Módulos e Pacotes

Até o momento nossos programas tem sido bastante simples, e utilizamos na maioria das vezes apenas um único arquivo python, no mundo real, no entanto, muitos dos projetos são relativamente grandes, de forma que manter tudo em um arquivo só tornaria a tarefa de dar manutenção a aquele pedaço de código uma verdadeira batalha.

Uma das formas de facilitar o trabalho da criação de um programa complexo, é separar as partes do programa em módulos, assim, além de ser mais fácil focar em uma área do software de cada vez, permite que o programador escreva código que possa ser reutilizado em diversas parte do programa, ou até em outros programas!

Em uma das aulas anteriores já tivemos uma ideia de como é utilizar um módulo em python, utilizando a palavra-chave `import`, podemos incluir em nossos scripts outros modulos que não estão carregados na biblioteca padrão da linguagem.

Criando módulos

Para criar um módulo utilizando a linguagem python, basta criar um arquivo e utilizar a extensão .py, simples não? Isso quer dizer que todos os scripts que escrevemos até agora são módulos? Na verdade a resposta é sim e não, nossos scripts até o momento são pequenos programas, mas se importados em outro script python, ai sim, estes seriam considerados módulos.

Vamos supor os seguintes arquivos: **modulo.py** que contem algumas variáveis classes e funções importantes, e **script.py** que é o script principal de um programa.

```
# modulo.py
a = 2
c = 3

def soma(a, b):
        return a + b
```

Para incluir esse módulo no nosso script.py basta utilizar a palavra chave import:

```
# script.py
import modulo

print(modulo.a) # 2
print(modulo.c) # 3
print(modulo.soma(2, 3)) # 5
```

Usando a sintaxe import e o nome do arquivo que será importado como módulo automaticamente trás todos os membros (variáveis, funções, classes, etc) daquele módulo para o escopo atual como membros do objeto modulo, no entanto podemos optar por importar apenas alguns dos membros, e trazes eles para o escopo do script, para isso basta usar a palavra from:

```
# script.py
from modulo import a, c

print(a) # 2
print(c) # 3
print(soma(2, 3)) # ERROR name soma is not defined
```

 Cuidado, importar membros específicos de um módulo pode causar efeitos adversos, como sobrescrever variáveis e funções.

Apesar de não ser recomendado, podemos importar todos os membros de um módulo utilizando o caractere *:

```
from modulo import *

print(a) # 2
print(c) # 3
print(soma(2, 3)) # 5
```

Como dito acima, esse tipo de importação pode facilmente levar a um problema quando temos variáveis com os mesmos nomes das variáveis do módulo, uma das formas de sanar esse problema é dar nomes alternativos as variáveis importadas:

```
# script.py
from modulo import a as mod_a, c as mod_c

print(mod_a) # 2
print(mod_c) # 3
print(soma(2, 3)) # ERROR name soma is not defined
```

Pacotes

Pacotes são basicamente conjuntos de vários módulos, que podem ser importados e utilizados em outros scripts python. Na maioria das vezes pacotes são muito maiores do que simples módulos, e podem ser compostos de diversos módulos dispostos em um complexo sistema de pastas e arquivos. O pacote abaixo contem um arquivo chamado __init__.py, um módulo chamado modulo.py e uma pasta chamada others contendo outro modulo chamado other.py.

```
pkg/
        __init__.py
        modulo.py
        other/
                others.py
```

Podemos importar os módulos deste pacote seguindo a estruturação das pastas:

```
import pkg.modulo
import pkg.others.other.py
print(pkg.modulo.soma(2,3)) # 5
print(pkg.others.other.sub(3,2)) # 1
```

O arquivo __init__.py pode ser utilizado pelo desenvolvedor do pacote para importar de maneira sistemática módulos e membros de módulos específicos:

```
# __init__.py
print('Importing modules for ', __name__.py)
import pkg.modulo, pkg.others.other
```

Depois é só importar no nosso script:

```
# script.py
import pkg # Importing modules for pkg
print(pkg.modulo.soma(2,3)) # 5
print(pkg.others.other.sub(3,2)) # 1
```

Usando módulos e pacotes de outras pastas

Por padrão, o interpretador python vai buscar módulos presentes no diretório atual, ou em diretórios definidos pela variável de ambiente path, para saber quais pastas o interpretador está buscando por módulos, podemos escrever um pequeno script:

```
import sys
print(sys.path)
['C:\\Users\\user\\Documents\\Python', 'C:\\Python38\\python38.zip', 'C:\\Python38\\DLLs',
'C:\\Python38\\lib', 'C:\\Python38', 'C:\\Python38\\lib\\site-packages']
```

Para garantir que um módulo externo possa ser utilizado em nossos scripts, precisamos disponibilizar o arquivo em uma dessas pastas acima, ou então, adicionar uma pasta a essa lista, para isso basta usar o método append:

```
import sys
print(sys.path)
['C:\\Users\\user\\Documents\\Python', 'C:\\Python38\\python38.zip', 'C:\\Python38\\DLLs',
'C:\\Python38\\lib', 'C:\\Python38', 'C:\\Python38\\lib\\site-packages']
sys.path.append(r'C:\\Users\\user\\minha_pasta')
print(sys.path)
```

Utilizando o método acima não estamos necessariamente modificando a variável de ambiente, portanto o caminho só fica válido enquanto o programa estiver sendo executado.