Módulos e pacotes

Em praticamente todos os paradigmas de programação atualmente - como a programação orientada a objeto, a programação funcional ou mesmo a programação procedural - um recurso muito importante é a possibilidade de modularizar o código e poder reaproveitá-lo em diversos projetos diferentes.

Os módulos podem conter diferentes componentes reutilizáveis, como classes ou funções prontas. Vamos começar com um ponto que já conhecemos: como utilizar um módulo já existente.

1. Biblioteca Padrão do Python

O Python possui uma política "pilhas inclusas", indicando que para começar a utilizá-lo para fazer diversas tarefas diferentes não é necessário qualquer tipo de configuração ou recursos adicionais, pois ele já vem com "tudo" pronto.

Parte desse "tudo" é uma extensa biblioteca com diversos módulos, cada qual com uma temática específica: estruturas de dados diversas, diferentes tipos de funções matemáticas e estatísticas, persistência de dados (incluindo um banco de dados simples, mas completo através do módulo sqlite3), processamento paralelo, conexões de rede e internet, manipulação de diferentes formatos de arquivo, interface gráfica, funcionalidades de criptografia... E a lista vai embora.

A documentação oficial do Python traz uma lista completa de todos os módulos, além de uma lista com descrições e pequenos exemplos de todas as classes e funções disponíveis em cada módulo: https://docs.python.org/pt-br/3/library/index.html

2. Importando um módulo

Para utilizar um módulo existente, podemos importá-lo através da palavra import. Sempre utilizaremos o nome do módulo para informar ao Python onde ele deve buscar aquela função ou classe. O exemplo abaixo utiliza funções trigonométricas disponíveis no módulo math (https://docs.python.org/pt-br/3/library/math.html).

import math

graus = float(input('Digite um ângulo: '))

# Função "radians" do módulo math - converte graus para radianos

radianos = math.radians(graus)

# funções "sin" e "cos" do módulo math - retornam seno e cosseno de um ângulo (dado em radianos)

seno = math.sin(radianos)

cosseno = math.cos(radianos)

print(seno, cosseno)

# função "sqrt" do módulo math - raiz quadrada

prova = math.sqrt(2)/2

print(prova)

Podemos dar um "apelido" para a biblioteca caso seja útil para melhorar a legibilidade de nosso código através da palavra as.

Isso é muito comum, por exemplo, na comunidade de ciência de dados: a maioria dos usuários das bibliotecas de ciência de dados (inclusive os criadores dos módulos) costumam renomear pandas para pd, matplotlib.pyplot para plt, numpy para np etc.

Mas use com cuidado para não gerar confusão em quem for ler o código.

import math as matematica

graus = float(input('Digite um ângulo: '))

# Função "radians" do módulo math - converte graus para radianos

radianos = matematica.radians(graus)

# funções "sin" e "cos" do módulo math - retornam seno e cosseno de um ângulo (dado em radianos)

seno = matematica.sin(radianos)

cosseno = matematica.cos(radianos)

print(seno, cosseno)

# função "sqrt" do módulo math - raiz quadrada

prova = matematica.sqrt(2)/2

print(prova)

3. Namespace e escopo

Você deve ter notado que sempre utilizamos o nome do módulo antes do nome da função. Isso ocorre porque o Python, como diversas outras linguagens, trabalha com a ideia de namespace.

De maneira simplificada, cada "nome" (de variável, de função, de classe, de objeto...) geralmente possui um escopo limitado. Quando criamos uma função ou classe dentro de um arquivo, o nome dessa função só existe dentro desse arquivo. Ao importarmos esse arquivo, temos mais de um namespace dentro de nosso projeto: o namespace global e o namespace do arquivo. Se digitarmos apenas o nome de uma função, esse nome será buscado no namespace global. Para procurar em outro namespace, precisamos indicar seu nome primeiro.

Podemos, porém, importar funções e classes de namespaces mais restritos diretamente para o namespace global. Uma forma é utilizando a forma from [módulo] import [funções/classes]:

from math import sqrt

x = float(input('Digite um número: '))

y = sqrt(x) # note que não precisamos escrever math.sqrt dessa vez

print(y)

Também podemos dar um novo nome para a função dentro do namespace global através da palavra as:

from math import sqrt as raiz\_quadrada

x = float(input('Digite um número: '))

y = raiz\_quadrada(x) # essa função é a math.sqrt

print(y)

É possível importar todas as funções de um módulo direto para o namespace global utilizando \*:

from math import \*

def cos(x):

print('coça coça coça')

graus = float(input('Digite um ângulo: '))

# Função "radians" do módulo math - note que não chamamos "math."

radianos = radians(graus)

# funções "sin" e "cos" do módulo math - note que não chamamos "math."

seno = sin(radianos)

cosseno = cos(radianos)

print(seno, cosseno)

# função "sqrt" do módulo math - note que não chamamos "math."

prova = sqrt(2)/2

print(prova)

Saída na tela:

Digite um ângulo: 45

coça coça coça

0.7071067811865476 None

0.7071067811865476

Note que no código acima a função cos executada foi a local, e não a do módulo math. Isso ocorre porque a primeira definição de cos foi a importação do módulo math. Mas ao esbarrar no def cos(x), o Python redefiniu cos como sendo essa nova função. Predomina a última definição.

Por conta disso, essas últimas opções devem ser utilizadas com muito cuidado, especialmente o \*. Vários programadores optam por nunca utilizar importação para o namespace global.

O Python utiliza a ideia de namespace justamente para evitar conflitos. Diferentes módulos podem ter diferentes funções ou classes com o mesmo nome, e o Python sempre saberá diferenciá-los através de seu namespace. A partir do momento que começamos a importar nossos recursos diretamente para o namespace global, existe a possibilidade de conflito.

Imagine que você importou os módulos chamados modulo1 e modulo2, nesta ordem. Se ambos os módulos possuem uma função com o mesmo nome, a função do segundo módulo importado irá sobrescrever a função do primeiro módulo.

4. Instalando novos pacotes

O Python possui um gerenciador próprio de pacotes, chamado de pip. Outras distribuições baseadas em Python, como o Anaconda, podem vir com seus próprios gerenciadores de pacotes. No caso do Anaconta, esse é o conda.

O papel desses gerenciadores é facilitar o download e instalação dos pacotes. Eles conseguem controlar a versão, atualizar pacotes já existentes para versões mais novas e baixar todas as dependências (outros pacotes) de um pacote em particular no ato de sua instalação.

Em um terminal qualquer (como o Prompt de Comando no Windows, os terminais padrão do MacOS e do Linux, ou até mesmo a aba "console" no Visual Studio Code) você pode digitar:

pip install nome-do-pacote

Ou, para garantir que estamos na versão 3 do Python (alguns sistemas vêm com mais de uma versão pré-instalada):

pip3 install nome-do-pacote

Por exemplo, se você quiser instalar o famoso pacote de visualização de dados matplotlib, basta digitar:

pip3 install matplotlib

Os pacotes são buscados em um repositório chamado de Python Package Index. Em seu site, é possível consultar todos os repositórios disponíveis e ler sua documentação. Esses pacotes não são feitos necessariamente pelos criadores/mantenedores do Python, mas por membros diversos da comunidade.

https://pypi.org/

Uma vez instalado em seu computador, você irá importar e utilizar os pacotes adicionais da mesma maneira que você utiliza os módulos da biblioteca padrão do Python.

5. Criando um módulo

Criar e utilizar um módulo simples é bastante direto: basta criar um arquivo Python (extensão .py) e colocá-lo na mesma pasta de seu programa.

Em seguida, no programa, utilize import nome-do-arquivo (lembrando de não colocar a extensão nessa linha).

Para testar os exemplos abaixo, coloque cada bloco de código em um arquivo diferente na IDE de sua preferência e salve com os nomes indicados.

Arquivo matematica.py:

def soma(a, b):

return a + b

def subtracao(a, b):

return a - b

def multiplicacao(a, b):

return a \* b

def divisao(a, b):

return a / b

def potencia(a, b):

return a \*\* b

Arquivo principal.py:

import matematica

x = 1/2

y = 1.732/2

um = matematica.soma(matematica.potencia(x,2), matematica.potencia(y,2))

print(um)

Tudo que você faz utilizando módulos pré-instalados pode ser feito com seus módulos personalizados, como importar apenas funções específicas ou trazer todas as funções para o namespace global.

5.1. Detectando o namespace atual

Existe uma variável global em todo projeto Python chamada \_\_name\_\_. Ela indica o namespace vinculado ao arquivo atual. Se o arquivo atual é o programa sendo executado, o seu valor será \_\_main\_\_.

Isso ajuda a resolver um problema. Imagine que o nosso arquivo matematica.py não tivesse sido planejado para ser um módulo, e sim um script independente, como no exemplo abaixo.

Arquivo matematica.py:

def soma(a, b):

return a + b

def subtracao(a, b):

return a - b

def multiplicacao(a, b):

return a \* b

def divisao(a, b):

return a / b

def potencia(a, b):

return a \*\* b

n1 = float(input('Digite o primeiro número: '))

n2 = float(input('Digite o segundo número: '))

op = input('Digite a operação: ')

if op == '+':

print(soma(n1, n2))

elif op == '-':

print(subtracao(n1, n2))

elif op == '\*':

print(multiplicacao(n1, n2))

elif op == '/':

print(divisao(n1, n2))

elif op == '\*\*' or op == '^':

print(potencia(n1, n2))

else:

print('Operação inválida!')

Ao importarmos o arquivo matematica.py em outros arquivos, tudo o que está fora das funções (inputs, condicionais, prints...) será executado imediatamente. Esse arquivo foi planejado para ser um script, não um módulo.

Porém, com uma pequena modificação, nosso matematica.py pode cumprir ambos os papéis:

def soma(a, b):

return a + b

def subtracao(a, b):

return a - b

def multiplicacao(a, b):

return a \* b

def divisao(a, b):

return a / b

def potencia(a, b):

return a \*\* b

if \_\_name\_\_ == '\_\_main\_\_':

n1 = float(input('Digite o primeiro número: '))

n2 = float(input('Digite o segundo número: '))

op = input('Digite a operação: ')

if op == '+':

print(soma(n1, n2))

elif op == '-':

print(subtracao(n1, n2))

elif op == '\*':

print(multiplicacao(n1, n2))

elif op == '/':

print(divisao(n1, n2))

elif op == '\*\*' or op == '^':

print(potencia(n1, n2))

else:

print('Operação inválida!')

A linha if \_\_name\_\_ == '\_\_main\_\_': está perguntando se o arquivo está sendo executado. A condição do if será False caso o arquivo esteja sendo importado, e por conta disso a lógica "avulsa" do arquivo não será executada, apenas suas funções serão disponibilizadas.

5.2. Criando pacotes

Ao falarmos sobre utilizar gerenciadores como o pip, não falamos em instalar módulos, mas pacotes. Qual a diferença?

Um pacote não é apenas 1 arquivo contendo funções e/ou classes, mas uma pasta contendo vários módulos ou mesmo subpacotes.

Um pacote deverá ter, no mínimo, um arquivo chamado \_\_init\_\_.py. Em vários casos ele estará em branco, mas alguns pacotes mais sofisticados, inclusive vários frameworks populares, utilizam esse arquivo para realizar tarefas de inicialização importantes, como pré-definir constantes ou instanciar objetos.

Por exemplo, imagine que você irá criar um pacote de matemática com as funções que criamos neste capítulo e duas classes novas para representar frações e números complexos:

class Fracao:

def \_\_init\_\_(self, num=1, den=1):

self.numerador = num

self.denominador = den

class Complexo:

def \_\_init\_\_(self, real=0, img=0):

self.real = real

self.imaginario = img

Você gostaria que um módulo possuísse operações básicas, e outros diferentes tipos de dados (classes) que você mesmo definiu em exercícios anteriores. A estrutura dos arquivos poderia ser:

matematica

|

|---> \_\_init\_\_.py

|

|---> operacoes.py

|

|---> tipos

|

|---> fracao.py

|

|---> complexo.py

Simplesmente importar matematica agora não resolve muita coisa. Você deverá usar "from" ou utilizar sintaxe de pontinho para importar arquivos específicos. O exemplo abaixo importa apenas o arquivo operacoes.py:

import matematica.operacoes

s = matematica.operacoes.soma(1, 2)

Naturalmente, você pode dar um apelido para ficar menos verboso:

import matematica.operacoes as op

s = op.soma(1, 2)

O exemplo abaixo importa o arquivo complexo.py dentro da subpasta tipos:

from matematica.tipos import complexo

z1 = complexo.Complexo(1, 2)

Já o exemplo abaixo importa diretamente a classe Complexo de dentro do arquivo complexo.py para o namespace global:

from matematica.tipos.complexo import Complexo

z1 = Complexo(1, 2)