Faculdade IMPACTA

Desempenho de Negócios

Aula 03 – Aquisição de Dados

Renato Rodrigues Oliveira da Silva
renato.silva@impacta.edu.br

Sumário

- Motivação
- Documentos não estruturados
- Documentos estruturados
  - CSV – Comma Separated Values
  - JSON – Javascript Object Notation
- Bases de Dados

Motivação

- Grande variedade e volume de informações disponíveis
  - Imagens, vídeos, textos, sons, sensores, etc
- Como obter e estruturar essas informações para posterior análise?
- As várias fontes de dados requerem diferentes estratégias para aquisição

Documentos não estruturados

- Textos livres, sem indicação (ou quase) de atributos para definir uma estrutura
  - Onde está o corpo do texto, título, autor, data de criação, categoria, etc?
- Fontes variadas
  - Listas de discussão
  - Relatórios
  - Manuais
  - Artigos, etc.

Leitura de arquivos em Python

- **open**: abertura de arquivo

  f = **open**(“nome_do_arquivo”, “modo”)

  – **f** é o descritor, usado para leitura

  – **nome_do_arquivo** é o caminho completo na árvore de diretórios

  – modo indica se o arquivo será aberto para leitura (r), escrita (w), se é texto (padrão) ou arquivo binário (b)

Leitura de arquivos em Python

- **readline**: lê uma linha do arquivo e avança a posição do descritor de arquivo

  linha = f.**readline**( )

  – **linha** é o string que contém a atualmente lida
    - Todos os caracteres da linha são lidos até encontrar um caractere de quebra de linha (“\n”)
    - A posição do descritor de arquivo é avançada automaticamente

Leitura de arquivos em Python

- **readlines**: lê todas as linhas do arquivo de uma vez

  linhas = f.**readlines**( )

  – **linhas:** lista com todas as linhas do arquivo

- **close**: fecha o arquivo, liberando recursos para o sistema

  – f.**close**()

Leitura de arquivos em Python

- **Exemplo**

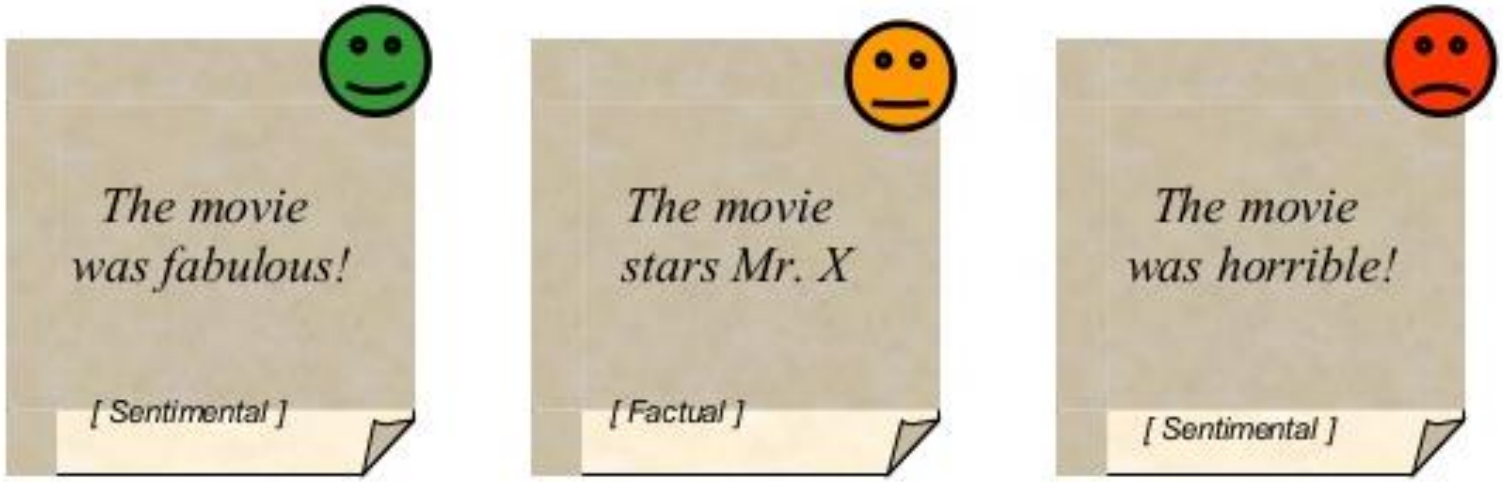
```
with open("hello.text", "r") as f:
    data = f.readlines()
    for line in data:
        words = line.split()
        print(words)
```

Leitura de arquivos em Python

- **Exercício**
  - Abre o arquivo “documento.txt”
  - Crie um dicionário que armazene a contagem de todas as palavras do texto
    - dicionario[“palavra”] = contagem_da_palavra

Leitura de arquivos em Python

- **Aplicação**
  - Análise de sentimento em posts do Twitter
    - Frequência de palavras, “peso” associado a cada palavra



Documentos estruturados: CSV

- Descrição textual de uma estrutura “tabular”
- Diversos campos bem definidos, separados por um caractere delimitador
  - Normalmente vírgulas ou ponto-e-vírgulas
- Facilidade de integração com bases de dados e planilhas

Documentos estruturados: CSV

- Utilizamos o módulo **csv** do Python
- reader = csv.reader(file, delimiter = ‘ ’)
  - **reader** é o controlador de leitura do arquivo csv
  - **file** é o descritor de arquivo (aberto previamente**)**
  - **delimiter** especifica o caractere delimitador de campos
- linha = next(reader)
  - Retorna em **linha** uma **lista** com os atributos lidos na posição atual do arquivo

Documentos estruturados: CSV

- **Exemplo**

```
import csv
with open("eggs.csv") as csvfile:
    reader = csv.reader(csvfile, delimiter=' ')
    for row in reader:
        print (', '.join(row))
```

Documentos estruturados: CSV

- reader = csv.DictReader(file, delimiter = ‘ ’)
  - O DictReader retorna um controlador de leitura em que cada leitura retorna um **dicionário** com informações sobre a linha lida
- linha = next(reader)
  - Retorna em **linha** uma **dicionário** com os atributos lidos na posição atual do arquivo
  - Para acessar o valor de um campo:
    - linha[‘nome_do_campo’]

Documentos estruturados: CSV

- **Exemplo**

```
import csv
with open('eggs.csv') as csvfile:
    reader = csv.DictReader(csvfile, delimiter=' ')
    for row in reader:
        print ('Name = {}'.format(row['name']))
```

Documentos estruturados: CSV

- **Exercício:**
  - Abrir e ler o arquivo **SalesJan2009.csv**
  - Contar quantas vendas utilizaram o cartão **Visa** como método de pagamento

Documentos estruturados: JSON

- É a notação para especificar um objeto em Javascript (Javascript Object Notation)
- Modelo de descrição de dados popular e estruturado
- Possui maior poder descritivo que o csv
  - Campos delimitados seguindo a estrutura de um dicionário
  - Pode descrever estruturas aninhadas, números, e listas

Documentos estruturados: JSON

- **Exemplo**

```
{
   "title": "Product",
   "description": "A product from Acme's  catalog",
   "type": "object",
   "properties": {
          "id": {
                "description": "The unique identifier for a  product",
                 "type": "integer "
           },
          "required": ["id", "name"]
     }
}
```

Documentos estruturados: JSON

- Utilizamos o módulo **json** para ler e escrever
- dados = json.load(f)
  - **dados**: dicionário contendo as associações chave/valor do arquivo
  - **f**: descritor do arquivo aberto
- dados = json.loads(string)
  - O método **loads** espera um texto JSON válido, e retorna o dicionário

Documentos estruturados: JSON

- **Exemplo**: Arquivo cores.json

```
import json
f = open("cores.json")
cores = json.load(f)
#cores['colors'] retorna a lista de registros
for cor in cores['colors']:
    print("Nome: {}, Codigo:  {}".format( \
            cor["color"], cor["code"]["hex"]))
```

Documentos estruturados: JSON

- **Exercício**: Ler o arquivo **countries.json**
  - Buscar e exibir o código do país “Bosnia”

- Estrutura do arquivo

```
{
   "countries": [
            {"name“ : "nome do pais",
              "code":" codigo do pais“
            },
             {"name" : .... }
   ]
}
```

Bases de Dados

- Estrutura normalmente utilizada para armazenar **grandes** volumes de dados em uma organização
- **Rapidez** na busca por informações
- **Flexibilidade** na construção de consultas
- **Robustez** e controle de **integridade**
- Porém é necessário utilizar o driver específico do SGBD

Bases de Dados: SQLite

- SQLite é um SGBD que acompanha a instalação do Python
- Simples e leve.
- A base é criada e mantida em um único arquivo.
- Não oferece todos os mecanismos dos sistemas mais completos como MySQL e PostgreSQL

Bases de Dados: SQLite

- Utilizamos o módulo **sqlite3**
- conn = connect('base.db')
  - Abre a base de dados (ou cria se não existir)
  - **conn**: Descritor da conexão
- cursor = conn.execute('query')
  - Executa uma consulta à base de dados
  - **cursor**: Controla o acesso aos valores retornados da consulta

Bases de Dados: SQLite

- **Exemplo**

```
import sqlite3
conn = sqlite3.connect('test.db')
cursor = conn.execute("SELECT id, name, \
                address, salary from COMPANY")
for row in cursor:
    print ("ID = ", row[0])
    print ("NAME = ", row[1])
    print ("ADDRESS = ", row[2])
    print ("SALARY = ", row[3], "\n")
conn.close()
```

Bases de Dados: SQLite

- **Exercício**
  - Executar o arquivo create_company.py
  - Ler todos os registros da tabela COMPANY e calcular a média de todos os salários

Web Services

- Possibilita a comunicação entre diferentes sistemas através da rede
  - Permite enviar e receber dados
- Normalmente a comunicação utiliza um formato padrão para troca de dados: CSV, JSON, XML
- Exemplo de tecnologias: REST e SOAP

Web Services

- SOAP (Simple Object Access Protocol)
  - Baseia-se na definição de uma estrutura XML para modelar o formato das mensagens
  - Utiliza o protocolo RPC ou HTTP para comunicação
- REST (Representational State Transfer)
  - Baseado no protocolo de comunicação HTTP
  - Não impõe restrições ao formato da mensagem
  - Possui maior flexibilidade

Web Services

- Em Python é necessário instalar o módulo **requests** e suas dependências
  - Entrar na linha de comando (com usuário de administrador)
  - Digitar: pip install requests
  - Aguardar o download e instalação do módulo

Web Services – Exemplo Correios ViaCEP

```
#Imprime os dados do endereço em JSON
import requests
cep = input("Informe o seu cep")
request =  \
requests.get("viacep.com.br/ws/%s/json/" \
                    %cep)
print (request.text)
```

Web Services – Exemplo Correios ViaCEP

- **Saída**:


```
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "unidade": "",
  "ibge": "3550308",
  "gia": "1004"
}
```

Exercício

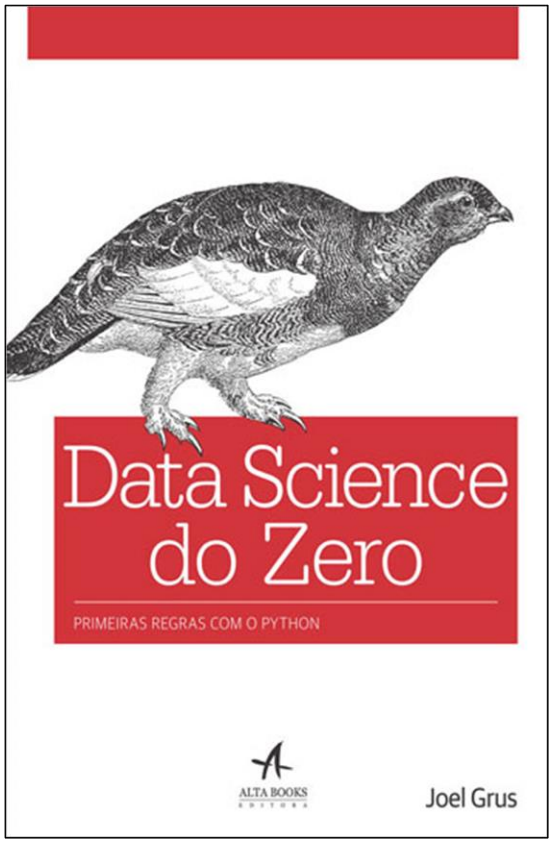
- Pedir para o usuário digitar um CEP
- Utilize o webservice ViaCEP para imprimir o logradouro, bairro e cidade do CEP informado

Outras fontes...

- Páginas Web
  - BeautifulSoap
- WebServices
  - SOAP
- XML
- Arquivos binários

Referências Bibliográficas

- Grus, J., **Data Science do Zero: Primeiras regras com o Python**



Referências Bibliográficas

- **Portal *Python for Beginners***
  - http://www.pythonforbeginners.com/files/reading-and-writing-files-in-python
- **Documentação oficial do Python**
  - https://docs.python.org/2/library/csv.html
- **Portal *Stack Abuse***
  - http://stackabuse.com/reading-and-writing-json-to-a-file-in-python/
- **Portal *Tutorials Point***
  - https://www.tutorialspoint.com/sqlite/sqlite_python.htm