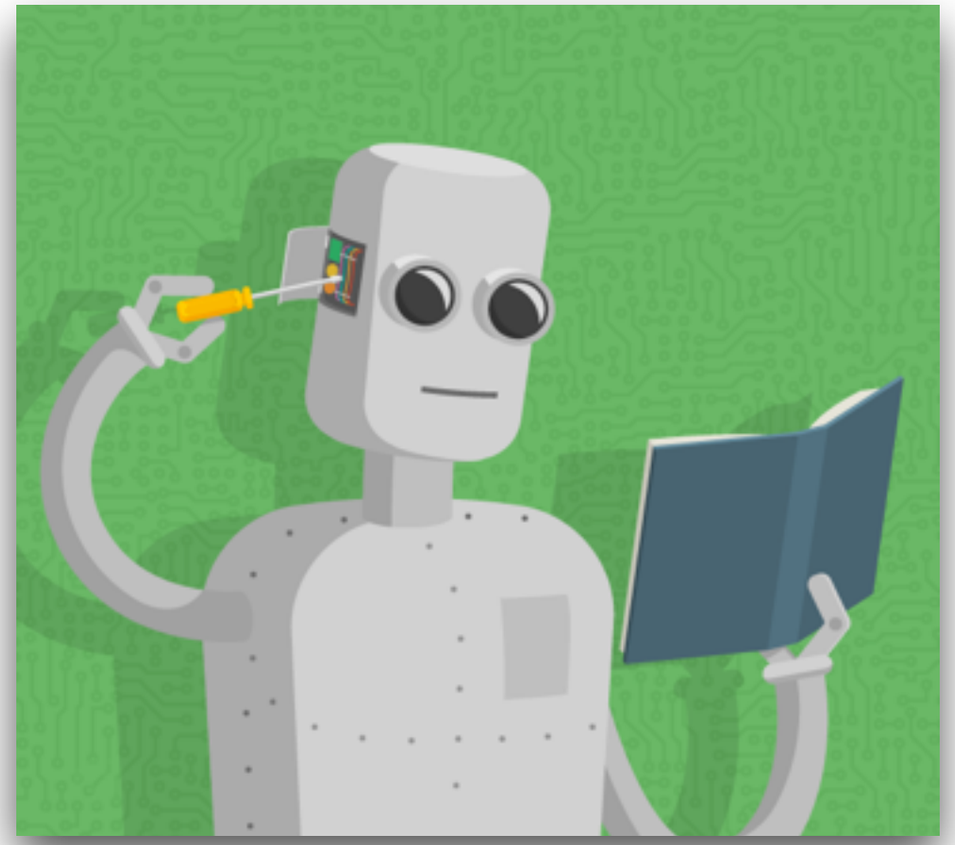
Disciplina de Inteligência Artificial (Aula 2)



Introdução ao Python

Apresentação

Francisco Nauber Bernardo Gois

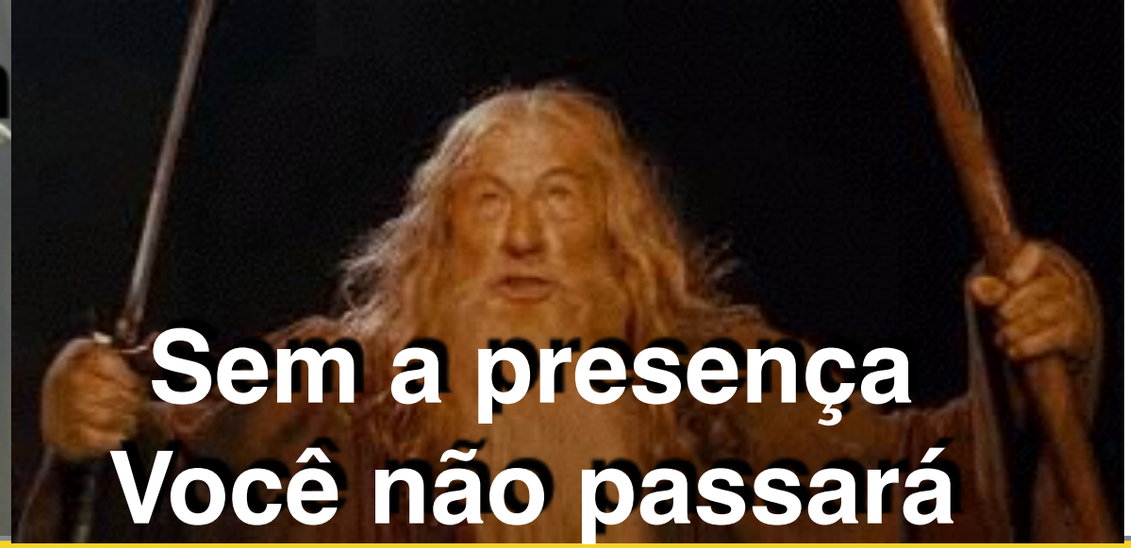
Analista aprendizado de máquina
no Serviço Federal de Processamento
de Dados

Doutorando em Informática Aplicada

Mestre em Informática Aplicada

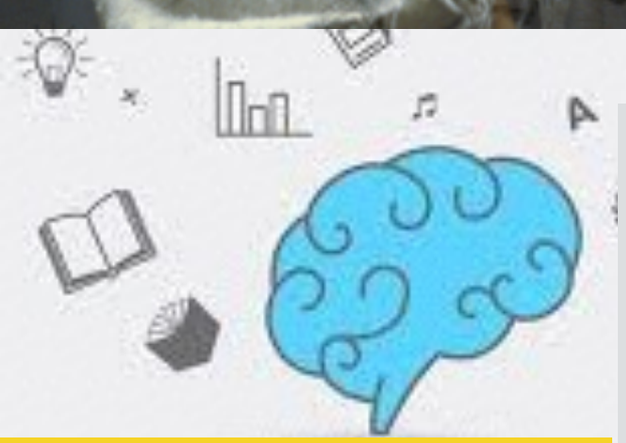
Especialista em desenvolvimento WEB

Dúvidas: naubergois@gmail.com



Para melhor desempenho na aula

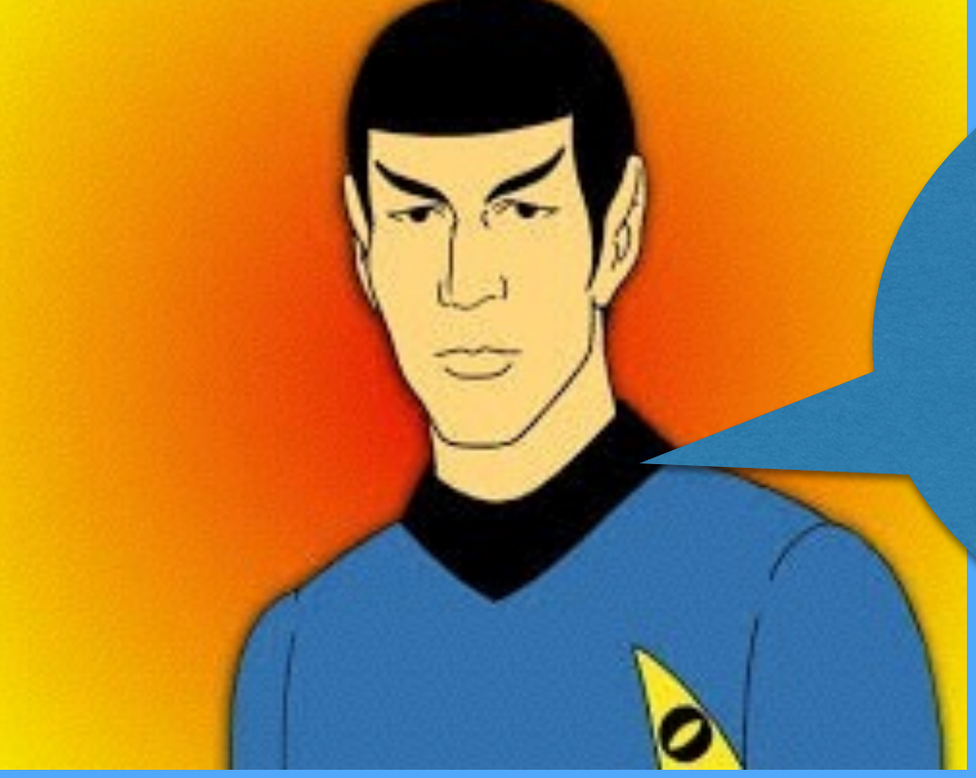
Cuidado com o Horário



Procure não conversar durante a aula

Buscar aprendizado ao invés de pontos

Não teremos pontuação fora dos trabalhos e provas da disciplina



OS trabalhos deveram ser entregues uma semana antes da prova

Um cadeira longa e prospera

Não teremos pontos após a prova não adianta pedir

Cronograma da Disciplina

Introdução a Inteligência Artificial

Introdução Aprendizado de Máquina

Introdução ao Python

O que vimos na aula passada

- Introdução ao Inteligência Artificial.
- Cronograma da disciplina.
- Como será a composição da nota.
- Principais áreas da inteligência artificial

Objetivo da Aula

- Introdução ao Python
- Conhecer o Numpy
- Conhecer conceitos de busca

Numpy Cheat Sheet

Python Package

Created By: Arianne Colton and Sean Chen

Numpy (Numerical Python)

What is NumPy?

Foundation package for scientific computing in Python

Why NumPy?

- Numpy **'ndarray'** is a much more efficient way of storing and manipulating **"numerical data"** than the built-in Python data structures.
- Libraries written in lower-level languages, such as C, can operate on data stored in Numpy **'ndarray'** without copying any data.

N-DIMENSIONAL ARRAY (NDARRAY)

What is NdArray?

Fast and space-efficient multidimensional array (container for homogeneous data) providing vectorized arithmetic operations

Create NdArray	`np.array(seq1)` # seq1 - is any sequence like object, i.e. [1, 2, 3]
Create Special NdArray	`1, np.zeros(10)` # one dimensional ndarray with 10 elements of value 0 `2, np.ones(2, 3)` # two dimensional ndarray with 6 elements of value 1 `3, np.empty(3, 4, 5)` * # three dimensional ndarray of uninitialized values `4, np.eye(N) or np.identity(N)` # creates N by N identity matrix
NdArray version of Python's `range`	`np.arange(1, 10)`
Get # of Dimension	`ndarray1.ndim`
Get Dimension Size	`dim1size, dim2size, .. = ndarray1.shape`
Get Data Type **	`ndarray1.dtype`
Explicit Casting	`ndarray2 = ndarray1.astype(np.int32)` ***

- Cannot assume empty() will return all zeros. It could be garbage values.

- ** Default data type is **'np.float64'**. This is equivalent to Python's float type which is 8 bytes (64 bits); thus the name 'float64'.
- *** If casting were to fail for some reason, **'TypeError'** will be raised.

SLICING (INDEXING/SUBSETTING)

- Slicing (i.e. `ndarray1[2:6]`) is a **'view'** on the original array. **Data is NOT copied.** Any modifications (i.e. `ndarray1[2:6] = 8`) to the 'view' will be reflected in the original array.
- Instead of a 'view', explicit copy of slicing via :

```
ndarray1[2:6].copy()
```

- Multidimensional array indexing notation :

```
ndarray1[0][2] or ndarray1[0, 2]
```

*** Boolean indexing** :

```
ndarray1[(names == 'Bob') | (names == 'Will'), 2:]
# "2:" means select from 3rd column on
```

- Selecting data by boolean indexing **ALWAYS** creates a copy of the data.
- The 'and' and 'or' keywords do NOT work with boolean arrays. Use & and |.

*** Fancy indexing** (aka "indexing using integer arrays')

Select a subset of rows in a particular order :

```
ndarray1[ [3, 8, 4] ]
ndarray1[ [-1, 6] ]
# negative indices select rows from the end
```

- Fancy indexing **ALWAYS** creates a copy of the data.

Numpy (Numerical Python)

Setting data with assignment :

```
ndarray1[ndarray1 < 0] = 0 *
```

- * If ndarray1 is two-dimensions, ndarray1 < 0 creates a two-dimensional boolean array.

COMMON OPERATIONS

1. **Transposing**
   - A special form of reshaping which returns a **'view'** on the underlying data without copying anything.

```
ndarray1.transpose()    or
ndarray1.T              or
ndarray1.swapaxes(0, 1)
```

2. **Vectorized wrappers (for functions that take scalar values)**
   - `math.sqrt()` works on only a `scalar`

```
np.sqrt(seq1) # any sequence (list, ndarray, etc) to return a ndarray
```

3. **Vectorized expressions**
   - `np.where(cond, x, y)` is a vectorized version of the expression 'x if condition else y'

```
np.where([True, False], [1, 2], [2, 3]) => ndarray (1, 3)
```

   - Common Usages :

```
np.where(matrixArray > 0, 1, -1)
=> a new array (same shape) of 1 or -1 values
```

```
np.where(cond, 1, 0).argmax() *
=> Find the first True element
```

- * `argmax()` can be used to find the index of the maximum element. Example usage is find the first element that has a "price > number" in an array of price data.

4. **Aggregations/Reductions Methods (i.e. mean, sum, std)**

Compute mean	`ndarray1.mean()` or `np.mean(ndarray1)`
Compute statistics over axis *	`ndarray1.mean(axis = 1)` `ndarray1.sum(axis = 0)`

- * axis = 0 means column axis, 1 is row axis.

5. **Boolean arrays methods**

Count # of 'Trues' in boolean array	`(ndarray1 > 0).sum()`
If at least one value is 'True'	`ndarray1.any()`
If all values are 'True'	`ndarray1.all()`

Note: These methods also work with non-boolean arrays, where non-zero elements evaluate to True.

6. **Sorting**

Inplace sorting	`ndarray1.sort()`
Return a sorted copy instead of inplace	`sorted1 = np.sort(ndarray1)`

7. **Set methods**

Return sorted unique values	`np.unique(ndarray1)`
Test membership of ndarray1 values in [2, 3, 6]	`resultBooleanArray = np.in1d(ndarray1, [2, 3, 6])`

- Other set methods : `intersect1d()`,`union1d()`, `setdiff1d()`,`setxor1d()`

8. **Random number generation** (np.random)
   - Supplements the built-in Python random * with functions for efficiently generating whole arrays of sample values from many kinds of probability distributions.

```
samples = np.random.normal(size =(3, 3))
```

- * Python built-in random ONLY samples one value at a time.

Created by Arianne Colton and Sean Chen
www.datasciencefree.com
Based on content from
'Python for Data Analysis' by Wes McKinney

Updated: August 18, 2016

Introdução ao Python

```
>>> a=1
>>> b=2
>>> a == b # == testa se a é igual a b
False
>>> a != b # != testa se a é diferente de b
True
>>> a <> b # <> também testa se a é diferente de b
True
>>> a > b # > testa se a é maior que b
False
```

Introdução ao Python

```
>>> 2**3 # É o mesmo que 2 elevado a 3 (dois ao cubo).
8
>>> 2**(3+6) # Dois elevado a 9
512
```

```
>>> valor1='Boa tarde!'
>>> valor1
'Boa tarde!'
```

```
>>> type(valor1)
<type 'str'>
```

Variáveis

```
>>> palavra='termodinâmica'
>>> palavra
'termodin\xe2mica'
```

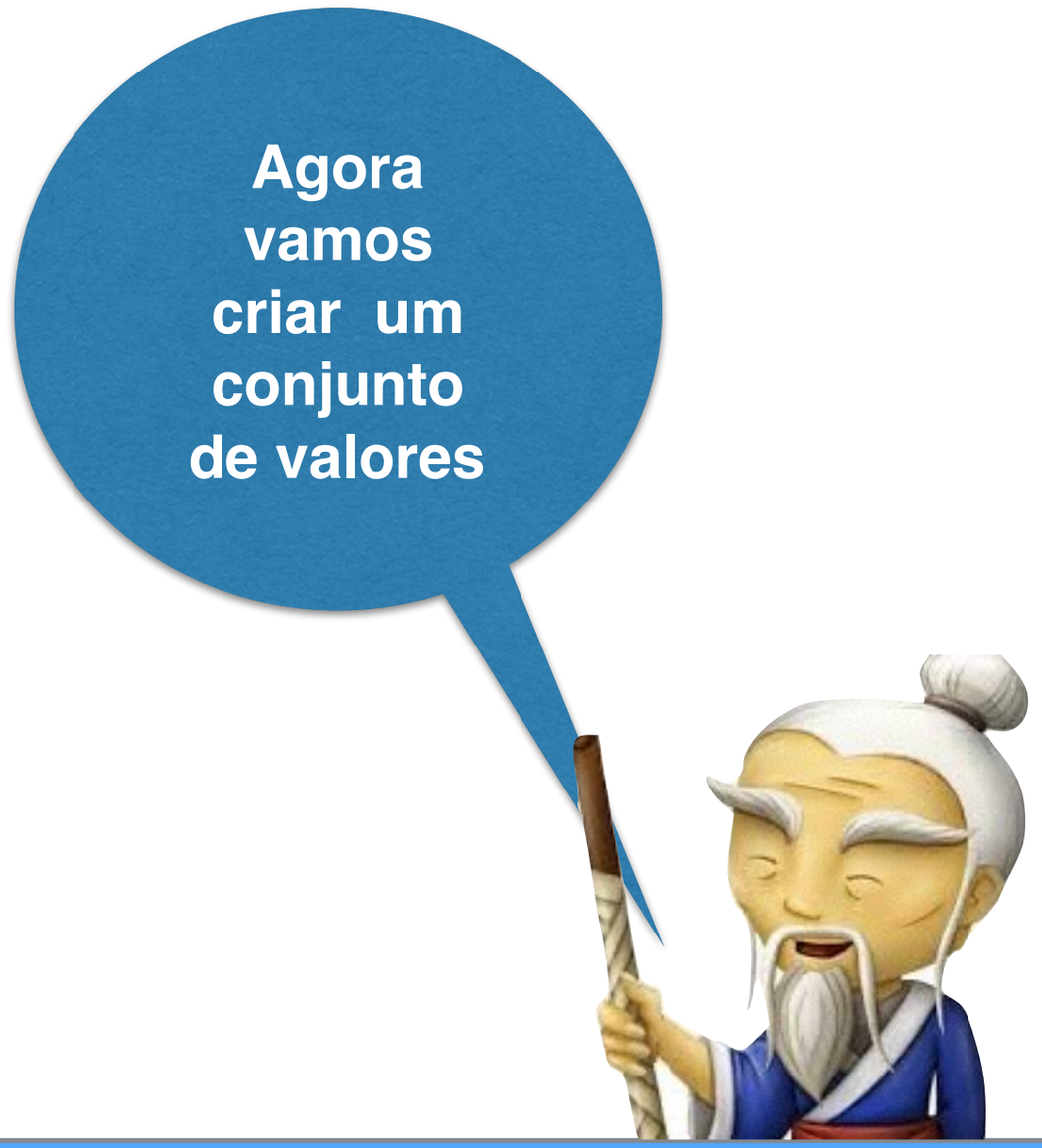
```
>>> print palavra
'termodinâmica'
```

Variáveis

```
>>> palavra[2]
'r'
>>> 2*palavra[0]
'tt'
```

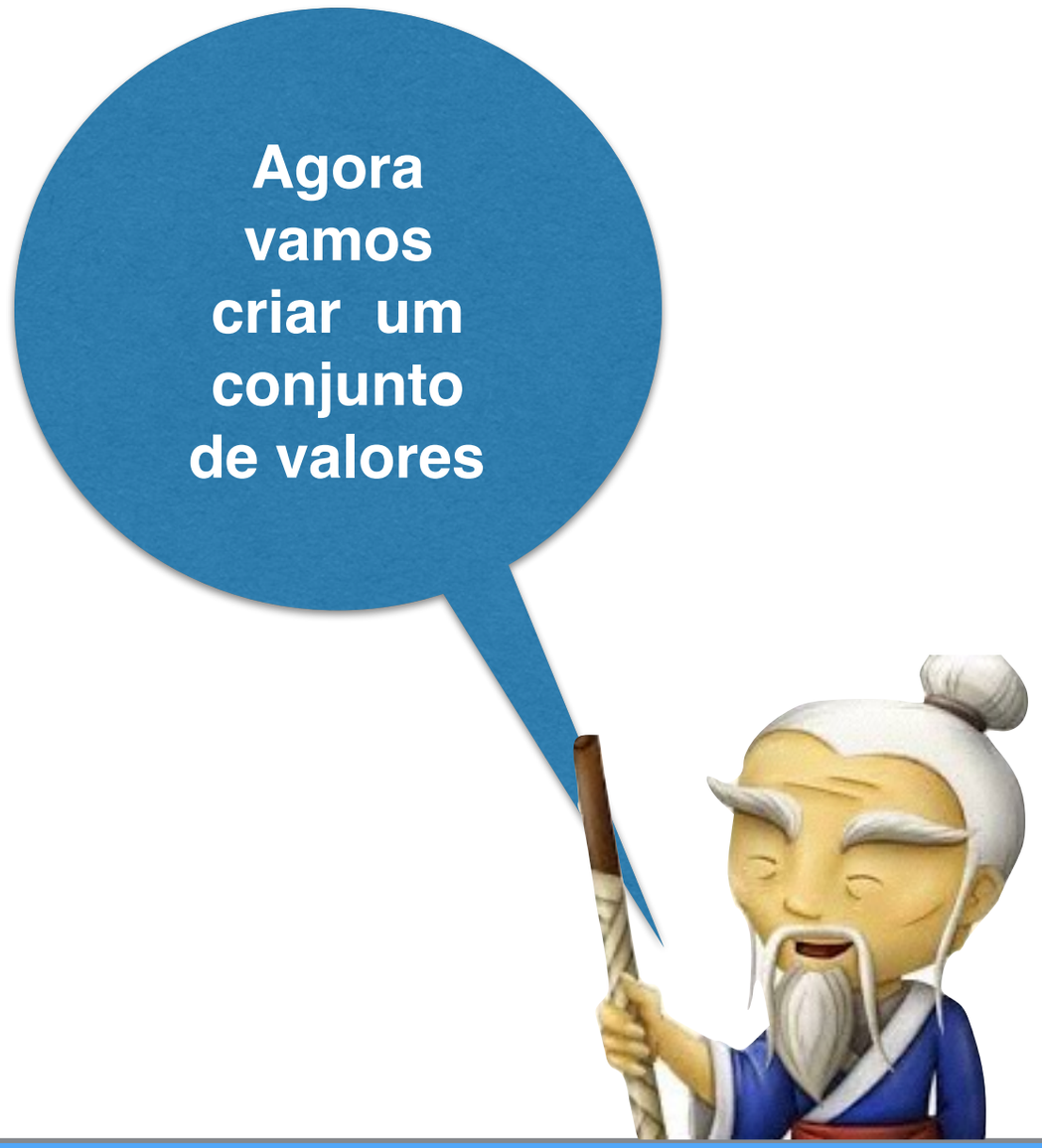
Listas

```
>>> lista=[1,2,3]
>>> lista
[1, 2, 3]
```



Listas

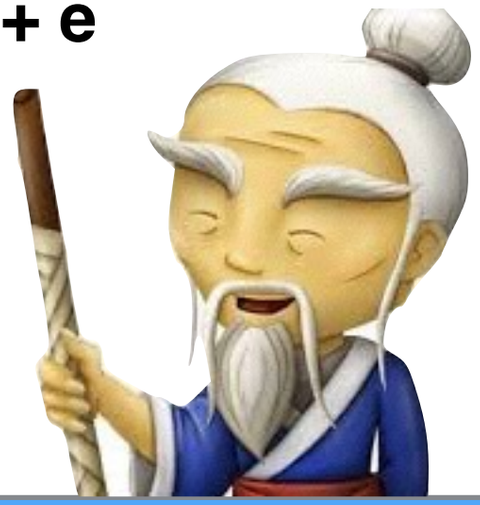
```
>>> lista[0]
1
>>> lista[0]+lista[2]
4
```



Introdução ao Numpy

NumPy é o pacote fundamental para a computação científica com o Python. Contém, entre outras coisas:

Um poderoso objeto de matriz N-dimensional
Funções sofisticadas (transmissão)
Ferramentas para integrar C / C ++ e código Fortran

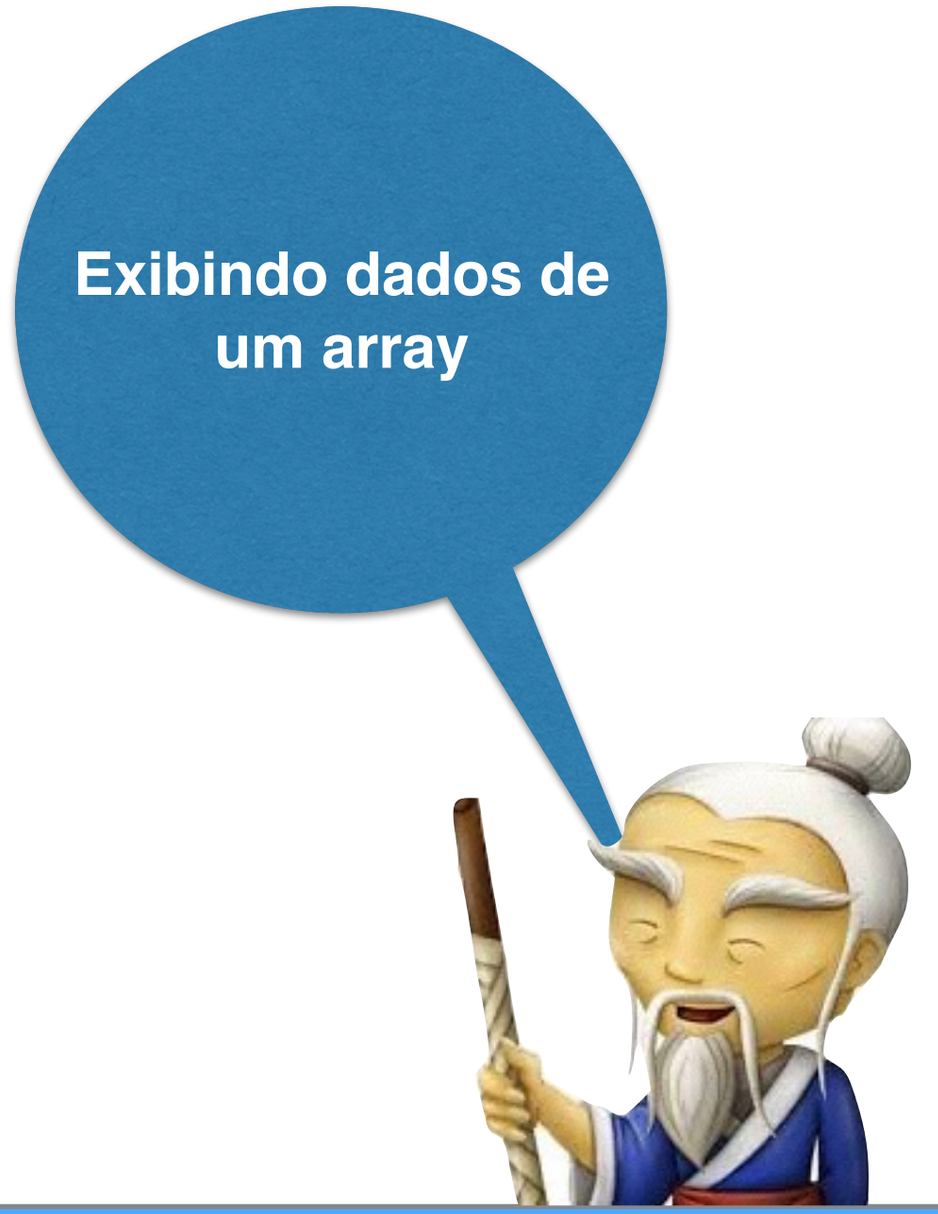


Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Arrays

```
>>> a[:2]
array([ 1., 4.])
>>> a[3]
8.0
>>> a[0] = 5.
>>> a
array([ 5., 4., 5., 8.])
```



Vídeo

https://youtu.be/cAICT4AI5Ow

Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1., 2., 3.],
 [ 4., 5., 6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Criando array do tipo float

Arrays

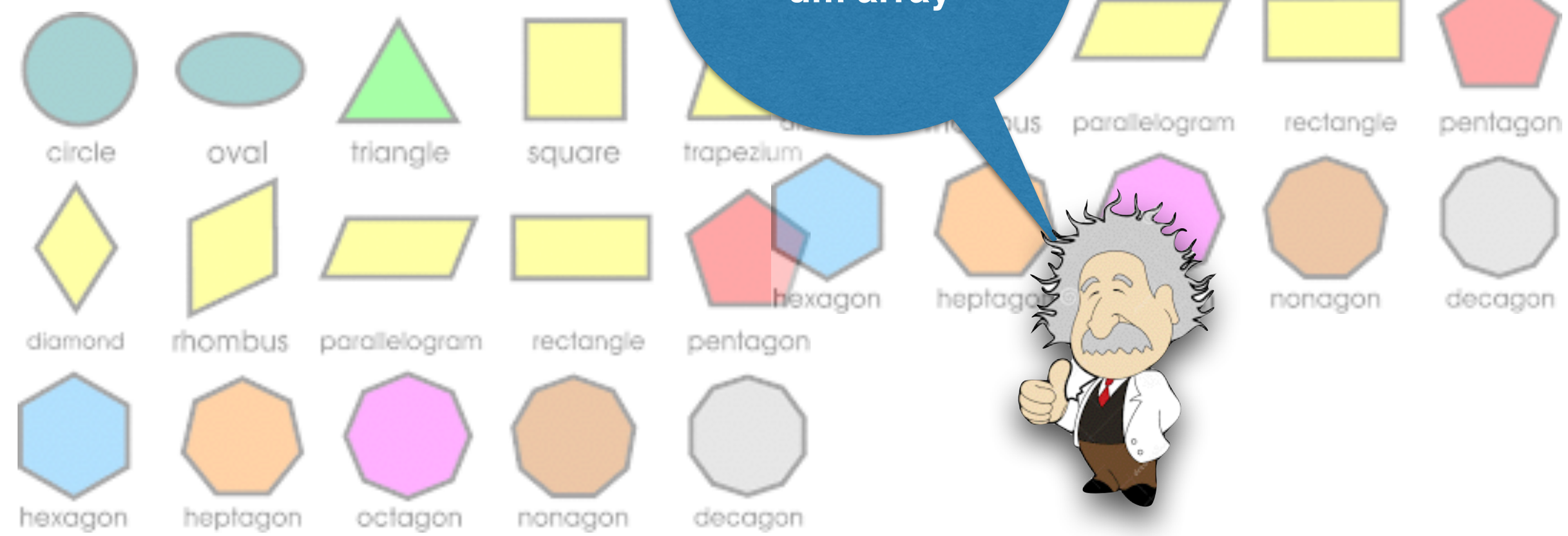
```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1., 2., 3.],
 [ 4., 5., 6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Criando array do tipo float

Shape de um array

```
>>> a.shape
(2, 3)
```

Imprimindo o Shape de um array

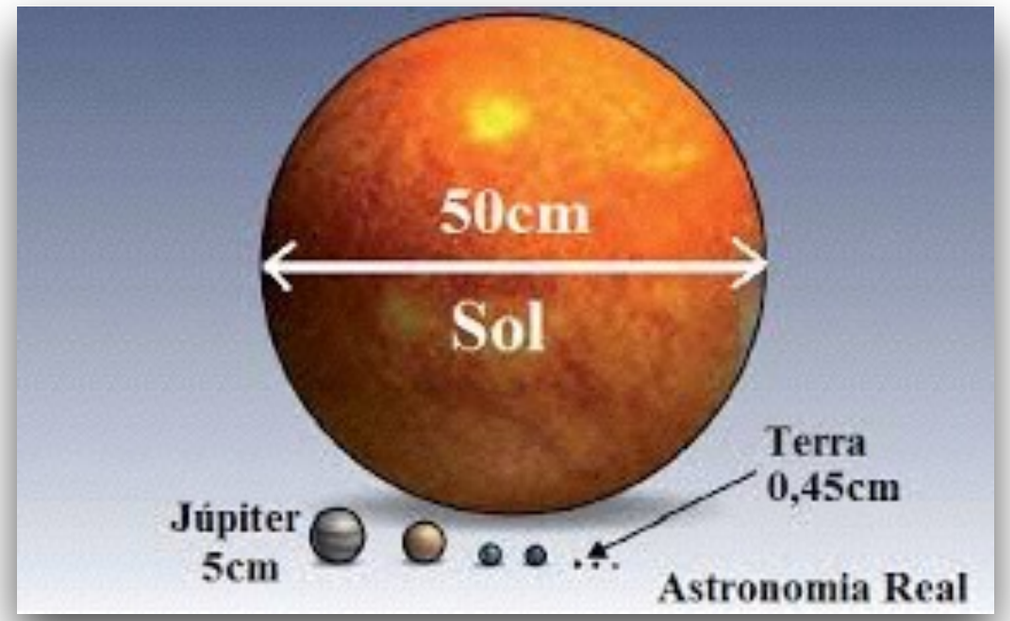


Tipo de um array

```
>>> a.dtype
dtype('float64')
```

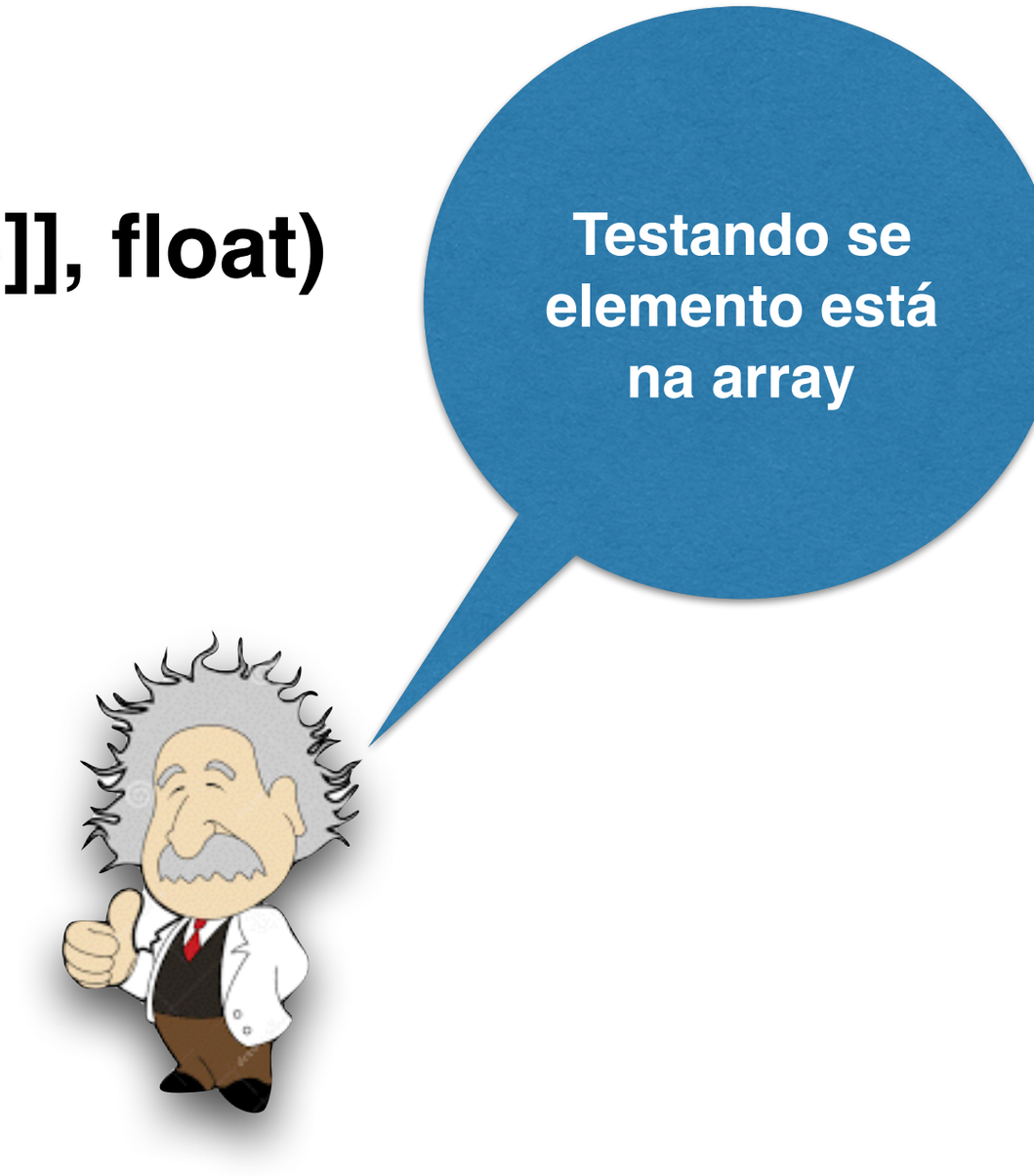
Tamanho de um array

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> len(a)
2
```



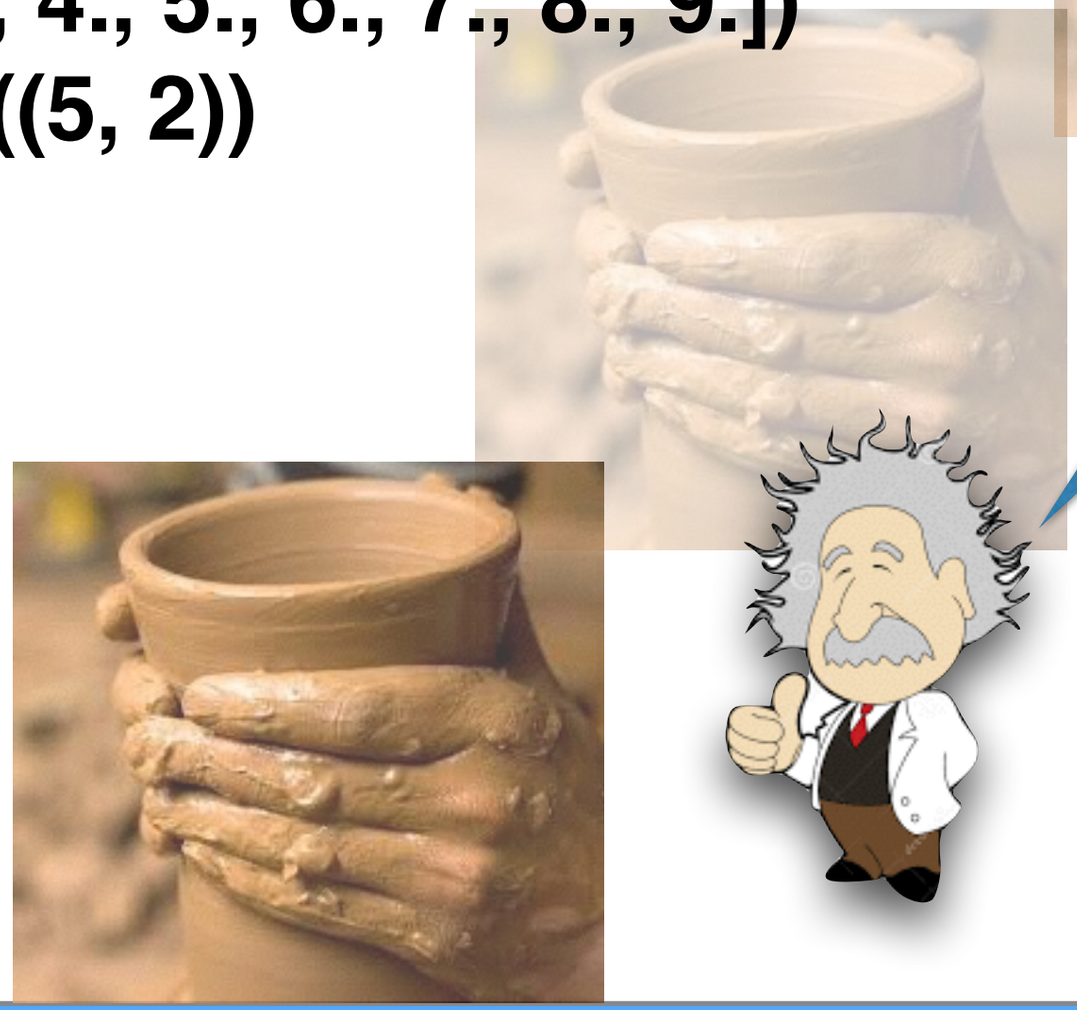
Testando se elemento está na array

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> 2 in a
True
>>> 0 in a
False
```



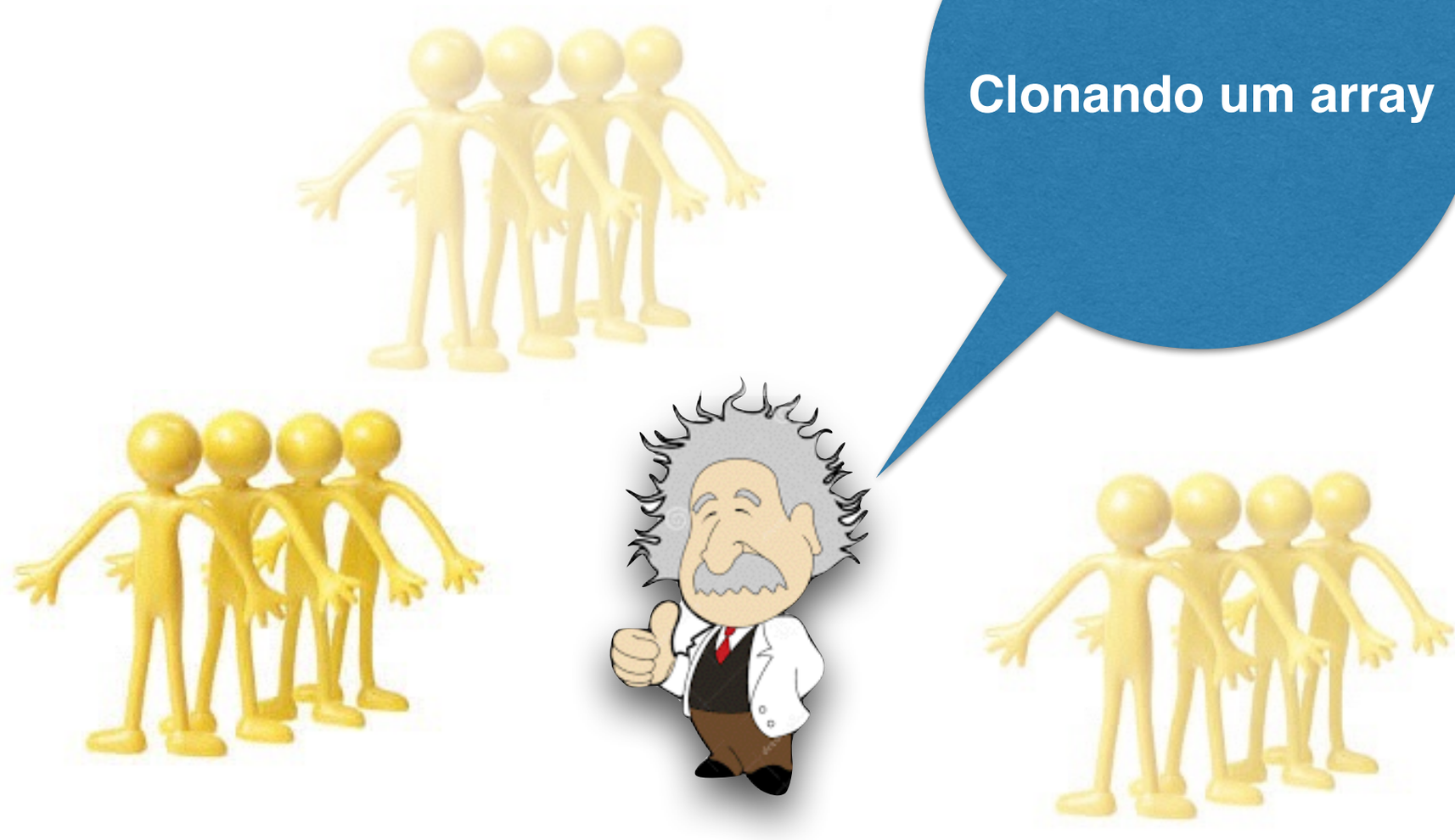
Reshaping

```
>>> a = np.array(range(10), float)
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> a = a.reshape((5, 2))
>>> a
array([[ 0., 1.],
 [ 2., 3.],
 [ 4., 5.],
 [ 6., 7.],
 [ 8., 9.]])
>>> a.shape
(5, 2)
```



Clonando array

```
>>> a = np.array([1, 2, 3], float)
>>> b = a
>>> c = a.copy()
>>> a[0] = 0
>>> a
array([0., 2., 3.])
>>> b
array([0., 2., 3.])
>>> c
array([1., 2., 3.])
```

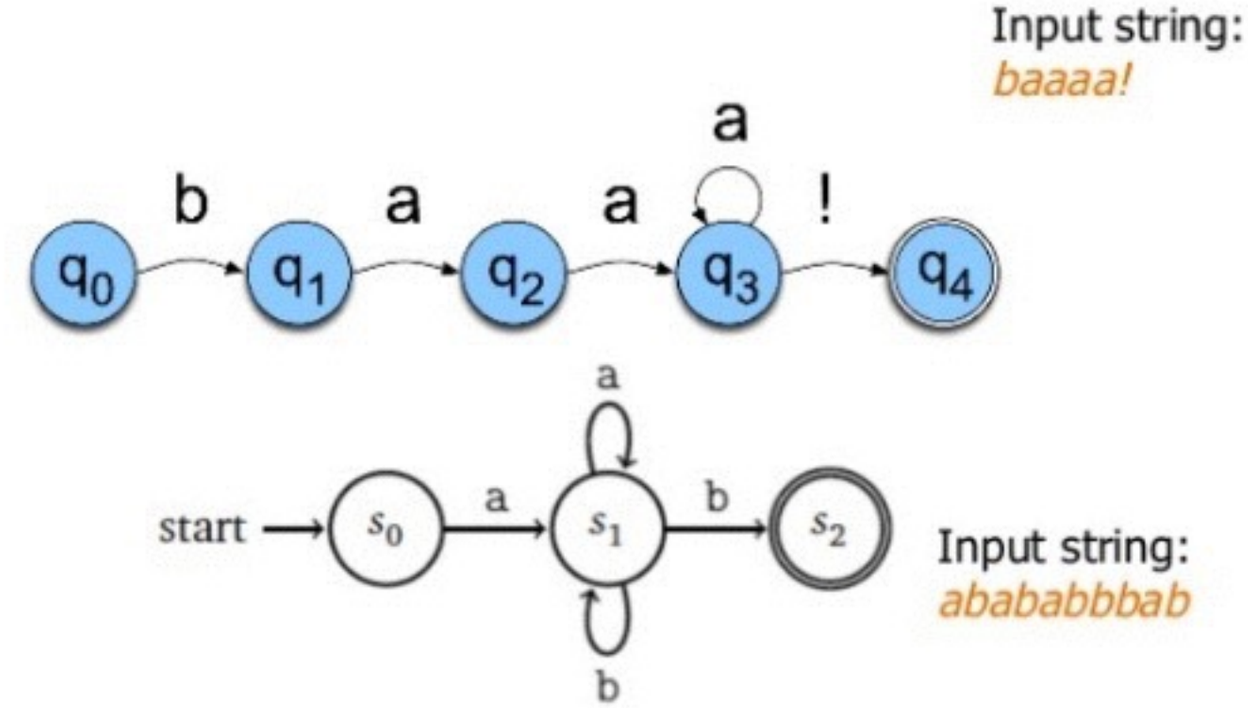


Transformando em Lista

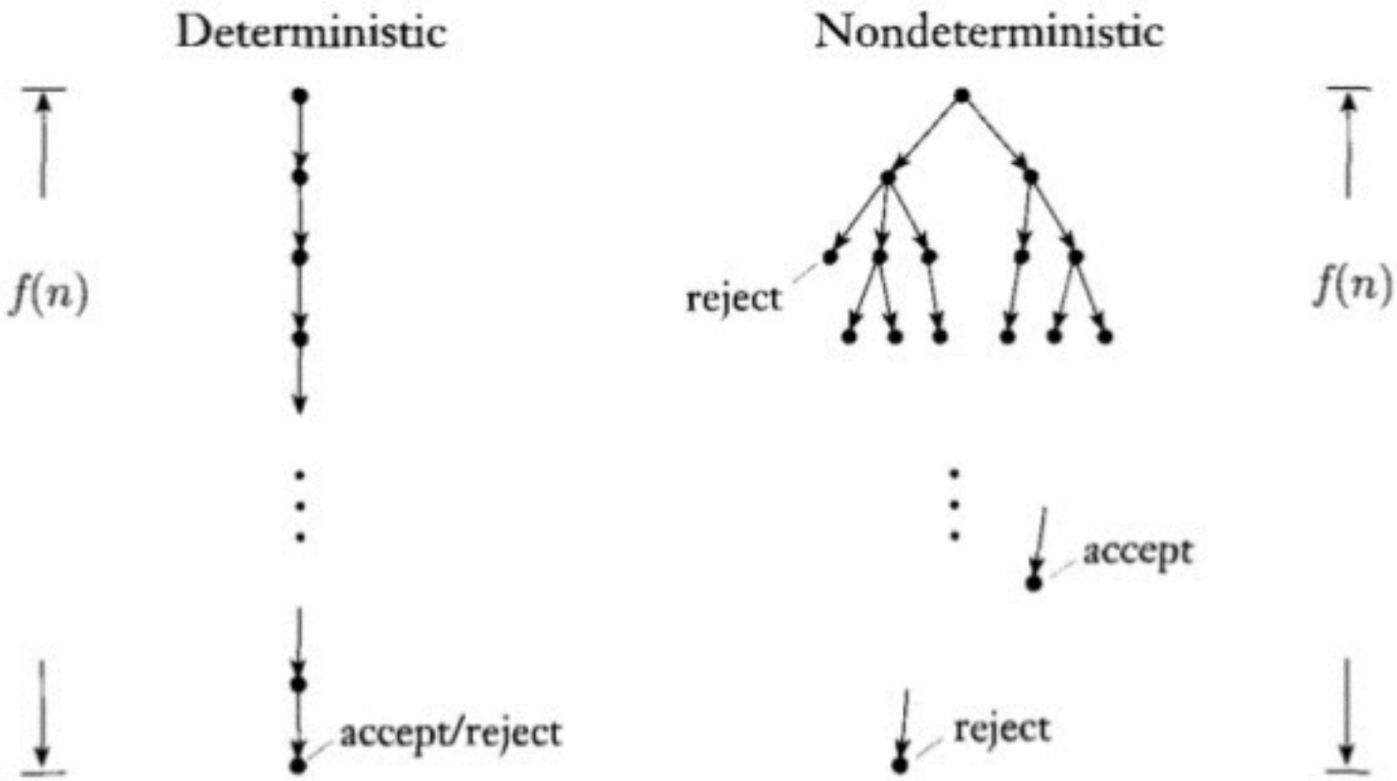
```
>>> a = np.array([1, 2, 3], float)
>>> a.tolist()
[1.0, 2.0, 3.0]
>>> list(a)
[1.0, 2.0, 3.0]
```

Problema Determinístico e não determinístico

Problema determinístico vs não deterministico



Problema Determinístico e não determinístico



Problema Determinístico e não determinístico

Tipos de Problemas

- **Para compreender o problema abordado neste trabalho, é necessário conhecer a classe de problemas NP-Completos. Para melhor compreender essa classe e também as classes P e NP, segue uma definição básica de cada uma delas.**

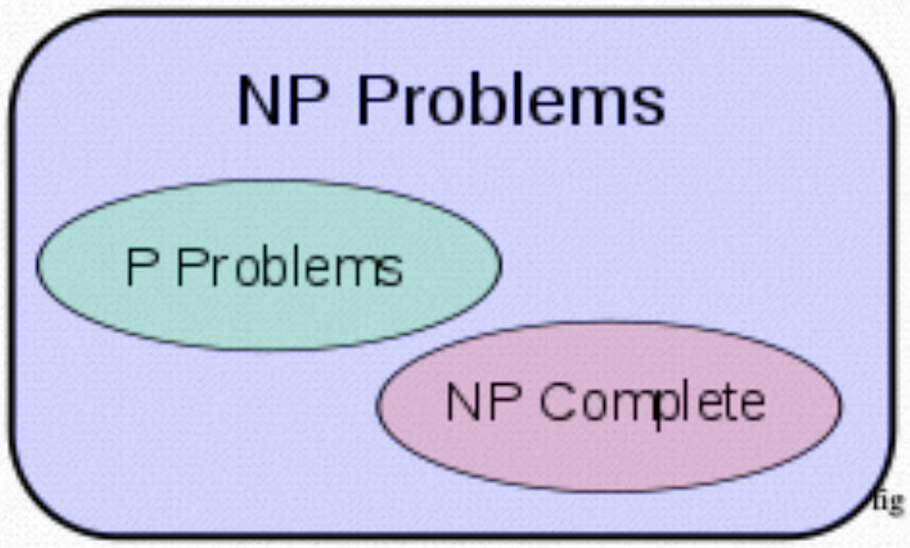
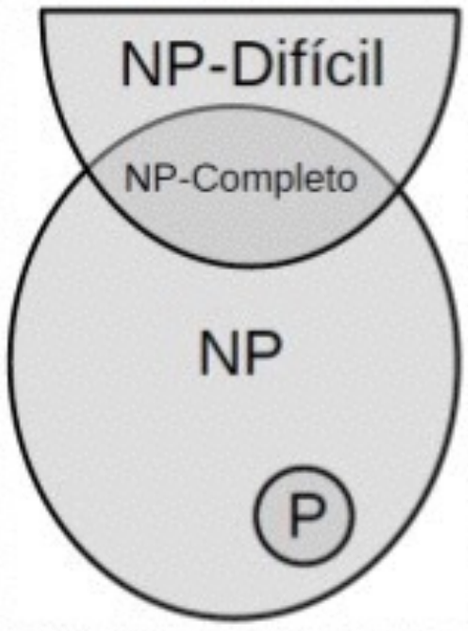


fig 1.



5

Tipos de Problemas

Problemas P, NP e NP-completo:

O problema P é um problema de decisão (ou seja, a resposta pode ser sim ou não), que possa ser resolvido em tempo polinomial.

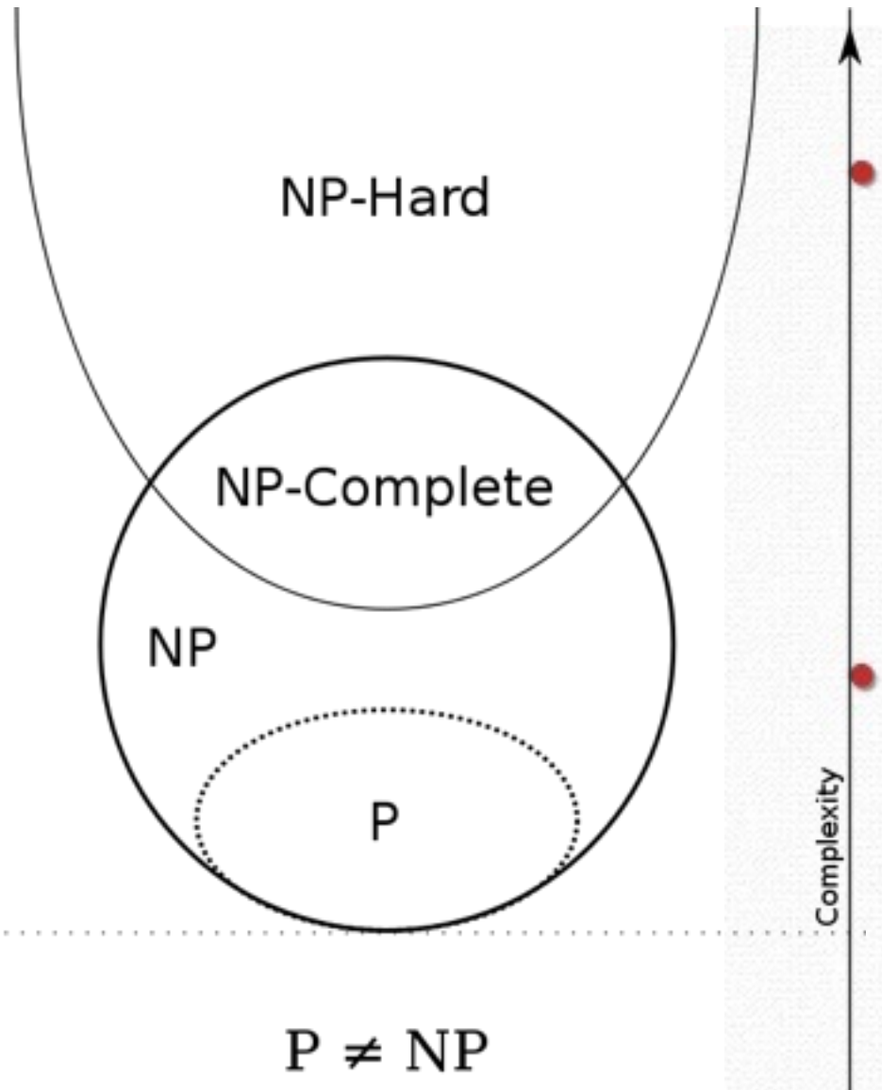
O problema NP (Non-Deterministic Polynomial time), ou seja "Tempo polinomial não-determinístico". O Problema NP pode ser um Problema P (na verdade o assunto é mais complexo), com a característica de "não determinístico", ou seja, ele pode ser provado em tempo polinomial.

Tipos de Problemas

2.1 A Classe P de problemas

- A classe P engloba todos os problemas decisórios que podem ser resolvidos por um algoritmo de tempo polinomial.
- Um problema cujo algoritmo é de ordem $O(n^k)$ com uma constante k muito grande é considerado eficiente, pois é provável, pelo que se vê ao longo da história dos algoritmos, que logo sejam encontrados novos algoritmos mais eficientes que resolvam tal problema com uma complexidade menor.

Tipos de Problemas



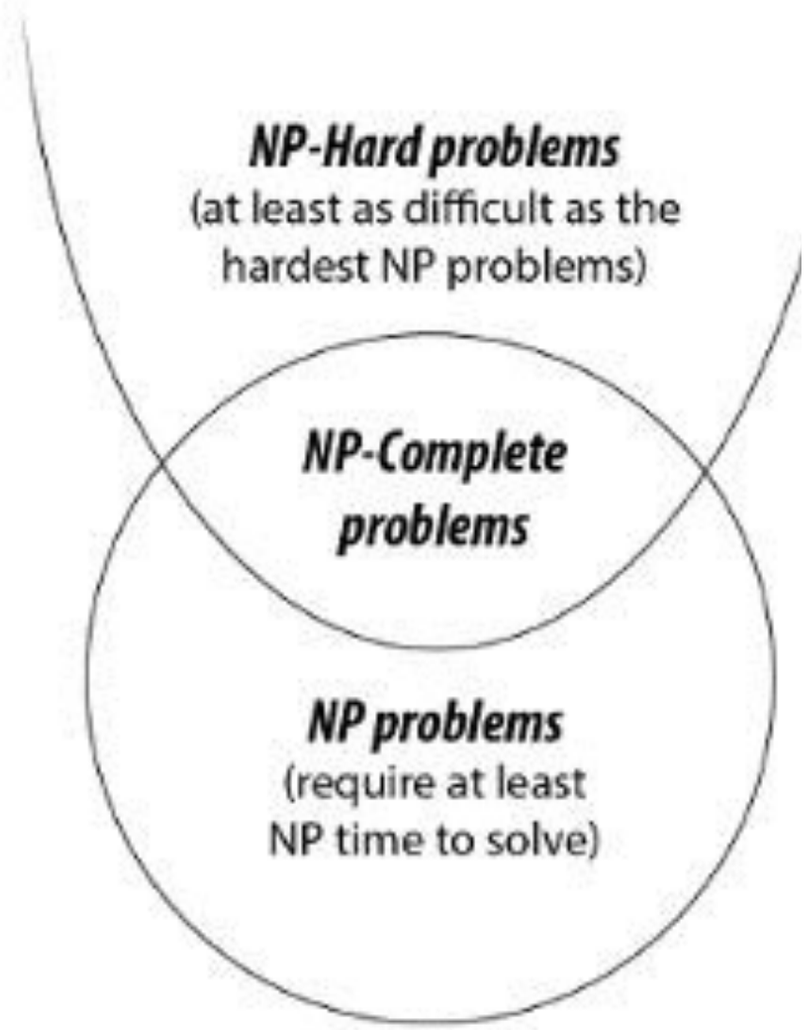
2.2 A Classe NP de problemas

- **A classe NP de problemas é o conjunto de todos os problemas de decisão para os quais existem verificadores polinomiais. Esta classe corresponde à classe dos problemas que poderíamos chamar "razoáveis".**

- **É importante lembrar que todo problema pertencente à classe P também pertence à classe NP, uma vez que se um problema pode ser resolvido em tempo polinomial, ele pode ser igualmente verificado em tempo polinomial.**

Complexity

Tipos de Problemas



Three classes of decision problems

- P is the set of decision problems that can be solved in polynomial time.3 Intuitively, P is the set of problems that can be solved quickly.

- NP is the set of decision problems with the following property: If the answer is YES, then there is a proof of this fact that can be checked in polynomial time.

- co-NP is the opposite of NP. If the answer to a problem in co-NP is NO, then there is a proof of this fact that can be checked in polynomial time.

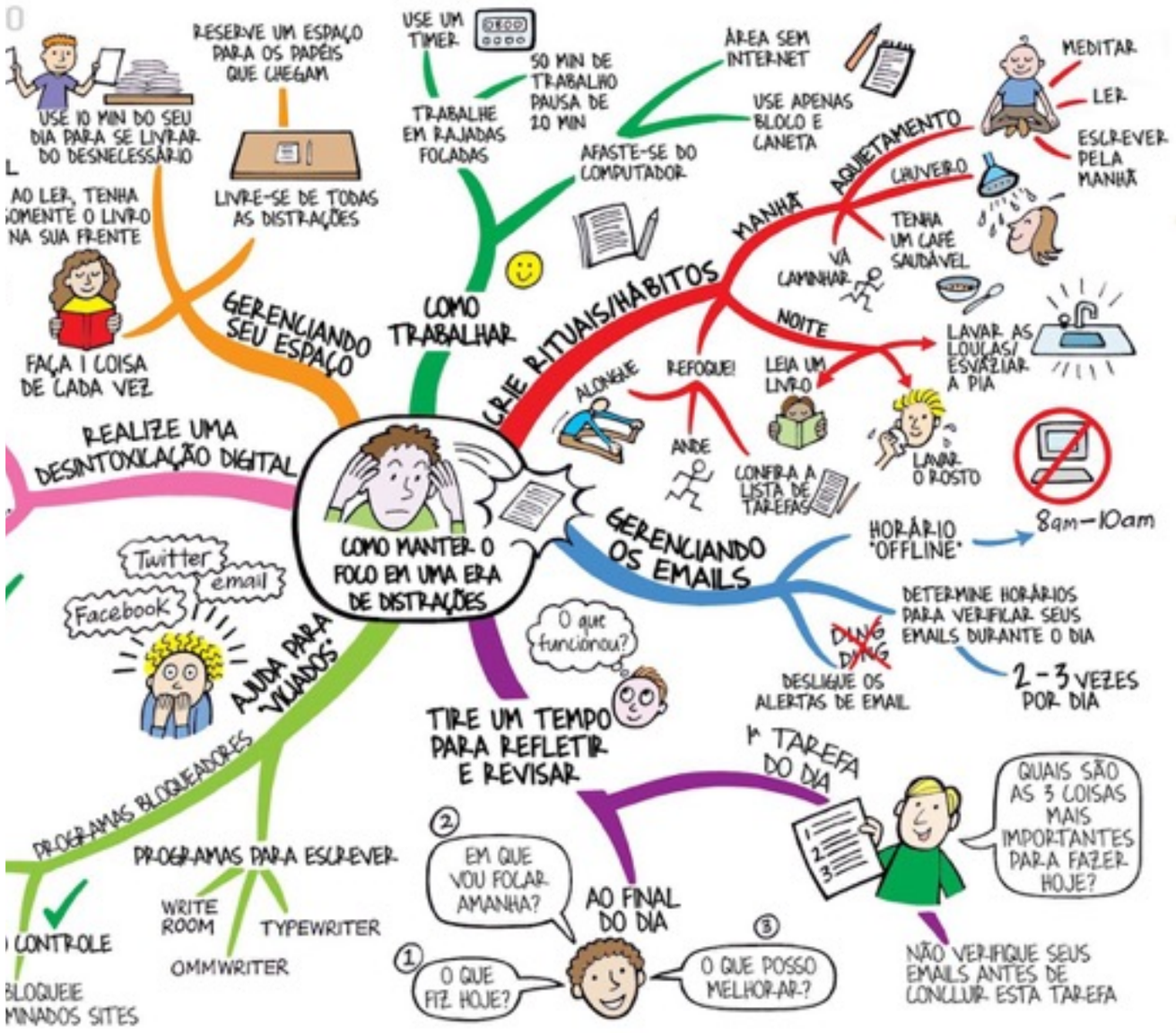
Tipos de Problemas

NP-Hard

- **What is NP-Hard?**
- *NP-Hard are problems that are at least as hard as the hardest problems in NP. Note that NP-Complete problems are also NP-hard. However not all NP-hard problems are NP (or even a decision problem), despite having 'NP' as a prefix. That is the NP in NP-hard does not mean 'non-deterministic polynomial time'. Yes this is confusing but its usage is entrenched and unlikely to change.*

Mapa Mental

Mapa mental, ou mapa da mente é o nome dado para um tipo de diagrama, sistematizado pelo psicólogo inglês Tony Buzan, voltado para a gestão de informações, de conhecimento e de capital intelectual



Exercício

Exercício 1

Separem-se em equipe de até 5 pessoas e criem um mapa mental descrevendo o que foi feito na aula (10 minutos).

Francisco Nauber Bernardo Gois
Email: naubergois@gmail.com