---

title: Reutilização de código e modularidade em Python

layout: lesson

slug: reutilizacao-de-codigo-e-modularidade-em-python

date: 2012-07-17

translation\_date: 2021-05-21

authors:

- William J. Turkel

- Adam Crymble

reviewers:

- Jim Clifford

editors:

- Miriam Posner

translator:

- Felipe Lamarca

translation-editor:

-

translation-reviewer:

- Suemi Higuchi

difficulty: 2

review-ticket:

activity:

topics: [python]

abstract: "Programas de computador podem se tornar longos, pesados e confusos sem mecanismos especiais para gerenciar a complexidade. Esta lição mostrará como reutilizar partes do seu código escrevendo funções e dividindo seus programas em módulos, a fim de mantê-los concisos e fáceis de serem depurados."

redirect\_from:

original: code-reuse-and-modularity

avatar\_alt: Three caricature heads

---

{% include toc.html %}

## Objetivos da lição

Programas de computador podem se tornar longos, pesados e confusos sem mecanismos especiais para gerenciar a complexidade. Esta lição mostrará como reutilizar partes do seu código escrevendo funções e dividindo seus programas em módulos, a fim de mantê-los concisos e fáceis de serem depurados. A possibilidade de remover um único módulo disfuncional pode economizar tempo e esforço.

### Funções

Frequentemente você descobrirá que deseja reutilizar um determinado conjunto de comandos, geralmente porque há uma tarefa que precisa ser realizada repetidas vezes. Programas são majoritariamente compostos de rotinas que são potentes e de usos gerais o suficiente para serem reutilizadas. Essas rotinas são conhecidas como funções, e o Python possui mecanismos que permitem a definição de novas funções. Vamos trabalhar com um exemplo muito simples de função. Suponha que você deseja criar uma função de uso geral para cumprimentar pessoas. Copie a definição de função a seguir no Komodo Edit e salve o documento como `cumprimento.py`.

```

# cumprimento.py

def cumprimentar\_entidade (x):

print("Olá " + x)

cumprimentar\_entidade("mundo")

cumprimentar\_entidade("Programming Historian")

```

A linha que começa com `def` é a declaração da função. Definiremos ("def") uma função, que nesse caso nomeamos "cumprimentar\_entidade". O `x` é o parâmetro da função. Seu funcionamento deve ficar claro num instante. A segunda linha contém o código da função. O número de linhas do código varia conforme a nossa necessidade, mas nesse caso é apenas uma única linha.

Note que a \*indentação\* é importante em Python. O espaço vazio antes do comando `print` informa ao interpretador que esse comando é parte da função que está sendo definida. Você aprenderá mais sobre isso à medida que prosseguirmos; por ora, certifique-se de manter a indentação da maneira como foi mostrada. Ao executar o programa, deverá ver algo assim:

```

Olá mundo

Olá Programming Historian

```

Este exemplo contém uma função: \*cumprimentar\_entidade\*. Essa função é então "chamada" (ou "invocada") duas vezes. Chamar ou invocar uma função significa apenas dizer ao programa para executar o código daquela função. É como dar ao cachorro sua guloseima com sabor de frango (\\*au\\* \\*au\\*). Nesse caso, para cada vez que chamamos a função informamos um parâmetro diferente. Tente editar `cumprimento.py` de forma que ele chame a função \*cumprimentar\_entidade\* uma terceira vez utilizando seu próprio nome como parâmetro. Execute o código novamente. Agora deve entender melhor o que `x` faz na declaração da função.

Antes de prosseguirmos para o próximo passo, edite `cumprimento.py` para deletar os chamados da função, deixando apenas sua declaração. Agora aprenderá como chamar uma função através de outro programa. Quando terminar, seu ficheiro `cumprimento.py` deverá estar assim:

```

# cumprimento.py

def cumprimentar\_entidade (x):

print("Olá " + x)

```

## Modularidade

Quando os programas são pequenos como o do exemplo acima, tipicamente ficam hospedados num único ficheiro. Quando quiser executar um dos seus programas, pode simplesmente enviar o ficheiro ao interpretador. À medida que os programas ficam maiores, faz sentido dividi-los em ficheiros separados conhecidos como módulos. Essa modularidade torna mais fácil o trabalho em seções de teus programas maiores. Aperfeiçoando cada seção do programa antes de unir todas as seções, torna-se mais fácil não apenas reutilizar módulos individuais em outros programas, como também torna mais fácil corrigir eventuais problemas ao ser capaz de identificar a origem do erro. Quando você divide um programa em módulos, também consegue ocultar os detalhes de como algo é feito dentro do módulo que o faz. Outros módulos não precisam saber como algo é feito se eles não são os responsáveis por fazê-lo. Esse princípio "need-to-know" (necessidade de saber) é chamado de "encapsulamento".

Suponha que você esteja construindo um carro. É possível começar a adicionar peças à vontade, mas faria mais sentido começar construindo e testando um módulo - talvez o motor - antes de passar para os outros. O motor, por sua vez, pode ser imaginado como consistindo em vários outros módulos menores, como o carburador e o sistema de ignição, e esses são compostos de módulos ainda menores e mais básicos. O mesmo se aplica à codificação. Você tenta quebrar um problema em pedaços menores e resolvê-los primeiro.

Um módulo já foi criado quando o programa `cumprimento.py` foi escrito. Agora você escreverá um segundo programa, `utilizando-cumprimento.py`, que importará o código do seu módulo e fará uso dele. O Python possui um comando especial, `import`, que permite que um programa tenha acesso ao conteúdo de outro ficheiro de programa. É isso que será utilizado.

Copie este código no Komodo Edit e salve-o como `utilizando-cumprimento.py`. Este ficheiro é o seu programa; `cumprimento.py` é o seu módulo.

```

# utilizando-cumprimento.py

import cumprimento

cumprimento.cumprimentar\_entidade("mundo")

cumprimento.cumprimentar\_entidade("programming historian")

```

Fizemos algumas coisas aqui. Primeiro, dissemos ao Python para ele importar (carregar), utilizando o comando `import`, o módulo `cumprimento.py` que criamos anteriormente.

Você perceberá que, embora antes conseguíssemos executar a função chamando-a pelo nome: \*cumprimentar\_entidade("mundo")\*, agora precisamos incluir o nome do módulo seguido por um ponto (.) na frente do nome da função. No português claro, isso significa: execute a função \*cumprimentar\_entidade\*, que deve ser encontrada no módulo `cumprimento.py`.

É possível executar o programa `utilizando-cumprimento.py` com o comando "Run Python" que você criou no Komodo Edit. Note que não é necessário executar o módulo… somente o programa que chama por ele. Se tudo correu bem, deverá ver o seguinte no painel de saída do Komodo Edit:

```

Olá mundo

Olá programming historian

```

Tenha certeza de que entende a diferença entre carregar um ficheiro de texto (ex.: olamundo.txt) e importar um ficheiro de programa (ex.: cumprimento.py) antes de prosseguir.

## Leituras recomendadas

- [Python Basics][] (em inglês)

[Python Basics]: http://www.astro.ufl.edu/~warner/prog/python.html

### Sobre os autores

William J. Turkel é Professor de História na University of Western Ontario.

Adam Crymble, University College London.

### Citação sugerida

William J. Turkel e Adam Crymble, “Reutilização de código e modularidad em Python”, traduzido por Felipe Marques Esteves Lamarca, The Programming Historian em português (2021), https://.