Programação de Computadores I – BCC701

Conteúdo Teórico em Linguagem Python

Módulo 6 – Funções definidas pelo usuário

2020/1

Elaborado pela Comissão de Unificação da Disciplina BCC701,
com a colaboração de professores e estagiários docentes
http://www.decom.ufop.br/bcc701/

DECOM – Departamento de Computação
ICEB – Instituto de Ciências Exatas e Biológicas
UFOP – Universidade Federal de Ouro Preto

Sumário

MÓDULO 6

Funções definidas pelo usuário

Nos módulos anteriores, aprendemos que o fluxo de um programa de computador é usualmente composto por três principais etapas: **Entrada**, **processamento** e **saída**. Aprendemos ainda a usar operações e funções definidos na própria linguagem **Python** para resolver nossos problemas. Relembre algumas funções definidas dentro do próprio **Python** que você já utilizou:

- Entrada e saída: **input** e **print**;
- Conversão de tipo: **int**, **float** e **str**;
- Cálculos matemáticos: **abs**, math.**sqrt**, math.**cos**, math.**ceil** e math.**pow**

Além das funções definidas dentro do próprio escopo da linguagem e seus módulos, podemos também definir **nossas próprias funções**. Isso permite reaproveitar, sempre que necessário, o mesmo **bloco de operações**.

As principais vantagens no uso de funções são:

- Códigos mais legíveis, especialmente para programas grandes;
- Manutenções e modificações no código são mais fáceis, com menor chance de prejudicar o que já foi feito;
- Possibilidade de reaproveitamento da mesma estrutura em diversas seções do código-fonte.

As funções são blocos que tomam **argumentos de entrada** (*input*), realizam operações e retornam um **valor de saída** (*output*), conforme ilustrado na Figura 1. As entradas e saídas podem ser variáveis de qualquer tipo.

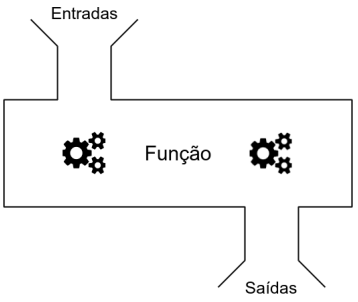

Figura 1: Representação ilustrativa de uma função.

6.1 Definição de Funções

Para que o **Python** reconheça um bloco como uma função, ele precisa estar organizado e indentado. O comando para definir funções é **def**. Abaixo apresentamos a sintaxe utilizada para esse fim:

```
def <nome da função> (<argumentos>):
    comando 1
    comando 2
    ...
        return <resultado>
    ...
    comando n
    return <resultado>
```

A palavra **def** determina que será definida uma função, que será composta de:

- **<nome da função>**: assim como as variáveis criadas pelo programador, as funções precisam ter um nome. As regras definidas são as mesmas das variáveis;
- **<argumentos>**: os argumentos determinam nomes de variáveis internas da função, automaticamente criadas quando a função for chamada, e responsáveis por receber os valores que devem ser fornecidos como entrada para a função. Uma função pode não receber argumento algum, ou um número arbitrário de argumentos. Isto ficará mais claro ao longo dos exemplos e exercícios apresentados a seguir;
- **corpo da função**: a corpo da função determina a sequência de comandos que realiza a tarefa necessária e gera os resultados esperados com a execução da função. Ele será composto de **comandos**, que realizam a tarefa, e cláusulas **return**, que encerram a execução da função e retornam os resultados de saída gerados. Estes resultados serão repassados para o *chamador* da função. Se a função não precisa retornar nada, a cláusula **return** não precisa ser usada, a menos que se deseje explicitamente definir o término da sua execução em algum momento. Quando esta cláusula não é usada, o término da execução ocorre da mesma forma que em um programa, ou seja, quando seu último comando for executado, e nada será retornado;
- A utilização dos parênteses, caracteres “(” e “)”, delimitando os argumentos, e do caractere “:” após o fechamento dos parênteses é obrigatória, assim como a indentação do código que representa o corpo da função.

6.2 Chamada de Funções

Para chamar uma função no **Python**, basta escrever o nome dela dentro do código, com os argumentos entre parêntesis. Isso você já faz há muito tempo, quando usa as funções definidas internamente no **Python**, exemplificadas no início deste módulo (como **print**, **int** e **abs**, por exemplo). Observe o programa abaixo:

```
1  # Definindo de funções (isto é só um comentário, não é obrigatório)
2  def my_function(x):
3      y = 2*x + 3
4      return y
5
6  # Programa principal (outro comentário, não obrigatório)
7  y1 = my_function(1)
8  y2 = my_function(2)
9  y3 = my_function(3)
10 print(f'y1 = {y1}, y2 = {y2}, y3 = {y3}!')
```

Neste programa temos a **definição da função**, entre as linhas 2 e 4, e o que denominamos **programa principal**, entre as linhas 7 e 9. As demais linhas são comentários ou estão em branco e são ignoradas na execução do programa, utilizados aqui apenas para organizar e facilitar o entendimento do programa. Ao iniciar a execução, primeiramente a definição da função será **interpretada**, todas as informações relacionadas à função (nome, argumentos e corpo) serão registradas na memória e deixadas lá para quando a função for chamada. Até agora ele não executou a função, apenas registrou para executar futuramente. A execução do programa de fato começa na atribuição da variável **y1**, linha 7, neste momento há uma chamada à função **my_function**. Devido à esta chamada, a função será de fato executada. Isso é feito com um **desvio de fluxo** de execução do programa, da linha 7 para a linha 2, onde será transferido o valor 1, passado como argumento de entrada para a função, para a variável **x** da função, cujo nome foi determinado na definição da função. Feita a transferência dos argumentos de entrada, inicia-se a execução do corpo da função. Neste caso, a linha 3 será executada, criando uma outra variável dentro da função, chamada de **y** e assumirá o valor $2 * 1 + 3 = 5$, já que $x = 1$. Concluída a atribuição, o próximo comando a ser executado encontra-se na linha 4, cujo resultado é o retorno do valor da variável **y** para quem fez a chamada da função. Neste momento, um novo **desvio de fluxo** é realizado, concluindo a execução da função e retornando para a linha 7, de onde a chamada ocorreu. Neste desvio, o valor de **y**, que é 5, assumirá o lugar da função, gerando a atribuição $y1 = 5$, cujo resultado é a criação da variável **y1**, com o valor 5. Esta ação conclui a execução da linha 7, o que faz com que seja realizada a execução da linha 8. De forma similar, tudo que aconteceu durante a execução da linha 7, acontecerá de novo agora, já que as duas linhas são muito parecidas. Porém, agora trata-se de outra chamada à função, com o argumento de entrada no valor 2, sendo assim, novamente dentro da função, será definido um novo valor para a variável **y**, $y = 2 * 2 + 3 = 7$, que define um novo valor de retorno, a ser atribuído a **y2**, ou seja, $y2 = 7$. Na linha 9, tudo acontece novamente, mas o resultado final agora será $y3 = 9$. O programa se encerrará após a execução da função **print**, linha 10, tendo executado a função em 3 ocasiões distintas, com valores de entrada e saída diferentes, atribuindo os valores 5, 7 e 9, às variáveis do programa principal **y1**, **y2** e **y3**, respectivamente, imprimindo a mensagem `"y1 = 5, y2 = 7, y3 = 9!"` no terminal.

Exemplo 1:

Crie um programa que realize a soma ou a subtração de dois números reais fornecidos pelo usuário, imprimindo em segui o resultado destas operações. Porém, deseja-se que as duas operações sejam realizadas por duas funções, uma para cada operação. As duas funções receberão como entrada os dois números e retornará como resultado a operação aplicada aos dois números.

Solução:

Não há limitação lógica do número de funções definidas em um programa, nem tampouco do número de argumentos passados a uma função. A única restrição é que as regras da linguagem sejam respeitadas. Sendo assim, podemos implementar o seguinte programa, que resolve o problema respeitando as suas restrições:

```
1  # Declaração de funções
2  def soma(num1, num2):
3      resultado = num1 + num2
4      return resultado
5
6  def subtracao(num1, num2):
7      return num1 - num2
8
9  # Programa principal
10 num1 = float(input('Digite o primeiro valor: '))
11 num2 = float(input('Digite o segundo valor: '))
12 s = soma(num1, num2)
13 print(f'A soma é {s}')
14 print(f'A subtração é {subtracao(num1, num2)}'}
```

Aproveitamos este exemplo para explicar vários conceitos relacionados às funções. Primeiro, em linhas gerais, definimos duas funções que têm como objetivo retornar uma soma e uma subtração, respectivamente. Em um programa principal, fizemos a leitura dos números, as chamadas das funções e a impressão dos resultados. Adotamos estratégias diferentes em relação às duas funções. Veja que, para a soma, utilizamos uma variável para armazenar o resultado da operação que, em seguida, é retornado como resultado da execução da função. Para a subtração, nós não utilizamos uma variável, simplesmente retornamos como resultado da função o resultado direto da expressão. Observe que, assim como para os programas, existem variadas formas de se implementar as funções, o importante é que elas cumpram a sua tarefa corretamente. Para isso, você pode usar os recursos disponíveis pela linguagem **Python** para fazer o que precisa ser feito pela função, isso inclui variáveis, operadores matemáticos, decisões, repetições, e até mesmo chamadas de funções.

Outro ponto importante explorado no exemplo é com relação às diferentes possibilidades de chamadas das funções. Observe que, mais uma vez, foi criada uma variável para a obtenção da soma. Na linha 12 a variável **s** será criada com o valor retornado pela chamada da função **soma** para depois ser impressa na tela. A variável não pode ter o mesmo nome da função, senão teremos um conflito que poderá causar um mal funcionamento do programa. Tome sempre muito cuidado com os nomes de variáveis e funções. Por outro lado, para a subtração não foi criada uma variável. Já que o objetivo é apenas imprimir o resultado na tela, a chamada da função **subtracao** foi feito dentro do **print**, para que o resultado retornado seja diretamente usado para compor a f-string utilizada para determinar a mensagem.

6.3 Escopo de variáveis

As variáveis criadas no programa principal e dentro das funções possuem o que chamamos de **escopo** diferentes. O **escopo** de uma variável é onde ela **existe**, ou seja, onde ela pode ser reconhecida e utilizada. Os argumentos de uma função e as variáveis criadas dentro dela estão **exclusivamente** dentro do escopo da função, ou seja, só existem e podem ser referenciadas dentro da função, elas não existem fora dela. Por isso, denominamos este escopo como **local**, pois as variáveis só existem localmente na função onde foram declaradas. As variáveis definidas dentro do programa principal possuem escopo **global**, ou seja, existem em todo o programa, incluindo as funções definidas dentro dele. Obviamente, uma variável só passa a existir após ter algum valor atribuído a ela.

Uma questão muito importante, que temos como uma **boa prática de programação**, é evitar fazer uso de variáveis globais dentro de funções. Embora acessíveis, recomenda-se que valores necessários para a execução de uma função sejam passados como argumentos de entrada. Já para os valores definidos dentro da função, que sejam tratados como valores de retorno da função. Ou seja, evite acessar ou definir valores de variáveis globais dentro de uma função.

Para finalizar, é preciso ressaltar que variáveis de escopo diferentes podem possuir o mesmo nome, sem embaralhar os seus respectivos valores. Variáveis globais **coexistem** com variáveis locais de mesmo nome. Porém, dentro de uma função prevalecem as variáveis **locais**, ou seja, as variáveis globais de mesmo nome não são acessíveis.

Exemplo 2:

Apresentamos uma segunda versão de programa para resolver o Exemplo 1 para observarmos algumas questões relacionadas ao escopo de variáveis.

```
1  # Declaração de funções
2  def soma(num1, num2):
3      resultado = num1 + num2
4      return resultado
5
6  def subtracao(num1, num2):
7      resultado = num1 - num2
8      return resultado
9
10 # Programa principal
11 num1 = float(input('Digite o primeiro valor: '))
12 num2 = float(input('Digite o segundo valor: '))
13 resultado = soma(num1, num2)
14 print(f'A soma é {resultado}')
15 resultado = subtracao(num1, num2)
16 print(f'A subtração é {resultado}'}
```

Agora temos inúmeras variáveis de mesmo nome, mas escopos diferentes. Temos 3 variáveis que se chamam **num1**, no programa principal e nas duas funções. Isso significa que em alguns momentos do fluxo de execução do programa nós teremos duas variáveis **num1**. Isso porque as variáveis locais deixam de existir quando as funções que as declararam terminam sua executar (lembrando que elas só serão criadas quando a função for executada. Como em nosso programa as funções **soma** e **subtracao** não podem estar executando ao mesmo tempo, suas variáveis não poderão coexistir durante nenhum fluxo de execução do programa. Porém, temos a variável global **num1**, que poderá então coexistir com uma das variáveis de mesmo nome das duas funções. O mesmo acontece com a variável **resultado**.

Apenas para ilustrar o que acontece com os escopos das variáveis e como os valores são manipulados, execute o código do programa a seguir para ver o resultado. Antes, tente definir o resultado que você esperaria ver apenas interpretando o código.

```
1  def f1(x):
2      print(x)
3      x = x * 2
4      print(x)
5      return x
6
7  def f2(x):
8      print(x)
9      x = x * 3
10     print(x)
11     return x
12
13 x = 5
14 print(x)
15 y = f1(x)
16 print(x)
17 print(y)
18 y = f2(y)
19 print(x)
20 print(y)
```

6.4 Retornando vários valores

Eventualmente é necessário que uma função retorne mais de um valor como resultado. Para fazer isso em **Python**, basta colocar todos os valores separados por vírgula na cláusula **return** e considerar esta questão na chamada da função. Veja o exemplo a seguir e os comentários que na sequência.

Exemplo 3:

Imagine que você quer fazer um programa que calcula as raízes de uma equação de segundo grau. Sabemos que o formato dessas equações é:

$$ax^2 + bx + c = 0$$

Ainda, sabemos que as raízes são calculadas usando a fórmula de Bháskara:

$$x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

E por último, sabemos que:

$$\Delta = b^2 - 4ac$$

Vamos criar um programa que tem duas funções definidas. Uma para calcular o delta e outra para calcular as raízes. Como são duas raízes, esta função deverá retornar dois valores.

É importante lembrar que precisamos verificar se as raízes reais existem ou não. **Observação:** Para simplificar, utilizaremos o tipo de dado **None** (que em **Python** representa um valor **nulo**) para indicar que não houve nenhum retorno para as raízes.

Vejamos então o programa:

```
1  import math
2
3  def delta(a, b, c):
4      d = (b**2) - (4*a*c)
5      return d
6
7  def raizes(a, b, c):
8      d = delta(a,b,c)
9      if (d < 0):
10         return None, None
11     else:
12         x1 = (-b + (math.sqrt(d)))/(2*a)
13         x2 = (-b - (math.sqrt(d)))/(2*a)
14         return x1, x2
15
16 # Entrada
17 a = int(input('Digite o valor do termo A: '))
18 b = int(input('Digite o valor do termo B: '))
19 c = int(input('Digite o valor do termo C: '))
20 # Processamento
21 r1, r2 = raizes(a,b,c)
22 # Saída
23 if (r1 != None):
24     print(f'Raízes: {r1:.2f} e {r2:.2f}')
25 else:
26     print('Não existem raízes reais')
```

Como a função **raizes** retorna dois valores, um para a primeira raiz e outro para a segunda, observe que a cláusula **return**, desta função, sempre retorna dois valores separados por vírgula. Por consequência, quando esta função é chamada, espera-se também receber dois valores, na atribuição, antes do operador de atribuição nós definimos duas variáveis para receber os valores retornados. Neste caso, a atribuição respeitará a ordem das variáveis e dos valores retornados. Implementando desta maneira, há restrições à chamada da função, recomendamos usar sempre a atribuição para receber os valores retornados diretamente para serem atribuídos a variáveis.

É importante ressaltar também que, no caso de **delta** ser um valor negativo, não existirão raízes reais, então, o resultado será uma mensagem indicando esta situação. Para identificá-la, a função **raizes** retorna dois valores nulos (**None**) e, no programa principal, uma decisão é tomada com base nisso. Se uma das raízes for diferente de **None**, enão imprime-se as raízes, caso contrário, imprime-se a mensagem de erro. Não é necessário testar as duas variáveis, com apenas uma delas já é possível tomar a decisão, neste caso, apenas a variável **r1** foi testada.

6.5 Exercícios Propostos

Exercício 1

Fazer uma função que calcula a distância euclidiana entre dois pontos bidimensionais e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função

implementa. Utilizar x_1, x_2, y_1 e y_2 como argumentos da função.

Fórmulas:

- $P_1 = (x_1, y_1)$
- $P_2 = (x_2, y_2)$
- $D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Exemplos de execução:

Exemplo 1:

```
Digite o valor da coordenada X1: 1
Digite o valor da coordenada Y1: 2
Digite o valor da coordenada X2: 2
Digite o valor da coordenada Y2: 3
A distância entre os pontos é 1.41
```

Exercício 2

Fazer uma função que converte uma entrada de temperatura em Celsius para Fahrenheit e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_F = (T_c.\frac{9}{5}) + 32$

Exemplos de execução:

Exemplo 1:

```
Digite uma temperatura em Celsius: 37
A temperatura em Fahrenheit é 98.6
```

Exercício 3

Fazer uma função que converte uma entrada de temperatura em Fahrenheit para Celsius e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_C = (T_F - 32).\frac{5}{9}$

Exemplos de execução:

Exemplo 1:

```
Digite a temperatura em Fahrenheit: 98.6
A temperatura em Celsius é 37.0
```

Exercício 4

Fazer uma função que calcula a extensão de uma mola vertical com constante elástica k ao pendurar um objeto de massa m. Utilizar k e m como argumentos da função e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmulas:

- $F_e = k.x$
- $F_g = m.g$
- $g = 9,807 m/s^2$

Exemplos de execução:

Exemplo 1:

```
Digite a massa do objeto: 5
Digite a constante da mola: 100
A extensão da mola é 0.49 m
```

Exercício 5

A inclinação de um telhado é dada por um percentual da largura, conforme a Figura 2.

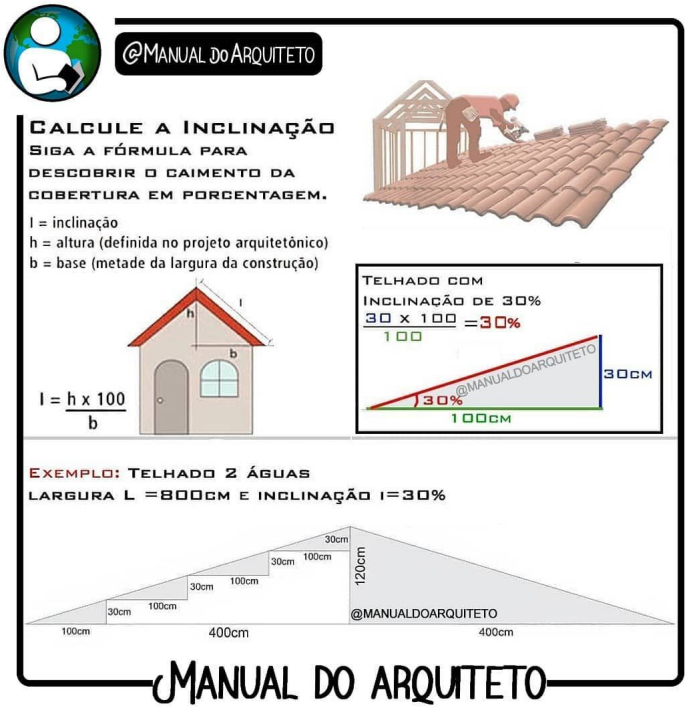

Figura 2: Esquema de inclinação de telhado.

Quando o telhado é de "duas águas", a largura deve ser dividida por dois antes de calcular a altura. Esses dados podem ser usados para calcular o comprimento do telhado, que é a hipotenusa do triângulo retângulo.

Fazer duas funções: uma função que calcula o comprimento c_1 de um telhado de duas águas e uma que calcula o comprimento c_2 de um telhado de uma água, dadas como entradas uma largura l e um percentual p.

Implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada às funções implementas.

Fórmulas (uma água):

- $h = \frac{p.l}{100}$
- $c_1 = \sqrt{l^2 + h^2}$

Fórmulas (duas águas):

- $h = \frac{p.(l/2)}{100}$
- $c_2 = 2.\sqrt{(l/2)^2 + h^2}$

Observação: Os resultados encontrados devem ser iguais (por semelhança de triângulos, é fácil observar essa relação).

Exemplos de execução:

Exemplo 1:

```
Digite a largura do telhado: 10
Digite a inclinação percentual: 30
O comprimento do telhado de uma água é 10.44
O comprimento do telhado de duas águas é 10.44
```

6.6 Solução dos Exercícios Propostos

Exercício 1

Fazer uma função que calcula a distância euclidiana entre dois pontos bidimensionais e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa. Utilizar x_1, x_2, y_1 e y_2 como argumentos da função.

Fórmulas:

- $P_1 = (x_1, y_1)$
- $P_2 = (x_2, y_2)$
- $D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Exemplos de execução:

Exemplo 1:

```
Digite o valor da coordenada X1: 1
Digite o valor da coordenada Y1: 2
Digite o valor da coordenada X2: 2
Digite o valor da coordenada Y2: 3
A distância entre os pontos é 1.41
```

Solução:

```
def euclidian_dist(x1,x2,y1,y2):
    return ((x1-x2)**2 + (y1-y2)**2)**(1/2)

X1 = float(input("Digite o valor da coordenada X1: "))
Y1 = float(input("Digite o valor da coordenada Y1: "))
X2 = float(input("Digite o valor da coordenada X2: "))
Y2 = float(input("Digite o valor da coordenada Y2: "))
dist = euclidian_dist(X1,X2,Y1,Y2)
print(f'A distância entre os pontos é {dist:.2f}')
```

Exercício 2

Fazer uma função que converte uma entrada de temperatura em Celsius para Fahrenheit e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_F = (T_c.\frac{9}{5}) + 32$

Exemplos de execução:

Exemplo 1:

```
Digite uma temperatura em Celsius: 37
A temperatura em Fahrenheit é 98.6
```

Solução:

```
1 def fahrenheit(celsius):
2     return (celsius*9/5) + 32
3
4 T_c = float(input('Digite uma temperatura em Celsius: '))
5 T_f = fahrenheit(T_c)
6 print(f'A temperatura em fahrenheit é {T_f:.1f}')
```

Exercício 3

Fazer uma função que converte uma entrada de temperatura em Fahrenheit para Celsius e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_C = (T_F - 32).\frac{5}{9}$

Exemplos de execução:

Exemplo 1:

```
Digite a temperatura em Fahrenheit: 98.6
A temperatura em Celsius é 37.0
```

Solução:

```
1 def celsius(fahrenheit):
2     return (fahrenheit-32)*5/9
3
4 T_f = float(input('Digite uma temperatura em Celsius: '))
5 T_c = celsius(T_f)
6 print(f'A temperatura em celsius é {T_c:.1f')
```

Exercício 4

Fazer uma função que calcula a extensão de uma mola vertical com constante elástica k ao pendurar um objeto de massa m. Utilizar k e m como argumentos da função e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmulas:

- $F_e = k.x$
- $F_g = m.g$
- $g = 9,807 m/s^2$

Exemplos de execução:

Exemplo 1:

```
Digite a massa do objeto: 5
Digite a constante da mola: 100
A extensão da mola é 0.49 m
```

Solução:

```
def extensao(m,k):
    g = 9.807
    return m*g/k

massa = float(input('Digite a massa do objeto: '))
const = float(input('Digite a constante da mola: '))
x = extensao(massa, const)
print(f'A extensão da mola é {x:.2f} m')
```

Exercício 5

A inclinação de um telhado é dada por um percentual da largura, conforme a Figura 2.

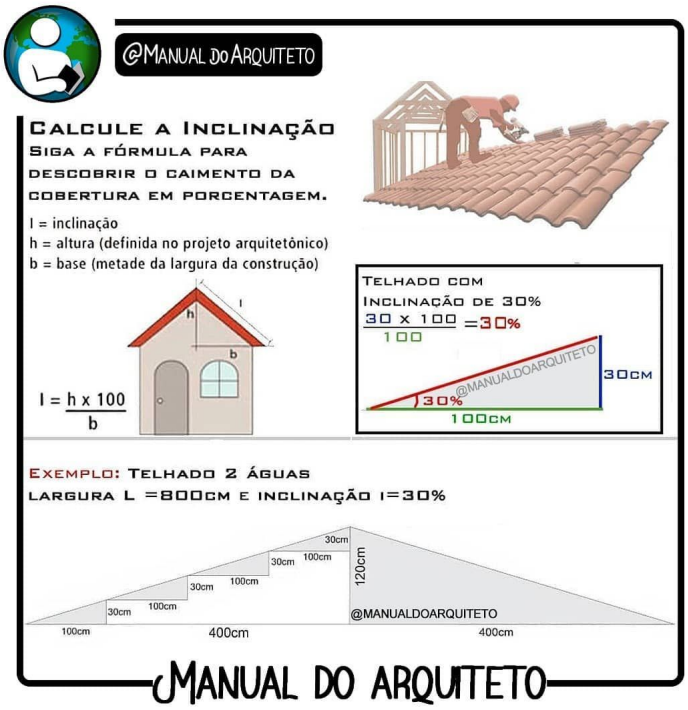

Figura 3: Esquema de inclinação de telhado.

Quando o telhado é de "duas águas", a largura deve ser dividida por dois antes de calcular a altura. Esses dados podem ser usados para calcular o comprimento do telhado, que é a hipotenusa do triângulo retângulo.

Fazer duas funções: uma função que calcula o comprimento c_1 de um telhado de duas águas e uma que calcula o comprimento c_2 de um telhado de uma água, dadas como entradas uma largura l e um percentual p.

Implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada às funções implementas.

Fórmulas (uma água):

- $h = \frac{p.l}{100}$
- $c_1 = \sqrt{l^2 + h^2}$

Fórmulas (duas águas):

- $h = \frac{p.(l/2)}{100}$
- $c_2 = 2.\sqrt{(l/2)^2 + h^2}$

Observação: Os resultados encontrados devem ser iguais (por semelhança de triângulos, é fácil observar essa relação).

Exemplos de execução:

Exemplo 1:

```
Digite a largura do telhado: 10
Digite a inclinação percentual: 30
O comprimento do telhado de uma água é 10.44
O comprimento do telhado de duas águas é 10.44
```

Solução:

```
1  def comprimento_1(l,p):
2      h = l*p/100
3      c = (l**2 + h**2)**(1/2)
4      return c
5
6  def comprimento_2(l,p):
7      h = (l/2)*p/100
8      c = ((l/2)**2 + h**2)**(1/2)
9      return 2*c
10
11 largura = float(input('Digite a largura do telhado: '))
12 percentual = float(input('Digite a inclinação percentual: '))
13 c1 = comprimento_1(largura, percentual)
14 c2 = comprimento_2(largura, percentual)
15 print(f'O comprimento do telhado de uma água é {c1:.2f}')
16 print(f'O comprimento do telhado de duas águas é {c2:.2f}')
```