UNIDADE 2
Tipos de Dados e Meios de Armazenamento

Disciplina: Tópicos Especiais III (DEC7553)
Prof. Alexandre L. Gonçalves
E-mail: a.l.goncalves@ufsc.br

- Pandas (DataFrame)

  - Representa uma tabela de dados retangular contendo uma coleção ordenada de colunas.
  - Cada coluna pode ter um tipo de valor diferente (numérico, *string*, booleano, etc).
  - Possui índices tanto para linha quanto para coluna. Pode ser visto como um dicionário de **Series**, todos compartilhando o mesmo índice.
  - Internamente, são armazenados como um ou mais blocos bidimensionais ao invés de uma lista, um dicionário ou outra coleção de *arrays* unidimensionais.

Pandas (DataFrame)

- Existe mais de uma forma de construir um DataFrame. Entre as mais comuns encontram-se os dicionários de listas de mesmo tamanho ou arrays NumPy.

```
data = {'estado': ['PR', 'SC', 'RS', 'RJ', 'MG', 'SP'],
        'ano': [2000, 2001, 2002, 2001, 2002, 2003],
        'desempenho': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
```

- Pandas (DataFrame)

  - O conjunto resultando possui um índice para as linhas atribuído automaticamente.

	estado	ano	desempenho
0	PR	2000	1.5
1	SC	2001	1.7
2	RS	2002	3.6
3	RJ	2001	2.4
4	MG	2002	2.9
5	SP	2003	3.2

Pandas (DataFrame)

- É possível determinar uma sequência específica de colunas do DataFrame.

```
pd.DataFrame(data, columns=['ano', 'estado'])
```

	ano	estado
0	2000	PR
1	2001	SC
2	2002	RS
3	2001	RJ
4	2002	MG
5	2003	SP

Pandas (DataFrame)

- Caso seja indicada uma coluna que não esteja no dicionário esta aparecerá com valores ausentes.

```
frame2 = pd.DataFrame(data, columns=['ano', 'estado', 'desempenho', 'débito'],
                      index=['um', 'dois', 'três', 'quatro',
                             'cinco', 'seis'])
frame2
```

	ano	estado	desempenho	débito
um	2000	PR	1.5	NaN
dois	2001	SC	1.7	NaN
três	2002	RS	3.6	NaN
quatro	2001	RJ	2.4	NaN
cinco	2002	MG	2.9	NaN
seis	2003	SP	3.2	NaN

Pandas (DataFrame)

- Uma coluna pode ser obtida como uma série usando uma notação de dicionário ou através de atributo.

```
print(frame2['estado'])
print(frame2.ano)
```

```
um       PR
dois     SC
três     RS
quatro    RJ
cinco     MG
seis     SP
Name: estado, dtype: object

um       2000
dois     2001
três     2002
quatro    2001
cinco     2002
seis     2003
Name: ano, dtype: int64
```

Pandas (DataFrame)

- As linhas podem ser obtidas com base na posição ou no nome do atributo através do método loc[].

```
frame2.loc['três']
```

```
ano          2002
estado         RS
desempenho    3.6
débito        NaN
```

Pandas (DataFrame)

- As colunas podem ser modificadas por atribuição. A variável débito poderia receber um escalar ou *array* de valores.

```
import numpy as np
frame2['débito'] = 16.5
frame2
```

	ano	estado	desempenho	débito
um	2000	PR	1.5	16.5
dois	2001	SC	1.7	16.5
três	2002	RS	3.6	16.5
quatro	2001	RJ	2.4	16.5
cinco	2002	MG	2.9	16.5
seis	2003	SP	3.2	16.5

```
frame2['débito'] = np.arange(6.)
frame2
```

	ano	estado	desempenho	débito
um	2000	PR	1.5	0.0
dois	2001	SC	1.7	1.0
três	2002	RS	3.6	2.0
quatro	2001	RJ	2.4	3.0
cinco	2002	MG	2.9	4.0
seis	2003	SP	3.2	5.0

Pandas (DataFrame)

- É possível também a atribuição de uma série utilizando a coluna de índice.

```
val = pd.Series([-1.2, -1.5, -1.7], index=['dois', 'quatro', 'cinco'])
frame2['débito'] = val
frame2
```

	ano	estado	desempenho	débito
um	2000	PR	1.5	NaN
dois	2001	SC	1.7	-1.2
três	2002	RS	3.6	NaN
quatro	2001	RJ	2.4	-1.5
cinco	2002	MG	2.9	-1.7
seis	2003	SP	3.2	NaN

Pandas (DataFrame)

- Ao fazer uma atribuição para uma coluna que não existe será criada uma nova coluna.

```
frame2['novo estado'] = frame2.estado == 'SC'
frame2
```

	ano	estado	desempenho	débito	novo estado
um	2000	PR	1.5	NaN	False
dois	2001	SC	1.7	-1.2	True
três	2002	RS	3.6	NaN	False
quatro	2001	RJ	2.4	-1.5	False
cinco	2002	MG	2.9	-1.7	False
seis	2003	SP	3.2	NaN	False

Pandas (DataFrame)

- Outro formato de dados utilizado para criar um DataFrame é um dicionário de dicionários aninhados.

```
pop = {'SP': {2001: 2.4, 2002: 2.9},
       'SC': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
frame3 = pd.DataFrame(pop)
frame3
```

	SP	SC
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

Pandas (DataFrame)

- É possível realizar a transposição dos dados.

frame3.T

	2000	2001	2002
SP	NaN	2.4	2.9
SC	1.5	1.7	3.6

Pandas (DataFrame)

- Caso **index** e **columns** de um DataFrame possuírem os atributos **name** definidos, esses serão exibidos.

```
frame3.index.name = 'ano'; frame3.columns.name = 'estado'
frame3
```

estado	SP	SC
ano		
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

Pandas (DataFrame)

- Assim como em uma série (series) o atributo **values** devolve os dados do DataFrame como um *array* bidimensional.

```
frame2.values
```

```
out ->
array([[2000, 'PR', 1.5, 0.0],
       [2001, 'SC', 1.7, 1.0],
       [2002, 'RS', 3.6, 2.0],
       [2001, 'RJ', 2.4, 3.0],
       [2002, 'MG', 2.9, 4.0],
       [2003, 'SP', 3.2, 5.0]], dtype=object)
```

Pandas (Objetos Index)

- Objetos Index são responsáveis por armazenar os rótulos dos eixos.

```
obj = pd.Series(range(3), index=['a', 'b', 'c'])
obj.index

out -> Index(['a', 'b', 'c'], dtype='object')
```

Pandas (Objetos Index)

- Os objetos Index são imutáveis.

```
obj.index[1] = 'd‘
```

out -> **TypeError**: Index does not support mutable operations

- A imutabidade torna o compartilhamento de objetos Index entre estruturas de dados mais seguro.

- Pandas (Reindexação)

  - Um método importante dos objetos do Pandas é **reindex**. Isto implica em criar um novo objeto com os dados de acordo com um novo índice.

```
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
obj
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
obj2
```

```
obj
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

```
obj2
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

Pandas (Reindexação)

- Nas séries temporais, talvez seja desejável fazer algum preenchimento de valores na reindexação.

```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
obj2

a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

```
obj3 = pd.Series(['azul', 'roxo', 'amarelo'],
index=[0, 2, 4])
obj3
obj3.reindex(range(6), method='ffill')

0       azul
1       azul
2       roxo
3       roxo
4    amarelo
5    amarelo
dtype: object
```

Pandas (Reindexação)

- Reindex pode alterar o índice (linha), as colunas, ou ambas.

```
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                     index=['a', 'c', 'd'],
                     columns=['SC', 'PR', 'RS'])
frame
frame2 = frame.reindex(['a', 'b', 'c', 'd'], columns=['SC', 'PR', 'RS'])
frame2
```

	SC	PR	RS
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

Pandas (Descartando entradas de um eixo)

- Após a criação de uma série é possível eliminar determinado conteúdo através da método **drop**.

```
obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
print(obj)
obj.drop(['d', 'c'])
```

```
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64
```

out ->

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

- Pandas (Descartando entradas de um eixo)

  - Através do DataFrame é possível apagar os valores de qualquer eixo.

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['SC', 'RS', 'PR', 'SP'],
                    columns=['um', 'dois', 'três', 'quatro'])
data
```

	um	dois	três	quatro
SC	0	1	2	3
RS	4	5	6	7
PR	8	9	10	11
SP	12	13	14	15

Pandas (Descartando entradas de um eixo)

Eliminando linhas

```
data.drop(['SC', 'RS'])
```

	um	dois	três	quatro
PR	8	9	10	11
SP	12	13	14	15

Eliminando colunas

```
data.drop('dois', axis=1)
data.drop(['dois', 'quatro'], axis='columns')
```

	um	três
SC	0	2
RS	4	6
PR	8	10
SP	12	14

Pandas (Indexação, Seleção e Filtragem)

- A indexação de séries é similar à indexação de *arrays* NumPy. Contudo, pode-se utilizar valores de índice da série ao invés de somente inteiros.

```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
print(obj)
print(obj['b'])
print(obj[2])
print(obj[2:4])
print(obj[['b', 'a', 'd']])
print(obj[[1, 3]])
print(obj[obj < 2])
```

```
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64

1.0

2.0

c    2.0
d    3.0
dtype: float64
```

```
b    1.0
a    0.0
d    3.0
dtype: float64

b    1.0
d    3.0
dtype: float64

a    0.0
b    1.0
dtype: float64
```

- Pandas (Indexação, Seleção e Filtragem)

  - É possível preencher a série com valores considerando um intervalo de índices.

```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
obj['b':'c'] = 5
obj
```

```
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

Pandas (Indexação, Seleção e Filtragem)

- A indexação em um DataFrame possibilita obter uma ou mais colunas, seja com um único valor ou uma sequência.

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['SC', 'RS', 'PR', 'SP'],
                    columns=['um', 'dois', 'três', 'quatro'])
data
data['dois']
data[['três', 'um']]
```

	três	um
SC	2	0
RS	6	4
PR	10	8
SP	14	12

- Pandas (Indexação, Seleção e Filtragem)
  - É possível realizar o fatiamento e a seleção booleana.

```
print(data[0:2])
data[data['três'] > 5]
```

	um	dois	três	quatro
SC	0	1	2	3
RS	4	5	6	7

	um	dois	três	quatro
RS	4	5	6	7
PR	8	9	10	11
SP	12	13	14	15

Pandas (Indexação, Seleção e Filtragem)

Atribuição utilizando uma seleção booleana.

```
print(data < 5)
data[data < 5] = 6
data
```

out ->

	um	dois	três	quatro
SC	True	True	True	True
RS	True	False	False	False
PR	False	False	False	False
SP	False	False	False	False

	um	dois	três	quatro
SC	6	6	6	6
RS	6	5	6	7
PR	8	9	10	11
SP	12	13	14	15

Pandas (Seleção com loc e iloc)

- Permitem a seleção de subconjuntos de linhas e colunas de um DataFrame usando rótulos de eixo (loc) ou inteiros (iloc).

```
data.loc['SC', ['dois', 'três']]
```

```
dois    1
três    2
Name: SC, dtype: int32
```

```
data.iloc[2, [3, 0, 1]]
data.iloc[2]
data.iloc[[1, 2], [3, 0, 1]]
```

	quatro	um	dois
RS	7	4	5
PR	11	8	9

Pandas (Seleção com loc e iloc)

```
print(data.loc[:'SC', 'dois'])
```

```
SC  1
Name: dois, dtype: int32
```

```
data.iloc[:, :4][data.quatro > 5]
```

	um	dois	três	quatro
RS	4	5	6	7
PR	8	9	10	11
SP	12	13	14	15

- Dados JSON

  - O JSON (*Javascript Object Notation*) vem se tornando um dos formatos padrões para requisições HTTP;

  - É um formato mais flexível em relação ao CSV;

- Dados JSON

  - Exemplo:

```
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
              {"name": "Katie", "age": 38,
               "pets": ["Sixes", "Stache", "Cisco"]}]
}
```

- Dados JSON

  - Os tipos básicos são objetos (dicionários), arrays (listas), strings, números, booleanos e valores nulos.
  - Todas as chaves em um objeto devem ser strings.
  - Existem algumas bibliotecas Python para ler e escrever dados JSON, entre elas a **json**.

Dados JSON

Criando uma string JSON.

```
import pandas as pd
import json
obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
              {"name": "Katie", "age": 38,
               "pets": ["Sixes", "Stache", "Cisco"]}]
}
"""
```

- Dados JSON

  - Convertendo uma string JSON em formato Python.

```
result = json.loads(obj)
result
```

```
{'name': 'Wes', 'places_lived': ['United States', 'Spain', 'Germany'],
    'pet': None, 'siblings': [{'name': 'Scott', 'age': 30, 'pets': ['Zeus',
    'Zuko']}, {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Stache',
    'Cisco']}]}
```

- Dados JSON

  - Convertendo um objeto Python novamente para JSON.

```
asjson = json.dumps(result)
```

- Dados JSON

  - É possível converte rum objeto JSON em um DataFrame.

```
siblings = pd.DataFrame(result['siblings'], columns=['name', 'age'])
siblings
```

	name	age
0	Scott	30
1	Katie	38

- Dados JSON

  - Através do Pandas é possível converter automaticamente conjuntos de dados JSON em Serie ou DataFrame.

```
data = pd.read_json('../../exemplos/example.json')
data
```

	quiz
maths	{'q1': {'question': '5 + 7 = ?', 'options': ['...
sport	{'q1': {'question': 'Which one is correct team...

- Dados JSON

  - O método to_json possibilita exportar dados do Pandas para o formato JSON.

```
data.to_json()
```

out ->

'{"quiz":{"maths":{"q1":{"question":"5 + 7 =
?","options":["10","11","12","13"],"answer":"12"},"q2":{"question":"1
2 - 8 =
?","options":["1","2","3","4"],"answer":"4"}},"sport":{"q1":{"question":
"Which one is correct team name in NBA?","options":["New York
Bulls","Los Angeles Kings","Golden State Warriros","Huston
Rocket"],"answer":"Huston Rocket"}}}}'

- Dados JSON

  - Obtendo determinado valor a partir de uma chave ou caminho de chaves.

```
print(data.get("quiz").get("maths").get("q1").get("question"))
```

out -> 5 + 7 = ?

- Dados XML

  - O XML (eXtensible Markup Language) é outro formato comum de dados que possibilita hierarquias e aninhamentos com metadados.
  - Através da biblioteca lxml é possível carregar e manipular arquivos em formato xml.

- Dados XML

  - Utilizando lxml.objectfy é possível realizar o parse do arquivo e a partir disso obter uma referência para o nó raiz através de getroot().

```
from lxml import objectify
import pandas as pd
import xml.etree.ElementTree as ET
import sys

sys.setrecursionlimit(10000)

path = '../../exemplos/performance.xml'
parsed = objectify.parse(open(path))
root = parsed.getroot()
```

- Dados XML

  - root.INDICATOR devolve um gerador que produz cada elemento XML<INDICATOR>.

```
data = []

#skip_fields = ['AGENCY_NAME']

for elt in root.INDICATOR:
    el_data = {}
    for child in elt.getchildren():
        if child.tag in skip_fields:
            continue
        el_data[child.tag] = child.pyval
    data.append(el_data)
```

Dados XML

- Utilizando Pandas é possível obter uma estrutura bidimensional (DataFrame).

```
perf = pd.DataFrame(data)
perf.head()
```

	AGENCY_NAME	CATEGORY	DECIMAL_PLACES	DESCRIPTION	DESIRED_CHANGE	FREQUENCY	INDICATOR_NAME
0	Metro-North Railroad	Service Indicators	1	Percent of commuter trains that arrive at thei...	U	M	On-Time Performance (West of Hudson)
1	Metro-North Railroad	Service Indicators	1	Percent of commuter trains that arrive at thei...	U	M	On-Time Performance (West of Hudson)

Dados XML

- Dados em XML podem ser complexos e cada *tag* pode ter metadados associados.

```
from io import StringIO
tag = '<a href="http://www.google.com">Google</a>'
root = objectify.parse(StringIO(tag)).getroot()

print(root.get('href'))
print(root.text)
```

out -> http://www.google.com
Google

- Formatos de Dados Binários

  - Uma das maneiras mais eficientes de serializar dados é utilizando a serialização embutida pickle do Python.

```
import pandas as pd
frame = pd.read_csv('../../exemplos/ex6.csv')
frame
frame.to_pickle('../../exemplos/frame_pickle')
```

- Formatos de Dados Binários

  - Uma vez salvo é possível recuperar o arquivo serializado para a memória.

```
frame = pd.read_pickle('../../exemplos/frame_pickle')
frame
```

	one	two	three	four	key
0	0.467976	-0.038649	-0.295344	-1.824726	L
1	-0.358893	1.404453	0.704965	-0.200638	B
2	-0.501840	0.659254	-0.421691	-0.057688	G
3	0.204886	1.074134	1.388361	-0.982404	R
4	0.354628	-0.133116	0.283763	-0.837063	Q
...					

- Formato HDF5

  - Formato de arquivo destinado a armazenar grandes quantidades de dados científicos em arrays.
  - Disponível através de uma biblioteca C, com interface para várias outras linguagens, entre elas, Java, Julia, MATLAB e Python.
  - HDF significa *Hierarchical Data Format*.
  - Cada arquivo HDF5 pode armazenar vários conjuntos de dados e pode aceitar metadados.

- Formato HDF5

  - Gerando dados e armazenando em formato HDF5.

```
import pandas as pd
import numpy as np
frame = pd.DataFrame({'a': np.random.randn(100)})
store = pd.HDFStore('../../exemplos/mydata.h5')
store['obj1'] = frame
store
```

Formato HDF5

- Gerando dados e armazenando em formato HDF5.

```
import pandas as pd
import numpy as np
frame = pd.DataFrame({'a': np.random.randn(100)})
store = pd.HDFStore('../../exemplos/mydata.h5')
store['obj1'] = frame
store

store['obj1']
```

	a
0	0.566643
1	0.307919
2	0.236928
3	-0.106868
4	1.629514
...	

- Formato HDF5

  - Armazenando e selecionando o conjunto de dados.

```
store.put('obj2', frame, format='table')
store.select('obj2', where=['index >= 10 and index <= 15'])
```

	a
10	-0.726472
11	0.489348
12	-0.351563
13	-0.293224
14	1.024278
15	-1.568169

- Formato HDF5

  - Armazenando e selecionando o conjunto de dados.

```
store.put('obj2', frame, format='table')
store.select('obj2', where=['index >= 10 and index <= 15'])
store.close()
```

	a
10	-0.726472
11	0.489348
12	-0.351563
13	-0.293224
14	1.024278
15	-1.568169

- Formato HDF5

  - Removendo o arquivo HDF salvo.

```
import os
os.remove('../../exemplos/mydata.h5')
```

- Lendo Arquivos do Microsoft Excel

  - O Pandas oferece suporte para ler dados tabulares a partir do Excel versão 2003 ou superior.
  - Utiliza a classe ExcelFile ou a função pandas.read_excel.
  - Internamente utilizam os pacotes add-on xlrd e openpyxl para ler arquivos XLS e XLSX

- Lendo Arquivos do Microsoft Excel

  - Lendo um arquivo XLSX e uma pasta específica.

```
import pandas as pd
xlsx = pd.ExcelFile('../../exemplos/ex1.xlsx')

pd.read_excel(xlsx, 'Plan1')
```

	Col1	Col2	Col3	Col4
0	1	2	1	2
1	2	3	4	1
2	1	1	2	1
3	2	2	1	1
4	4	2	1	1

- Lendo Arquivos do Microsoft Excel

  - Gravando um arquivo XLSX a partir de um data frame.

```
writer = pd.ExcelWriter('../../exemplos/ex2.xlsx', engine='openpyxl')
frame.to_excel(writer, index=False, sheet_name='Plan1', engine =
    'openpyxl')
writer.save()

writer.close()
```

- Interagindo com Web APIs

  - Vários sites possuem APIs públicas que oferecem feeds de dados usando JSON ou outro formato.
  - Uma forma de acessar tais dados é através do pacote **requests**.

```
import requests
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
resp = requests.get(url)
resp
```

```
out -> <Response [200]>
```

- Interagindo com Web APIs

  - Convertendo o conteúdo em formato JSON e acessando determinada célula de dados.

```
data = resp.json()
data[0]['title']

out -> 'DOC: Improve melt example (#23844)'
```

Interagindo com Web APIs

- Cria um data frame a partir dos dados recuperados através do request.

```
import pandas as pd
issues = pd.DataFrame(data, columns=['number', 'title',
                                     'labels', 'state'])
issues
```

	number	title	labels	state
0	28006	DOC: Improve melt example (#23844)	[]	open
1	28005	Explode on identical index causes duplication	[]	open
2	28004	Feature request: DataFrame.mse()	[]	open

- Interagindo com Banco de Dados

  - Em ambientes organizacionais uma grande parte dos dados reside em bancos de dados relacionais baseados na linguagem SQl.
  - Exemplos são o SQL Server, PostgreSQL, MySQL, Oracle, DB2, entre outros.
  - A escolha do banco de dados depende em geral da necessidade em relação ao tipo de aplicação e desempenho.

- Interagindo com Banco de Dados

  - Conectando com um banco de dados MySQL.

```
import mysql.connector

cnx = mysql.connector.connect(user='root', password='master',
                              host='localhost',
                              database='bd')
```

Interagindo com Banco de Dados

- Definindo uma estrutura para armazenar o esquema das tabelas para serem criadas no banco de dados.

```
import mysql.connector
from mysql.connector import errorcode

TABLES = {}
TABLES['empregado'] = (
    "CREATE TABLE `empregado` ("
    "  `emp_id` int(11) NOT NULL AUTO_INCREMENT,"
    "  `nome` varchar(100) NOT NULL,"
    "  `data_nascimento` date NOT NULL,"
    "  `data_contratacao` date NOT NULL,"
    "  PRIMARY KEY (`emp_id`)"
    ") ENGINE=InnoDB")
```

Interagindo com Banco de Dados

Criando tabelas a partir de uma lista de definições.

```
cursor = cnx.cursor()

for table_name in TABLES:
    table_description = TABLES[table_name]
    try:
        print("Criando tabela {}: ".format(table_name), end='')
        cursor.execute(table_description)
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
            print("Tabela já existe.")
        else:
            print(err.msg)
    else:
        print("OK")

cursor.close()
```

out -> Criando tabela empregado: OK

- Interagindo com Banco de Dados
  - Inserindo dados em uma tabela no banco de dados.

```
from datetime import date, datetime, timedelta
cursor = cnx.cursor()

datetime = datetime.now().date()
add_employee = ("INSERT INTO empregado "
                "(nome, data_nascimento, data_contratacao) "
                "VALUES (%s, %s, %s)")

data_employee = ('Funcionário 1', date(1977, 6, 14), datetime)

# Insert new employee
cursor.execute(add_employee, data_employee)
emp_no = cursor.lastrowid
print(emp_no)

cnx.commit()
cursor.close()
```

out -> True

- Interagindo com Banco de Dados
  - Selecionando dados a partir de uma tabela do banco de dados.

```
cursor = cnx.cursor()

query = ("SELECT emp_id, nome, data_nascimento, data_contratacao FROM empregado "
        "WHERE data_nascimento BETWEEN %s AND %s")

hire_start = date(1977, 1, 1)
hire_end = date(1977, 12, 31)

cursor.execute(query, (hire_start, hire_end))

for (emp_id, nome, data_nascimento, data_contratacao) in cursor:
  print("{}, {} foi contrato em {:%d/%m/%Y}".format(
    emp_id, nome, data_contratacao))

cursor.close()
```

out -> 1, Funcionário 1 foi contrato em 18/08/2019

- Interagindo com Banco de Dados

  - Ao final é necessário finalizar a conexão com o banco de dados.

```
cnx.close()
```

- Interagindo com Banco de Dados NoSQL

  - Termo genérico que define bancos de dados não-relacionais (**N**ot **O**nty **SQL**);
  - O termo NoSQL foi inicialmente cunhado em 1998;
  - Segundo Carlo Strozzi (criador do **Strozzi NoSQL**) o movimento NoSQL **"é completamente distinto do modelo relacional e portanto deveria ser mais apropriadamente chamado "NoREL" ou algo que produzisse o mesmo efeito"**.

- Interagindo com Banco de Dados NoSQL

  - Criando uma sessão com o banco de dados.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
```

- Interagindo com Banco de Dados NoSQL

  - Criando o acesso a determinado banco de dados e coleção.

```
db = client['ciencia_dados']

collection = db['unidade_2']
```

- Interagindo com Banco de Dados NoSQL

  - Inserindo determinado conteúdo em formato JSON.

```
import datetime
post = {"author": "Mike",
        "text": "My first blog post!",
        "tags": ["mongodb", "python", "pymongo"],
        "date": datetime.datetime.utcnow()}

post_id = collection.insert_one(post).inserted_id
post_id
```

- Interagindo com Banco de Dados NoSQL

  - Listando a coleção corrente.

```
db.list_collection_names()

out -> ['unidade_2']
```

- Interagindo com Banco de Dados NoSQL

  - Imprimindo o primeiro documento localizado.

```
import pprint
pprint.pprint(collection.find_one())

{'_id': ObjectId('5d59ebf68d0a68e5e566a12b'),
 'author': 'Mike',
 'date': datetime.datetime(2019, 8, 19, 0, 23, 15, 743000),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

- Interagindo com Banco de Dados NoSQL

  - Localizando determinado documento através de algum conteúdo de filtro.

```
collection.find_one({"author": "Mike"})

{'_id': ObjectId('5d59ebf68d0a68e5e566a12b'),
 'author': 'Mike',
 'date': datetime.datetime(2019, 8, 19, 0, 23, 15, 743000),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

- Interagindo com Banco de Dados NoSQL

  - Localizando determinado documento através de algum conteúdo de filtro.

```
post_id

pprint.pprint(collection.find_one({"_id": post_id}))

{'_id': ObjectId('5d5a15550eca4af8d5676864'),
 'author': 'Mike',
 'date': datetime.datetime(2019, 8, 19, 3, 19, 46, 885000),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

Interagindo com Banco de Dados NoSQL

Localizando todos os documentos de uma coleção.

```
for post in collection.find():
    pprint.pprint(post)
```

```
{'_id': ObjectId('5d59ebf68d0a68e5e566a12b'),
 'author': 'Mike',
 'date': datetime.datetime(2019, 8, 19, 0, 23, 15, 743000),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
{'_id': ObjectId('5d59ee0e8d0a68e5e566a12c'),
 'author': ‘John',
 'date': datetime.datetime(2019, 8, 19, 0, 32, 10, 310000),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
....
```

- Interagindo com Banco de Dados NoSQL

  - Contando o número de documentos de uma coleção.

```
collection.count_documents({})
```

Bons Estudos!