa corporal dado. Acrescente uma chamada a esta função na função principal, para obter o nome da categoria. Volte a testar.
4. Execute e analise o programa `poly.py`. Acrescente-lhe uma função para calcular o polinómio $p(x) = x^2 + 2x + 3$ e modifique a função `main` para mostrar os valores de $p(1)$, $p(2)$, $p(10)$ e $g(1 + p(3))$. Confira os resultados.
5. Defina uma função que devolva o maior dos seus dois argumentos. Por exemplo, `max2(4, 3)` deve devolver 4 e `max2(-3, -2)` deve devolver -2. Não pode usar a função pré-definida `max`. Use uma instrução `if` ou uma **expressão condicional**. Teste a função com vários conjuntos de argumentos.
6. No mesmo programa, crie uma função `max3` que devolva o maior dos seus 3 argumentos. Não pode usar a função `max`, nem instruções ou expressões condicionais. Recorra apenas à função `max2` que definiu atrás. Teste a nova função.
7. Escreva uma função, `tax(r)`, que implemente a seguinte função de ramos:

$$tax(r) = \begin{cases} 0.1r & \text{se } r \leq 1000 \\ 0.2r - 100 & \text{se } 1000 < r \leq 2000 \\ 0.3r - 300 & \text{se } 2000 < r \end{cases}$$

Use uma instrução `if-elif-else` e evite condições redundantes. Teste a função para diversos valores de `r` e confirme os resultados. Que valores deve testar?

8. Escreva uma função `intersects(a, b, c, d)` que devolva `True` se os intervalos $[a, b]$ e $[c, d]$ se intersectarem e devolva `False`, caso contrário. Pode admitir que $a \leq b \wedge c \leq d$. Experimente responder no [CodeCheck](#). *Sugestão: é mais simples definir quando os intervalos não se intersectam.*
9. Analise e execute o programa `dates.py`. Faça as correções indicadas abaixo.
 - a. A função `isLeapYear` deveria indicar quando um ano é bissexto, mas está errada. Corrija-a. Um **ano é bissexto** se for **múltiplo de 4**, com exceção dos fins de século

(múltiplos de 100), que só são bissextos se forem múltiplos de 400. Por exemplo: 1980, 1984, 2004 foram bissextos; 1800 e 1900 foram anos comuns, mas 2000 foi bissexto.

- b. A função `monthDays`, para determinar o número de dias de um mês, também está errada. Quando o mês é fevereiro, invoque a função anterior para determinar se o ano é bissexto e devolva 29 dias nesse caso.
 - c. Corrija a função `nextDay` para devolver o dia seguinte corretamente.
10. Complete a função `hms2sec` de forma a devolver o número de segundos correspondente a h horas, m minutos e s segundos. Faça no [CodeCheck](#).
 11. Complete a função `sec2hms` de forma a converter um número de segundos em horas, minutos e segundos. Faça no [CodeCheck](#). Note que a função pode devolver vários valores usando uma instrução `return h, m, s`, por exemplo. Quando chamar a função, pode atribuir o resultado a um tuplo de três variáveis, de forma a “desempacotar” o resultado.
 12. ** Escreva uma função `countdown(N)` que imprima uma contagem decrescente a partir de um número positivo N. Note que pode imprimir N e depois chamar `countdown(N-1)`. Teste a função com diversos valores de N.
 13. ** O algoritmo de Euclides para determinar o máximo divisor comum de dois números naturais baseia-se na igualdade seguinte:

$$mdc(a,b) = \begin{cases} b & \text{se } r=0 \\ mdc(b,r) & \text{se } r>0 \end{cases}$$

onde r é o resto da divisão de a por b . Escreva uma função para calcular o m.d.c. e teste-a com diversos pares de valores.