Curso Python Avançado

Módulos, Pacotes e Classes

olist

Módulos

Módulos

Definindo Módulos

Módulos em Python são arquivos “.py” contendo declarações e definições. Todo arquivo de código Python pode ser considerado um módulo.

Considerando o um módulo com o nome de arquivo “fibo.py” e o seguinte código.

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Código disponível em:
https://pastebin.com/uUe0Mzkp

Módulos

Importando Módulos

O módulo pode ser importado (utilizando) das seguintes maneiras.

```
>>> import fibo
>>> fibo.fib(1000)
>>> fibo.fib2(100)
>>> fibo.__name__



>>> from fibo import fib, fib2
>>> fib(100)
```

Módulos

Importando Módulos

Também é possível importar todos os nomes declarados em um módulo utilizando `*`. Obs: nomes iniciando com "`_`" *não* serão importados.

```
>>> from fibo import *
>>> fib(200)
```

O uso deste modo de importação não é recomendado.

Módulos

Importando Módulos

Uso de `as` após o nome de módulo ou de uma declaração sendo importada faz com que a importação seja identificada por um *alias*.

```
>>> import fibo as fib
>>> fib.fib(500)

>>> from fibo import fib as fibonacci
>>> fibonacci(500)
```

Módulos

Executando Módulos como Scripts

É possível executar um módulo como um script. Para isso basta executar o interpretador passando o arquivo de módulo como parâmetro.

```
python3 fibo.py 50
```

É possível identificar se o módulo está sendo usado como um script utilizando a variável “__name__”.

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

Pacotes

Pacotes

Definição

Pacotes são uma maneira de estruturar o “espaço de nomes” dos módulos Python, usando “nomes de módulo com pontos”.

Por exemplo, “A.B.C” define um módulo *C* dentro de um sub-pacote *B* dentro de um pacote *A*.

Pacotes

Definição

Considere o seguinte exemplo extraído da documentação do Python, onde é definida a estrutura de um pacote para tratamento de arquivos de som.

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Pacotes

Definição

O arquivo “__init__.py” é necessário para que o Python reconheça a pasta como sendo um pacote ou sub-pacote. Este arquivo pode ser vazio, mas também pode conter código de inicialização do pacote.

Pacotes

Importando Pacotes

Para este exemplo um módulo de um pacote pode ser importado das seguintes maneiras:

```
>>> import sound.effects.echo
>>> sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)

>>> from sound.effects import echo
>>> echo.echofilter(input, output, delay=0.7, atten=4)

>>> from sound.effects.echo import echofilter
>>> echofilter(input, output, delay=0.7, atten=4)
```

Pacotes

Importando Pacotes

Em outras aulas iremos estudar quais as implicações de usar o formato “from package import item” quando “item” for um sub-pacote e não um módulo.

Também iremos ver quais as consequências de se usar “from sound.effects import *”.
Neste momento iremos evitar importar desta maneira, não só porque é considerada uma má prática, mas também porque involve mais regras que não serão utilizadas neste momento.

Classes

Classes

Revisão de Escopos

Código disponível em: https://pastebin.com/mVg7ZAd0

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Classes

Revisão de Escopos

Resultado:

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Classes

Definindo uma Classe

Um exemplo de definição de classe:

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

Classes

Definindo uma Classe

Para obter uma instância:

```
>>> x = MyClass()
```

Classes

Definindo uma Classe

Definição de um *"construtor"*:

```
def __init__(self):
    self.data = []
```

Classes

Definindo uma Classe

Como em outras linguagens, é possível passar argumentos adicionais para o método de inicialização de uma instância (aqui estou chamando de construtor).

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```