Introdução à linguagem Python

Thiago Martins

Módulos e Pacotes

- São uma maneira de organizar código reutilizável em Python.
- Situação: Você cria uma função para ajudá-lo a processar dados
  - Exemplo: Uma função que recupera um objeto em formato JSON na rede:

```
def read_json(url):
  response = urllib.request.urlopen(url)
  encoding='UTF-8'
  for content_spec in response.headers['Content-Type'].split(';'):
    if content_spec.startswith('charset'):
      encoding = content_spec.split('=')[1].lstrip()
    return json.loads(response.read().decode(encoding))
```

- É natural que este tipo de código seja usado mais de uma vez!
- É indesejável reinserir isso a cada uso.

Módulos e Pacotes

- Código declarado em um módulo pode ser *importado* no contexto do interpretador.
- A diretriz `import <nome>` procura um arquivo com nome.py *no caminho atual* de módulos e, se encontra, executa este arquivo.
- No exemplo, suponha o arquivo funcoesuteis.py no caminho dos módulos.

```
def read_json(url):
  response = urllib.request.urlopen(url)
  encoding='UTF-8'
  isjson = False
  for content_spec in response.headers['Content-Type'].split(';'):
    if content_spec=='application/json':
      isjson = True
    elif content_spec.startswith('charset'):
      encoding = content_spec.split('=')[1].lstrip()
    return json.loads(response.read().decode(encoding))
```

funcoesuteis.py

```
import funcoesuteis

funcoesuteis.read_json(readjson.read_json('https://worldtimeapi.org/api/ip')
```

Módulos e *namespaces*

- Declarações dentro de um módulo são importadas dentro de um novo *namespace* com o mesmo nome do módulo.
- Exemplo:

```
a = ‘variavel 1’
b = ‘variavel 2’
def f():
  print(‘funcao 3’)

class minhaclasse():
  def __init__(self, val)
    self._v = val
```

meumodulo.py

```
import meumodulo

print(meumodulo.a)

meumodulo.f()

x = meumodulo.minhaclasse()
```

Módulos e *namespaces*

- Declarações dentro de um módulo são importadas dentro de um novo *namespace* com o mesmo nome do módulo.
- Exemplo:

```
a = ‘variavel 1’
b = ‘variavel 2’
def f():
  print(‘funcao 3 ’ + str(b))

class minhaclasse():
  def __init__(self, val)
    self._v = val
```

meumodulo.py

```
import meumodulo

print(meumodulo.a)

meumodulo.f()

x = meumodulo.minhaclasse()
```

- *Dentro* do módulo os seus nomes são referenciados normalmente!

Módulos e *namespaces*

- *Todas* as instruções em um módulo são executadas quando este é importado.
- Exemplo:

```
print(“modulo carregado”)
```

meumodulo2.py

```
>>> import meumodulo2
modulo carregado
```

- Útil para fazer inicializações mais complexas no módulo

Módulos e a variável __name__

- A variável __name__ contém o nome do módulo importado.
- Exemplo:

```
print(“Carregando o módulo ” + __name__)
```

meumodulo3.py

```
>>> import meumodulo3
Carrengando o módulo meumodulo3
>>> print(meumodulo3.__name__)
meumodulo3
```

- **O que significa `if __name__ == "__main__":` ??**
- Quando um arquivo é executado diretamente pelo interpretador (ao invés de ser importado como um módulo) a variável **`__name__`** adota o valor `"__main__"`. Este recurso é empregado para construir programas.

Caminho de busca de módulos

- A variável sys.path (ela mesma contida no módulo sys) contém os caminhos atuais de busca por módulos.
- O seu conteúdo depende do ambiente Python.
- No caso do interpretador, contém para além de diretórios padrão, também o conteúdo da variável de ambiente PYTHONPATH e o caminho do arquivo atualmente executado (ou do diretório a partir do qual o interpretador foi invocado).
- E no colab?

Caminho de busca de módulos

- A variável sys.path (ela mesma contida no módulo sys) contém os caminhos atuais de busca por módulos.
- O seu conteúdo depende do ambiente Python.
- No caso do interpretador, contém para além de diretórios padrão, também o conteúdo da variável de ambiente PYTHONPATH e o caminho do arquivo atualmente executado (ou do diretório a partir do qual o interpretador foi invocado).
- E no colab?

Escrevendo módulos no colab

- O interpretador do colab existe em uma máquina virtual *efêmera*.
- Todo o conteúdo é destruído ao final da execução.
- Uma alternativa para escrever e carregar módulos é usar o google drive...

    ... a mesma alternativa empregada na matéria.

```
from google.colab import drive
import sys
drive.mount('/content/gdrive', force_remount=True)
sys.path.append('/content/gdrive/My Drive/...')
```

Pacotes

- Um *pacote* é um conjunto de módulos organizado em diretórios.
- Um diretório pode conter o arquivo __init__.py que é executado quando o pacote é carregado.

  Exemplo:

```
pacote/
    __init__.py
    modulo1.py
    subpacote1/
        __init__.py
        modulo2.py
import pacote.modulo1
import pacote.subpacote.modulo2
```

Pacotes

- Um *pacote* é um conjunto de módulos organizado em diretórios.
- Um diretório pode conter o arquivo __init__.py que é executado quando o pacote é carregado.

  Exemplo:

```
pacote/
    __init__.py
    modulo1.py
    subpacote1/
        __init__.py
        modulo2.py
import pacote.modulo1
import pacote.subpacote.modulo2
```

Pacotes

- import funciona um pouco diferente para módulos que estão DENTRO de um pacote.
- Para um módulo importar outro dentro do *mesmo* pacote, é necessário incluir o nome do pacote no import:

  Exemplo

```
pacote/
    modulo1.py
    modulo2.py
```

```
import pacote.modulo2
```

modulo1.py

- Este import FALHARIA se o modulo.py fosse executado diretamente pelo interpretador!!!

Pacotes

- import relativo: SOMENTE para módulos dentro de pacotes.
- Use “.” para especificar o diretório atual. “..” para dois níveis acima, e assim por diante...

  Exemplo

```
pacote/
    modulo1.py
    subpacote/
        modulo2.py
        modulo3.py
```

```
from . import modulo2
from .. import modulo1
```

modulo3.py

- NOVAMENTE, este import FALHA se modulo3.py for chamado diretamente pelo interpretador.

Pacotes

- Este import FALHA se modulo3.py for chamado diretamente pelo interpretador.

```
pacote/
    modulo1.py
    subpacote/
        modulo2.py
        modulo3.py
```

```
from . import modulo2
from .. import modulo1
```

modulo3.py

- É possível executar um arquivo como se fosse um módulo:

  python -m pacote.subpacote.modulo3

Pacotes

- Símbolos definidos em __init__.py são incluídos em um namespace com o nome do pacote

```
a = ‘variavel 1’
```

pacote/__init__.py

```
import pacote
print(pacote.a)
```

Mais sobre import

- É possível importar somente símbolos específicos.

```
a = ‘variavel 1’
b = ‘variavel 2’
def f():
  print(‘funcao 3 ’ + str(b))

class minhaclasse():
  def __init__(self, val)
    self._v = val
```

meumodulo.py

```
from meumodulo import f

f()
```

- É possível dar novos nomes a símbolos:

```
from meumodulo import f as g

g()
```

Mais sobre import

```
from modulo import *
```

- Comportamento padrão: Insere TODOS os símbolos (exceto os iniciados por “_”) do módulo no namespace atual!

  Você TEM CERTEZA de que quer fazer isso?

  Variável __all__definida dentro do módulo modifica este comportamento: Lista de nomes de símbolos (strings) a serem importados com “`import *`”

```
from meumodulo import *
print(a)
print(c) # ERRO!
```

```
__all__ = [‘a’, ‘b’]
a = “variavel 1”
b = “variavel 2”
c = “variavel 3”
```

meumodulo.py

Mais sobre import

```
from pacote import *
```

- Comportamento padrão: Não importa *nada* para o namespace local (mas roda __init__.py)

    Variável __all__definida dentro do arquivo __init__.py modifica este comportamento: Lista de nomes de módulos (strings) a serem importados com "`import *`"

meumodulo.py

Mais sobre import

- Cuidado ao escrever módulos!
- Uma vez um módulo importado via import, subsequentes chamadas são *ignoradas*!
- Truque: Usar a função reload do módulo importlib.

```
import importlib
importlib.reload(meumodulo)
```

Mais sobre import

- Cuidado ao escrever módulos!
- Uma vez um módulo importado via import, subsequentes chamadas são *ignoradas*!
- Truque: Usar a função reload do módulo importlib.

```
import importlib
importlib.reload(meumodulo)
```

A Biblioteca padrão Python

- Diz-se que python tem “pilhas inclusas” (*batteries included*). A biblioteca padrão.
- I/O de arquivos, Expressões Regulares, Matemática, redes, rpc via xml, json, servidor web, sql...
  - https://docs.python.org/3/library/