Módulos

Módulos

▼ Importar um módulo (**import**)

> ‼️ Verificar módulos do Python: https://docs.python.org/3/py-modindex.html

- O método `import` importa todo o módulo
- Os elementos do módulo precisam ser acessados com `módulo` `.` `nome_do_elemento`
- Exemplo para importar módulo **sys**:

```
import sys

print(sys.platform)  # Exibe o sistema operacional
sys.exit()  # Encerra o programa
```

▼ Importar partes do módulo (**from** módulo **import** elemento)

- Permite importar apenas partes do módulo, tornando o acesso mais direto, sem precisar usar o nome do módulo como prefixo
- Vantagens:
  - Importa apenas o que é necessário, economizando memória

    - Nomes menores e código mais limpo, sem necessidade de usar `módulo . elemento`
- Desvantagens:
    - Não possui o namescape do módulo
    - Pode causar conflitos de nome se houver variáveis/funções locais com o mesmo nome
- Exemplo para importar **exit** de sys:

```
from sys import exit, platform

print(platform)  # Exibe o sistema operacional
exit()  # Encerra o programa
```

▼ Apelidar / renomear módulo (**as**)

- O **alias** (`as`) permite dar um nome diferente ao módulo ou função importada
- Vantagens:
    - Pode encurtar nomes longos, tornando o código mais limpo
    - Evita conflitos ao escolher nomes personalizados
- Desvantagens:
    - Pode reduzir a legibilidade do código se os nomes forem muito diferentes do original
    - Pode ficar fora do padrão da linguagem
- Exemplo abreviando nome de módulo:

```
import sys as s

print(s.platform)  # Exibe o sistema operacional
s.exit()  # Encerra o programa
```

- Exemplo abreviando nome de variável do módulo:

```
from sys import platform as pl

print(pl)  # Exibe o sistema operacional
```

▼ Importar tudo de um módulo (**from** módulo **import ***)

‼️ Isso é considerado uma má prática de programação.

- Esse método importa **todos** os elementos do módulo, permitindo usá-los diretamente **sem precisar do nome do módulo**
- Vantagens:
  - Menos código a escrever, pois não é necessário o prefixo do módulo
- Desvantagens:
  - Pode causar conflitos de nome, pois sobrescreve variáveis/funções existentes
  - Dificulta a leitura do código, pois não fica claro de qual módulo veio cada elemento
  - Importa tudo, consumindo mais memória
- Exemplo:

```
from sys import *

print(platform)  # Exibe o sistema operacional
exit()  # Encerra o programa
```

▼ Formas de importar um módulo em um diretório

- Supondo a seguinte estrutura de diretórios:

```
projeto/
├── main.py
├── packet/        ← Pacote
```

```
│   │── __init__.py  ← Indica que "packet" é um pacote
│   │── modulo.py    ← Módulo dentro do pacote
```

  - E supondo o conteúdo de `modulo.py` como:

```
# packet.modulo.py
def saudacao(nome):
    return f"Olá, {nome}!"
```

- Importar o módulo inteiro dentro do pacote:

```
# main
import packet.modulo

print(packet.modulo.saudacao("João"))  # Olá, João!
```

- Importar parte do módulo, no exemplo, apenas a função `saudacao`:

```
# main
from packet.modulo import saudacao

print(saudacao("João"))  # Olá, João!
```

- Importar módulo inteiro, mas agora direto do pacote, não precisando utilizar o nome do diretório no samespace:

```
# main
from packet import modulo

print(modulo.saudacao("João"))  # Olá, João!
```

- Importar tudo do módulo para o espaço atual:

```
# main
from packet.modulo import *

print(saudacao("João"))  # Olá, João!
```

    - Má prática de programação

▼ Como criar um módulo

- Um módulo é simplesmente um arquivo `.py` contendo funções, classes ou variáveis que podem ser reutilizadas em outros arquivos. Para criar um módulo:
    - Crie um arquivo chamado `meu_modulo.py`
- O módulo criado pode ser importado por outro, sendo considerado como `__main__` o primeiro módulo executado
- Exemplo de criação de módulo:

```
# meu_modulo.py
def saudacao(nome):
    return f"Olá, {nome}!"

PI = 3.1415
```

▼ Importar o módulo criado

- Supondo um módulo main e um módulo chamado `meu_modulo.py` , para importá-lo no main é possível fazer:

```
# meu_modulo.py
def saudacao(nome):
    return f"Olá, {nome}!"

PI = 3.1415
```

```
# main
import meu_modulo

print(meu_modulo.saudacao("João"))  # Olá, João!
print(meu_modulo.PI)  # 3.1415
```

- Para importar partes específicas:

```
# main
from meu_modulo import saudacao, PI

print(saudacao("Maria"))  # Olá, Maria!
print(PI)  # 3.1415
```

‼️ Isso funciona quando o módulo está no mesmo diretório ou no `sys.path`. Assim, caso o módulo importado esteja em um diretório externo ao módulo main, ele só poderá ser importado utilizando `sys.path`. Porém, caso seja um arquivo ou um diretório “vizinho” do módulo main, poderá ser importado.

▼ Arquivo/módulo main (**__main__**)

- Um arquivo main é o primeiro módulo executado em um código Python
  - Exemplo: `print ( __name__ )`
  - Exibe: `__main__`
- Quando um script Python é executado, ele possui um atributo especial chamado `__name__`. Se o arquivo for executado diretamente, `__name__` será `"__main__"`. Isso permite criar um bloco de código que só roda quando o script é executado diretamente, e não quando é importado como módulo.
- Exemplo:

```
# meu_modulo.py
def saudacao(nome):
    return f"Olá, {nome}!"

if __name__ == "__main__":
    print(saudacao("Mundo"))  # Só será executado se rodarmos direta
```

▼ Limitar importação (**__all__**)

- Podemos usar `__all__` para especificar o que pode ser importado em um módulo

    - Isso permite restringir alguns elementos de serem importados ao utilizar `import *` para importar tudo
    - Pode ser utilizado para selecionar funções, variáveis, etc
- Exemplo:

```
# modulo.py
__all__ = ["funcao_publica"]  # Apenas essa função será importada co

def funcao_publica():
    return "Essa função pode ser importada!"

def _funcao_privada():
    return "Essa função deveria ser privada!"
```

```
# main
from modulo import *

print(funcao_publica())  # OK
print(_funcao_privada())  # Erro! Não foi importada.
```

▼ Inicialização de um pacote (**__init__.py**)

- O arquivo `__init__.py` é usado para indicar que um diretório deve ser tratado como um pacote (package). Ele pode ser um arquivo vazio ou conter código de inicialização para o pacote.
- O `__init__.py` permite que um diretório se comporte de forma semelhante a um módulo, pois transforma esse diretório em um pacote. Isso significa que ele pode ser importado e referenciado como se fosse um único módulo.
- Possibilita executar um código de inicialização, contendo um código que é executado quando o pacote é importado.
- Permite especificar o que será importado com `from` `pacote` `import *`, definindo `__all__` dentro de `__init__.py`, controlando quais módulos são importados.
- Exemplo de criação de um pacote simples:

```
meu_pacote/
├── __init__.py
├── modulo1.py
├── modulo2.py
```

```
# modulo1.py

def func1():
    return "Função 1 do módulo 1"
```

```
# modulo2.py

def func2():
    return "Função 2 do módulo 2"
```

```
# __init__.py

# Código opcional para execução ao importar o pacote
print("Pacote 'meu_pacote' inicializado!")

# Importando funções diretamente para o pacote
from .modulo1 import func1
from .modulo2 import func2

# Define o que será importado com 'from meu_pacote import *'
__all__ = ["func1", "func2"]
```

- Agora é possível importar um pacote, semelhante a forma como os módulos são importados:

  ```
  import meu_pacote

  print(meu_pacote.func1())  # "Função 1 do módulo 1"
  print(meu_pacote.func2())  # "Função 2 do módulo 2"
  ```

- Agora é possível importar um pacote, semelhante a forma como os módulos são importados:

```
from meu_pacote import *

print(func1())  # "Função 1 do módulo 1"
print(func2())  # "Função 2 do módulo 2"
```

  - Isso funciona porque `__all__` foi definido no `__init__.py`.
- Desse modo, ao realizar `import meu_pacote`, os comando de inicialização escritos em `__init__.py` serão executados na inicialização:

```
import meu_pacote  # "Pacote carregado!" será impresso automat
```

▼ Recarregar um módulo que já foi importado + Singleton (**importlib.reload( )**)

- Em Python, **um módulo é um singleton**, o que significa que **ele é carregado na memória apenas uma vez por processo**. Se o módulo for importado várias vezes em diferentes partes do código, o Python **não cria uma nova instância**, ele apenas usa a versão já carregada.
  Exemplo:

```
import meu_modulo
import meu_modulo
# O Python não recarrega, apenas reutiliza o mesmo módulo já carreg
# Isso descarta alterações feitas no módulo após sua importação
```

- Quando um módulo é importado, ele é carregado na memória e **não será recarregado automaticamente** se o arquivo for modificado. Se você quiser que as mudanças tenham efeito sem reiniciar o interpretador, pode usar `importlib.reload()`.
- Exemplo:
  - Primeiro momento:

```
# meu_modulo.py
mensagem = "Versão original"
```

```
# main
import meu_modulo
print(meu_modulo.mensagem)  # Exibe: Versão original
```

- Segundo momento:

```
# meu_modulo.py
mensagem = "Versão modificada"
```

```
# main
import importlib
importlib.reload(meu_modulo)  # Recarrega o módulo
print(meu_modulo.mensagem)  # Exibe: Versão modificada
```