

Python, o que é e porque usar

Gustavo Fernandes dos Santos
gfdsantos@inf.ufpel.edu.br

27 de novembro de 2016

UFPEL - Universidade Federal de Pelotas

- Programador Python há 6 anos
- Pregador do software livre há 4 anos
- Curioso por natureza
- Estudante de Engenharia de Computação

Índice

Historinha do Python

Ferramentas Interessantes

A Linguagem

Paradigmas de Programação e Python 3.x

Estruturas de Dados em Python

Pandas

Hydra - Python Multitarefa

Coisas Legais no Python

Adicionando cache aos programas

Compilando programas em Python

Aprenda Mais!

Historinha do Python

O que é Python?

Python é uma linguagem de programação de propósito geral. É uma linguagem de script, logo obedece às regras de linguagens de script. Python é, muitas vezes, definida como uma **linguagem de script orientada a objetos**. A definição anterior acaba limitando o escopo atual do Python, que pode ser definido mais precisamente como uma **linguagem de programação de propósito geral, que suporta os paradigmas de programação procedural, funcional e orientado a objetos**.

Porque as Pessoas Usam Python?

Esta é uma das primeiras questões que aparecem quando as pessoas pensam em usar Python. Existem tantas linguagens de programação por aí hoje que escolher uma linguagem específica para um determinado projeto torna-se uma tarefa difícil.

Mas não há regras. É uma questão pessoal. Mas há uma série de fatores que acabam chamando a atenção dos desenvolvedores para a linguagem da cobrinha.

Porque as Pessoas Usam Python?

- **Produtividade.** Python é uma linguagem que proporciona que o mesmo que seria feito em alguma linguagem compilada e estaticamente tipada faria, mas com muito mais rapidez, reduzindo drasticamente o tempo de desenvolvimento do projeto. O código em Python é geralmente 3 ou 5 vezes menor que o código equivalente em C++. Programas em Python rodam imediatamente após serem escritos, não necessitando de nenhum processo de compilação, ligação ou montagem.
- **Portabilidade.** A maioria dos programas escritos em Python rodam na maioria dos sistemas operacionais para computadores. Um código pode ser totalmente portátil entre Linux, Mac e Windows, se não utilizar ferramentas específicas destes sistemas operacionais. Python é interpretado, logo um sistema apenas necessita implementar o interpretador para rodar o código em Python. Existem também diversas bibliotecas de interface de usuário portáteis. Característica que permite que um programa tenha a mesma interface independente de onde seja executado.

Porque as pessoas usam Python?

- **Suporte.** Python vem por padrão com uma coleção incrível de baterias. Contudo há uma infinidade de bibliotecas para as distribuições de Python, como bibliotecas de deep learning - *Tensorflow*, *Theano* e etc; visão computacional, tendo como destaque a biblioteca *OpenCV*. Python também também tem bibliotecas para computação numérica, como o *NumPy*, análise de dados com o *Pandas*, desenvolvimento web com *Django*, *Flask* e muitas outras.
- **Integração.** Python consegue facilmente se integrar com partes de alguma outra aplicação, como invocar códigos C e C++ e ser invocado por programas escritos em C ou C++. Pode se integrar com componentes Java e .NET. Python não é uma ferramenta singular, Python pode conviver facilmente com outros ecossistemas.

Existe alguma parte ruim?

Python é uma linguagem de script, é interpretada. O fato de Python ser interpretado implica que nem sempre os programas em Python irão rodar com a mesma velocidade que programas equivalentes escritos em C/C++. Isto é relativamente raro hoje, mas ainda acontece.

A implementação em Python padrão hoje (3.5.2) traduz o código fonte para um código intermediário. O código intermediário - bytecode - é interpretado pela máquina virtual. Como o código em bytecode é interpretado, em vez de rodar diretamente no processador, há perdas de performance se comparados com código objeto gerado pelo gcc, por exemplo.

```
In [1]: def mult(x, y):  
...:     z = x * y  
...:     return z  
...:
```

```
In [2]: from dis import dis
```

```
In [3]: dis(mult)
```

2	0	LOAD_FAST	0	(x)
	3	LOAD_FAST	1	(y)
	6	BINARY_MULTIPLY		
	7	STORE_FAST	2	(z)
3	10	LOAD_FAST	2	(z)
	13	RETURN_VALUE		

Existe alguma parte ruim?

O tempo de execução sempre irá depender do quanto você está disposto a abrir mão em prol das outras características boas do Python. No fim das contas, você irá decidir se vale mais a pena abrir mão de alguns milissegundos do tempo de execução ou dispende mais tempo com o desenvolvimento e manutenção da aplicação em alguma outra linguagem compilada, como C/C++.

Considerando a evolução das CPUs e a velocidade com que os dados são processados atualmente, na grande maioria dos casos, optar por uma linguagem como Python torna-se natural e seguro. Ainda mais considerando que é possível escrever as partes mais críticas do software em C/C++ e integrar com sua aplicação em Python.

Quem usa Python?

Python dispõe de uma grande base de usuários e uma comunidade bastante ativa. Há vários anos diferentes ranks de uso de linguagens de programação citam Python como uma das 5 linguagens mais usadas.

- **Google.** Google usa largamente Python na sua ferramenta de indexação - Google Search. A empresa também usa Python para desenvolver seus sistemas de inteligência artificial.
- **Youtube.** Como um serviço do Google, o Youtube também usa largamente Python por baixo dos panos.
- **Dropbox.** O cliente do Dropbox para desktops é escrito em Python.
- **Pixar.** A Pixar (Toy Story) usa Python na produção de seus filmes de animação.
- **NSA.** A agência americana usa Python para análise e criptografia dos dados.
- **Netflix.** A Netflix usa Python na sua infraestrutura de software.

O poder do Python

Python está entre uma linguagem de script e linguagem de desenvolvimento de sistemas. Python contém toda a facilidade e simplicidade de linguagens de script (como Perl, Ruby e Scheme) e também ferramentas poderosas de engenharia de software, encontradas tipicamente em linguagens compiladas (como Java e C++).

- **Tipagem dinâmica.** Nomes (variáveis) em Python são apenas referências a objetos, objetos não ficam atrelados a um tipo específico de dado, são mutáveis.
- **Gerenciamento de memória automático.** Python contém um coletor de lixo embutido na máquina virtual. Além disso, Python consegue alocar automaticamente objetos. O desenvolvedor não precisa se preocupar com problemas de baixo nível, a máquina virtual trata todos estes problemas automaticamente.

O poder do Python

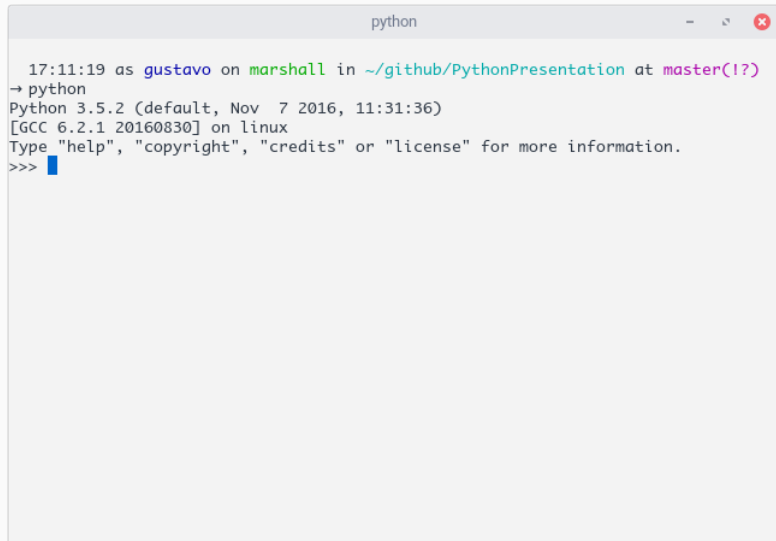
- **Estruturas de dados incluídas.** Python vêm com as estruturas de dados mais utilizadas como listas, dicionários e strings como parte da linguagem.
- **Programação escalável.** Python contém ferramentas para desenvolvimento de sistemas de larga escala, como módulos, classes e exceções.
- **Ferramentas de terceiros.** Python é livre e aberto. Desenvolvedores são encorajados a contribuir com ferramentas. Você consegue encontrar ferramentas para acesso a bancos de dados, programação numérica, desenvolvimento de sistemas inteligentes, desenvolvimento web e muito mais.

Ferramentas Interessantes

CPython é a distribuição padrão do Python. O interpretador da linguagem é escrito em ANSI C, o que dá a esta implementação maior poder de portabilidade. O desenvolvimento é encabeçado pelo criador da linguagem Python: Guido van Rossum e pelo grupo principal de desenvolvimento da linguagem. A distribuição padrão vêm com um tradutor de código fonte para bytecode e com uma máquina virtual, chamada de **Python Virtual Machine**.

Além do CPython, existem outras implementações conhecidas da linguagem, como **JPython**, **IronPython**, **Stackless** e **PyPy**.

CPython

A terminal window titled 'python' with standard window controls. The terminal shows the execution of the 'python' command, displaying version information and a prompt. The text is as follows:

```
17:11:19 as gustavo on marshall in ~/github/PythonPresentation at master(!?)  
→ python  
Python 3.5.2 (default, Nov  7 2016, 11:31:36)  
[GCC 6.2.1 20160830] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

O IPython pode ser visto como o REPL padrão do Python, com esteroides. O objetivo do IPython é disponibilizar um ambiente exploratório de computação interativa. Esta apresentação foi construída quase que em sua totalidade utilizando o IPython para debug dos trechos de códigos mostrados ao longo da apresentação.

IPython

```
IPython: github/PythonPresentation

17:14:55 as gustavo on marshall in ~/github/PythonPresentation at master(!?)
→ ipython
Python 3.5.2 (default, Nov 7 2016, 11:31:36)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: █
```

PyPy é uma implementação alternativa do CPython, focada na performance. A grande diferença entre o PyPy e o CPython padrão é que o PyPy vem com um compilador JIT - Just In Time. O compilador JIT é acoplado ao PVM como uma extensão, traduzindo partes do código fonte, apenas quando necessário. O PyPy consegue executar programas longos escritos em Python, entre 3-5 vezes mais rápido que a implementação padrão - CPython.

```
pypy

17:16:25 as gustavo on marshall in ~/github/PythonPresentation at master(!?)
→ pypy
Python 2.7.12 (aff251e54385, Nov 12 2016, 22:03:47)
[PyPy 5.6.0 with GCC 6.2.1 20160830] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

A Linguagem

Tipos básicos de dados

```
In [1]: type(2**512)
```

```
Out[1]: int
```

```
In [2]: type([1, 2, 3])
```

```
Out[2]: list
```

```
In [3]: type((1, 2, 3))
```

```
Out[3]: tuple
```

```
In [4]: type({1: "Marie", 2: "Maribel", 3: "Meggie", 4: "Elisa"})
```

```
Out[4]: dict
```

```
In [5]: type("oi, tudo bem?")
```

```
Out[5]: str
```

Tipos básicos de datos

```
In [6]: type(lambda x: x + x)
```

```
Out[6]: function
```

```
In [7]: type(False)
```

```
Out[7]: bool
```

```
In [8]: type(1.23e-4)
```

```
Out[8]: float
```

```
In [9]: type(range(10))
```

```
Out[9]: range
```

```
In [10]: type(enumerate([1, 2, 3]))
```

```
Out[10]: enumerate
```


Operações aritméticas

```
In [29]: 1 + 2
```

```
Out[29]: 3
```

```
In [30]: 1 * 2
```

```
Out[30]: 2
```

```
In [31]: 1 / 2
```

```
Out[31]: 0.5
```

```
In [32]: 1 ** 2
```

```
Out[32]: 1
```

```
In [33]: 1 >> 2
```

```
Out[33]: 0
```

```
In [34]: 1 << 2
```

```
Out[34]: 4
```

```
In [35]: 1 % 2
```

```
Out[35]: 1
```

Referências a objetos

Python não possui variáveis, em vez disto, possui referências a objetos, que são conhecidos como nomes.

```
In [11]: g = lambda x: x*x + 2*x
```

```
In [12]: a = 4
```

```
In [13]: b = 5
```

```
In [14]: l = [1, 2, 3, 4, 5]
```

```
In [15]: t = (1, 2, 3, 4, 5)
```

```
In [16]: d = {0: "oi", "tudo": 1, 2: "bem?"}
```

Operador de identificação

```
In [36]: n1 = 4
```

```
In [37]: n2 = 4
```

```
In [38]: n1 == n2
```

```
Out[38]: True
```

```
In [39]: n1 is n2
```

```
Out[39]: True
```

```
In [40]: n3 = 5
```

```
In [41]: n1 is n3
```

```
Out[41]: False
```

Operador de identificação

```
In [42]: l1 = [1, 2, 3, 4]
```

```
In [43]: l2 = [1, 2, 3, 4]
```

```
In [44]: l1 == l2
```

```
Out[44]: True
```

```
In [45]: l1 is l2
```

```
Out[45]: False
```

```
In [46]: l1 = l2
```

```
In [47]: l1 is l2
```

```
Out[47]: True
```

Operações lógicas

```
In [52]: True or False
```

```
Out[52]: True
```

```
In [53]: True and False
```

```
Out[53]: False
```

```
In [54]: True or False and True
```

```
Out[54]: True
```

```
In [55]: True and False and True
```

```
Out[55]: False
```

Operadores de comparação

```
In [56]: x = 1; y = 2
```

```
In [57]: x == y
```

```
Out [57]: False
```

```
In [58]: x != y
```

```
Out [58]: True
```

```
In [59]: x < y
```

```
Out [59]: True
```

```
In [60]: x >= y
```

```
Out [60]: False
```

```
In [61]: 0 < x < y
```

```
Out [61]: True
```

Operadores de comparação

```
In [62]: t = ("gustavo", 1994, [1, 2, 3], lambda n: n < 1)
```

```
In [63]: 1994 in t
```

```
Out[63]: True
```

```
In [65]: (lambda n: n < 1) in t
```

```
Out[65]: False
```

```
In [66]: [1, 2, 3] in t
```

```
Out[66]: True
```

```
In [67]: "gustavo" in t
```

```
Out[67]: True
```

Fluxo de controle

```
if <expres_booleana>:  
    <comandos>  
elif <expres_booleana>:  
    <comandos>  
else:  
    <comandos>
```

```
In [69]: if 3 > 1:  
...:     print("3 > 1")  
...: elif 2 < 5:  
...:     print("2 < 5")  
...: else:  
...:     print("oops")  
...:  
3 > 1
```


Laço de repetição while

```
while <expres_booleana>:  
    <comandos>
```

```
In [78]: while n != 0:  
...:     n = int(input("chute um numero:"))  
...:     if n == 0:  
...:         print("acertou!")  
...:  
chute um numero:5  
chute um numero:1  
chute um numero:0  
acertou!
```

```
In [79]:
```

Laço de repetição for

```
for <variavel> in <iteravel>:  
    <comandos>
```

```
In [80]: for i in range(5):  
...:     print("i+i=", i+i)  
...:  
i+i= 0  
i+i= 2  
i+i= 4  
i+i= 6  
i+i= 8
```

Excessões

```
In [81]: def f():  
...:     i = input("digite um inteiro: ")  
...:     try:  
...:         i2 = int(i)  
...:         print("o valor eh:", i2)  
...:     except:  
...:         print("o valor digitado nao eh um inteiro")  
...:
```

```
In [82]: f()  
digite um inteiro: 4  
o valor eh: 4
```

```
In [83]: f()  
digite um inteiro: oi  
o valor digitado nao eh um inteiro
```

Executando scripts

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  if __name__ == "__main__":
5      nome = input("> ")
6      print("Ola, ", nome)
```

Se você está usando Linux, precisa apenas dar a permissão de execução para o script, então ele irá invocar a PVM automaticamente baseado no caminho especificado na primeira linha do script.

```
$ chmod +x hello_world.py
$ ./hello_world.py
> Gustavo Santos
Ola, Gustavo Santos
```

Executando scripts

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  import sys
4
5  if __name__ == "__main__":
6      nome = sys.argv[1:]
7      print("Ola,", " ".join(nome))
```

```
$ chmod +x hello_world_2.py
$ ./hello_world_2.py Gustavo Santos
Ola, Gustavo Santos
```

```
In [5]: def fatorial(x):  
...:     if x < 0:  
...:         return None  
...:     elif x < 2:  
...:         return x  
...:     else:  
...:         return x*fatorial(x-1)  
...:
```

```
In [6]: fatorial(3)
```

```
Out[6]: 6
```

```
In [7]: fatorial(-4)
```

Paradigmas de Programação e Python 3.x

3 Pilares das Linguagens Funcionais

- `map`
- `filter`
- `reduce`

Dada uma coleção e uma função tal que $C \mapsto U$, a função `map` transforma uma coleção de dados em outra.

```
In [1]: map(lambda x: x * x, [1, 2, 3, 4, 5])
```

```
Out[1]: <map at 0x7f1d2535bdd8>
```

```
In [2]: list(map(lambda x: x * x, [1, 2, 3, 4, 5]))
```

```
Out[2]: [1, 4, 9, 16, 25]
```

A função `map` é preguiçosa no Python 3.x. Somente irá avaliar o resultado quando preciso. `map` retorna um iterador.

Dada uma coleção e uma função predicado, a função `filter` irá "filtrar" todos os dados da coleção de entrada que retorne `True` quando aplicados na função predicado.

```
In [3]: filter(lambda x: x%2 == 0, [1, 2, 3, 4, 5])
```

```
Out[3]: <filter at 0x7f1d25381e48>
```

```
In [4]: list(filter(lambda x: x%2 == 0, [1, 2, 3, 4, 5]))
```

```
Out[4]: [2, 4]
```

A função `filter` também é preguiçosa. `filter` também retorna um iterador.

A função `reduce` foi removida do Python 3.x por detalhes de performance. Uma composição com laços de repetição é mais performática do que o uso desta função.

```
In [5]: from functools import reduce
```

```
In [7]: reduce(lambda x, acum: x * acum, [1, 2, 3, 4, 5])
```

```
Out [7]: 120
```

Entretanto está disponível na biblioteca `functools`.

Python não é uma boa linguagem funcional

O paradigma de programação funcional exige diversas características. Python não obedece a estas características, logo não pode ser taxada como uma linguagem funcional, nem mesmo como tendo suporte a programação funcional, como Scala.

Estruturas de Dados em Python

Hydra - Python Multitarefa

Coisas Legais no Python

Adicionando cache aos programas

```
1 from functools import lru_cache
2
3 @lru_cache(maxsize=None)
4 def fib_cache(x):
5     if x >= 0 and x < 2:
6         return x
7     else:
8         return fib_cache(x - 1) +
           fib_cache(x - 2)
```

```
1 def fib_rec(x):
2     if x >= 0 and x < 2:
3         return x
4     else:
5         return fib_rec(x - 1) +
           fib_rec(x - 2)
```


Adicionando cache aos programas

```
In [15]: from fib_rec import fib_rec
```

```
In [16]: from fib_cache import fib_cache
```

```
In [17]: %time fib_rec(35)
```

```
CPU times: user 12.6 s, sys: 3.33 ms, total: 12.6 s
```

```
Wall time: 12.7 s
```

```
Out[17]: 9227465
```

```
In [18]: %time fib_cache(35)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
```

```
Wall time: 59.6  $\mu$ s
```

```
Out[18]: 9227465
```

Adicionando cache aos programas

```
1  from fib_rec import fib_rec
2  from fib_cache import fib_cache
3  import time
4
5  def test_cache(valores):
6      tempos = []
7      for x in valores:
8          t1 = time.time()
9          r = fib_cache(x)
10         t2 = time.time()
11         tempos.append(t2 - t1)
12     return tempos
13
14
15 def test_rec(valores):
16     tempos = []
17     for x in valores:
18         t1 = time.time()
19         r = fib_rec(x)
20         t2 = time.time()
21         tempos.append(t2 - t1)
22     return tempos
```

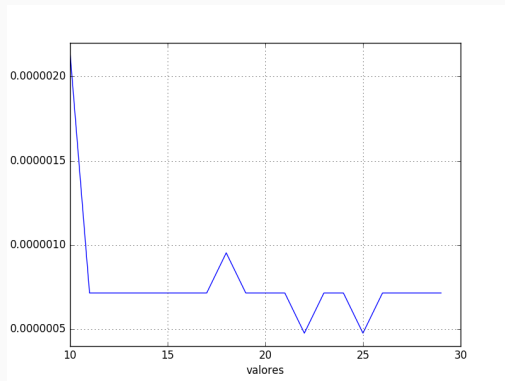
Adicionando cache aos programas

```
In [8]: valores = [i for i in range(10, 30)]
```

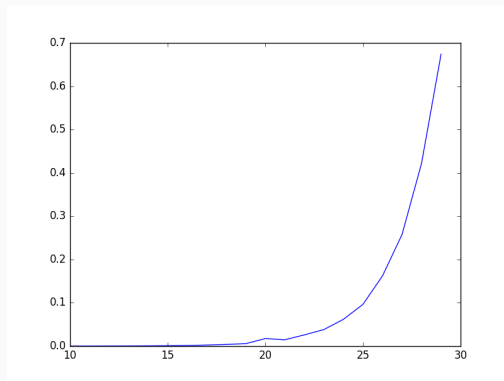
```
In [9]: tempos_cache = fib_test.test_cache(valores)
```

```
In [10]: tempos_rec = fib_test.test_rec(valores)
```

Adicionando cache aos programas



(a) Implementação com cache



(b) Implementação sem cache

Figura 1: Tempos das implementações recursivas da função de Fibonacci com e sem cache

É necessário o módulo PyInstaller, disponível para Linux, Mac e Windows.

```
|| $ pip install PyInstaller
```

Compilando programs em Python

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  def main():
4      print("Oi, tudo bem?")
5      v1 = int(input("primeiro inteiro: "))
6      v2 = int(input("segundo inteiro: "))
7      print(v1 + v2)
8
9  if __name__ == "__main__":
10     main()
```

Compilando programs em Python

```
$ pyinstaller python_compilado.py
79 INFO: PyInstaller: 3.2
79 INFO: Python: 3.5.2
85 INFO: Platform: Linux-4.8.10-1-ARCH-x86_64-with-arch
85 INFO: wrote /home/gustavo/github/PythonPresentation/python_scripts/
    python_compilado.spec
87 INFO: UPX is not available.
93 INFO: Extending PYTHONPATH with paths
['/home/gustavo/github/PythonPresentation/python_scripts',
 '/home/gustavo/github/PythonPresentation/python_scripts']
93 INFO: checking Analysis
93 INFO: Building Analysis because out00-Analysis.toc is non existent
94 INFO: Initializing module dependency graph...
```

Compilando programs em Python

```
$ cd dist
$ cd python_compilado
$ pwd
~/github/PythonPresentation/python_scripts/dist/python_compilado
$ ./python_compilado
Oi, tudo bem?
primeiro inteiro: 8
segundo inteiro: 9
17
```


Aprenda Mais!

- **Learning Python.** *Mark Lutz*. O'Reilly.
- **Learning IPython for Interactive Computing and Data Visualization.** *Cyrille Rossant*. Packt Publishing Ltd. Birmingham, UK. 2015. ISBN 978-1-78398-698-9.
- **Mastering Pandas.** *Femi Anthony*. Packt Publishing Ltd. Birmingham, UK. 2015. ISBN 978-1-78398-196-0.

Esta apresentação e todos os scripts apresentados estão disponíveis no repositório no GitHub: <https://github.com/gustavofsantos/PythonPresentation>

Python, o que é e porque usar

Gustavo Fernandes dos Santos
gfdsantos@inf.ufpel.edu.br

27 de novembro de 2016

UFPEL - Universidade Federal de Pelotas