Utilizando módulos e pacotes para implementar uma funcionalidade mais avançada

Principais conceitos a serem abordados

- Utilizando classes de biblioteca
  - random, NumPy, PIL
- Lendo a documentação

Módulos

- Para pequenos programas, podemos colocar uma classe em um arquivo
- Para grandes projetos, é difícil encontrar uma classe em muitas classes definidas
- Precisamos usar módulos
  - classes relacionadas

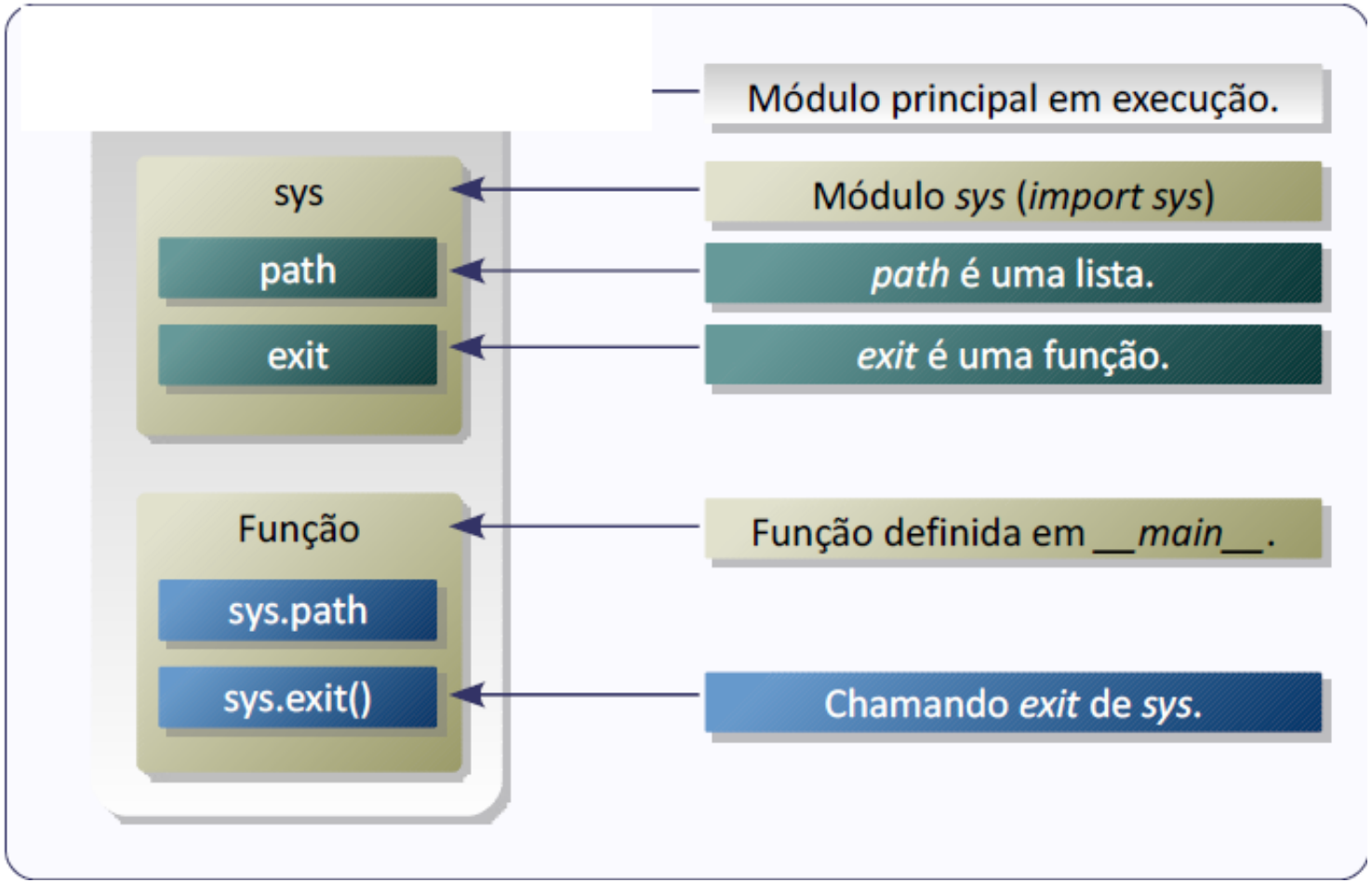
Módulos

- Módulos são simplesmente arquivos Python.
- Um simples arquivo com um pequeno programa é um módulo.
- Dois arquivos Python são dois módulos.

Módulos

- Se tivermos dois arquivos python em uma mesma pasta,
- Podemos carregar uma classe de um módulo para usar em outro módulo.

Módulos


Módulo principal em execução.
sys
path
exit
Módulo sys (import sys)
path é uma lista.
exit é uma função.
Função
sys.path
sys.exit()
Função definida em __main__.
Chamando exit de sys.

Módulos

- Podemos colocar todas as classes e funções relacionadas com acesso a um banco de dados em um arquivo separado (database.py).

- Então, outros módulos podem importar classes desse módulo para o acesso ao banco de dados.

Comando import

Existem muitas variações na sintaxe da declaração import que é usada para acessar as classes

Veremos as principais importações

Comando import

Importando todo o módulo sem especificar as classes:

```
import database
db = database.Database()
# faça consultas em db
```

Comando import

Importando o módulo e especificando uma classe:

```
from database import Database
db = Database()
# faça consultas em db
```

Comando import

Importando o módulo, especificando e nomeando uma classe:

```
from database import Database as DB
db = DB()
# faça consultas em db
```

Comando import

Importando o módulo e especificando duas classes:

from database import Database, Query

Comando import

Importando todas as classes do módulo:

*from database import **

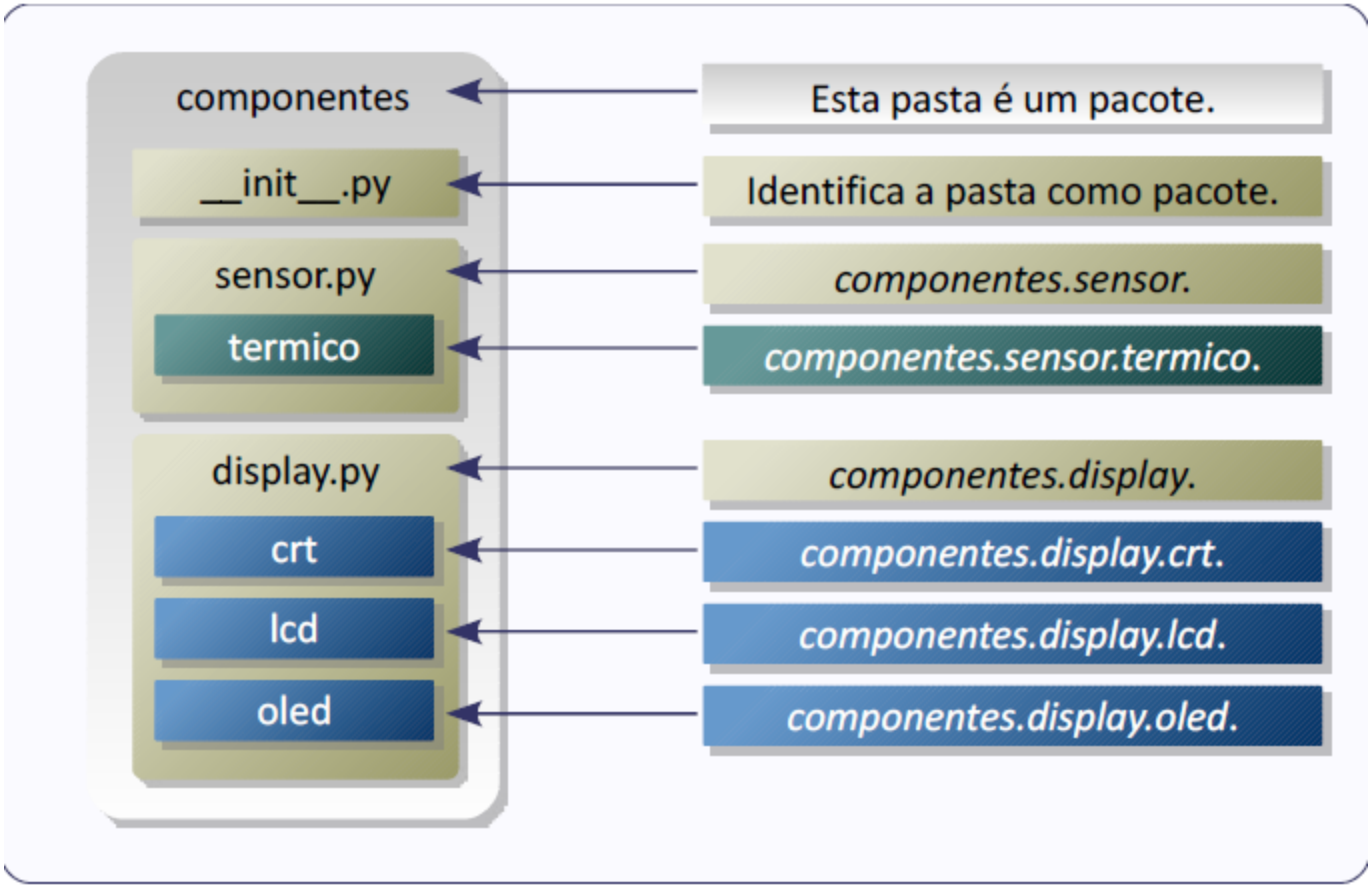
Pacotes

- Como o projeto cresce em uma coleção de módulos, precisamos adicionar um outro nível de abstração
- Não podemos colocar módulos dentro de módulos
- Módulos são apenas arquivos Python.

Pacotes

- Arquivos podem ser colocados em pastas, assim podemos colocar módulos.
- Um **pacote** é uma coleção de módulos em uma pasta.
- O nome do pacote é o nome da pasta.

Pacotes


componentes
__init__.py
sensor.py
termico
display.py
crt
lcd
oled
Esta pasta é um pacote.
Identifica a pasta como pacote.
componentes.sensor.
componentes.sensor.termico.
componentes.display.
componentes.display.crt.
componentes.display.lcd.
componentes.display.oled.

Pacotes

- parent_directory/
  main.py
  ecommerce/
    __init__.py
    database.py
    products.py
  payments/
    __init__.py
    square.py
    stripe.py

Imports absolutos

imports absolutos especifica o caminho completo para o módulo, função, ou caminho que queremos importar.

```
import ecommerce.products
product = ecommerce.products.Product()
```

ou

```
from ecommerce.products import Product
product = Product()
```

Imports relativos

- Imports relativos são basicamente a maneira de encontrar uma classe, função, ou módulo que está posicionado relativo ao módulo corrente

Imports relativos

- u*sando o módulo database dentro do pacote corrente*

from .database import Database

Imports relativos

- u*sando o pacote database dentro do pacote pai*

from ..database import Database

Organizando conteúdo de módulos

- Classes da biblioteca devem ser importadas utilizando uma instrução *import*.
- Elas podem então ser utilizadas como classes do projeto atual.

Acesso aos dados

- Classes podem ser organizadas em pacotes e módulos.
- Classes únicas podem ser importadas:
  `from random import Random`
- Módulos completos podem ser importados:
  `from random import *`

Utilizando `Random`

- A classe da biblioteca `Random` pode ser utilizada para gerar números aleatórios.

- As funções fornecidas no módulo random são métodos de ligação de uma instância escondida da classe Random

Utilizando `Random`

```
# Programa para gerar um número aleatório entre 0 e 9

#importando o modulo random
import random

print(random.randint(0,9))
```

Utilizando `Random`

- Se pode instanciar objetos Random para obter geradores que não compartilham estado.
- Classe Random também pode ser uma superclasse se vc precisa usar um gerador diferente próprio.
- Nesse caso, sobrecarregue os métodos random(), seed(), getstate(), e setstate().

Utilizando `Random`

- O módulo random também fornece a classe SystemRandom que usa a função sistema os.urandom() para gerar números aleatórios de fontes providas pelo sistema operacional.

Processamento numérico

- No Python, além dos recursos matemáticos que fazem parte da distribuição padrão,

- O processamento numérico pode ser feito através do NumPy e outros pacotes que foram construídos a partir dele.

NumPy

- NumPY é um pacote que inclui:
  - Classe array.
  - Classe matrix.
  - Várias funções auxiliares.

Arranjos

- A classe array implementa um arranjo homogêneo mutável com número arbitrário de elementos

- Semelhante à lista comum do Python, porém mais poderosa.

Arranjos

```
import numpy
# Criando arranjos
print (‘Arranjo criado a partir de uma lista:’)
a = numpy.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
print (a)
# [0 1 2 3 4 5 6 7 8]
print (‘Arranjo criado a partir de um intervalo:’)
z = numpy.arange(0., 4.5, .5)
print (z)
#[0. 0.5 1. 1.5 2. 2.5 3. 3.5 4.]
```

Arranjos

```
print (‘Multiplicando cada elemento por um escalar:’)
print (5 * z)
#[ 0. 2.5 5. 7.5 10. 12.5 15. 17.5 20.]
print (‘Redimensionando o arranjo:' )
z.shape = 3, 3
print (z)
#[[0. 0.5 1.]
# [1.5 2. 2.5]
#[3. 3.5 4.]]
```

Arranjos

```
print (‘Arranjo transposto:’)
print (z.transpose())
#[[0. 1.5 3.]
# [0.5 2. 3.5]
#[1. 2.5 4.]]
print (‘"Achata" o arranjo:’)
print (z.flatten())
#[0. 0.5 1. 1.5 2. 2.5 3. 3.5 4.]
print (‘O acesso aos elementos funciona como nas
    listas:' )
print (z[1])
#[1.5 2. 2.5]
```

Arranjos

```
# Dados sobre o arranjo
print (‘Formato do arranjo:' )
print (z.shape)
# (3, 3)
print (‘Quantidade de eixos:' )
print (z.ndim)
#2
print (‘Tipo dos dados:' )
print (z.dtype)
# float64
```

Processamento de imagem

- Python Imaging Library (PIL) é uma biblioteca de processamento de imagens matriciais para Python.
- PIL possui módulos que implementam:
  - Ferramentas para cortar, redimensionar e mesclar imagens.
  - Algoritmos de conversão, que suportam diversos formatos.
  - Filtros, tais como suavizar, borrar e detectar bordas.

Processamento de imagem

- PIL possui módulos que implementam (cont.):
  - Ajustes, incluindo brilho e contraste.
  - Operações com paletas de cores.
  - Desenhos simples em 2D.
  - Rotinas para tratamento de imagens: equalização, auto-contraste, deformar, inverter e outras.

Tratamento de imagem

```
# -*- coding: latin-1 -*-
"""
    Cria miniaturas suavizadas para cada JPEG na
    pasta corrente
    """
import glob
# Módulo principal do PIL
import Image
# Módulo de filtros
import ImageFilter
```

Tratamento de imagem

```
# Para cada arquivo JPEG
for fn in glob.glob(“*.jpg"):
    # Retorna o nome do arquivo sem extensão
    f = glob.os.path.splitext(fn)[0]
    print (‘Processando:', fn)
    imagem = Image.open(fn)
    # Cria thumbnail (miniatura) da imagem
    # de tamanho 256x256 usando antialiasing
    imagem.thumbnail((256, 256), Image.ANTIALIAS)
    # Filtro suaviza a imagem
    imagem = imagem.filter(ImageFilter.SMOOTH)
    # Salva como arquivo PNG
    imagem.save(f + '.png', 'PNG')
```

Escrevendo a documentação da classe

- Suas classes devem ser documentadas da mesma maneira como as classes da biblioteca.
- Outras pessoas devem ser capazes de utilizar sua classe sem ler a implementação.
- Torne sua classe uma ‘classe de biblioteca’!

Elementos da documentação (1)

A documentação de uma classe deve incluir:

- o nome da classe;
- um comentário descrevendo o propósito geral e as características da classe;
- um número da versão;
- o nome do autor (ou autores); e
- a documentação para cada construtor e cada método.

Elementos da documentação (2)

A documentação de cada construtor e método deve incluir:

- o nome do método;
- o tipo de retorno;
- o nome e os tipos de parâmetros;
- uma descrição do propósito e da função do método;
- uma descrição de cada parâmetro; e
- uma descrição do valor retornado.

Público *versus* privado

- Entidades públicas (atributos, construtores, métodos) são acessíveis a outras classes.
- Atributos não devem ser públicos.
- Entidades privadas são acessíveis apenas dentro da mesma classe.
- Somente os métodos concebidos para outras classes devem ser públicos.

Ocultamento de informações

- Dados que pertencem a um objeto são ocultados de outros objetos.
- Sabem *o que* um objeto pode fazer, não *como* ele faz isso.
- Ocultamento de informações aumenta o nível de *independência*.
- Independência dos módulos é importante para grandes sistemas e para manutenção.

Revisão

- Python tem uma extensa biblioteca de classes.
- Um bom programador precisa conhecer a biblioteca.
- A documentação informa o que precisamos saber para utilizar uma classe (interface).
- A implementação é ocultada (ocultamento de informações).
- Documentamos nossas classes para que a interface possa ser lida por si própria (comentários de classe, comentários de método).