# Big Data & Data Science

## Introdução à Linguagem Python

# *Data Scientist*

O cientista de dados, por Josh Wills

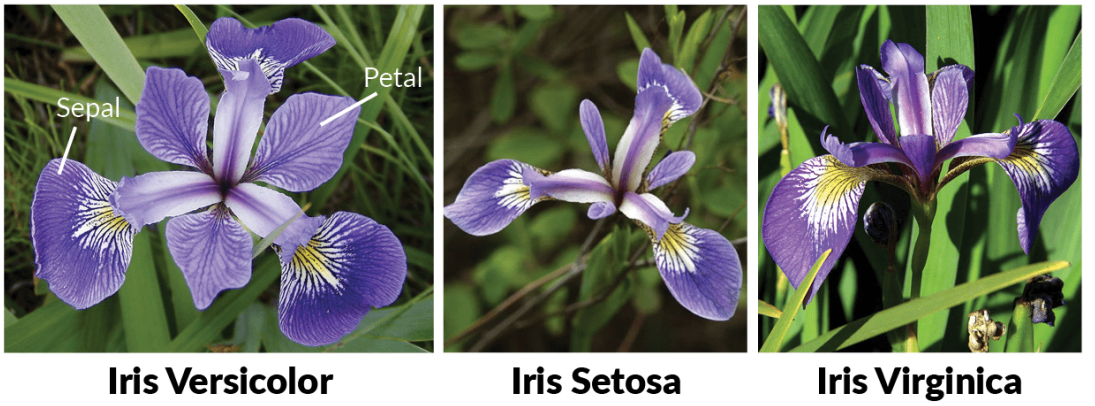
(fundador: Apache Crunch, Cloudera ML)

*“Person who is better at statistics than any software engineer and better at software engineering than any statistician”*

https://twitter.com/josh_wills/status/19809351214995865 6

# Motivação

5.0,3.5,1.6,0.6,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
...
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
...
6.3,3.3,6.0,2.5,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica




*https://www.datacamp.com/community/tutorials/machine-learning-in-r*

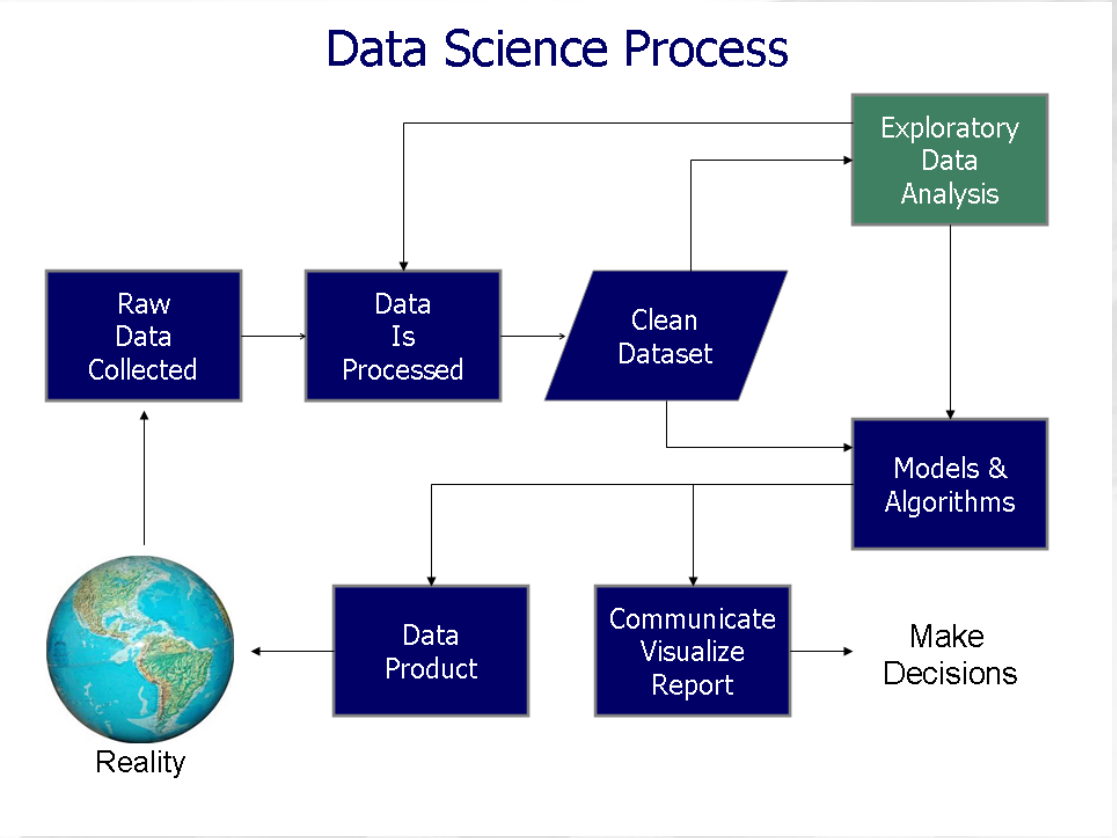
*Fonte: (http://archive.ics.uci.edu/ml/datasets/Iris)*

# Motivação

```
1. Title: Iris Plants Database
      Updated Sept 21 by C.Blake - Added discrepency information
2. Sources:
     (a) Creator: R.A. Fisher
     (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
     (c) Date: July, 1988
…
5. Number of Instances: 150 (50 in each of three classes)

6. Number of Attributes: 4 numeric, predictive attributes and the class

7. Attribute Information:
   1. sepal length in cm
   2. sepal width in cm
   3. petal length in cm
   4. petal width in cm
   5. class:
      -- Iris Setosa
      -- Iris Versicolour
      -- Iris Virginica

8. Missing Attribute Values: None

Summary Statistics:
                Min  Max   Mean    SD   Class Correlation
   sepal length: 4.3  7.9   5.84  0.83    0.7826
    sepal width: 2.0  4.4   3.05  0.43   -0.4194
   petal length: 1.0  6.9   3.76  1.76    0.9490  (high!)
    petal width: 0.1  2.5   1.20  0.76    0.9565  (high!)
```

*http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.names*

# Motivação




Data science process flowchart from "Doing Data Science", Cathy O'Neil and Rachel Schutt, 2013
https://en.wikipedia.org/wiki/Data_analysis

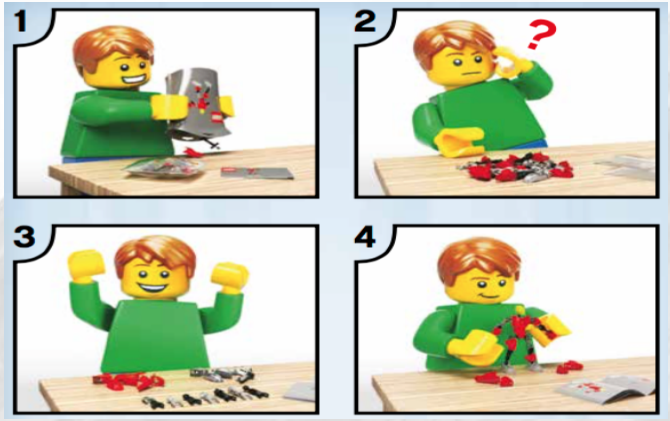
# Motivação

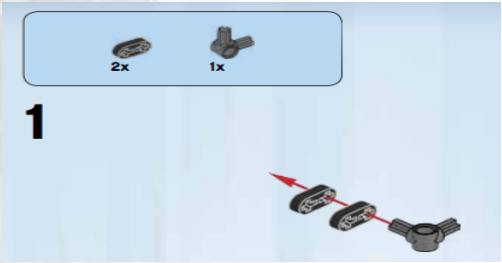
# Motivação

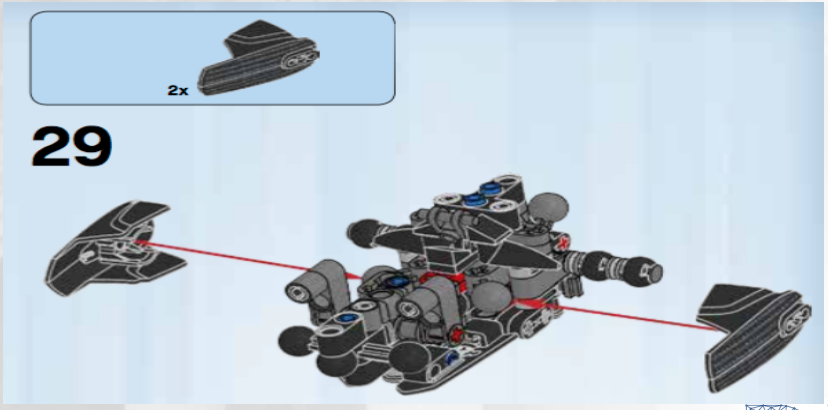
- Como cozinhar macarrão *al dente*?

# Motivação


LEGO
STAR WARS
75111
BUILDABLE FIGURES
LEGO.com/starwars
Disney


1
2
3
4


2x
1x
1


2x
29

# Motivação

Entrada


Saída

# Motivação

- Algoritmos estão presentes no dia-a-dia:
  - Bula de remédios (posologia, instr. de uso)
  - Configuração de dispositivos eletrônicos
  - Receitas culinárias
- Objetivo:
  - Estabelecer um conjunto de passos que possa ser reproduzido (facilmente?)
  - Ex.: troca de pneus

# Revisão

Programa:

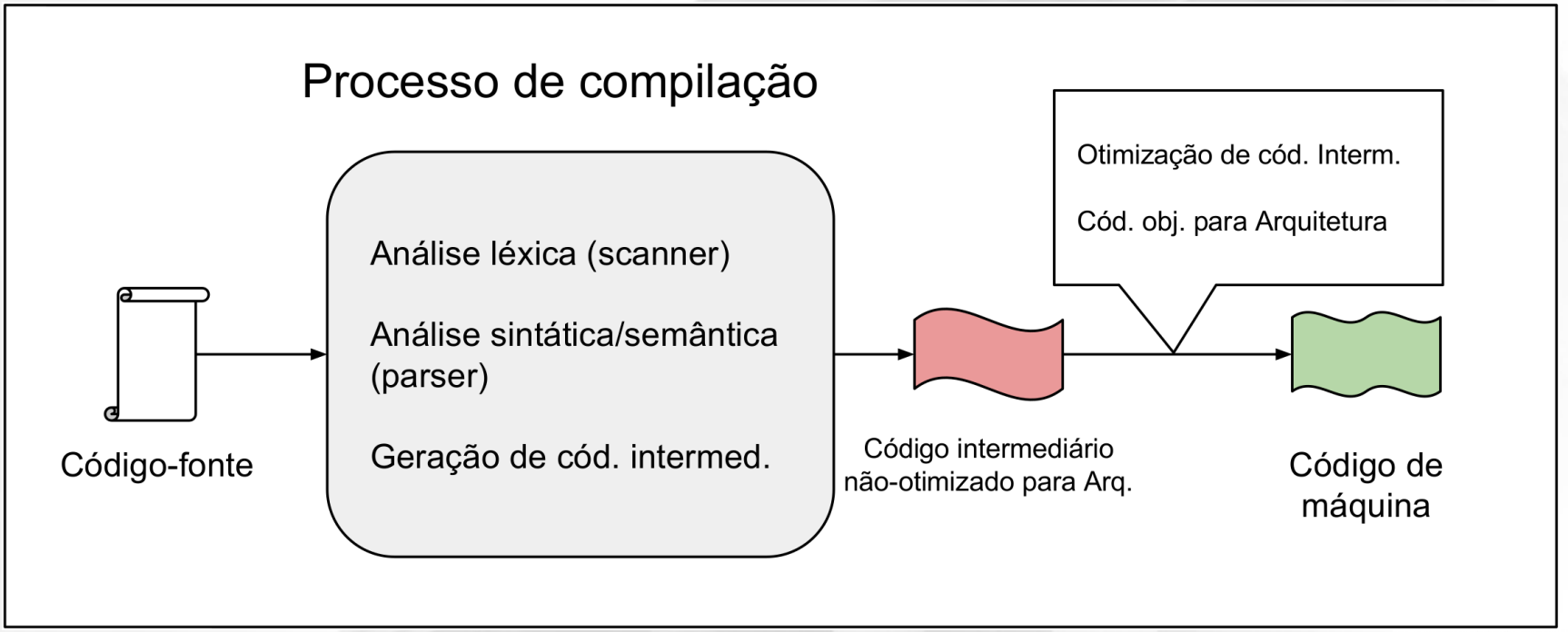
- Sequência de instruções para resolver um dado problema com:
  - ENTRADA
  - ALGORITMO
  - SAÍDA

# Do programador para a máquina

1. Projeto da solução/algoritmo
2. Escolha da linguagem
3. Programação (em alto nível)
4. Geração de código de máquina
5. Execução do programa

Simples?

# Processo de compilação

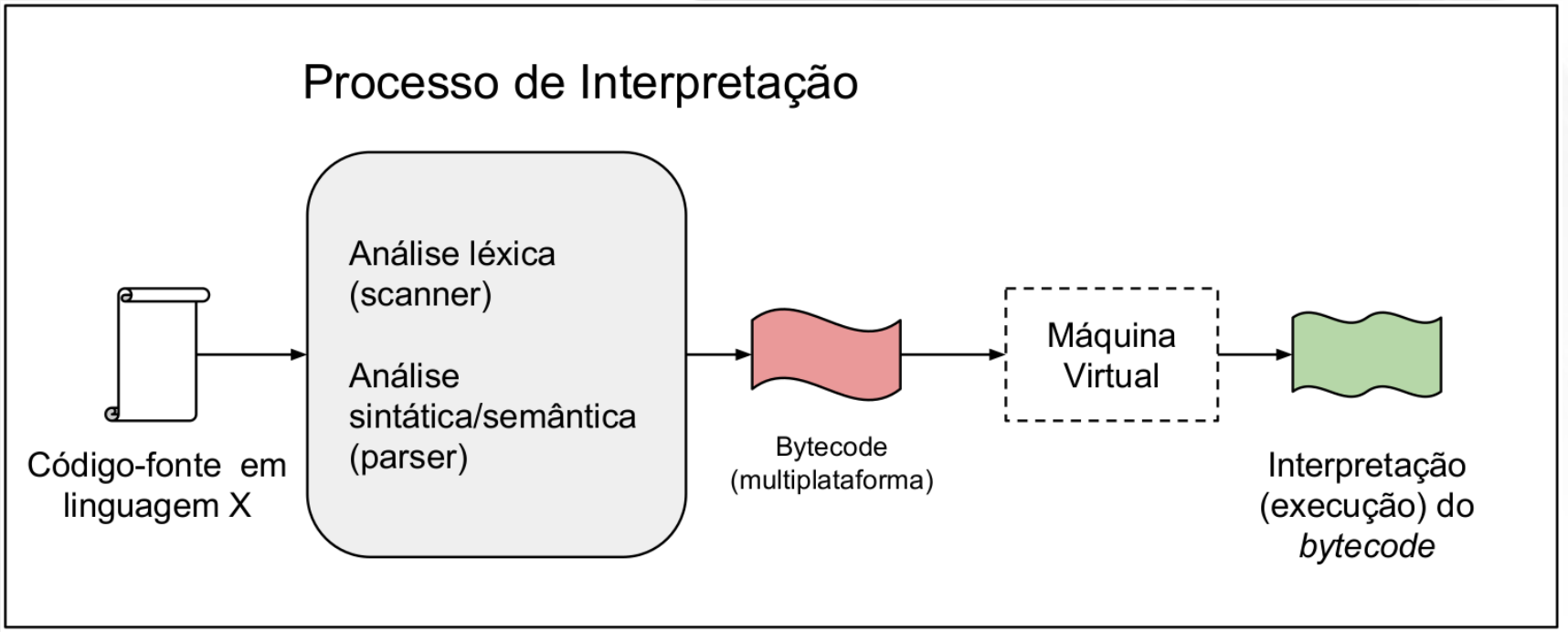


# Compilador x Interpretador

Ambos atuam como “tradutores”

| Compilador | Interpretador |
|---|---|
| Recebe o programa inteiro como ENTRADA | Recebe uma instrução como ENTRADA |
| Gera código intermediário | Não gera código intermediário |
| Condicionais executam mais rápido | Condicionais executam mais lentamente |
| Requer + memório (para o código objeto) | Requer – memória |
| Erros no final da checagem do programa inteiro | Erros mostrados para cada instrução interpretada |
| Executável criado independente | Interpretador necessário para o programa rodar |

# Processo de interpretação



# Bytecode?

- Um *conjunto de instruções* projetado para execução eficiente por um interpretador (Wikipedia)

- Representação interna de um programa no compilador
  - Códigos numéricos compactos
  - Constantes
  - Referências (endereços)

# Bytecode!

```
4           0 LOAD_CONST               1 ('Boa noite!')
            3 PRINT_ITEM
            4 PRINT_NEWLINE
            5 LOAD_CONST               0 (None)
            8 RETURN_VALUE
```

Colunas:
1. linha original do código-fonte
2. endereço da intrução do bytecode
3. nome da instrução
4. índice do argumento no nome do bloco e tabela de constante
5. mapeamento do índice de argumento “legível”

# A linguagem Python

- Linguagem de programação de altíssimo nível
- Desenvolvida nos anos 90 por Guido van Rossum
- Objetivo: escrever pequenos programas (ou *scripts*) que não precisavam rodar em velocidade “ótima” para executar tarefas repetitivas...

# Interpretador Python

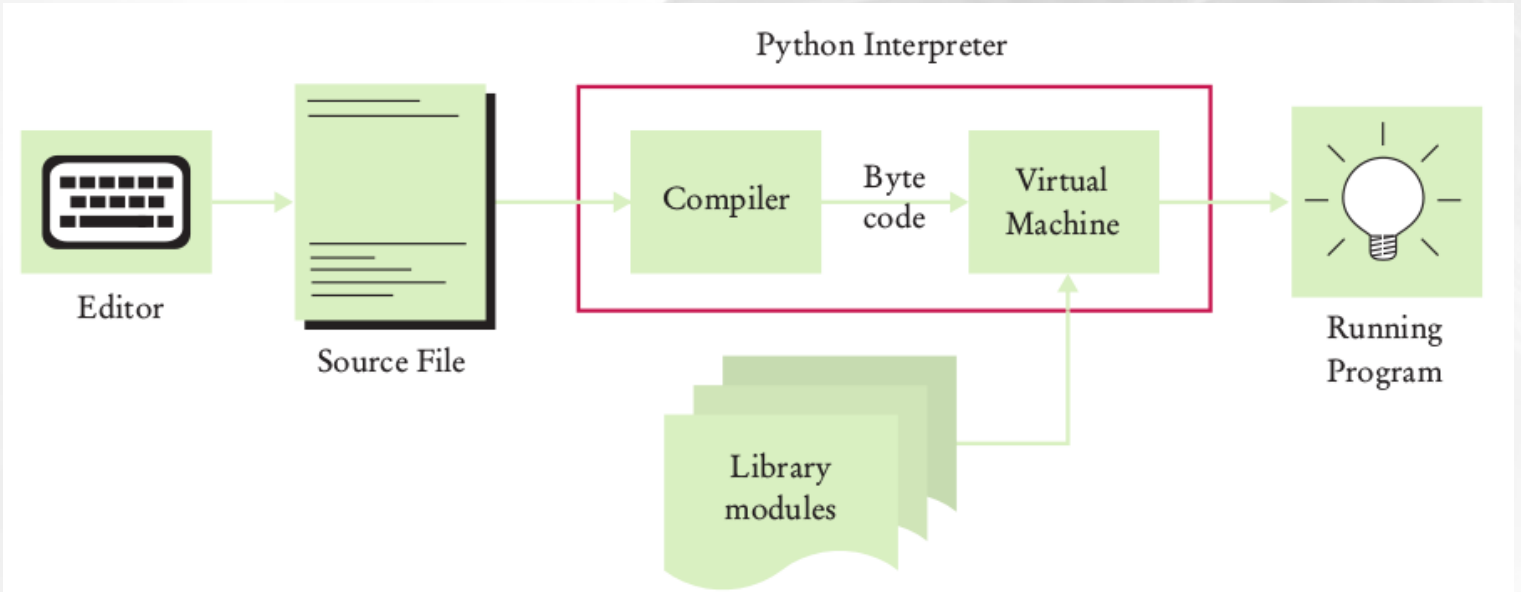



**Figure 7** From Source Code to Running Program

C. Horstmann, R. D. Necaise. *Python for Everyone*. Wiley, 2014

# Interpretador Python

Python possui dois *forks*: Python 2 e Python 3

```
$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# Interpretador Python

Uma instrução simples:

- $ python
  - Python 2.7.12 (default, Nov 20 2017, 18:23:56)
  - >>> **print "oi**"
  - oi
- $ python3
  - Python 3.5.2 (default, Nov 23 2017, 16:37:01)
  - >>> **print("oi")**
  - oi

# Operadores Aritméticos

Operadores são símbolos especiais para realizar cálculos

- +: adição
- -: subtração
- /: divisão
- *: multiplicação
- **: exponenciação

# Variáveis

Identificadores para os quais são atribuídos valores de **tipos distintos** e com os quais se podem fazer operações

```
>>> a = 5
>>> b = 3
>>> a+b
8
>>> a**b
125
>>> a/b
1.6666666666666667
```

# Tipos de valores/variáveis

Tipo é a classe a qual a variável pertence:

- A = 5, B = 3
  - Integer
- “oi”, C = ‘5’
  - String
- N = A/B = 1.666…
  - Float

```
>>> type(a)
    <class 'int'>

>>> type(c)
    <class 'str'>

>>> type("oi")
    <class 'str'>

>>> type(a/b)
    <class 'float'>
```

# Meu primeiro .py

- Em um editor de texto qualquer, abra um arquivo novo e o nomeie com a extensão do Python, por exemplo “teste.py”:

```
1)# meu primeiro programa em Python
2)print(“oi”)
```

- Comentários precedidos por “#”
- Execução na linha de comando: `python teste.py`

# Função básica - print()

O comando *print( )*:

- Sintaxe:
  - print(valor1, valor2, …, valorN)
  - Se nenhum argumento é passado, imprime uma linha em branco
  - Os valores a serem impressos são separados por um espaço:
    - print(“Resposta:”, 6 * 7, “!!!”)

# Variáveis

- São nomes (identificadores) que referenciam algum valor (tipo de dado a ser armazenado)
- O processo de “guardar” um valor em uma variável é chamado de **atribuição**

```
>>> resposta = 42
>>> pi = 3.1415
>>> nome
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'nome' is not defined
>>> nome = "Andre"
>>> nome
'Andre'
>>> resposta
42
>>> pi
3.1415
```

# Nomes de variáveis

- De preferência, devem significar algo
- Podem ser longos
- Podem conter letras, números e “_”
- **NÃO** podem começar com números
- **NÃO** podem ter certos caracteres especiais, como @, &
- **NÃO** podem ser palavras reservadas *(keywords)*

# Nomes de variáveis

Palavras reservadas:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

```
>>> 1var = "eh uma string"
SyntaxError: invalid syntax

>>> def = 1
SyntaxError: invalid syntax

>>> uma_var = "eh uma string"
>>> _var = 1

>>> a@ = 1
  File "<stdin>", line 1
    a@ = 1
      ^
SyntaxError: invalid syntax
```

# Estilo de codificação

Python é extremamente dependente de **IDENTAÇÃO**

- Funções e blocos **NÃO** usam chaves…
- Comandos e funções requerem “:” ao final (veremos depois)
- Pode-se usar <TAB> ou espaços para criar níveis:
  - Separação de funções, blocos ou aninhamentos
- Comentários:
  - Linha → # um comentário
  - Múltiplas linhas → **'''** texto em várias linhas **'''**

# Condicionais básicos

- if (*condição*):
  - Instrução 1
  - Instrução 2
- elif (*outra condição*):
  - Instrução 3
- else:
  - Instrução 4
- Instrução 5 (fora do IF-ELIF-ELSE)

# Laços básicos

- for i in range(5):
  - print(“%d “ %i)

    0

    1

    2

    3

    4

- while True:
  - print(“laço infinito!”)

# Próxima aula...

Mão na massa no laboratório!

- Instalação do *framework* para programação de aplicações científicas “Anaconda”
- Uso do interpretador Python para programas básicos
- Uso de editor de texto (vi, emacs, spyder) para codificação de programas em Python