Aula 2 - numpy e pandas

Jayme Anchante

23 de fevereiro de 2021

montando ambiente

software

- git
- anaconda
- virtual environments
- reproducibility
- jupyter

numpy

pacotes que usam numpy

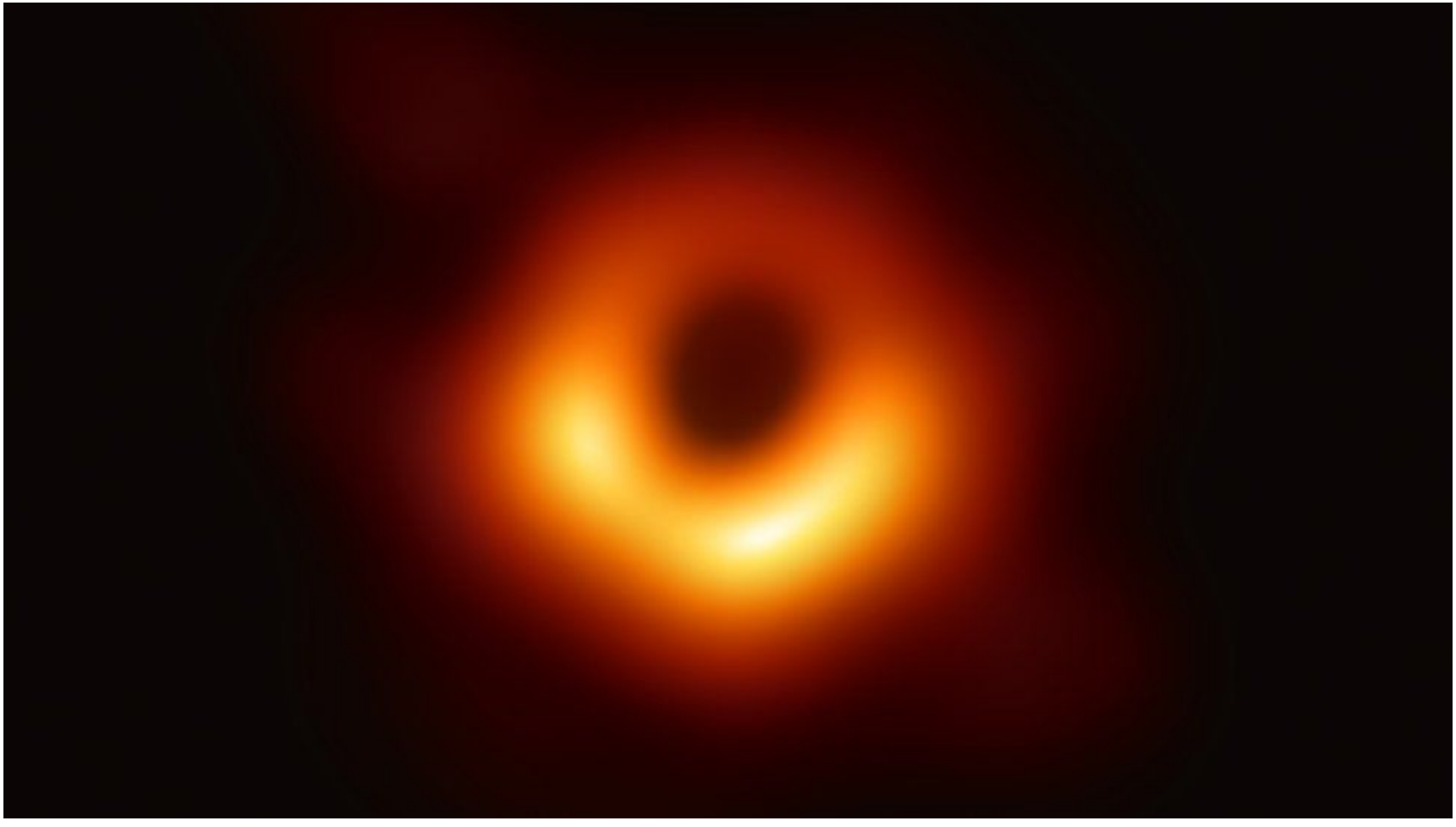
Quantum Computing	Statistical Computing	Signal Processing	Image Processing	Graphs and Networks	Astronomy Processes	Cognitive Psychology
						
QuTiP PyQuil Qiskit	Pandas statsmodels Seaborn	SciPy PyWavelets	Scikit-image OpenCV	NetworkX graph-tool igraph PyGSP	AstroPy SunPy SpacePy	PsychoPy

Bioinformatics	Bayesian Inference	Mathematical Analysis	Simulation Modeling	Multi-variate Analysis	Geographic Processing	Interactive Computing
						
BioPython Scikit-Bio PyEnsembl	PyStan PyMC3	SciPy SymPy cvxpy FEniCS	PyDSTool	PyChem	Shapely GeoPandas Folium	Jupyter IPython Binder

The fundamental package for scientific computing with Python, NumPy website

fronteira da ciência

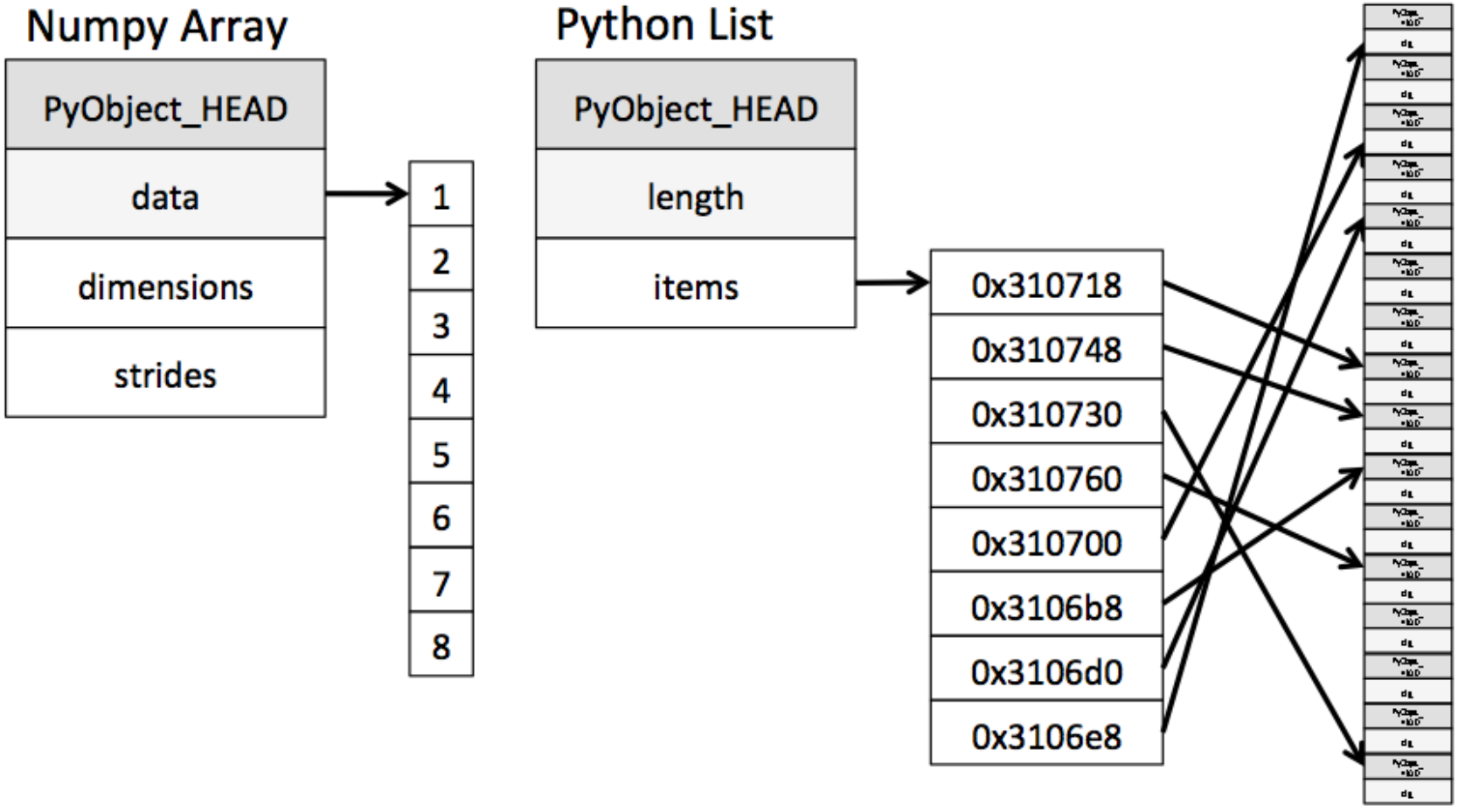
Case Study: First Image of a Black Hole



listas vs arrays vs numpy arrays

- listas aceitam qualquer tipo de dados (flexibilidade)
- arrays tem tipo fixo (armazenamento eficiente)
- numpy arrays tem tipo fixo e otimizações (armazenamento e cálculo eficiente)

lista vs numpy array



criando arrays

```
1  np.zeros(10)
2  np.ones((2,2))
3  np.full((3,1), 3.14)
4  np.arange(5)
5  np.linspace(0, 1, 5)
```

gerador de números pseudo-aleatórios

```
1  np.random.RandomState(42)
2  np.random.seed(42)
3  np.random.<tab>
```

acessando elementos

```
x1 = np.random.randint(10, size=6)       # 1 dim
x2 = np.random.randint(10, size=(3, 4)) # 2 dim
x1[0]
x1[-2]
x2[0, 0]
x2[2, -1]
x2[0, 0] = 12
```

fatiamento de elementos

```
# x[start:stop:step]
x1[:5]    # primeiros cinco elementos
x1[::2]   # cada dois elementos
x1[::-1] # inversão dos elementos
```

cópia de objetos

```
x2_sub = x2[:2, :2]
x2_sub[0, 0] = 99
# o que aconteceu com x2?
x2_sub_copy = x2[:2, :2].copy() # fazendo uma cópia
x2_sub_copy[0, 0] = 42
# o que aconteceu com x2?
```

reformatação de objetos

```
1  x = np.array([1, 2, 3])
2  x.reshape((1, 3)) # row vector via reshape
3  x[np.newaxis, :]  # row vector via newaxis
4  x.reshape((3, 1)) # column vector via reshape
5  x[:, np.newaxis]  # column vector via newaxis
```

junção e separação de objetos

```
1 np.concatenate
2 np.vstack
3 np.hstack
4 np.split
5 np.vsplit
6 np.hsplit
```

operações com numpy

```
1  def compute_reciprocals(values):
2      output = np.empty(len(values))
3      for i in range(len(values)):
4          output[i] = 1.0 / values[i]
5      return output
6  big_array = np.random.randint(1, 100, size=1000000)
7
8  %timeit compute_reciprocals(big_array)
9
10 %timeit 1.0 / big_array
```

operações com numpy

```
# operações agregação
x = np.arange(1, 6)
np.add.reduce(x)          # soma dos elementos
np.add.accumulate(x)      # soma acumulada
np.multiply.outer(x, x) # produto cartesiano
```

sumarizando np.array

```
a = np.random.random(100)
sum(a)
np.sum(a)
a.max()
# soma com dados faltantes
np.nansum(a)
```

máscaras np.array

```
a = np.random.random(10)
a > 5
a[a>5]
a[(a>5) & (a<7)]
# suporte ao |/or e ~/not
```

dado o seguinte np.array

```
1  a = np.arange(25).reshape(5,5)
```

1. Retorne os valores pares positivos menores que 14.
2. Qual a média da segunda coluna?
3. Qual a soma da quarta linha?
4. Separe a última linha e transforme em um vetor coluna.
5. Salve o objeto contendo o np.array em disco.

pandas

história

Iniciado por Wes McKinney em 2008 quando ele trabalhava no mercado financeiro

Começou como uma implementação em Python da API de dataframe do R

Código aberto em 2009 e posterior apoio pela NumFocus

Livro base Pyhon para Análise de dados

pd.Series

```
1  import pandas as pd
2  data = pd.Series([0, 2, 4, 6])
3  # valores e índice são np.array
4  data.values
5  data.index
6  # acessando elementos pelo índice
7  data[-1]
8  data[:2]
```

pd.Series vs np.array

Uma das grandes diferenças está no índice. Ele pode ser não numérico, não sequencial.

```
1  data = pd.Series([0.25, 0.5, 0.75, 1.0],
2                   index=['a', 'b', 'c', 'd'])
3  data['b']
```

pd.Series vs dict

```
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)
population['California':'Illinois']
```

pd.DataFrame

Se o pd.Series pode ser comparados a um vetor unidimensional, o pd.DataFrame pode ser comparado a uma matrix bidimensional.

O pd.DataFrame é como uma sequencia de pd.Series que compatilham o mesmo índice.

```
area_dict = {'California': 423967, 'Texas': 695662,
             'New York': 141297, 'Florida': 170312,
             'Illinois': 149995}
area = pd.Series(area_dict)
```

pd.DataFrame a partir de pd.Series

```
states = pd.DataFrame({'population': population,
                       'area': area})
```

exercícios de casa

lista

Façam um pipelines para processamento de uma dataset escolhido