

O Essencial da Linguagem **Python**

... **Python** nome inspirado nos “Monty Python”!



“its the...” **Monty Python**... desde 5.Out.1969

Com Eric Idle, Graham Chapman, John Cleese, Michael Palin e Terry Jones.

Actores britânicos com novo estilo de texto, e representação, marcadamente **anárquico** e pautado pelo completo **surrealismo** das cenas.

Também contavam com Terry Gilliam, caricaturista na revista *Mad*.



...”it’s the Monty Python”

“***and now for something completely different...***”

... a influência dos “Monty Python” no Python



*Despite all the reptile icons in Python world, the truth is that Python creator **Guido van Rossum** named it after the BBC comedy series Monty Python’s Flying Circus. He is a fan of Monty Python, as are many software developers (indeed, there seems to almost be a symmetry between the two fields).*

This legacy inevitably adds a humorous quality to Python code examples. For instance, the traditional “foo” and “bar” for generic variable names become “spam” and “eggs” in the Python world; the occasional “Brian,” “ni,” and “shrubbery” likewise owe their appearances to this namesake.

“Spam” is a reference from a famous Monty Python skit in which people trying to order food in a cafeteria are drowned out by a chorus of Vikings singing about spam.

Spam = abreviatura de “spiced ham” (presunto condimentado).



Mark Lutz, “Learning Python”, 3rd edition (Python 2.5), O’Reilly, 2008.

Para onde está o **Python** mais vocacionado?

É linguagem de programação genérica portanto aplicável a qualquer problema.
No entanto tem evoluído colocando bastante ênfase em:

Inteligência Artificial – sistemas periciais (PyKE, PyCLIPS), redes semânticas
redes neuronais (nn), tratamento língua natural (NLTK), etc.

Jogos – processamento gráfico (pygame), imagem (PIL), robots (PyRo), etc.

Acesso Internet – tarefas na rede como cliente e servidor; tratamento HTML,
XML, RDF (rdflib), envio, recepção, composição e análise de e-mail, etc.

Programação Numérica e Científica – matrizes biblioteca matemáticas (NumPy).

Programação de Sistemas – integração com serviços do sistema operativo.

Interface Gráfica – interoperabilidade usando API “Tkinter” (e “Dabo”).

Integração de Sistemas – possível embeber em Java, (Jython) .NET
(IronPython), C, C++; tem ferramentas, SWIG e SIP, para simplificar integração.

Acesso Bases de Dados – todos os conhecidos; e.g., MySQL e SQLite.

Construção Rápida de Protótipos – simplicidade de escrita e de teste.

Quem (“importante”, i.e. com “notoriedade”) usa Python?

Google → web search, and **employs Python's creator**.

YouTube → video sharing service.

BitTorrent → peer-to-peer file sharing system.

Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, IBM → hardware testing.

Pixar, Industrial Light & Magic, and others → movie animation.

JPMorgan Chase, UBS, Getco, Citadel → market forecasting.

NASA, Los Alamos, Fermilab, JPL, and others → scientific programming tasks.

iRobot → commercial robotic vacuum cleaners.

ESRI → as an end-user customization tool for its GIS mapping products.

NSA → cryptography and intelligence analysis.

IronPort email server → more than 1 million lines of Python code to do its job.

One Laptop Per Child (OLPC) project → user interface and activity model.

... and so on.

“top ten” – índice de popularidade das linguagens

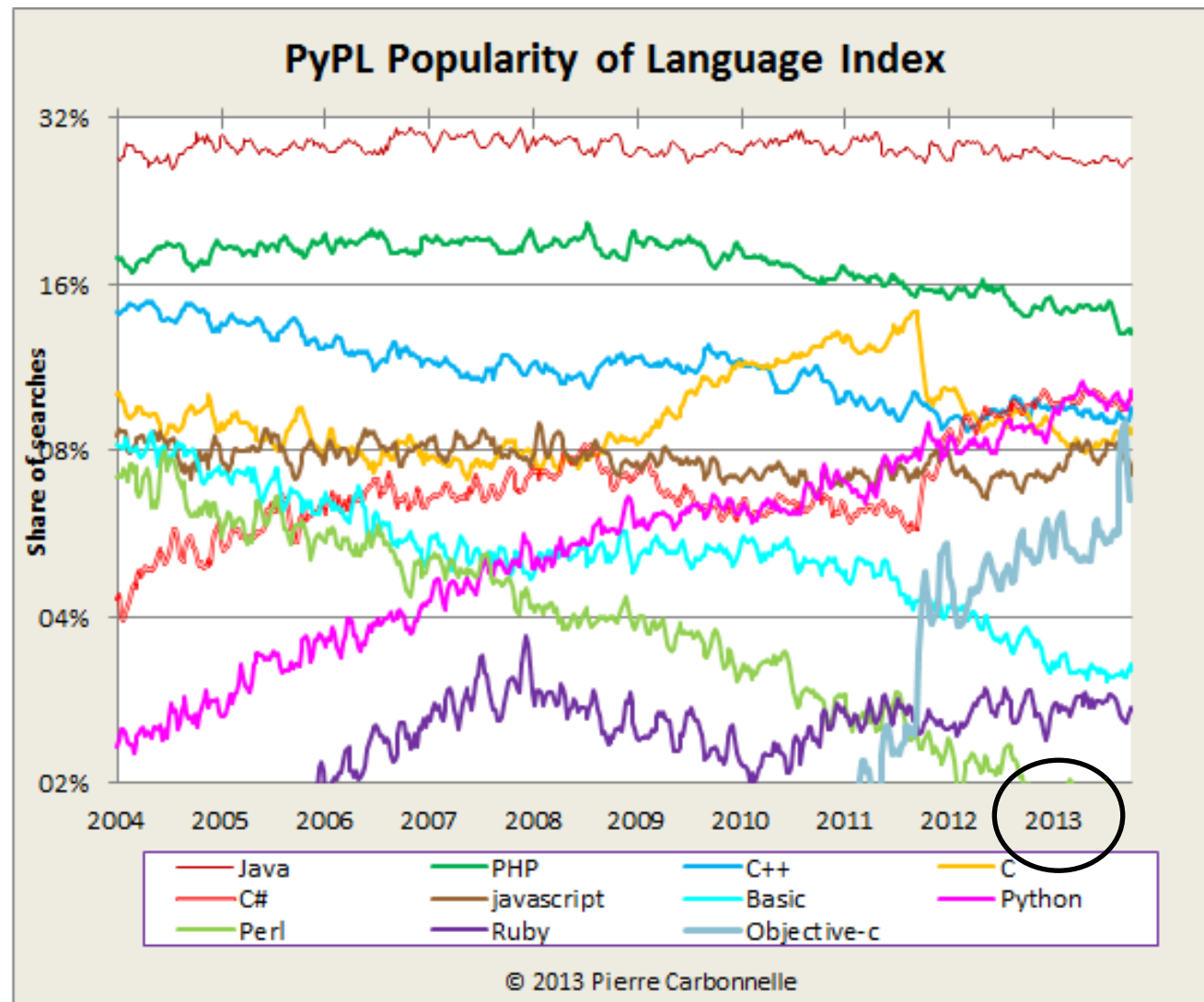
PYPL Popularity of Programming Language index

	Position Nov 2013	Position Nov 2012	Delta in position	Programming language	Share in Nov 2013	Twelve month trends
Java →	1	1		Java	27.0 %	-1.2 %
	2	2		PHP	13.0 %	-1.1 %
Python →	3	6	↑↑↑	<u>Python</u>	10.3 %	+1.4 %
C# →	4	3	↓	C#	10.0 %	+0.0 %
	5	5		C++	9.6 %	-0.6 %
	6	4	↓↓	C	8.6 %	-0.5 %
	7	7		<u>Javascript</u>	7.2 %	+0.8 %
	8	8		<u>Objective-C</u>	6.6 %	+1.7 %
	9	9		Visual Basic	3.3 %	-0.4 %
	10	10		<u>Ruby</u>	2.7 %	+0.1 %

actualização mensal em:

<https://sites.google.com/site/pydatalog/pypl/PyPL-Popularity-of-Programming-Language>

... “top ten” – de 2004 a 2013



Python

... e agora, uns pensamentos e obter o **Python**

There are two ways of constructing a software design:

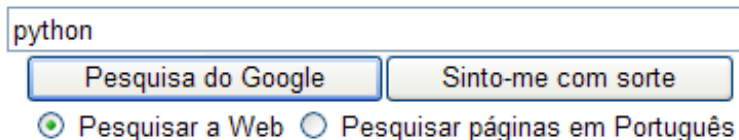
- *one way is to make it so simple that there are obviously no deficiencies;*
- *the other is to make it so complicated that there are no obvious deficiencies.*

C. A. R. Hoare

Para obter o **Python**

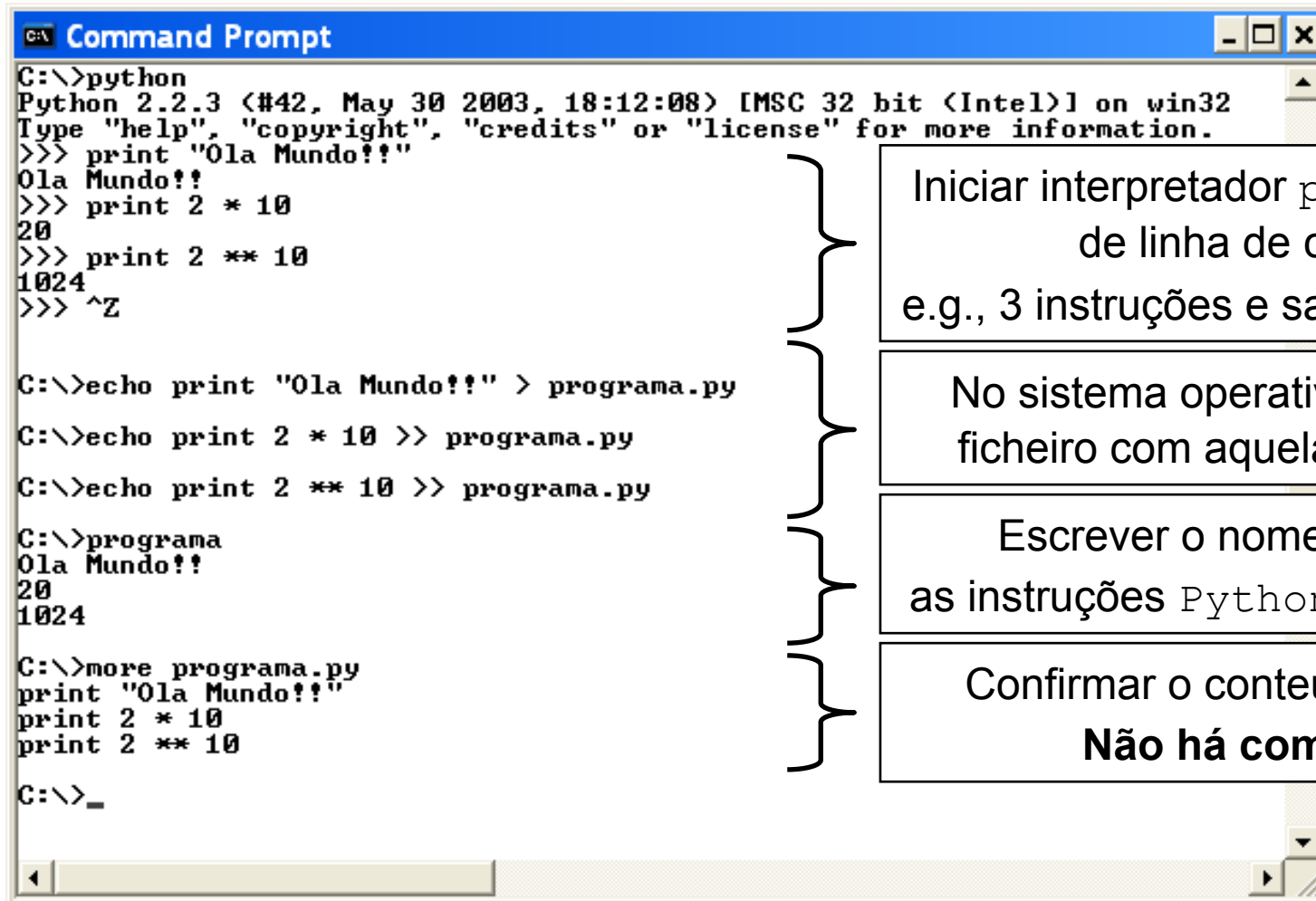
`http://www.python.org/download/`
código fonte aberto (“open source”)

Para mais informação:



A screenshot of a Google search interface. At the top, a search bar contains the word "python". Below the search bar are two buttons: "Pesquisa do Google" and "Sinto-me com sorte". At the bottom, there are two radio buttons: the first is selected and labeled "Pesquisar a Web", and the second is unselected and labeled "Pesquisar páginas em Português".

O interpretador Python – um exemplo de utilização



```
C:\>python
Python 2.2.3 (#42, May 30 2003, 18:12:08) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Ola Mundo!!"
Ola Mundo!!
>>> print 2 * 10
20
>>> print 2 ** 10
1024
>>> ^Z

C:\>echo print "Ola Mundo!!" > programa.py
C:\>echo print 2 * 10 >> programa.py
C:\>echo print 2 ** 10 >> programa.py

C:\>python programa
Ola Mundo!!
20
1024

C:\>more programa.py
print "Ola Mundo!!"
print 2 * 10
print 2 ** 10

C:\>_
```

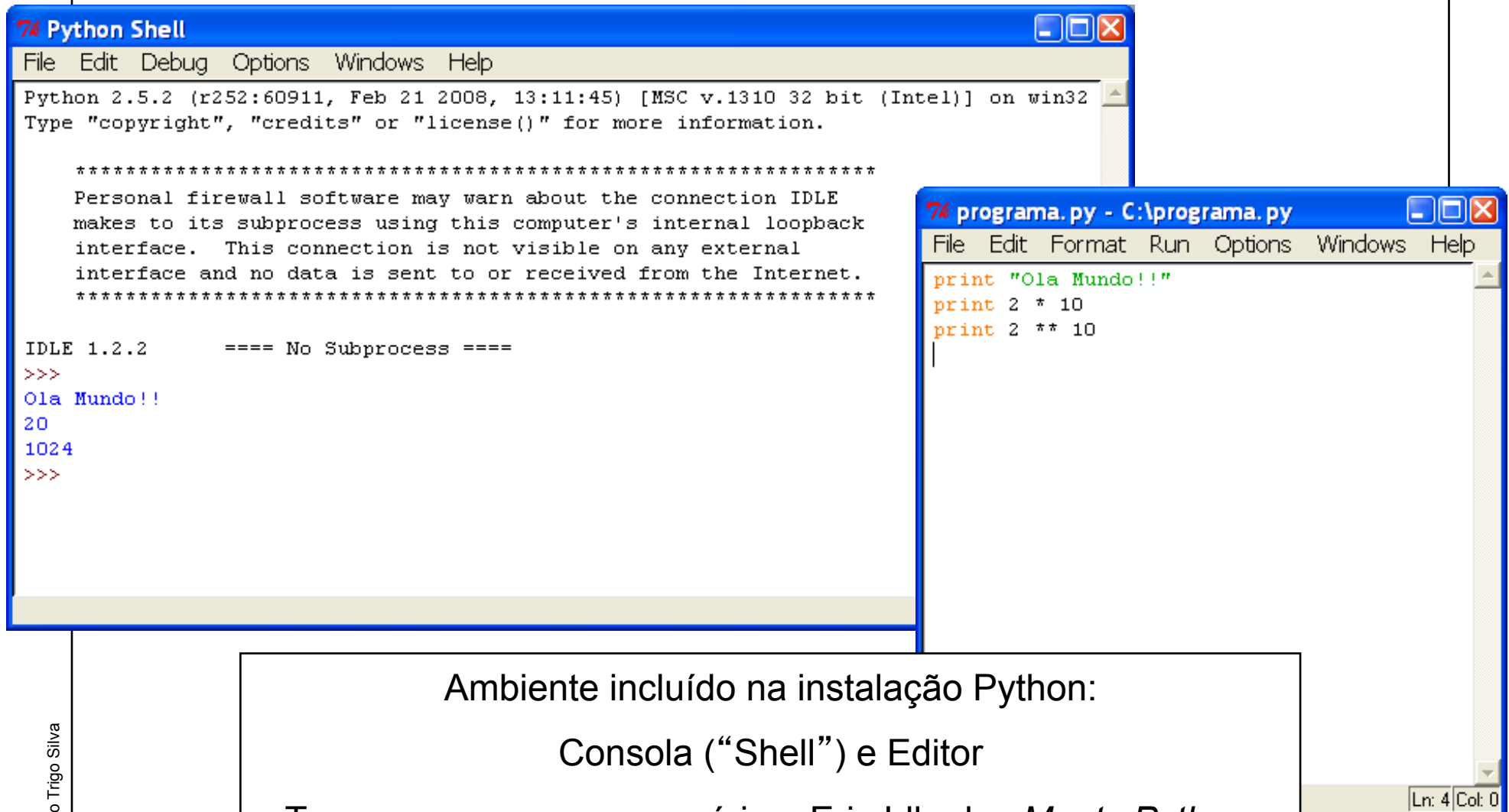
Iniciar interpretador `python` em modo de linha de comando; e.g., 3 instruções e sair do interpretador

No sistema operativo construir um ficheiro com aquelas 3 instruções.

Escrever o nome do ficheiro... as instruções `Python` são executadas.

Confirmar o conteúdo do ficheiro.
Não há compilação.

IDLE – ambiente de desenvolvimento para o Python



Ambiente incluído na instalação Python:

Consola (“Shell”) e Editor

Tem nome IDLE em memória a Eric Idle dos *Monty Python*.

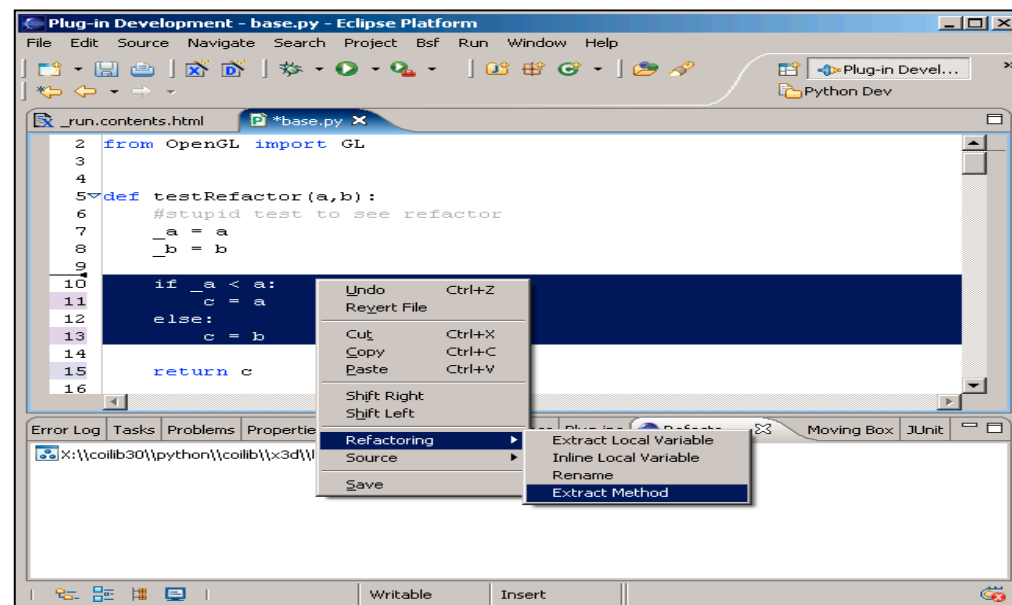
Outros ambientes de desenvolvimento para o Python



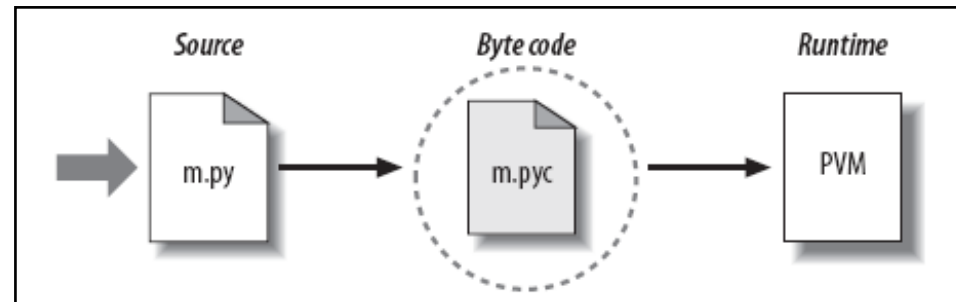
Pydev is a plugin to use Eclipse for Python and Jython development.

It comes with many goodies such as code completion, syntax highlighting, syntax analysis, refactor, debug and many others.

<http://pydev.sourceforge.net/>



Modelo de execução do Python



`m.py` ≡ código fonte

`m.pyc` ≡ tradução do código fonte para “byte code”

PMV ≡ máquina virtual Python “Python Virtual Machine”

A tradução para “byte code” (ficheiro `.pyc`) ocorre durante a primeira execução.

Se o código fonte não se alterar não se volta a construir o “byte code”

A “Python Virtual Machine” (PVM) não é um programa separado nem precisa de ser instalado separadamente.

PVM é o último passo do “interpretado Python”; é apenas um “grande ciclo” que itera sobre cada uma das instruções do “byte code”.

A estrutura de um programa **Python**

1. um programa é composto por módulos (*a ver posteriormente*)
2. um módulo contém instruções
3. uma instrução contém expressões
4. uma expressão cria e/ou processa objectos

uma
instrução
em Java

```
if (x > y) {  
    x = 1;  
    y = 2;  
}
```

uma
instrução
em Python

```
if x > y:  
    x = 1  
    y = 2
```

Muito importante:
no Python não há delimitadores de bloco,
a indentação delimita os blocos,
os parêntesis são opcionais,
cada instrução escreve-se numa linha.

O código Python é simples de escrever e de ler

```
while True:  
    reply = raw_input('Enter text:')  
    if reply == 'stop': break  
    print reply.upper()
```

... e agora para ... o essencial da linguagem

As estruturas:

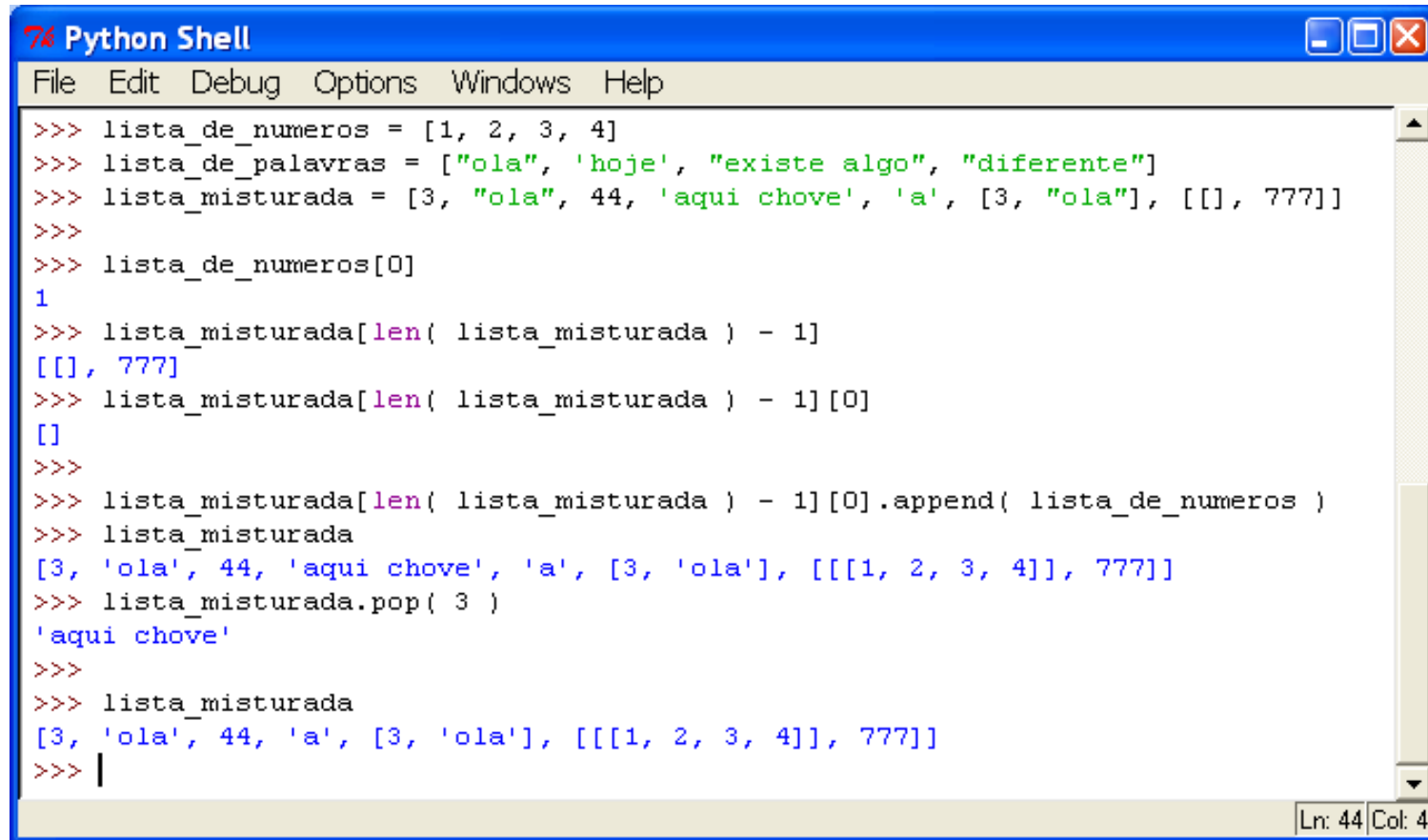
Listas

Dicionários

Tuplos

... e conversões entre as anteriores

Alguns aspectos do Python – as listas: [e1, ...]



```
Python Shell
File Edit Debug Options Windows Help

>>> lista_de_numeros = [1, 2, 3, 4]
>>> lista_de_palavras = ["ola", 'hoje', "existe algo", "diferente"]
>>> lista_misturada = [3, "ola", 44, 'aquí chove', 'a', [3, "ola"], [[], 777]]
>>>
>>> lista_de_numeros[0]
1
>>> lista_misturada[len( lista_misturada ) - 1]
[[], 777]
>>> lista_misturada[len( lista_misturada ) - 1][0]
[]
>>>
>>> lista_misturada[len( lista_misturada ) - 1][0].append( lista_de_numeros )
>>> lista_misturada
[3, 'ola', 44, 'aquí chove', 'a', [3, 'ola'], [[[1, 2, 3, 4]], 777]]
>>> lista_misturada.pop( 3 )
'aquí chove'
>>>
>>> lista_misturada
[3, 'ola', 44, 'a', [3, 'ola'], [[[1, 2, 3, 4]], 777]]
>>> |
```

Ln: 44 | Col: 4

O potencial para a “anarquia”?

Talvez, mas com alguma “disciplina” (regras de “boas práticas”) não chega aí e torna-se extremamente útil.

O essencial das listas: `[e1, ...]`

Operation	Interpretation
<code>L1 = []</code>	An empty list
<code>L2 = [0, 1, 2, 3]</code>	Four items: indexes 0..3
<code>L3 = ['abc', ['def', 'ghi']]</code>	Nested sublists
<code>L2[i]</code>	Index, index of index, slice, length
<code>L3[i][j]</code>	
<code>L2[i:j]</code>	
<code>len(L2)</code>	
<code>L1 + L2</code>	Concatenate, repeat
<code>L2 * 3</code>	

O essencial das listas: [e1, ...] (cont.)

Operation	Interpretation
<code>for x in L2</code> <code>3 in L2</code>	Iteration, membership
<code>L2.append(4)</code> <code>L2.extend([5,6,7])</code> <code>L2.sort()</code> <code>L2.index(1)</code> <code>L2.insert(i, x)</code> <code>L2.reverse()</code>	Methods: grow, sort, search, insert, reverse, etc.
<code>del L2[k]</code> <code>del L2[i:j]</code> <code>L2.pop()</code> <code>L2.remove(2)</code> <code>L2[i:j] = []</code>	Shrinking
<code>L2[i] = 1</code> <code>L2[i:j] = [4,5,6]</code>	Index assignment, slice assignment
<code>range(4)</code> <code>xrange(0, 4)</code>	Make lists/tuples of integers
<code>L4 = [x**2 for x in range(5)]</code>	List comprehensions

Exemplo – listas: [e1, ...]

dimensão, aumento e pertença

```
% python
>>> len([1, 2, 3])           # Length
3
>>> [1, 2, 3] + [4, 5, 6]    # Concatenation
[1, 2, 3, 4, 5, 6]
>>> ['Ni!'] * 4              # Repetition
['Ni!', 'Ni!', 'Ni!', 'Ni!']
>>> 3 in [1, 2, 3]           # Membership
True
>>> for x in [1, 2, 3]: print x, # Iteration
...
1 2 3
```

operador + aplicado a listas e “strings”; conversão lista-”string” e “string”-lista

```
>>> str([1, 2]) + "34"      # Same as "[1, 2]" + "34"
'[1, 2]34'
>>> [1, 2] + list("34")     # Same as [1, 2] + ["3", "4"]
[1, 2, '3', '4']
```

Exemplo – listas: [e1, ...] (cont.)

indexar e projectar

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[2]                # Offsets start at zero
'SPAM!'
>>> L[-2]               # Negative: count from the right
'Spam'
>>> L[1:]               # Slicing fetches sections
['Spam', 'SPAM!']
```

indexar e projectar com atribuições

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs'       # Index assignment
>>> L
['spam', 'eggs', 'SPAM!']
>>> L[0:2] = ['eat', 'more'] # Slice assignment: delete+insert
>>> L                    # Replaces items 0,1
['eat', 'more', 'SPAM!']
```

Exemplo – métodos sobre listas: [e1, ...]

aumento, e ordenação

```
>>> L.append('please')           # Append method call
>>> L
['eat', 'more', 'SPAM!', 'please']
>>> L.sort()                     # Sort list items ('S' < 'e')
>>> L
['SPAM!', 'eat', 'more', 'please']
```

aumento de múltiplos elementos, remoção do topo, e inversão

```
>>> L = [1, 2]
>>> L.extend([3,4,5])           # Append multiple items
>>> L
[1, 2, 3, 4, 5]
>>> L.pop()                     # Delete and return last item
5
>>> L
[1, 2, 3, 4]
>>> L.reverse()                 # In-place reversal
>>> L
[4, 3, 2, 1]
```

Exemplo – métodos sobre listas: [e1, ...] (cont.)

remoção de um ou mais elementos

```
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> del L[0]                # Delete one item
>>> L
['eat', 'more', 'please']
>>> del L[1:]               # Delete an entire section
>>> L                       # Same as L[1:] = []
['eat']
```

inserir uma lista vazia é diferente de atribuir uma lista vazia a uma projecção

```
>>> L = ['Already', 'got', 'one']
>>> L[1:] = []
>>> L
['Already']
>>> L[0] = []
>>> L
[[]]
```

Os dicionários: {chave:valor, ...}

The screenshot shows a Python Shell window with the following code and output:

```
>>>
>>> idade = {"Miguel":24, "Ana":28, 'Adriana':31}
>>> idade
{'Miguel': 24, 'Ana': 28, 'Adriana': 31}
>>> idade["Ana"]
28
>>> idade[1]

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    idade[1]
  KeyError: 1
>>> idade.has_key("Ana")
True
>>> idade["Tu"]=10
>>> idade["Eu"]="10"
>>> idade
{'Eu': '10', 'Miguel': 24, 'Tu': 10, 'Ana': 28, 'Adriana': 31}
>>> del idade["Eu"]
>>> idade
{'Miguel': 24, 'Tu': 10, 'Ana': 28, 'Adriana': 31}
>>> idade.items()
[('Miguel', 24), ('Tu', 10), ('Ana', 28), ('Adriana', 31)]
>>> idade.keys()
['Miguel', 'Tu', 'Ana', 'Adriana']
>>> |
```

Annotations in the image include a circle around the error message and an arrow pointing from the text box to the dictionary definition line.

o dicionário (tabela de dispersão, “hash table”) não tem relação de ordem;
o acesso faz-se por valor de chave.

O essencial dos dicionários: `{chave:valor, ...}`

Operation	Interpretation
<code>D1 = {}</code>	Empty dictionary
<code>D2 = {'spam': 2, 'eggs': 3}</code>	Two-item dictionary
<code>D3 = {'food': {'ham': 1, 'egg': 2}}</code>	Nesting
<code>D2['eggs']</code> <code>D3['food']['ham']</code>	Indexing by key
<code>D2.has_key('eggs')</code> <code>'eggs' in D2</code> <code>D2.keys()</code> <code>D2.values()</code> <code>D2.copy()</code> <code>D2.get(key, default)</code> <code>D2.update(D1)</code> <code>D2.pop(key)</code>	Methods: membership test, keys list, values list, copies, defaults, merge, delete, etc.
<code>len(D1)</code>	Length (number of stored entries)
<code>D2[key] = 42</code> <code>del D2[key]</code>	Adding/changing keys, deleting keys
<code>D4 = dict.fromvalues(['a', 'b'])</code> <code>D5 = dict(zip(keylist, valslst))</code> <code>D6 = dict(name='Bob', age=42)</code>	Alternative construction techniques

Exemplo – dicionários: {chave:valor, ...}

construir e dada uma chave obter correspondente valor

```
% python
>>> d2 = {'spam': 2, 'ham': 1, 'eggs': 3}    # Make a dictionary
>>> d2['spam']                               # Fetch a value by key
2
>>> d2                                       # Order is scrambled
{'eggs': 3, 'ham': 1, 'spam': 2}
```

dimensão, pertença por chave e construção de lista de chaves

```
>>> len(d2)                                # Number of entries in dictionary
3
>>> d2.has_key('ham')                       # Key membership test
True
>>> 'ham' in d2                             # Key membership test alternative
True
>>> d2.keys()                               # Create a new list of my keys
['eggs', 'ham', 'spam']
```


Exemplo – dicionários: {chave:valor, ...} (cont.)

alterar, eliminar e inserir pares chave:valor

```
>>> d2['ham'] = ['grill', 'bake', 'fry']          # Change entry
>>> d2
{'eggs': 3, 'spam': 2, 'ham': ['grill', 'bake', 'fry']}

>>> del d2['eggs']                                # Delete entry
>>> d2
{'spam': 2, 'ham': ['grill', 'bake', 'fry']}

>>> d2['brunch'] = 'Bacon'                         # Add new entry
>>> d2
{'brunch': 'Bacon', 'spam': 2, 'ham': ['grill', 'bake', 'fry']}
```

métodos – obter lista de valores e obter lista com os pares (chave, valor)

```
>>> d2 = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> d2.values()
[3, 1, 2]
>>> d2.items()
[('eggs', 3), ('ham', 1), ('spam', 2)]
```

Exemplo – métodos sobre dicionários: {chave:valor}

obter valor a partir de chave e tratamento da ausência de chave

```
>>> d2.get('spam')           # A key that is there
2
>>> d2.get('toast')          # A key that is missing
None
>>> d2.get('toast', 88)      88
```

valor a devolver quando
a chave não existe

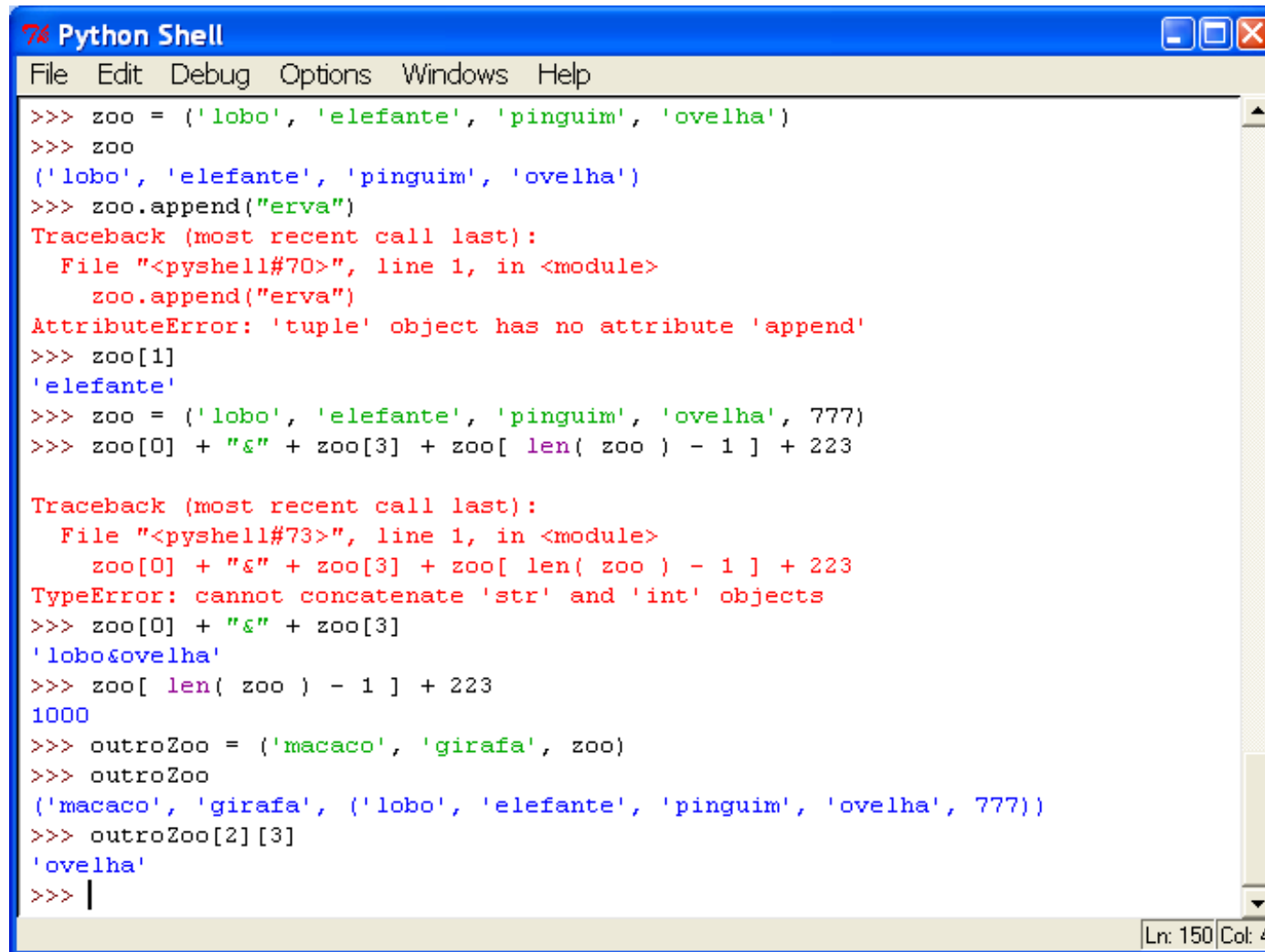
idêntico à concatenação mas para dicionários; sobrepõe valores da mesma chave

```
>>> d2
{'eggs': 3, 'ham': 1, 'spam': 2}
>>> d3 = {'toast': 4, 'muffin': 5}
>>> d2.update(d3)
>>> d2
{'toast': 4, 'muffin': 5, 'eggs': 3, 'ham': 1, 'spam': 2}
```

eliminar par
chave:valor

```
>>> d2
{'toast': 4, 'muffin': 5, 'eggs': 3, 'ham': 1, 'spam': 2}
>>> d2.pop('muffin')
5
>>> d2.pop('toast')          # Delete and return from a key
4
>>> d2
{'eggs': 3, 'ham': 1, 'spam': 2}
```

Os tuplos: (e1, . . .) – como listas mas imutáveis

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
>>> zoo = ('lobo', 'elefante', 'pinguim', 'ovelha')
>>> zoo
('lobo', 'elefante', 'pinguim', 'ovelha')
>>> zoo.append("erva")
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    zoo.append("erva")
AttributeError: 'tuple' object has no attribute 'append'
>>> zoo[1]
'elefante'
>>> zoo = ('lobo', 'elefante', 'pinguim', 'ovelha', 777)
>>> zoo[0] + "&" + zoo[3] + zoo[ len( zoo ) - 1 ] + 223

Traceback (most recent call last):
  File "<pyshell#73>", line 1, in <module>
    zoo[0] + "&" + zoo[3] + zoo[ len( zoo ) - 1 ] + 223
TypeError: cannot concatenate 'str' and 'int' objects
>>> zoo[0] + "&" + zoo[3]
'lobo&ovelha'
>>> zoo[ len( zoo ) - 1 ] + 223
1000
>>> outroZoo = ('macaco', 'girafa', zoo)
>>> outroZoo
('macaco', 'girafa', ('lobo', 'elefante', 'pinguim', 'ovelha', 777))
>>> outroZoo[2][3]
'ovelha'
>>> |
```

The status bar at the bottom right shows "Ln: 150 Col: 4".

O essencial dos tuplos: (e1, ...)

Operation	Interpretation
()	An empty tuple
t1 = (0,)	A one-item tuple (not an expression)
t2 = (0, 'Ni', 1.2, 3)	A four-item tuple
t2 = 0, 'Ni', 1.2, 3	Another four-item tuple (same as prior line)
t3 = ('abc', ('def', 'ghi'))	Nested tuples
t1[i] t3[i][j] t1[i:j] len(t1)	Index, index of index, slice, length
t1 + t2 t2 * 3	Concatenate, repeat
for x in t 'spam' in t2	Iteration, membership

Exemplo – tuplos: (e1, ...)

aumentar o tuplo

```
>>> (1, 2) + (3, 4)           # Concatenation
(1, 2, 3, 4)

>>> (1, 2) * 4                 # Repetition
(1, 2, 1, 2, 1, 2, 1, 2)
```

indexar e projectar o tuplo

```
>>> T = (1, 2, 3, 4)           # Indexing, slicing
>>> T[0], T[1:3]
(1, (2, 3))
```

particularidade – definir tuplo de um único elemento “,” distingue a expressão

```
>>> x = (40)                   # An integer
>>> x
40
>>> y = (40,)                  # A tuple containing an integer
>>> y
(40,)
```

Conversões tuplo-lista e lista-tuplo

tuplo-lista

```
>>> T = ('cc', 'aa', 'dd', 'bb')
>>> tmp = list(T)           # Make a list from a tuple's items
>>> tmp.sort()              # Sort the list
>>> tmp
['aa', 'bb', 'cc', 'dd']
```

lista-tuplo

```
>>> T = tuple(tmp)          # Make a tuple from the list's items
>>> T
('aa', 'bb', 'cc', 'dd')
```

tuplo-lista

construção de uma lista por compreensão (“*list comprehension*”)

```
>>> T = (1, 2, 3, 4, 5)
>>> L = [x + 20 for x in T]
>>> L
[21, 22, 23, 24, 25]
```

... “*list comprehension*” – tratamentos em compreensão

```
Python Shell
File Edit Debug Options Windows Help
>>> l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l1
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l2 = [ e * 10 for e in l1 if e > 4 ]
>>> l2
[50, 60, 70, 80, 90]
>>>
>>> l1 = range( 1, 10 )
>>> l1
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> dict( [ ('chave1', 22), ('chave2', 'valor2') ] )
{'chave1': 22, 'chave2': 'valor2'}
>>>
>>> tabuadaDo7 = dict( [ ( e, e*7 ) for e in l1 ] )
>>> tabuadaDo7
{1: 7, 2: 14, 3: 21, 4: 28, 5: 35, 6: 42, 7: 49, 8: 56, 9: 63}
>>> tabuadaDo7.items()
[(1, 7), (2, 14), (3, 21), (4, 28), (5, 35), (6, 42), (7, 49), (8, 56), (9, 63)]
>>>
>>> eEsta = [ { e : e[0] + e[1] } for e in tabuadaDo7.items() ]
>>> eEsta
[{(1, 7): 8}, {(2, 14): 16}, {(3, 21): 24}, {(4, 28): 32}, {(5, 35): 40}, {(6, 42): 48},
 {(7, 49): 56}, {(8, 56): 64}, {(9, 63): 72}]
>>> |
```

conversão de uma lista
de tuplos num dicionário!

forma:

[expressão for variável in lista]

OU

[expressão for variável in lista if condição]

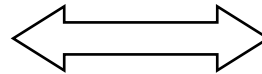
O essencial da “*list comprehension*”

ciclo

```
>>> L = [1, 2, 3, 4, 5]
>>> for i in range(len(L)):
...     L[i] += 10
...
>>> L
[11, 12, 13, 14, 15]
```

“list comprehension”

```
>>> L = [x + 10 for x in L]
>>> L
[21, 22, 23, 24, 25]
```



exemplo – um filtro sobre o conteúdo de ficheiros

```
>>> f = open('script1.py')
>>> lines = f.readlines()
>>> lines
['import sys\n', 'print sys.path\n', 'x = 2\n', 'print 2 ** 33\n']
```

```
>>> lines = [line.rstrip() for line in lines]
>>> lines
['import sys', 'print sys.path', 'x = 2', 'print 2 ** 33']
```

exemplo – sintaxe estendida

```
>>> lines = [line.rstrip() for line in open('script1.py') if line[0] == 'p']
>>> lines
['print sys.path', 'print 2 ** 33']
```

condição a
satisfazer

Trabalhar com ficheiros

Operation	Interpretation
<code>output = open('/tmp/spam', 'w')</code>	Create output file ('w' means write)
<code>input = open('data', 'r')</code>	Create input file ('r' means read)
<code>input = open('data')</code>	Same as prior line ('r' is the default)

Operation	Interpretation
<code>aString = input.read()</code>	Read entire file into a single string
<code>aString = input.read(N)</code>	Read next N bytes (one or more) into a string
<code>aString = input.readline()</code>	Read next line (including end-of-line marker) into a string
<code>aList = input.readlines()</code>	Read entire file into list of line strings
<code>output.write(aString)</code>	Write a string of bytes into file
<code>output.writelines(aList)</code>	Write all line strings in a list into file
<code>output.close()</code>	Manual close (done for you when file is collected)
<code>outout.flush()</code>	Flush output buffer to disk without closing
<code>anyFile.seek(N)</code>	Change file position to offset N for next operation

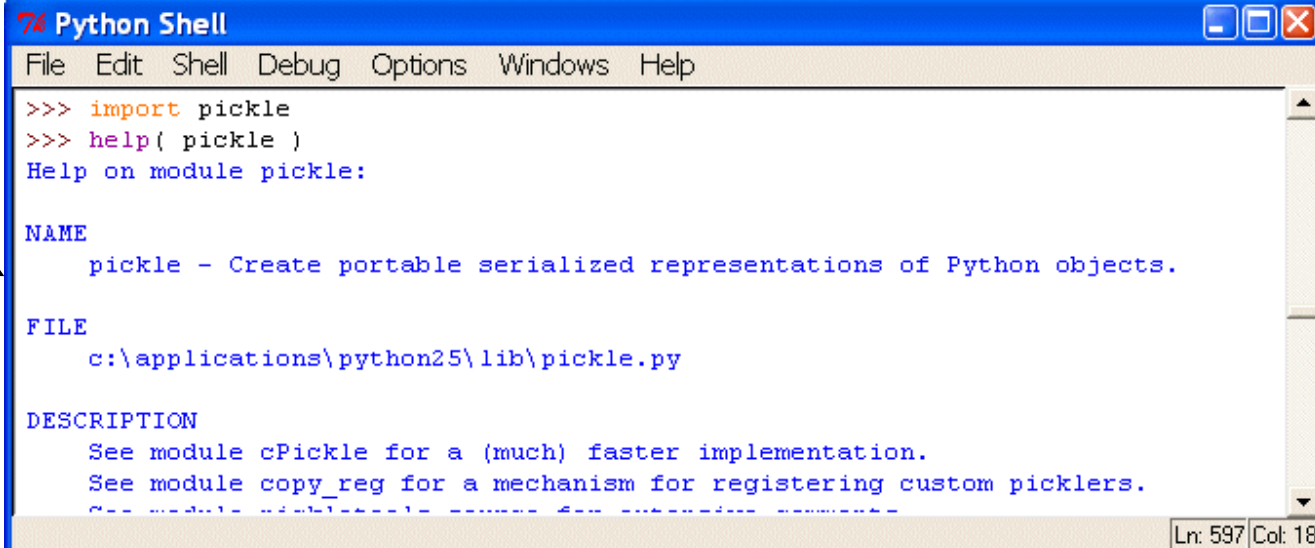
Escrever e ler objectos Python em ficheiros

para seriar objectos (“serialization”) usar o módulo “pickle”
e.g., se tivermos um dicionário (objecto) `D={ 'a' : 1, 'b' : 2 }`

```
>>> F = open('datafile.txt', 'w')
>>> import pickle
>>> pickle.dump(D, F)           # Pickle any object to file
>>> F.close()
```

```
>>> F = open('datafile.txt')
>>> E = pickle.load(F)          # Load any object from file
>>> E
{'a': 1, 'b': 2}
```

para mais
informação



The screenshot shows a 'Python Shell' window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The command prompt shows the user typing `import pickle` and `help(pickle)`. The output displays the help text for the pickle module, including its name, file location, and description. The status bar at the bottom right indicates 'Ln: 597 Col: 18'.

```
Python Shell
File Edit Shell Debug Options Windows Help

>>> import pickle
>>> help( pickle )
Help on module pickle:

NAME
    pickle - Create portable serialized representations of Python objects.

FILE
    c:\applications\python25\lib\pickle.py

DESCRIPTION
    See module cPickle for a (much) faster implementation.
    See module copy_reg for a mechanism for registering custom picklers.
    See module pickletools for information on the pickle protocol.
```

... e agora para ... o essencial para escrever um programa

A escrita de um programa:

Controlo de fluxo

Funções

Classes

Módulos

... e outros aspectos relacionados

Controlo de fluxo: escolha – if

Sintaxe geral:

```
if <test1>:                # if test
    <statements1>          # Associated block
elif <test2>:              # Optional elifs
    <statements2>
else:                      # Optional else
    <statements3>
```

alguns exemplos simples

```
>>> if 1:
...     print 'true'
...
true
```

```
>>> if not 1:
...     print 'true'
... else:
...     print 'false'
...
false
```

```
>>> x = 'killer rabbit'
>>> if x == 'roger':
...     print "how's jessica?"
... elif x == 'bugs':
...     print "what's up doc?"
... else:
...     print 'Run away! Run away!'
...
Run away! Run away!
```

Controlo de fluxo: escolher uma de múltiplas alternativas

Não existe uma instrução para escolher uma entre múltiplas alternativas i.e., não existe o equivalente ao `switch`, de outras linguagens de programação.

Aninhar diversos
`if-elif`.

```
>>> if choice == 'spam':
...     print 1.25
... elif choice == 'ham':
...     print 1.99
... elif choice == 'eggs':
...     print 0.99
... elif choice == 'bacon':
...     print 1.10
... else:
...     print 'Bad choice'
...
1.99
```

Outra técnica **bem mais interessante**:
usar um dicionário;
que até se pode construir em tempo de execução!

```
>>> choice = 'ham'
>>> print {'spam': 1.25,          # A dictionary-based 'switch'
...       'ham': 1.99,           # Use has_key or get for default
...       'eggs': 0.99,
...       'bacon': 1.10}[choice]
1.99
```

```
>>> branch = {'spam': 1.25,
...           'ham': 1.99,
...           'eggs': 0.99}
>>> print branch.get('spam', 'Bad choice')
1.25
>>> print branch.get('bacon', 'Bad choice')
Bad choice
```

Controlo de fluxo: iteração `while`

Sintaxe geral:

```
while <test1>:
    <statements1>
    if <test2>: break           # Exit loop now, skip else
    if <test3>: continue       # Go to top of loop now, to test1
else:
    <statements2>             # Run if we didn't hit a 'break'
```

alguns exemplos simples

```
>>> while True:
...     print 'Type Ctrl-C to stop me!'
```

```
>>> x = 'spam'
>>> while x:                       # While x is not empty
...     print x,
...     x = x[1:]                 # Strip first character off x
...
spam pam am m
```

```
>>> a=0; b=10
>>> while a < b:                   # One way to code counter loops
...     print a,
...     a += 1                    # Or, a = a + 1
...
0 1 2 3 4 5 6 7 8 9
```

Controlo de fluxo: iteração for

Sintaxe geral:

```
for <target> in <object>:           # Assign object items to target
    <statements>
    if <test>: break                 # Exit loop now, skip else
    if <test>: continue             # Go to top of loop now
else:
    <statements>                   # If we didn't hit a 'break'
```

técnica
importante

alguns exemplos simples

```
>>> sum = 0
>>> for x in [1, 2, 3, 4]:
...     sum = sum + x
...
>>> sum
10
>>> prod = 1
>>> for item in [1, 2, 3, 4]: prod *= item
...
>>> prod
24
```

```
>>> S = "lumberjack"
>>> T = ("and", "I'm", "okay")

>>> for x in S: print x,
...
l u m b e r j a c k

>>> for x in T: print x,
...
and I'm okay
```

```
>>> for i in range(3):
...     print i, 'Pythons'
...
0 Pythons
1 Pythons
2 Pythons
```

```
>>> T = [(1, 2), (3, 4), (5, 6)]
>>> for (a, b) in T:
...     print a, b
...
1 2
3 4
5 6
```

Escrever o mecanismo `do-until` com `while-break`

O mecanismo `do-until` garante que o corpo do ciclo se executa pelo menos uma vez.

Isso pode ser importante para alguns problemas.

Por exemplo, para fazer uma pesquisa num espaço de estados é preciso explorar pelo menos o estado inicial.

Para emular o mecanismo `do-until` usa-se uma combinação `while-break`

```
while True:
    ...loop body...
    if exitTest(): break
```

Algumas instruções com efeito colateral em ciclos:

`break` – abandona o âmbito do ciclo

`continue` – salta para o próximo passo do ciclo

`pass` – não faz nada; é uma directiva vazia

O essencial sobre funções

Statement	Examples
Calls	<code>myfunc("spam", "eggs", meat=ham)</code>
def, return, yield	<code>def adder(a, b=1, *c): return a+b+c[0]</code>
global	<code>def changer(): global x; x = 'new'</code>
lambda	<code>Funcs = [lambda x: x**2, lambda x: x*3]</code>

`def` define a função

```
def intersect(seq1, seq2):  
    res = []                # Start empty  
    for x in seq1:          # Scan seq1  
        if x in seq2:       # Common item?  
            res.append(x)   # Add to end  
    return res
```

invocação da função, usando como parâmetro “strings” mas poderiam ser listas!

```
>>> s1 = "SPAM"  
>>> s2 = "SCAM"  
  
>>> intersect(s1, s2)                # Strings  
['S', 'A', 'M']
```

A definição de uma função ocorre em tempo de execução

Tudo é feito em tempo de execução.

Assim, definir uma função corresponde a avaliar a função (“built-in”) `def`

Ou seja, em qualquer momento se pode definir uma função!

Exemplo de definição de uma função quando determinada condição é válida

```
if test:
    def func( ):          # Define func this way
        ...
else:
    def func( ):          # Or else this way
        ...
...
func( )                  # Call the version selected and built
```

Exemplo de atribuição do nome de uma função a uma variável que em seguida se usa para invocar a função

```
othername = func          # Assign function object
othername( )              # Call func again
```

O essencial sobre classes

Definição da classe

variável de classe

construtor

variável de instância

```
class MixedNames:
    data = 'spam'
    def __init__(self, value):
        self.data = value
    def display(self):
        print self.data, MixedNames.data
```

Define class
Assign class attr
Assign method name
Assign instance attr
Instance attr, class attr

Construção de objectos (instâncias) daquela classe

```
>>> x = MixedNames(1)
>>> y = MixedNames(2)
>>> x.display(); y.display()
```

Make two instance objects
Each has its own data
self.data differs, Subclass.data is the same

1 spam
2 spam

... classe e super-classe – relações de herança

```
class Super:
    def method(self):
        print 'in Super.method'
    def delegate(self):
        self.action()

class Inheritor(Super):
    pass

class Replacer(Super):
    def method(self):
        print 'in Replacer.method'

class Extender(Super):
    def method(self):
        print 'starting Extender.method'
        Super.method(self)
        print 'ending Extender.method'

class Provider(Super):
    def action(self):
        print 'in Provider.action'
```

Diversas técnicas

comportamento do pai

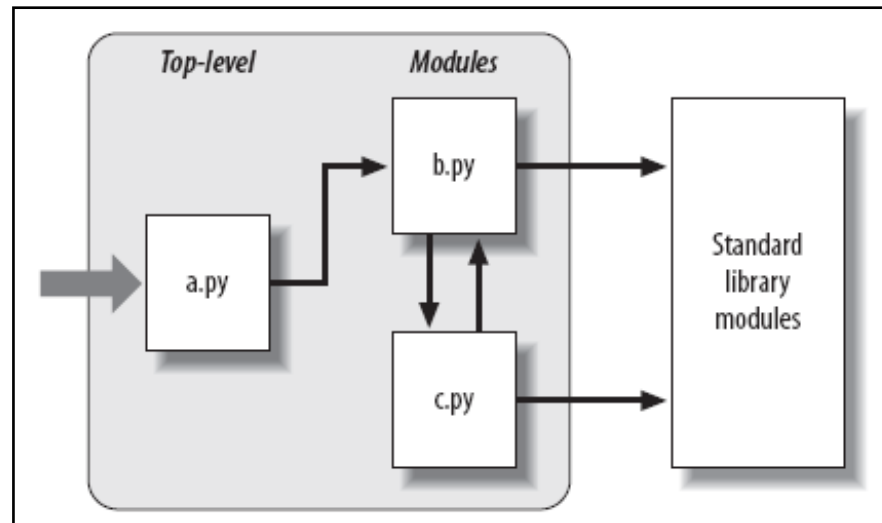
aceita tudo do
comportamento do pai

redefine
comportamento do pai

estende
comportamento do pai

implementa
comportamento
delegado pelo pai

O essencial sobre a estrutura de um programa Python



```
import b
b.spam('gumby')
```

a.py

```
def spam(text):
    print text, 'spam'
```

b.py

```
import b
```

≡ “se o módulo `b` ainda não foi carregado, então carrega-o e dá acesso ao que lá está definido através do nome `b`.”

Como funciona o `import` ?

```
import a
```

1. **procurar** o ficheiro `a.py`
2. **traduzir**, se necessário, para “byte code” `a.pyc`
3. **executar** o código para construir os objectos aí definidos.

... o procedimento de procura de um módulo

procurar

1. directório corrente (“home directory of the program”)
2. directorias em `PYTHONPATH` (se estiver definida)
3. directorias das bibliotecas standard (`lib` no directório de instalação)
4. o conteúdo de qualquer ficheiro `.pth` (caso exista)

Nota: a variável `sys.path` contém a concatenação dos pontos anteriores.

```
>>> import sys
>>> sys.path
['C:\\Applications\\Python25\\Lib\\idlelib', 'C:\\IBMTTOOLS\\utils\\support', 'C:\\IBMTTOOLS\\utils\\logger', 'C:\\WINDOWS\\system32\\python25.zip', 'C:\\Applications\\Python25\\DLLs', 'C:\\Applications\\Python25\\lib', 'C:\\Applications\\Python25\\lib\\plat-win', 'C:\\Applications\\Python25\\lib\\lib-tk', 'C:\\Applications\\Python25', 'C:\\Applications\\Python25\\lib\\site-packages']
>>> |
```

... o procedimento de tradução de um módulo

traduzir

1. se não existir código fonte (`.py`) considerar apenas o “byte code” (`.pyc`)
2. se apenas existir código fonte (`.py`) construir o “byte code” (`.pyc`)
3. se existir `.py` e `.pyc` apenas traduzir `.py` caso este tenha sido alterado desde a sua última tradução (comparar “timestamps”)

Nota: a tradução ocorre no momento em que o módulo é importado; assim não é comum ver o `.pyc` do módulo de topo (não é importado por outros).

... o procedimento de execução de um módulo

executar

1. enquanto existirem instruções no ficheiro `.pyc`
2. executar a instrução

Por exemplo as instruções `def` apenas definem funções.

Posteriormente as funções poderão ser invocadas pelo módulo que as importou

`import m` `e` `from m import x` `e` `from m import *`

```
def printer(x):  
    print x
```

Module attribute

`module1.py`

```
>>> import module1  
>>> module1.printer('Hello world!')
```

Get module as a whole

Qualify to get names

necessário qualificar

```
>>> from module1 import printer  
>>> printer('Hello world!')
```

Copy out one variable

No need to qualify name

indicar o que se pretende
desnecessário qualificar

```
>>> from module1 import *  
>>> printer('Hello world!')
```

Copy out all variables

não indicar o que se pretende
desnecessário qualificar

Módulo que pode ser: de topo ou importado

É possível escrever código que pode servir:

- como ficheiro de topo (que importa módulos)
- como módulo a ser importado por outros

Para isso é preciso usar dois atributos especiais: `__name__` e `__main__`

```
def tester():  
    print "It's Christmas in Heaven..."  
  
if __name__ == '__main__':  
    tester()
```

`runme.py`

módulo importado
e função invocada

```
% python  
>>> import runme  
>>> runme.tester()  
It's Christmas in Heaven...
```

módulo executado
directamente

```
% python runme.py  
It's Christmas in Heaven...
```

Construir e obter documentação

Form	Role
# comments	In-file documentation
The <code>dir</code> function	Lists of attributes available in objects
Docstrings: <code>__doc__</code>	In-file documentation attached to objects
PyDoc: The <code>help</code> function	Interactive help for objects
PyDoc: HTML reports	Module documentation in a browser
Standard manual set	Official language and library descriptions
Web resources	Online tutorials, examples, and so on
Published books	Commercially available reference texts

≡ marca de início de comentário;
cada instrução escreve-se numa linha pelo
que também só existem comentários de linha

... função `dir`

Apresenta uma lista de todos os atributos disponíveis num objecto.
Apenas se pode invocar sobre um objecto que tenha atributos.

```
>>> import sys
>>> dir(sys)
['_displayhook__', '__doc__', '__excepthook__', '__name__',
'__stderr__', '__stdin__', '__stdout__', '_getframe', 'argv',
'builtin_module_names', 'byteorder', 'copyright', 'displayhook',
'dllhandle', 'exc_info', 'exc_type', 'excepthook',
...more names omitted...]
```

... comentários `"""x"""` e atributo `__doc__`

Comentários no início do módulo e no início de cada função delimitados por `"""` e `"""` são atribuídos a `__doc__`

```
"""
Module documentation
Words Go Here
"""

spam = 40

def square(x):
    """
    function documentation
    can we have your liver then?
    """
    return x **2

class employee:
    "class documentation"
    pass

print square(4)
print square.__doc__
```

```
>>> import docstrings
16

    function documentation
    can we have your liver then?

>>> print docstrings.__doc__

Module documentation
Words Go Here

>>> print docstrings.square.__doc__

    function documentation
    can we have your liver then?

>>> print docstrings.employee.__doc__
class documentation
```

... a função `help`

A função `help` simplifica a utilização de `__doc__`

A função `help` está definida no módulo “built-in” `PyDoc`

O `PyDoc` fornece uma interface gráfica que simplifica a análise dos módulos

```
>>> help(dict)
Help on class dict in module __builtin__:

class dict(object)
| dict() -> new empty dictionary.
| ...more omitted...

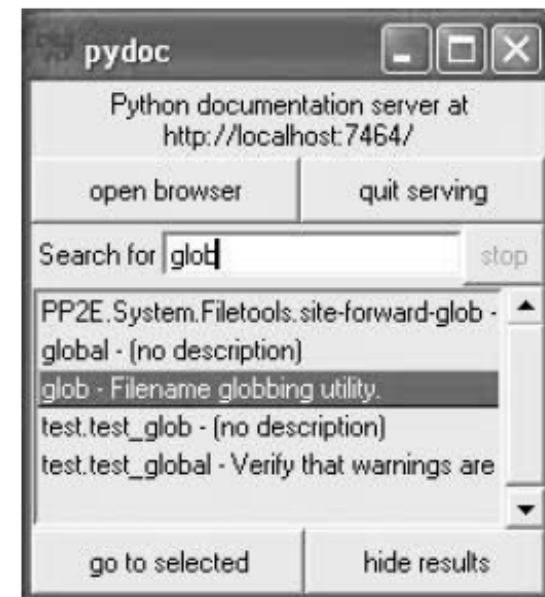
>>> help(str.replace)
Help on method_descriptor:

replace(...)
    S.replace(old, new[, maxsplit]) -> string

    Return a copy of string S with all occurrences
    ...more omitted...

>>> help(ord)
Help on built-in function ord:

ord(...)
    ord(c) -> integer
```



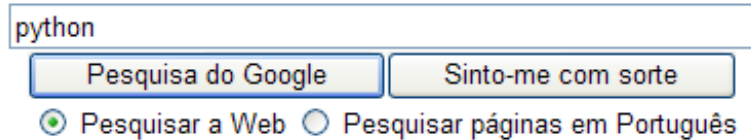
Informação adicional

Mark Lutz; “Learning Python”; 3rd edition; Out.2007; O’ Reilly.

Swaroop; “A Byte of Python”; 2005 (não comercial)

Ambos os documentos foram usados nesta apresentação

E ainda...



A screenshot of a Google search interface. The search bar contains the text "python". Below the search bar are two buttons: "Pesquisa do Google" and "Sinto-me com sorte". Below these buttons are two radio buttons: "Pesquisar a Web" (which is selected) and "Pesquisar páginas em Português".