FIAP ON

PROTOTYPING

QUEM TEM O NOME

NA LISTA?

ANDRÉ DE FREITAS DAVID

04

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 QUEM TEM O NOME NA LISTA?

1.1 Um novo mundo se apresenta!

Sua visão do mundo da programação está prestes a se expandir! Assim como um hobbit que sai do Condado e ganha a Terra Média, ou um hacker que descobre que é o escolhido, nada mais em sua vida será como antes!

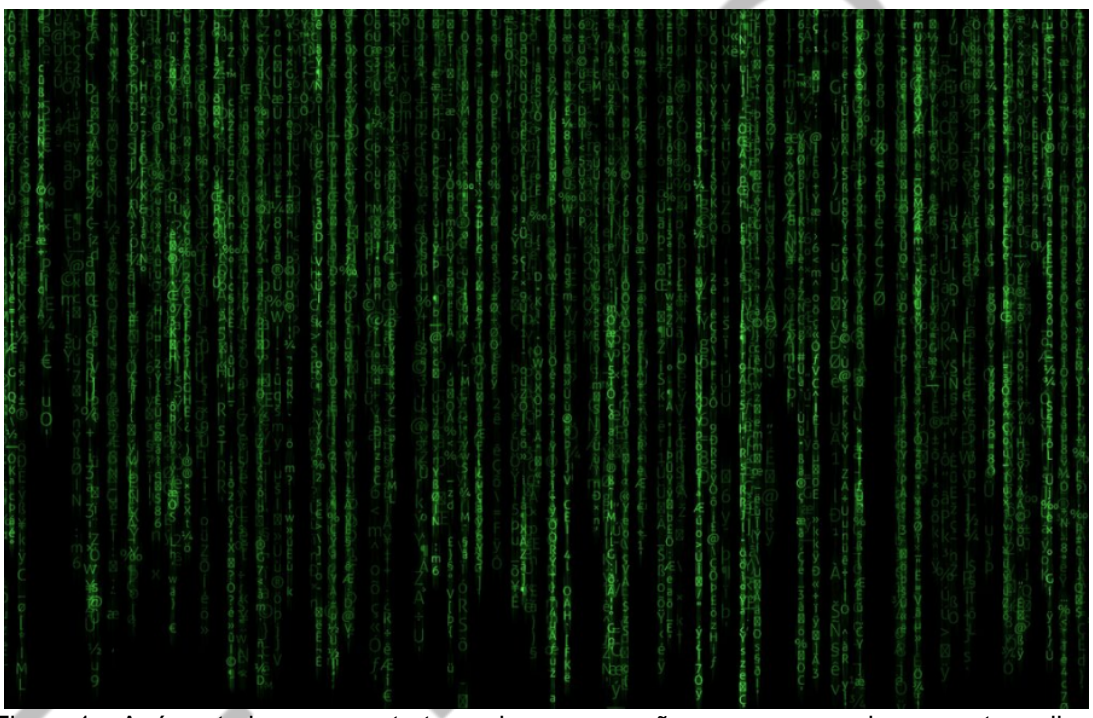

Figura 1 – Após estudar novas estruturas de programação, vemos o mundo com outros olhos
Fonte: Pixabay (2020)

Neste capítulo, você descobrirá o que deve fazer para armazenar dados com maior volume, como preparar o seu programa para reutilizar trechos de código e de que maneira efetivamente fazer essa reutilização.

Tire uma foto e despeça-se dos seus velhos programas, porque eles nunca mais serão os mesmos!

1.2 Agora eu tenho muitos dados...

Do início do nosso estudo até este momento, trabalhamos somente com algoritmos que manipulavam dados em pouca quantidade, tais como: 2 valores para saber qual era o maior, 3 valores para saber o tipo do triângulo ou algo do gênero. Até mesmo quando trabalhamos com os loops para calcular uma média, não nos importava preservar cada valor digitado.

Na sua vida como programador, porém, em diversas ocasiões você terá de lidar com um volume maior de dados que, muitas vezes, têm dimensões incertas. Um exemplo bem simples é pensarmos em um e-commerce qualquer. A lista de produtos digitada pelo usuário ou oriunda de um banco de dados precisa ser armazenada temporariamente em algum local para ser exibida na tela, certo? Imagine se fôssemos tentar realizar essa operação usando diversas variáveis: quantas seriam necessárias? Devo criar variáveis a mais? A menos? E se a quantidade de produtos mudar?

Para solucionar problemas como esse, as linguagens de programação possuem estruturas de dados que permitem armazenar mais de um valor simultaneamente.

Um bom exemplo para ilustrar essa ideia é: uma única variável é como se fosse um copo, no qual só é possível adicionar uma única bebida. As estruturas com que vamos trabalhar são como a bandeja de um garçom, repleta de copos e pratos com conteúdo diferentes.

Figura 2 – A variável é um único copo, mas, às vezes, precisamos de uma bandeja completa
Fonte: Pixabay (2020)

Agora que você entendeu o problema, vamos para a solução? Em linguagem Python, quem resolve esse problema são as listas!

1.2.1 Everybody loves lists!

Para solucionar o problema de armazenamento temporário de múltiplos valores simultaneamente, a linguagem Python possui uma estrutura chamada lista.

Uma lista nada mais é do que uma estrutura de conteúdo mutável (podemos alterar os valores ao longo do código) e tamanho variável (podemos incluir novos ou excluir valores antigos) que pode armazenar diversos valores.

Para criar uma lista, basta utilizarmos colchetes envolvendo os valores que desejamos armazenar.

Da seguinte forma:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]
```

Código-fonte 1 – Criação de lista em linguagem Python
Fonte: Elaborado pelo autor (2020)

Nossa lista agora é uma estrutura que contém todos esses valores que foram fornecidos.

Podemos exibir nossa lista facilmente com um print:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#exibindo a lista com um print
print(jedi)
```

Código-fonte 2 – Exibição de lista em linguagem Python
Fonte: Elaborado pelo autor (2020)

A lista, porém, foi exibida exatamente como ela é, com os colchetes, os valores entre aspas e separados por vírgula. Apesar de essa exibição ser correta, muitas vezes vamos preferir exibir cada um dos valores da lista separadamente.

Podemos fazer isso de inúmeras formas, mas vamos nos focar nas duas principais: exibir um valor específico ou exibir todos os valores.

Para exibir um valor específico da lista, podemos utilizar seu índice! Cada valor que está presente na lista pode ser identificado por um número inteiro (chamado índice) que inicia por 0. Em nosso exemplo, o valor “Yoda” tem índice 0, enquanto o valor “Luke” possui índice 1, o valor “Obi-Wan” tem índice 2 e o valor “Anakin” possui índice 3.

Figura 3 – Cada valor da lista possui um índice
Fonte: Elaborado pelo autor (2020)

Pela existência desses índices, podemos exibir apenas um dos valores da lista ao indicarmos a posição dentro do comando print:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#exibindo um valor específico da lista
print(jedi[2])
```

Código-fonte 3 – Exibição de um valor específico da lista em Python
Fonte: Elaborado pelo autor (2020)

Para exibirmos a lista completa, um valor por vez, podemos utilizar nosso velho conhecido *loop for*. Basta que informemos a lista, e não a função range, com o os valores que a variável pode assumir:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#A variável assumirá cada um dos valores da lista
for nome in jedi:
    #para cada volta do loop, exibir o valor assumido
    print(nome)
```

Código-fonte 4 – Exibição de um valor específico da lista em Python
Fonte: Elaborado pelo autor (2020)

Já deu para entender como as listas são fáceis de usar, não é? Que tal avançarmos um pouco mais em nosso estudo?

1.2.2 Me coloca na lista, cara!

Já sabemos o que são as listas e como elas funcionam, mas você deve estar se perguntando “e aquele papo de que a lista pode ser aumentada?”. Afinal, até agora, as nossas listas estão todas estáticas! Não se aflija, pois temos mais de uma solução para esse terrível problema!

A primeira solução (e a mais simples) é utilizar o método *append.* Esse método permite que um valor seja inserido no fim da nossa lista.

Veja este exemplo:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#incluindo um valor no final da lista
jedi.append("Mace Windu")

#A variável assumirá cada um dos valores da lista
for nome in jedi:
    #para cada volta do loop, exibir o valor assumido
    print(nome)
```

Código-fonte 5 – Inclusão de um valor no fim da lista
Fonte: Elaborado pelo autor (2020)

Se quisermos que o usuário digite o valor que será colocado no fim da lista, basta substituirmos o valor que está entre parênteses por um comando de input.

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#incluindo um valor digitado no final da lista
jedi.append(input("Digite o nome de um jedi"))

#A variável assumirá cada um dos valores da lista
for nome in jedi:
    #para cada volta do loop, exibir o valor assumido
    print(nome)
```

Código-fonte 6 – Inclusão de um valor pelo usuário no fim da lista
Fonte: Elaborado pelo autor (2020)

Pronto! Mais uma operação em que a lista da linguagem Python esbanja facilidade de uso.

Dissemos, porém, que havia mais uma forma de incluir valores na lista, portanto, se você quiser incluir um valor em uma posição específica de sua lista, o método *append* não será útil. Usaremos o método *insert*, o qual exige que o programador informe o índice em que o valor deve ser inserido.

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#incluindo um valor em uma posição específica da lista
jedi.insert(2, "Luminara")

#A variável assumirá cada um dos valores da lista
for nome in jedi:
    #para cada volta do loop, exibir o valor assumido
    print(nome)
```

Código-fonte 7 – Inserção de um valor em um índice específico da lista
Fonte: Elaborado pelo autor (2020)

Tenha em mente que, ao utilizar esse método, os índices dos outros valores são modificados para que a sequência se mantenha. Em nosso exemplo, o valor “Obi-Wan” tinha índice 2 e, após o método *insert,* passou a ter índice 3.

Para permitirmos que o usuário digite o valor, seguimos a mesma lógica aplicada anteriormente:

```
      #criação de uma lista com os nomes dos Jedi
      jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

      #incluindo um valor pelo usuário em uma posição
específica da lista
      jedi.insert(2, input("Digite o nome de um Jedi"))

      #A variável assumirá cada um dos valores da lista
      for nome in jedi:
          #para cada volta do loop, exibir o valor assumido
          print(nome)
```

Código-fonte 8 – Inserção de um valor pelo usuário em um índice específico da lista
Fonte: Elaborado pelo autor (2020)

Você já deve estar imaginando todas as aplicações possíveis para esses comandos, mas se segure porque a lista pode ser ainda mais poderosa!

1.2.3 O que esse nome está fazendo aí?

Se uma lista pode suportar a adição de novos valores, certamente podemos escolher alguns valores para remover dela. A tarefa de remover, assim como a de incluir, pode ser feita de duas formas: indicando a posição ou removendo o último valor inserido.

Para removermos o último valor inserido, basta utilizarmos o método *pop*. Vejamos:

```
      #criação de uma lista com os nomes dos Jedi
      jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

      #Removendo o último valor inserido na lista
      jedi.pop()

      #A variável assumirá cada um dos valores da lista
      for nome in jedi:
          #para cada volta do loop, exibir o valor assumido
          print(nome)
```

Código-fonte 9 – Remoção do último valor inserido na lista
Fonte: Elaborado pelo autor (2020)

Nesse caso, será removido o valor “Anakin”. Para podermos indicar uma posição específica, podemos utilizar o método *pop* indicando um índice entre parênteses:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#Removendo um valor em uma posição específica
jedi.pop(2)

#A variável assumirá cada um dos valores da lista
for nome in jedi:
    #para cada volta do loop, exibir o valor assumido
    print(nome)
```

Código-fonte 10 – Remoção de um valor em um índice específico na lista
Fonte: Elaborado pelo autor (2020)

Com essa alteração, fomos capazes de remover o valor “Obi-Wan” da lista, mas existe um recurso que vai fazer você se apaixonar de vez por essa estrutura tão flexível: remover um valor específico, independentemente da sua posição!

O método *remove* permite que indiquemos qual valor deve ser localizado e removido de qualquer posição da lista! Ele funciona assim:

```
#criação de uma lista com os nomes dos Jedi
jedi = ["Yoda", "Luke", "Obi-Wan", "Anakin"]

#Removendo um valor específico da lista
jedi.remove("Yoda")

#A variável assumirá cada um dos valores da lista
for nome in jedi:
    #para cada volta do loop, exibir o valor assumido
    print(nome)
```

Código-fonte 11 – Remoção de um valor específico na lista
Fonte: Elaborado pelo autor (2020)

E se você acha que a lista não tem mais segredos, está completamente enganado! Vamos explorar alguns métodos extremamente úteis que podem ser utilizados.

1.2.4 Um cinto de utilidades!

Seria um desperdício de tempo tentar decorar todos os métodos que estão disponíveis para a estrutura de lista em linguagem Python por duas razões: nem todos os métodos possuem correspondentes em outras linguagens, e todos os métodos estão disponíveis na documentação.

Apesar disso, é legal você ficar atento a alguns métodos tidos como “coringa” e que podem vir a ser úteis em inúmeras situações. São eles:

- count() – retorna a quantidade de vezes que um elemento aparece na lista.
- sort() – organiza a lista em ordem crescente/alfabética.
- reverse() – inverte a ordem dos elementos de uma lista.

Vamos colocar os três métodos em um único script para compreender seu uso?

```
      #valores fora de ordem
      valores = [1, 7, 7, 19, 3, 2, 11, 15, 6, 1, 5]

      #exibição da lista
      print("A lista foi criada assim: {}".format(valores))

      #contagem de elementos
      contagem = valores.count(7)
      print("Nessa lista o número 7 aparece {}
vezes".format(contagem))

      #invertendo a lista
      valores.reverse()
      print("A lista agora está invertida:
{}".format(valores))

      #ordenando a lista
      valores.sort()
      print("A lista agora está ordenada: {}".format(valores))
```

Código-fonte 12 – Métodos úteis da classe list em Python
Fonte: Elaborado pelo autor (2020)

Além desses métodos da lista, algumas funções do Python podem ajudá-lo, como as funções *len* (retorna o tamanho de um objeto) e s*um* (realiza a soma dos elementos presentes em um objeto).

Veja as duas em funcionamento:

```
#valores de uma lista
valores = [2, 3, 5, 10]

#tamanho da lista
tamanho = len(valores)
print("A lista tem {} elementos".format(tamanho))

#soma dos elementos
soma = sum(valores)
print("A soma dos elementos é {}".format(soma))
```

Código-fonte 13 – Funções úteis usadas com a lista em Python
Fonte: Elaborado pelo autor (2020)

A estrutura de lista é uma ferramenta poderosíssima, e com grandes poderes vêm... ótimas ideias de algoritmos para criar! Não deixe esse conhecimento parado e coloque a mão na massa.

2 COMO ESSE NEGÓCIO FUNCIONA?

Você trabalha em uma empresa de transportes como desenvolvedor e consegue criar um algoritmo para calcular a velocidade média que os caminhões da empresa desenvolvem para cumprir um determinado trajeto. Entretanto, há uma pegadinha: você precisa usar esse mesmo algoritmo em diferentes partes do seu sistema!

A solução mais tradicional seria utilizar o famoso “CTRL+C, CTRL+V” e copiar seu algoritmo em todos os lugares em que for necessário, correto? Mas hoje você vai aprender por que essa é uma péssima solução.

Os dois principais problemas são: o seu programa vai ter muito mais linhas do que ele precisaria ter, e a manutenção do código será fortemente dificultada, pois cada alteração que seu algoritmo tiver precisará ser replicada em todas as cópias.

Para resolver esse problema de forma inteligente, existem as funções, que podem ser consideradas “microprogramas” que têm uma única função específica e podem ser convocadas quando necessárias.

Vamos entender melhor?

2.1 Minha primeira função!

Para compreender como criar uma função e de que maneira ela funciona, vamos primeiramente escrever aquele algoritmo de velocidade média que tínhamos pensado:

```
      #solicitar distância e tempo
      distancia    =    float(input("Digite    a    distância
percorrida"))
      tempo = float(input("Digite o tempo da viagem"))
      #calcular a velocidade média
      velocidade_media = distancia/tempo
      #exibir o resultado
      print("A       velocidade       média       é       {}
km/h".format(velocidade_media))
```

Código-fonte 14 – Algoritmo de velocidade média sem função
Fonte: Elaborado pelo autor (2020)

O que faremos agora é indicar ao Python que queremos criar uma nova função. Isso é feito por meio da palavra reservada “def”, que exige que indiquemos um nome para nossa função, seguido de parênteses:

```
      def calcularVelocidadeMedia():
```

Código-fonte 15 – Criando uma nova função em Python
Fonte: Elaborado pelo autor (2020)

Uma vez que nossa função tenha sido criada, vamos pegar todo o código do nosso algoritmo e colocá-lo dentro dela:

```
      #Função que calcula a velocidade média
      def calcularVelocidadeMedia():
          #solicitar distância e tempo
          distancia = float(input("Digite a distância
percorrida"))
          tempo = float(input("Digite o tempo da viagem"))
          #calcular a velocidade média
          velocidade_media = distancia/tempo
          #exibir o resultado
          print("A velocidade média é {}
km/h".format(velocidade_media))
```

Código-fonte 16 – Colocando o algoritmo dentro de uma função em Python
Fonte: Elaborado pelo autor (2020)

Se você tentar executar esse script, notará que nada acontece! Isso ocorre porque uma função só entra em funcionamento quando é convocada. Por isso, fora da função e dentro do programa principal, faremos uma chamada para ela.

```
      #Função que calcula a velocidade média
      def calcularVelocidadeMedia():
          #solicitar distância e tempo
          distancia = float(input("Digite a distância
percorrida"))
          tempo = float(input("Digite o tempo da viagem"))
          #calcular a velocidade média
          velocidade_media = distancia/tempo
          #exibir o resultado
          print("A velocidade média é {}
km/h".format(velocidade_media))

      #aqui começa o programa principal
      calcularVelocidadeMedia()
```

Código-fonte 17 – Chamada para a função criada em Python
Fonte: Elaborado pelo autor (2020)

Pronto! Não só criamos nossa função como aprendemos a fazer uma chamada. Agora, toda vez que quisermos calcular a velocidade média, bastará escrevermos “calcularVelocidadeMedia()”.

Mas não vá embora ainda! Temos alguns pontos para melhorar.

2.2 Nada de input na função!

E se quisermos integrar nossa função em um outro sistema que recupera a distância e o tempo de um banco de dados? E se preferirmos recuperar tais valores diretamente do computador de bordo do caminhão?

A forma como criamos a nossa função faz com que ela solicite dois valores por meio do comando input e, apesar de não estar incorreta, essa prática acarreta algumas dificuldades para o uso futuro, pois estamos limitando sua utilização a um cenário em que o usuário seja obrigado a digitar pelo teclado os dois valores.

Por essa razão, não é uma boa prática fazer com que a função solicite valores durante seu funcionamento. A saída é solicitarmos tais valores fora da função e passarmos para ela por meio de argumentos ou parâmetros.

Veja que legal:

```
    #Função que calcula a velocidade média
    def calcularVelocidadeMedia(distancia, tempo):
        #calcular a velocidade média
        velocidade_media = distancia/tempo
        #exibir o resultado
        print("A velocidade média é {} km/h".format(velocidade_media))
```

Código-fonte 18 – Modificando a função para aceitar argumentos
Fonte: Elaborado pelo autor (2020)

Perceba que nossa função parte do pressuposto de que os valores para distância e tempo serão informados no momento da chamada da função e, por isso, não usamos os inputs dentro dela.

A chamada da função no programa principal também deve ser alterada:

```
      #Função que calcula a velocidade média
      def calcularVelocidadeMedia(distancia, tempo):
          #calcular a velocidade média
          velocidade_media = distancia/tempo
          #exibir o resultado
          print("A      velocidade      média      é      {}
km/h".format(velocidade_media))

      #aqui começa o programa principal
      distancia = float(input("Informe a distância"))
      tempo = float(input("Informe o tempo"))
      #chamando a função com valores definidos pelo usuário
      calcularVelocidadeMedia(dist, temp)

      #chamando a função com valores definidos pelo programador
      calcularVelocidadeMedia(15,2)
```

Código-fonte 19 – Modificando a chamada da função para passagem de argumentos
Fonte: Elaborado pelo autor (2020)

Viu como nossos programas estão ficando cada vez mais refinados? Falta aprendermos mais um detalhe para termos funções realmente poderosas!

2.3 Retornar ou não retornar: eis a questão

Existem cenários nos quais desejamos que uma função execute uma tarefa que não gere dados ao final, como fechar uma conexão com o banco de dados ou remover um arquivo do HD. Em outras ocasiões, porém, o objetivo da nossa função é retornar alguma informação valiosa, como no caso da nossa função que calcula a velocidade média.

Se prestarmos bastante atenção à nossa função, ela “amarra” seu funcionamento a um cenário no qual vamos sempre printar o resultado na tela. Mas e se quiséssemos apenas gravar esse valor em um arquivo ou se quiséssemos utilizá-lo em outro cálculo?

Quando nos vemos diante de uma função que tem por objetivo retornar um dado, devemos abrir mão do uso de “prints” dentro da função e utilizar o comando return.

O comando return faz com que uma função seja encerrada e um determinado valor seja devolvido para o local onde ocorreu a chamada da função. Ficou confuso? Então vamos devagar.

Nossa primeira tarefa é fazer o print desaparecer e substituí-lo por um return:

```
        #Função que calcula a velocidade média
        def calcularVelocidadeMedia(distancia, tempo):
            #calcular a velocidade média
            velocidade_media = distancia/tempo
            #retornar o resultado
            return velocidade_media
```

Código-fonte 20 – Modificando a função para retornar um valor
Fonte: Elaborado pelo autor (2020)

Agora precisamos adequar a chamada da função. Afinal de contas, será retornado um valor com o qual precisaremos lidar. Portanto:

```
        #Função que calcula a velocidade média
        def calcularVelocidadeMedia(distancia, tempo):
            #calcular a velocidade média
            velocidade_media = distancia/tempo
            #exibir o resultado
            return velocidade_media

        #aqui começa o programa principal
        dist = float(input("Informe a distância"))
        temp = float(input("Informe o tempo"))
        #chamando a função com valores definidos pelo usuário
        print("A           velocidade           média           é
{}".format(calcularVelocidadeMedia(dist, temp)))
```

Código-fonte 21 – Modificando a chamada da função que retorna um valor
Fonte: Elaborado pelo autor (2020)

O uso de funções é um grande diferencial para qualquer desenvolvedor, e uma ótima forma de exercitar é refazendo todos os algoritmos que você concluiu até hoje adequando a esse formato.

2.4 A fronteira final!

Depois de aprendermos a lidar com funções, existe mais um espaço importantíssimo para concluir nossa caminhada rumo à modularização completa dos nossos programas: o uso de módulos.

Módulos são scripts Python que contêm funções e estruturas que podem ser incorporadas em outros scripts. Seremos capazes de utilizar nossas funções não apenas dentro de um mesmo script, mas entre diversos scripts que compõem um único sistema. Vamos lá?

2.4.1 Criando o módulo

Para começar nossa experiência, vamos partir do seguinte cenário: um script chamado *calc.py* (que conterá apenas as funções necessárias para as quatro operações básicas) e um script chamado *teste.py* (que utilizaremos para testar o uso de módulos), sendo os dois criados dentro de um mesmo projeto.

No script *calc.py*, vamos escrever nossas funções:

```
#função de soma
def somar(a, b):
    return float(a) + float(b)

#função de subtração
def subtrair(a, b):
    return float(a) - float(b)

#função de divisão
def dividir(a, b):
    if b==0:
        return 0
    return float(a) / float(b)

#função de multiplicar
def multiplicar(a, b):
          return float(a) * float(b)
```

Código-fonte 22 – Módulo calc.py
Fonte: Elaborado pelo autor (2020)

Pronto! Sem aprendermos nenhum comando novo, desenvolvemos nosso primeiro módulo! Resta-nos aprender como utilizá-lo.

2.4.2 Utilizando o módulo

Para utilizarmos um módulo que foi criado, usaremos o comando chamado *import*. Com ele seremos capazes de indicar o arquivo ao qual queremos incorporar funções e estruturas.

Em nosso script chamado teste.py, teremos a seguinte linha:

```
#importação do módulo calc.py
import calc
```

Código-fonte 23 – Importação do módulo calc.py
Fonte: Elaborado pelo autor (2020)

Uma vez que fizermos a importação, o uso do módulo deverá ser realizado obedecendo à estrutura: *nomedomodulo.nomedafuncao*. Em nosso caso, podemos exemplificar assim:

```
#importação do módulo calc.py
import calc

#solicitando valores ao usuário
valor1 = input("Digite um valor: ")
valor2 = input("Digite outro valor: ")

#armazenando a soma
soma = calc.somar(valor1, valor2)
#exibindo a soma
print("A soma é {}".format(soma))
```

Código-fonte 24 – Uso da função soma do módulo calc.py
Fonte: Elaborado pelo autor (2020)

Se formos utilizar muitas funções de um determinado módulo, podemos fazer a importação de cada uma delas utilizando a seguinte estrutura:

```
#importação de funções específica do módulo calc.py
from calc import somar, subtrair
```

Código-fonte 25 – Importação de funções específicas do módulo calc.py
Fonte: Elaborado pelo autor (2020)

Dessa maneira, a chamada da função poderá ser realizada sem a necessidade de escrever o nome do módulo e o sinal de ponto. Veja:

```
#importação de funções específica do módulo calc.py
from calc import somar, subtrair

#solicitando valores ao usuário
valor1 = input("Digite um valor: ")
valor2 = input("Digite outro valor: ")

#armazenando a soma
soma = somar(valor1, valor2)
#exibindo a soma
print("A soma é {}".format(soma))
```

Código-fonte 26 – Chamada de uma função especificada na importação do módulo calc.py
Fonte: Elaborado pelo autor (2020)

Ainda é possível importar todas as funções de uma só vez, mas esse recurso deve ser usado com cautela, para que não tenhamos um programa desnecessariamente lento:

```
#importação de funções específica do módulo calc.py
from calc import *
```

Código-fonte 27 – Importação de todas as funções do módulo calc.py
Fonte: Elaborado pelo autor (2020)

Já passamos pelos conceitos-chave de algoritmos, desde as variáveis até a modularização, passando por desvios condicionais, laços de repetição e estruturas de armazenamento.

Agora chegou a sua hora de brilhar, avaliando todos os problemas e encontrando as melhores soluções algorítmicas para eles.

REFERÊNCIAS

PIVA JÚNIOR, Dilermando et al. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, Sandra; RISSETTI, Gerson. **Lógica de programação e estrutura de dados**. São Paulo: Pearson Prentice Hall, 2009.

RAMALHO, Luciano. **Python fluente**: programação clara, concisa e eficaz. São Paulo: Novatec, 2015.