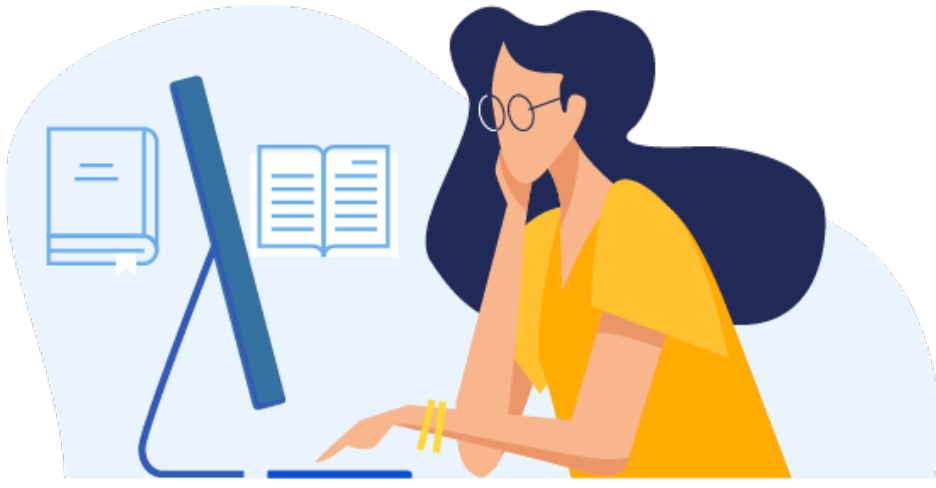
Bibliotecas e módulos em Python

Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

Implementamos algoritmos, nas diversas linguagens de programação, para automatizar soluções e acrescentar recursos digitais, como interfaces gráficas e processamento em larga escala. Uma solução pode começar com algumas linhas de códigos, mas, em pouco tempo, se tornar centenas, milhares e até milhões delas. Nesse cenário, trabalhar com um único fluxo de código se torna inviável, dando origem a técnicas de implementação para organizar a solução.



Fonte: Shutterstock.

Organização em módulos

Uma opção para organizar o código é implementar funções, contexto em que cada bloco passa a ser responsável por uma determinada funcionalidade. Outra forma é utilizar a orientação a objetos e criar classes que encapsulam características e comportamentos de um determinado objeto. Conseguimos utilizar ambas as técnicas para melhorar nosso código, mas, ainda assim, estamos falando de toda a solução agrupada em um arquivo Python (.py).

Atenção

Considerando a necessidade de implementar uma solução, o mundo ideal é:

Modular uma solução ou seja fazer a separação em funções ou classes e ainda realizar a separação em vários arquivos .py (PSF, 2020b).

Segundo a documentação oficial do Python, é preferível implementar, em um arquivo separado, uma funcionalidade que você possa reutilizar, criando assim um módulo.

"Um módulo é um arquivo contendo definições e instruções Python. O nome do arquivo é o nome do módulo acrescido do sufixo .py." (PSF, 2020b, [s.p.]).

Veja a seguir essa ideia, com base na qual uma solução original implementada em um único arquivo .py é transformada em três módulos que, inclusive, podem ser reaproveitados, conforme aprenderemos.

Contínuo *versus* módulo

solucao.py

```
def funcao1():
   pass

class Classe1:
```

Fonte: elaborada pela autora.

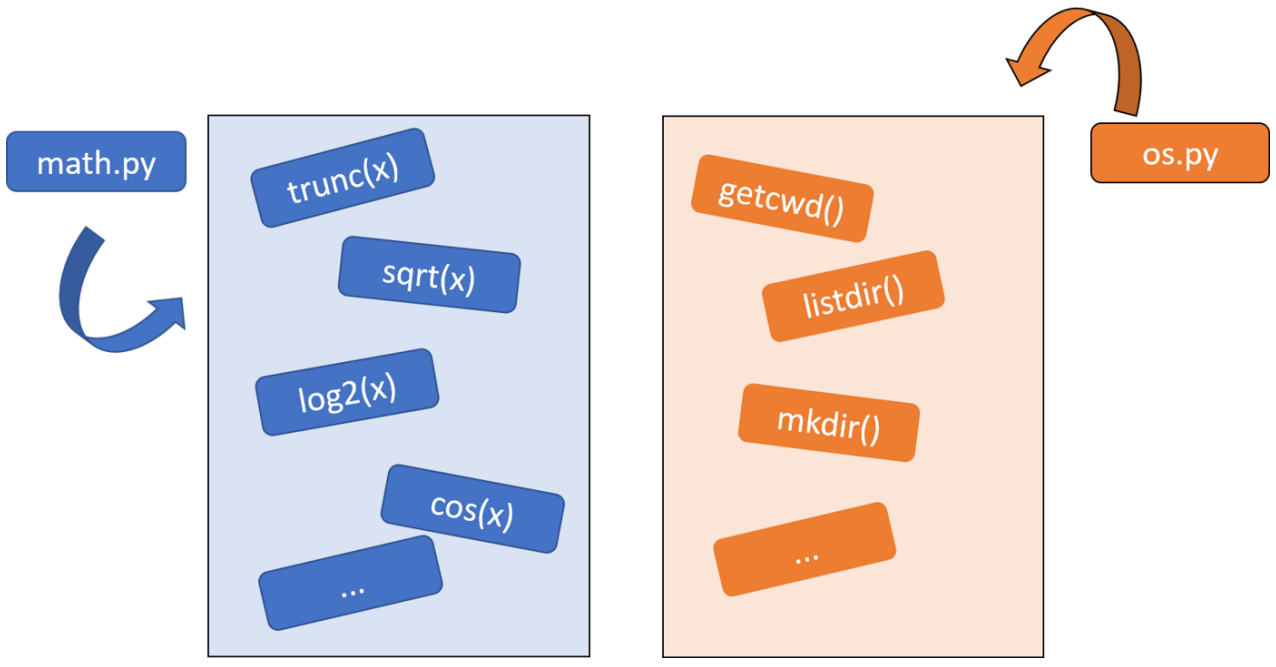
Bibliotecas

Falamos em módulo, mas em Python se ouve muito o termo biblioteca. O que será que eles têm em comum?

Na verdade, um módulo pode ser uma biblioteca de códigos! Veja a seguir que temos o módulo *math*, que possui diversas funções matemáticas, e o módulo *os*, que possui funções de sistema operacional, como capturar o caminho (getcwd), listar um diretório (listdir), criar uma nova pasta (mkdir), dentre inúmeras outras.

Esses módulos são bibliotecas de funções pertinentes a um determinado assunto (matemática e sistema operacional), as quais possibilitam a reutilização de código de uma forma elegante e eficiente.

Módulo como biblioteca



Fonte: elaborada pela autora.

Como utilizar um módulo

Para utilizar um módulo, é preciso importá-lo para o arquivo. Essa importação pode ser feita de maneiras distintas:

» `1. import moduloXX`

`1.2 import moduloXX as apelido`

Desta forma todas as funcionalidades de um módulo são carregadas na memória.

> A diferença entre elas é que, na primeira, usamos o nome do módulo e, na segunda, atribuímos um apelido (as = alias) ao módulo.

» `2. from moduloXX import itemA, itemB`

Nesta forma de importação, somente funcionalidades específicas de um módulo são carregadas na memória. A forma de importação também determina a sintaxe para utilizar a funcionalidade. Com podemos observar nos exemplo a seguir:

Exemplo 1: Importando todas as funcionalidades da biblioteca

```
1  import math
2
3  math.sqrt(25)
4  math.log2(1024)
5  math.cos(45)
```

Exemplo 2: Importando todas as funcionalidades da biblioteca com alias

```
1  import math as m
2
3  math.sqrt(25)
4  math.log2(1024)
5  math.cos(45)
```

Exemplo 3: Importando funcionalidades específicas de uma biblioteca

```
1  from math import sqrt, log2, cos
2
3  sqrt(25)
4  log2(1024)
5  cos(45)
```

Classificação dos módulos (bibliotecas)

Podemos classificar os módulos (bibliotecas) em três categorias:

» **Módulos *built-in***
São embutidos no interpretador. Ao instalar o interpretador Python, também é feita a instalação de uma biblioteca de módulos, que pode variar um de sistema operacional para outro.

» **Módulos de terceiros**
São criados por terceiros e disponibilizados via PyPI.
PyPI é a abreviação para *Python Package Index*, que é um repositório para programas Python. Programadores autônomos e empresas podem criar uma solução em Python e disponibilizá-la em forma de biblioteca no repositório PyPI. Dessa forma, todos usufruem e contribuem para o crescimento da linguagem.

» **Módulos próprios**

São criados pelo desenvolvedor. Cada módulo pode importar outros módulos, tanto os pertencentes ao mesmo projeto, quanto os built-in ou de terceiros.

Pesquise mais

No Capítulo 7 (*Namespaces*) do livro a seguir indicado, você encontrará uma explicação sobre o uso de módulos como *namespaces*. Nele, mais precisamente entre as páginas 238 e 242, o autor discorre sobre o caminho de busca do módulo.

LJUBOMIR, P. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

Fonte: Shutterstock.