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

Estruturas de dados e sequências

- Embora bibliotecas *add-ons* como Pandas e NumPy forneçam funcionalidades avançadas para o processamento de grandes conjuntos, elas são projetadas para serem utilizadas com estruturas embutidas de manipulação de dados do Python;

Tupla

- Sequência imutável, de tamanho fixo, de objetos Python. A forma mais fácil para criar uma tupla é através da atribuição de valores separados por vírgula.

```
tup = 4, 5, 6
tup
out -> (4, 5, 6)
```

- Em geral quando expressões são mais complicadas, geralmente é necessário a utilização de parênteses.

```
nested_tup = (4, 5, 6), (7, 8)
nested_tup
out -> ((4, 5, 6), (7, 8))
```

Tupla

- Pode-se converter qualquer sequência ou iterador utilizando a função *tuple*.

```
tup = tuple([4, 0, 2])
print(tup)
tup = tuple('string')
tup
 out -> (4, 0, 2)
 out -> ('s', 't', 'r', 'i', 'n', 'g')
```

Tupla

- Uma tupla é imutável, mas objetos armazenados na tupla são mutáveis.

```
tup = tuple(['foo', [1, 2], True])
tup[2] = False

TypeError Traceback (most recent call last)
<ipython-input-5-11b694945ab9> in <module>
      1 tup = tuple(['foo', [1, 2], True])
      ----> 2 tup[2] = False
TypeError: 'tuple' object does not support item assignment
```

Tupla

- Como objetos em uma tupla são mutáveis, por exemplo, uma lista, pode-se modificá-la *in-place.*

```
tup[1].append(3)
tup
out -> ('foo', [1, 2, 3], True)
```

- Pode-se concatenar tuplas usando o operador “+”.

```
(4, None, 'ciência') + (6, 0) + ('dados',)
out -> (4, None, 'ciência', 6, 0, 'dados')
```

Tupla

- Multiplicar uma tupla por um inteiro, tem o efeito de concatenar a estrutura *n* vezes.

(‘ciência', ‘dados') * 4
('ciência', 'dados', 'ciência', 'dados', 'ciência', 'dados', 'ciência', 'dados')

- A atribuição de uma tupla a uma variável produz o desempacotamento da tupla.

```
#desempacotamento de tuplas
tup = (4, 5, 6)
a, b, c = tup
b
out -> 5
```

```
#desempacotamento de tuplas
tup = (4, 5, (6, 7))
a, b, (c, d) = tup
a, c
out -> (4, 6)
```

Tupla

- Case se deseje contar o número de ocorrências de determinado valor utiliza-se o método count().

```
a = (1, 2, 2, 2, 3, 4, 3)
a.count(2)
out -> 3
```

Arquivos e o Sistema Operacional

- O objetivo desta seção é apresentar os elementos básicos sobre arquivos em Python, apesar de existirem bibliotecas especializadas como o Pandas que serão analisadas mais adiante.

- Para abrir um arquivo para leitura ou escrita a função *open* deve ser utilizada com um caminho de arquivo relativo ou absoluto.

```
file = 'ciência_de_dados.txt'
f = open(file)
```

Arquivos e o Sistema Operacional

- As linhas são extraídas do arquivo com os marcadores de fim de linha (EOL) e com isso podem ser atribuídas para uma lista.

```
lines = [x.rstrip() for x in open(file)]
lines
```

- Ao término do acesso a determinado arquivo o mesmo deve ser fechado liberando o recurso para o Sistema Operacional.

```
f.close()
```

Arquivos e o Sistema Operacional

- Um forma de liberação do arquivo é através do bloco *with*.

```
with open(file) as f:
    lines = [x.rstrip() for x in f]
```

- Isto fechará o arquivo assim que ocorrer a saída do bloco *with*.

Arquivos e o Sistema Operacional

- Para arquivos em modo de leitura os métodos mais comuns são *read*, *seek* e *tell*. *read* faz a posição do *handle* do arquivo avançar o número de *bytes* lidos.

```
f = open(file)
f.read(12)
f2 = open(file, 'rb')  # Binary mode
f2.read(10)
```

- *tell* devolve a posição atual do arquivo.

```
f.tell()
f2.tell()
```

Arquivos e o Sistema Operacional

- *seek* altera a posição do arquivo para o *byte* indicado.

```
f.seek(3)
f.read(1)
out -> ‘n’
```

- Para escrever texto em um arquivo deve-se utilizar os métodos *write* ou *writelines*.

```
with open('tmp.txt', 'w') as handle:
    handle.writelines(x for x in open(file) if len(x) > 1)
with open('tmp.txt') as f:
    lines = f.readlines()
lines
```

Arquivos e o Sistema Operacional

- O método *remove* da biblioteca **os** possibilita a eliminação de um arquivo.

```
import os
os.remove('tmp.txt')
```

Arquivos e o Sistema Operacional

Modo	Descrição
r	Modo de leitura
w	Modo somente de escrita; cria um novo arquivo apagando os dados de qualquer arquivo com o mesmo nome
x	Modo somente de escrita; cria um novo arquivo, mas falha se já existir um arquivo no mesmo caminho
a	Concatena no arquivo existente (cria um arquivo caso ele ainda não exista)
r+	Leitura e escrita
b	Deve ser adicionado ao modo para arquivos binários (isto é, ‘rb’ ou ‘wb’)
t	Modo texto para arquivo (decodifica *bytes* automaticamente para Unicode). É o padrão se o modo não for especificado

NumPy: Arrays e Processamento Vetorizado

- NumPy, abreviatura de *Numerical Python*, é um dos pacotes básicos mais importantes para processamento numérico em Python.
- Vários pacotes de processamento científico utilizam objetos *array* do NumPy para a troca de dados.
- Recursos:
  - *ndarray*: *array* multidimensional que oferece operações aritméticas rápidas;
  - Funções matemáticas para operações rápidas em *arrays* de dados inteiros, sem que exista a necessidade de laços;
  - Ferramentas para ler/escrever dados de *array* em disco e trabalhar com arquivos mapeados em memória;
  - Recursos para álgebra linear, geração de números aleatórios e transformadas de Fourier;
  - Uma API C para conectar o NumPy a bibliotecas escritas em C, C++ ou FORTRAN.

NumPy: Arrays e Processamento Vetorizado

- As operações do NumPy sobre *arrays* são mais rápidas em relação a manipulação de estruturas nativas do Python.

```
import numpy as np
my_arr = np.arange(1000000)
my_list = list(my_arr)

%time for _ in range(10): my_arr2 = my_arr * 2
#print(my_arr2)
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
#print(my_list2)
```

```
Wall time: 39.2 ms
Wall time: 1.69 s
```

- Objeto Array Multidimensional

  - Um dos principais recursos do NumPy é o *ndarray* (objeto *array N*-dimensional).
  - *Container* para manipulação de grandes conjuntos de dados.
  - Permite realizar operações matemáticas em blocos inteiros de dados (em lote).

Objeto Array Multidimensional

```
import numpy as np
# Generate some random data
data = np.random.randn(2, 3)
data
out -> array([[-1.88271181, 1.43897225, 1.4529078 ],
              [ 0.68628268, 0.87018197, 0.49220185]])

res = data * 10
print(res)
res = data + data
print(res)
out ->
[[-18.82711813 14.38972249 14.529078 ]
 [ 6.86282683 8.70181972 4.92201854]]
[[-3.76542363 2.8779445 2.9058156 ]
 [ 1.37256537 1.74036394 0.98440371]]
```

- *ndarrays*

  - A maneira mais fácil de criar um *array* é através da função *array*.
  - Aceita qualquer objeto do tipo sequência e gera um novo *array* NumPy contendo os dados recebidos.

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
out ->
array([6. , 7.5, 8. , 0. , 1. ])
```

- *ndarrays*

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
arr2
out ->
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])

print(arr2.ndim)
print(arr2.shape)
out ->
2
(2, 4)
```

- *ndarrays*

```
print(arr1.dtype)
print(arr2.dtype)
out ->
float64
int32

print(np.zeros(10))
print(np.zeros((3, 6)))
print(np.ones(10))
print(np.empty((3, 6, 2)))
print(np.eye(4,4))
```

- *arrange* é uma versão da função embutida *range* do Python que devolve um *ndarray* ao invés de uma lista:

```
np.arange(15)
out ->
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
```

- Pandas

  - Contém estruturas de dados e ferramentas para manipulação de dados.
  - Projetada para facilitar a limpeza e análise de dados em Python.
  - Utilizada frequentemente com NumPy, SciPy, bibliotecas de análise como statsmodels e scikit-learn e bibliotecas de visualização de dados como matplotlib.
  - Pandas difere do NumPy por ter sido projetado para trabalhar com dados tabulares e heterogêneos.

- Pandas

  - Já o NumPy é mais apropriado para trabalhar com dados numéricos homogêneos em arrays.
  - Desde 2010, quando se tornou código aberto, o projeto evoluiu bastante e possui uma comunidade ativa.

- Séries

  - Uma série é um tipo de *array* unidimensional que possui uma sequência de valores e um *array* associado de rótulos, chamado de índice.
  - A série mais simples é composta apenas por um *array* de dados.

```
import pandas as pd
import numpy as np
obj = pd.Series([4, 7, -5, 3])
obj
out ->
```

```
0    4
17
2-5
33
dtype: int64
```

- Séries

  - A representação de uma série mostra o índice a esquerda e os valores à direita.
  - O índice varia de 0 a N – 1.
  - Pode-se obter a representação do *array* e o objeto de índice da série através de seus valores (*values*) e índice (*index*), respectivamente.

```
obj.values
out -> array([ 4, 7, -5, 3], dtype=int64)

obj.index
out -> RangeIndex(start=0, stop=4, step=1)
```

- Séries

  - Pode ser desejável criar uma série com um índice que identifique cada ponto de dado com um rótulo.

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(obj2)
obj2.index
out ->
d 4
b 7
a -5
c 3
dtype: int64
Index(['d', 'b', 'a', 'c'], dtype='object')
```

- Séries

  - Pode-se utilizar os rótulos no índice para selecionar valores únicos ou um conjunto de valores.

```
obj2['a']
out -> -5

obj2['d'] = 6

obj2[['c', 'a', 'd']]
out ->
c 3
a -5
d 6
dtype: int64
```

- Séries

  - Existe a possibilidade de filtragem booleana, multiplicação por um escalar ou aplicação de funções matemáticas.

```
print(obj2[obj2 > 0])
print(obj2 * 2)
print(np.exp(obj2))
```

```
out ->
d 6
b 7
c 3
dtype: int64
```

```
out ->
d 12
b 14
a -10
c 6
dtype: int64
```

```
out ->
d 403.428793
b 1096.633158
a 0.006738
c 20.085537
dtype: float64
```

- Séries

  - Pode-se também pensar em uma série como um dicionário ordenado de tamanho fixo. Pode ser utilizada em vários contextos em substituição a um dicionário.

'b' in obj2
out -> true

'e' in obj2
out -> false

- Séries

  - Uma série pode ser criada a partir de um dicionário.

```
sdata = {'SC': 35000, 'RS': 71000, 'PR': 80000, 'SP': 120000}
obj3 = pd.Series(sdata)
obj3
out ->
SC 35000
RS 71000
PR 80000
SP 120000
dtype: int64
```

- Séries
  - É possível sobrescrever os índices da série.

```
states = ['RJ', 'SC', 'PR', 'RS']
obj4 = pd.Series(sdata, index=states)
obj4
out ->
RJ NaN
SC 35000.0
PR 80000.0
RS 71000.0
dtype: float64
```

- Séries

  - As funções isnull e notnull são utilizadas no pandas para detectar dados ausentes.

```
pd.notnull(obj4)
RJ False SC True PR True RS True dtype: bool

obj4.isnull()
RJ True SC False PR False RS False dtype: bool
```

- Séries

  - Possibilita o alinhamento automático pelo rótulo do índice em operações aritméticas.

```
obj3 + obj4

out ->
PR 160000.0
RJ NaN
RS 142000.0
SC 70000.0
SP NaN
dtype: float64
```

- Séries
  - É possível indicar um título para a série e para os índices.

```
obj4.name = 'population'
obj4.index.name = 'state'
obj4
```

```
out ->
state
RJ NaN
SC 35000.0
PR 80000.0
RS 71000.0
Name: population, dtype: float64
```

- Séries

  - Um índice de uma série pode ser alterado por atribuição.

```
obj
obj.index = ['João', 'Maria', 'Alex', 'Victor']
obj
```

```
out ->
João 4
Maria 7
Alex -5
Victor 3
dtype: int64
```

- Carga e Armazenamento de Dados

  - Acessar dados é o primeiro passo necessário para a utilização de bibliotecas que irão manipular os dados.
  - A entrada e a saída de dados geralmente possuem as seguintes categorias de dados: arquivos em formato de texto ou outros formatos, dados a partir de um banco de dados e interação com fontes de dados disponíveis na web.

- Carga e Armazenamento de Dados

  - Pandas possui um série de funções para ler dados tabulares na forma de um objeto DataFrame.

Modo	Descrição
read_csv	Carrega dados delimitados de um arquivo ou um URL; utiliza vírgula como delimitador padrão
read_table	Carrega dados delimitados de um arquivo ou um URL; utiliza '\t' como delimitador padrão; descontinuado
read_fwf	Lê dados em formato de coluna com tamanho fixo (sem delimitadores)
read_clipboard	Versão de read_table que lê dados da área de transferência
read_excel	Lê dados tabulares de um arquivo Excel xls ou xlsx

Carga e Armazenamento de Dados

Modo	Descrição
read_hdf	Lê arquivos HDF5 escritos pelo Pandas
read_html	Lê todas as tabelas que se encontram em um determinado documento HTML
read_json	Lê dados de uma representação em string JSON (*JavaScript Object Notation*)
read_msgpack	Lê dados codificados pelo Pandas no formato binário MessagePack
read_pickle	Lê um objeto arbitrário armazenado no formato pickle do Python
read_sas	Lê um conjunto de dados SAS (Sistema SAS)
read_sql	Lê o resultado de uma consulta SQL na forma de um DataFrame do Pandas
read_stata	Lê um conjunto de dados no formato de arquivo Stata
read_feather	Lê o formato de arquivo binário feather

- Carga e Armazenamento de Dados

  - Para as funções anteriores existem vários argumentos (parâmetros) que se enquadram em algumas das seguintes categorias:
    - Indexação
    - Conversão de dados
    - Parsing de data e hora
    - Iteração parcial de grandes arquivos
    - Problemas com dados sujos/faltantes

Carga e Armazenamento de Dados

- Lendo um arquivo CSV. Como o separador no arquivo não é o padrão (‘,’) e na carga não foi indicado o separador desejado, os dados ficaram juntos.

```
df = pd.read_csv('../../exemplos/ex1.csv')
df
```

	a;b;c;d;mensagem
0	1;2;3;4;hello
1	5;6;7;8;world
2	9;10;11;12;foo

Carga e Armazenamento de Dados

- Lendo arquivo csv sem definir o cabeçalho e alterando o cabeçalho.

```
pd.read_csv('../../exemplos/ex2.csv', sep=';', header=None)
pd.read_csv('../../exemplos/ex2.csv', sep=';', names=['Col A', 'Col B', 'Col C',
    'Col D', 'Mensagem'])
```

	Col A	Col B	Col C	Col D	Mensagem
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Carga e Armazenamento de Dados

- Atribuindo determinada coluna para a coluna de índice.

```
names=['Col A', 'Col B', 'Col C', 'Col D', ‘Mensagem']
pd.read_csv('../../exemplos/ex2.csv', sep=';', names=names,
    index_col=Mensagem')
```

Mensagem	Col A	Col B	Col C	Col D
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

Carga e Armazenamento de Dados

- Indexando mais de uma coluna e criando uma quebra em colunas que possuem valores iguais em diferentes linhas.

```
parsed = pd.read_csv('../../exemplos/csv_mindex.csv', sep=';',
    index_col=[‘Chave 1', ‘Chave 2'])
parsed
```

		Valor 1	Valor 2
Chave 1	**Chave 2**		
Um	**a**	1	2
	b	3	4
	c	5	6
	d	7	8
two	**a**	9	10
	b	11	12
	c	13	14
	d	15	16

Carga e Armazenamento de Dados

- Lendo um arquivo texto e formatando o mesmo através de múltiplos espaços.

```
result = pd.read_csv('../../exemplos/ex3.txt', sep='\s+')
result
```

	A	B	C
aaa	-0.264438	-1.026059	-0.619500
bbb	0.927272	0.302904	-0.032399
ccc	-0.264273	-0.386314	-0.217601
ddd	-0.871858	-0.348382	1.100491

Carga e Armazenamento de Dados

- Lendo um arquivo CSV e ignorando determinadas linhas.

```
pd.read_csv('../../exemplos/ex4.csv', skiprows=[0, 2, 3])
```

	a	b	c	d	mensagem
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Carga e Armazenamento de Dados

- Testando se cada uma das células da tabela são ou não nulas.

```
result = pd.read_csv('../../exemplos/ex5.csv', sep='\t')
result
pd.isnull(result)
```

	Col A	Col B	Col C	Col D	Col E	Mensagem
0	False	False	False	False	False	True
1	False	False	False	True	False	False
2	False	False	False	False	False	False

Carga e Armazenamento de Dados

- na_values determina a lista ou um conjunto de strings que serão considerados como valores ausentes (nulos).

```
result = pd.read_csv('../../exemplos/ex5.csv', sep='\t', na_values=['NULL'])
result
```

	Col A	Col B	Col C	Col D	Col E	Mensagem
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

Carga e Armazenamento de Dados

- Atribuindo valores nulos para determinadas linhas e colunas.

```
new_values = {'Mensagem': ['foo'], 'Col A': ['two']}
pd.read_csv('../../exemplos/ex5.csv', sep='\t', na_values=new_values)
```

	Col A	Col B	Col C	Col D	Col E	Mensagem
0	one	1	2	3.0	4	NaN
1	NaN	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

Lendo Arquivos em Pedaços

- Efetuando a leitura em blocos de 1000 linhas.

```
chunker = pd.read_csv('../../exemplos/ex6.csv', chunksize=1000)

tot = pd.Series([])
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)

tot = tot.sort_values(ascending=False)
tot[:10]
```

```
out ->
E 368.0
X 364.0
L 346.0
O 343.0
Q 340.0
M 338.0
J 337.0
F 335.0
K 334.0
H 330.0
dtype: float64
```

Lendo e Gravando

- Realizando a leitura e a escrita de arquivos em formato CSV.

```
data = pd.read_csv('../../exemplos/ex5_1.csv')
data
```

```
data.to_csv('../../exemplos/out.csv')
data = pd.read_csv('../../exemplos/out.csv')
data
```

	Col A	Col B	Col C	Col D	Col E	Mensagem
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

Bons Estudos!