Introdução a Python

Gustavo Serra Scalet gsscalet@gmail.com

Encontros GrupySP

30 de Maio, 2008

Roadmap

Um pouco da história e futuro

- Criada por Guido van Rossum na Holanda para ser um sucessor de uma linguagem chamada ABC
- O nome Python veio de "Monty Python's Flying Circus", que Guido van Rossum estava lendo durante a implementação. Ele precisava de um nome curto, único e levemente misterioso.
- Surgiu em torno dos anos 90 e após 2001, já na sua versão 2.0, python tem sua própria licença compatível com a GPL com a criação da PSF (Python Software Foundation)
- Python está na versão 2.6 e há desenvolvimento de uma versão 3.0 com diversas alterações
- Última versão (08/05/2008): 2.6alpha3 e 3.0alpha5

E o que Python tem de tão especial?

- É livre!!
- Multiplataforma (roda em Windows, Linux/Unix, OS/2, Mac, .Net e S60...)
- Multiparadigma (Estrutural, orientada a objetos, funcional)
- É interpretada
- Linguagem limpa, com sintaxe legível e de fácil aprendizado
- Gerenciamento automático de memória: Esqueça os frees!
- Nível altíssimo de tipos de dados dinâmicos
- Extensas bibliotecas por padrão e muitos módulos *third party* para praticamente qualquer tarefa
- Usada como linguagem de scripts por diversos sistemas (GIMP, Blender)

Algumas aplicações e bibliotecas externas

- Web
  - Frameworks (Django, TurboGears)
  - Servidor de aplicações (Zope)
  - Sistemas de gerenciamentos (Plone)
  - Suporte a diversos protocolos (CGI, http, ftp, smtp)
- Graphic User Interface (GUI)
  - Tk (comum em muitos portes do python)
  - wxWidgets
  - GTK+
  - Qt
  - Microsoft Foundation Classes
  - Delphi
- Banco de Dados
  - MySQL
  - Oracle
  - MS SQL Server
  - PostgreSQL
  - SybODBC
  - SQLite
- Científico e numérico
  - Processamento de imagens (VTK, PIL)
  - Cálculos e precisos (SciPy, NumPy, GmPy)
- Jogos
  - Pygame
  - PyOpenGL
- Entre muitos outros...

O que dizem por aí...

- "Python é nos permite produzir novas funcionalidades em tempo recorde com o mínimo de desenvolvedores" - Cuong Do, Arquiteto de software, YouTube.com.
- "Python foi importante para o Google desde o seu início, permanecendo até hoje. Muitos de nossos engenheiros usam Python e nós estamos procurando mais pessoas com habilidades nessa linguagem." - Peter Norvig, Diretor de qualidade de busca, Google, Inc.
- "Nós ensinamos Python para estudantes da graduação e da pós em nossos cursos de semânticas Web simplesmente porque não há nada tão flexível e com tantas bibliotecas web." - Prof. James A. Hendler, University of Maryland

Tradução livre de `http://www.python.org/about/quotes/`

Roadmap

Apresentando os tipos

Em python podemos separar os tipos de dados nativos em dois grandes grupos:

Apresentando os tipos

Em python podemos separar os tipos de dados nativos em dois grandes grupos:

1. **Imutáveis** - Inteiros, inteiros longos, pontos flutuantes, complexos, strings e tuplas

Apresentando os tipos

Em python podemos separar os tipos de dados nativos em dois grandes grupos:

1. **Imutáveis** - Inteiros, inteiros longos, pontos flutuantes, complexos, strings e tuplas
2. **Mutáveis** - Listas e dicionários

Apresentando os tipos

Em python podemos separar os tipos de dados nativos em dois grandes grupos:

1. **Imutáveis** - Inteiros, inteiros longos, pontos flutuantes, complexos, strings e tuplas
2. **Mutáveis** - Listas e dicionários

Fora eles tem o None, que seria um NULL.

Apresentando os tipos

Em python podemos separar os tipos de dados nativos em dois grandes grupos:

1. **Imutáveis** - Inteiros, inteiros longos, pontos flutuantes, complexos, strings e tuplas
2. **Mutáveis** - Listas e dicionários

Fora eles tem o None, que seria um NULL.
Mas qual a diferença entre esses mutáveis e imutáveis?

Mutáveis e imutáveis?!

Analisando um tipo imutável (no caso string) vemos que:

```
>>> str = "Gustavo"
>>> str2 = str
>>> str = str + " Serra"
>>> str
'Gustavo Serra'
>>> str2
'Gustavo'
```

Isso acontece porque a ao incrementarmos a `str` com `" Serra"` um novo objeto foi criado, e apenas `str` apontava para ele, enquanto que `str2` apontava para o objeto antigo

Mutáveis e imutáveis?!

Mas se fosse um tipo imutável (listas):

```
>>> lista = ['abacaxi',(0,0),2]
>>> lista2 = lista
>>> lista.append('Novo Elemento')
>>> lista
['abacaxi', (0, 0), 2, 'Novo Elemento']
>>> lista2
['abacaxi', (0, 0), 2, 'Novo Elemento']
```

Como o tipo lista pode se modificar, ele foi alterado e não recriado como outro objeto, sendo assim a lista2 e lista apontam para o mesmo objeto mesmo após a sua alteração

Roadmap

Seu melhor amigo

Use como calculadora, peça ajuda, pergunte métodos de objetos, teste trechos de códigos, recrie ambientes... use e abuse!

```
[21:10:31] gut@quasar ~ $ python
Python 2.4.4 (#1, Apr  6 2008, 07:48:32)
[GCC 4.1.2 (Gentoo 4.1.2 p1.0.2)] on linux2
Type "help", "copyright", "credits" or "license" for more informat
>>>
```

Roadmap

Um erro incomum

Python é orientado a indentação, o que quer dizer que um código indentado não é só elegante mas também é necessário para o reconhecimento dos laços.

```
>>> if 1 == 2:
...     print "Temos problemas..."
...     print "...dos grandes!"
...  else:
  File "<stdin>", line 3
    else:
       ^
IndentationError: unindent does not match any outer indentation le
>>>
```

Exemplo simples

Vendo quais elementos são iguais em duas listas

```
>>> l = [1,2,3,4,5]
>>> p = [3,4,5,6,7]
>>> for x in l:
...   if x in p:
...     print x,
...
3 4 5
```

Melhorando...

Usando conjuntos!

```
>>> l = set([1,2,3,4,5])
>>> p = set([3,4,5,6,7])
>>> l & p  # '&' eh o operador logico AND
set([3, 4, 5])
```

Deixando mais pythônico

```
>>> l = [1,2,3,4,5]
>>> p = [3,4,5,6,7]
>>> [x for x in l if x not in p]
[1, 2]
```

Por que os programadores tentam fazer tudo em uma linha?!

Padronizações

É costume incluir nos cabeçalhos de seus programas as seguintes linhas:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

É também uma boa maneira não interpretar seu código diretamente:

```
def main():
    # Agora sim! Programe aqui dentro
    ...

if __name__ == "__main__":
    main()
```

Deste jeito seu código fica muito mais modularizado

Palavras chaves

Explore-as! Há muitos poder nelas

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Python tem até easter egg!

Use o módulo `this`:

```
>>> import this
```

The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

Roadmap

Referências

1. http://www.python.org/
2. http://www.gustavobarbieri.com.br/python/aulas_python/
3. http://www.dmat.furg.br/~python/aspectos.html

Aprenda Mais!

- http://www.pythonbrasil.com.br/moin.cgi/DocumentacaoPython
- http://www.pythonbrasil.com.br/moin.cgi/CookBook
- http://docs.python.org/