

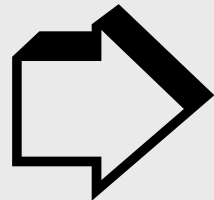


2º CURSO DE PYTHON

uma iniciativa **evolux**

Lucas
Castro

evolux

Evolux  Call Center

evolux



Evolux ➡ Call Center

evolux



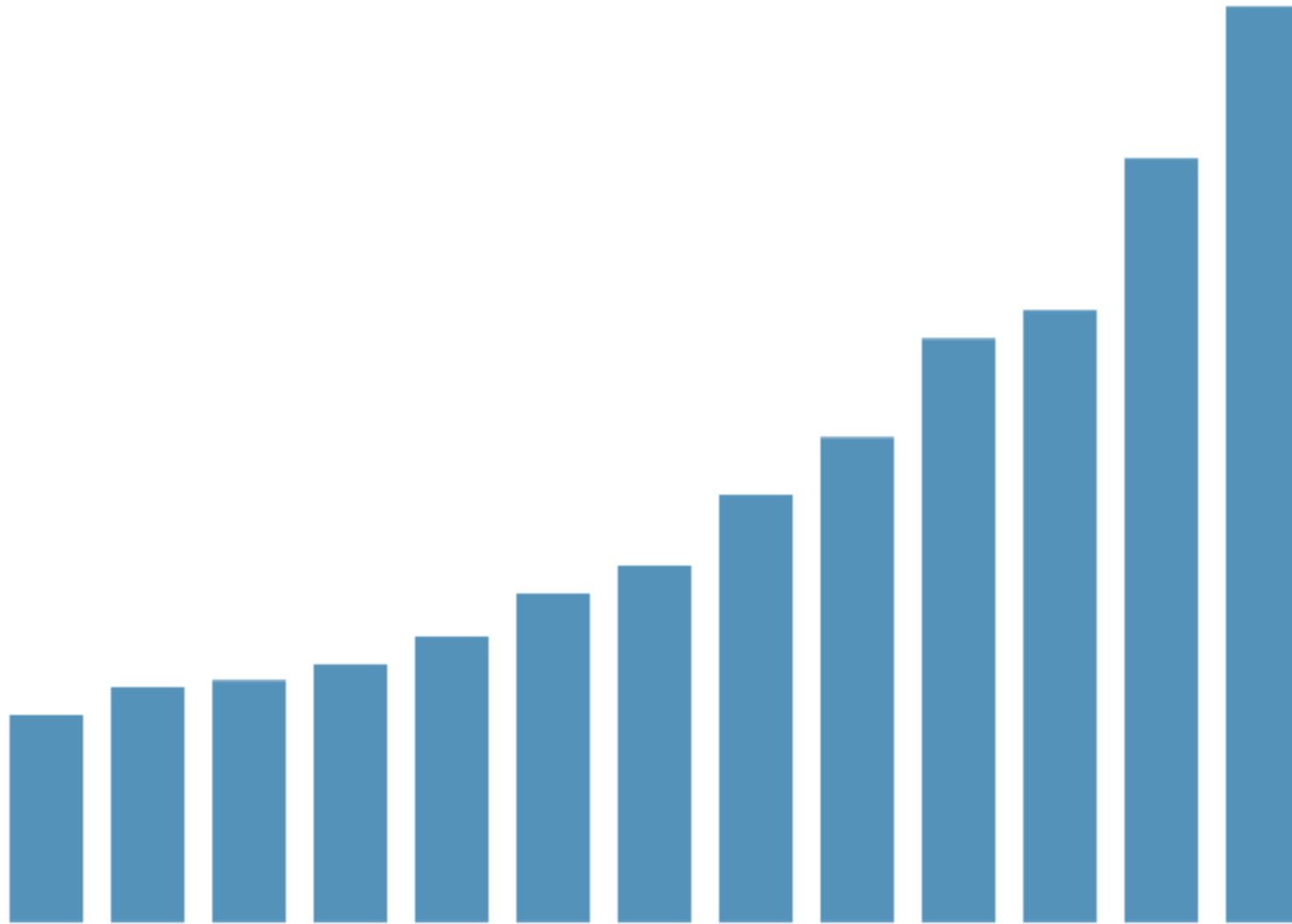
Evolux Call Center



WE ARE

evolux

WE ARE



evolux

WE ARE

evolux

WE ARE

HIRING

evolux

Python

Guido Van Rossum

Guido Van Rossum



Comunidade Brasileira

Luciano Ramalho

Luciano Ramalho



Fernando Masanori

Fernando Masanori



evolux

Oswaldo Santana

Oswaldo Santana



evolux

Rodrigo Senra

Rodrigo Senra



Propósito Genérico

Alto Nivel

evolux

Múltiplos Paradigmas

Múltiplos Paradigmas

 **imperativo**

evolux

Múltiplos Paradigmas

⚛ imperativo ⚛ funcional

Múltiplos Paradigmas

✱ imperativo ✱ funcional
✱ orientado a objetos

Legibilidad

Tipagem Dinâmica e Forte

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.



Dropbox

Google™



evolux

Sublime Text

Mercurial

Cinema 4D

Bit Bucket

rdio

OpenStack

evolux

RECAP

RECAP

Python is
AWESOME

evolux



Python 2.7

Stable

evolux



Python 2.7

Stable

```
$ python --version  
Python 2.7.3
```

CPython

PyPy (just-in-time)

Jython Iron Python

```
$ which python  
/usr/bin/python
```

```
$ sudo easy_install pip  
$ which pip  
/usr/bin/pip
```

```
$ pip install flask  
$ pip freeze  
Fabric==1.6.1  
Flask==0.10.1  
...
```

```
$ python
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
...
```


docs.python.org/2.7/

The screenshot shows the Python 2.7.5 documentation page. The left sidebar contains navigation links: 'Download' (with a sub-link 'Download these documents'), 'Docs for other versions' (listing 'Python 3.3 (stable)', 'Python 3.4 (in development)', and 'Old versions'), 'Other resources' (listing 'PEP Index', 'Beginner's Guide', 'Book List', and 'Audio/Visual Talks'), and 'Quick search' (with a search bar and a 'Go' button). The main content area is titled 'Python v2.7.5 documentation' and includes a welcome message: 'Welcome! This is the documentation for Python 2.7.5, last updated Oct 18, 2013.' Below this, a section 'Parts of the documentation:' lists several links with descriptions: 'What's new in Python 2.7?' (or all 'What's new' documents since 2.0), 'Tutorial' (start here), 'Library Reference' (keep this under your pillow), 'Language Reference' (describes syntax and language elements), 'Python Setup and Usage' (how to use Python on different platforms), 'Python HOWTOs' (in-depth documents on specific topics), 'Extending and Embedding' (tutorial for C/C++ programmers), 'Python/C API' (reference for C/C++ programmers), 'Installing Python Modules' (information for installers & sys-admins), 'Distributing Python Modules' (sharing modules with others), and 'FAQs' (frequently asked questions (with answers!)).

Python » 2.7.5 » Documentation »

Download

Download these documents

Docs for other versions

- Python 3.3 (stable)
- Python 3.4 (in development)
- Old versions

Other resources

- PEP Index
- Beginner's Guide
- Book List
- Audio/Visual Talks

Quick search

Enter search terms or a module, class or function name.

Python v2.7.5 documentation

Welcome! This is the documentation for Python 2.7.5, last updated Oct 18, 2013.

Parts of the documentation:

- What's new in Python 2.7?**
or all "What's new" documents since 2.0
- Tutorial**
start here
- Library Reference**
keep this under your pillow
- Language Reference**
describes syntax and language elements
- Python Setup and Usage**
how to use Python on different platforms
- Python HOWTOs**
in-depth documents on specific topics
- Extending and Embedding**
tutorial for C/C++ programmers
- Python/C API**
reference for C/C++ programmers
- Installing Python Modules**
information for installers & sys-admins
- Distributing Python Modules**
sharing modules with others
- FAQs**
frequently asked questions (with answers!)

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

Caminho para interpretador

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"  
  
    foo(5, 10)  
  
    print '=' * 5  
    print 'Diretório atual é ' + os.getcwd()
```

Caminho para interpretador

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!  
    print "aspas simples" dentro de aspas duplas"
```

```
foo(5, 10)
```

```
print '=' * 5
```

```
print 'Diretório atual é ' + os.getcwd()
```

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"  
  
    foo(5, 10)  
  
    print '=' * 5  
    print 'Diretório atual é ' + os.getcwd()
```

Import carrega um
módulo

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
```

```
    print 'Hello World!'
```

```
    print " 'espas simples' dentro de aspas duplas"
```

```
    foo(5, 10)
```

```
print '=' * 5
```

```
print 'Diretório atual é ' + os.getcwd()
```

Import carrega um
módulo

O nome `main` é apenas
uma convenção

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```



```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"  
  
    foo(5, 10)  
  
    print '=' * 5  
    print 'Diretório atual é ' + os.getcwd()
```

Não há ponto e
vírgula

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"
```

```
foo(5, 10)
```

Não há chaves. Ao invés, indentação

Não há ponto e vírgula

```
+ os.getcwd()
```

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"  
  
    foo(5, 10)  
  
    print '=' * 5  
    print 'Diretório atual é ' + os.getcwd()
```

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

print automaticamente
adiciona nova linha

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"
```

print automaticamente
adiciona nova linha

o nome `main` é apenas
uma convenção

```
foo(5, 10)
```

```
print '=' * 5
```

```
print 'O caminho atual é ' + os.getcwd()
```

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

foo(5, 10) é uma
chamada de função

```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

```
print '=' * 5 irá
mostrar: '====='
```

foo(5, 10) é uma
chamada de função


```
#!/usr/bin/python
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():
    print 'Hello World!'
    print "'aspas simples' dentro de aspas duplas"

    foo(5, 10)

    print '=' * 5
    print 'Diretório atual é ' + os.getcwd()
```

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"  
  
    foo(5, 10)  
  
    print '=' * 5  
    print 'Diretório atual é ' + os.getcwd()
```

+ para concatenação
de strings

```
#!/usr/bin/python  
# -*- coding: utf8 -*-
```

```
import os
```

```
def main():  
    print 'Hello World!'  
    print "'aspas simples' dentro de aspas duplas"  
  
    foo(5, 10)  
  
    print '=' * 5  
    print 'Diretório atual é ' + os.getcwd()
```

+ para concatenação
de strings

os.getcwd() chama
função do módulo

```
counter = 0
counter += 1

fruits = ['manga', 'pitomba', 'caju']

for f in fruits:
    | print 'Eu gosto de comer ' + f

print 'Contando números:'
for i in range(5):
    | print i
```

```
counter = 0  
counter += 1
```

```
fruits = ['manga', 'pitomba', 'figo']
```

```
for f in fruits:  
    print 'Eu gosto de comer ' + f
```

```
print 'Contando números:'  
for i in range(5):  
    print i
```

counter precisa ser
inicializada primeiro

```
counter = 0  
counter += 1
```

```
fruits = ['manga', 'pitomba', 'figo']  
  
for f in fruits:  
    print 'Eu gosto de comer ' + f
```

counter precisa ser
inicializada primeiro

foods é uma lista
com três strings

```
counter = 0
counter += 1

fruits = ['manga', 'pitomba', 'caju']

for f in fruits:
    | print 'Eu gosto de comer ' + f

print 'Contando números:'
for i in range(5):
    | print i
```

```
counter = 0  
counter += 1
```

```
fruits = ['manga', 'pitomba', 'jujuba']
```

```
for f in fruits:  
    print 'Eu gosto de comer ' + f
```

```
print 'Contando números:'  
for i in range(5):  
    print i
```

f leva uma fruta a
cada iteração


```
counter = 0  
counter += 1
```

```
fruits = ['manga', 'pitomba', 'juá']
```

```
for f in fruits:  
    print 'Eu gosto de comer ' + f
```

f leva uma fruta a
cada iteração

bloco acaba quando
identação recua

```
counter = 0
counter += 1

fruits = ['manga', 'pitomba', 'caju']

for f in fruits:
    | print 'Eu gosto de comer ' + f

print 'Contando números:'
for i in range(5):
    | print i
```

```
counter = 0  
counter += 1
```

```
fruits = ['manga', 'pitomba', 'abacaxi']
```

```
for f in fruits:  
    print 'Eu gosto de comer ' + f
```

```
print 'Contando números:'  
for i in range(5):  
    print i
```

range(5) retorna
lista [0,1,2,3,4]

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

foo é uma outra
função

```
def foo(param1, param2):  
    res = param1 + param2
```

```
    print '%s mais %s é igual a %s' % (param1, param2, res)
```

```
    if res < 50:
```

```
        print 'foo'  
    elif (res >= 50) and (param1 == 42) or (param2 == 24):  
        print 'bar'  
    else:  
        print 'moo'
```

foo é uma outra
função

foo está no mesmo
escopo que main

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

comparações funcionam
como em C


```
def foo(param1, param2):  
    res = param1 + param2
```

```
    print '%s mais %s é igual a %s' % (param1, param2, res)
```

```
    if res < 50:
```

```
        print 'foo'  
        if res >= 50 and (param1 == 42) or (param2 == 24):  
            print 'bar'  
        else:  
            print 'moo'
```

comparações funcionam
como em C

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

Operadores booleanos
and e or são palavras

```
def foo(param1, param2):
    res = param1 + param2

    print '%s mais %s é igual a %s' % (param1, param2, res)

    if res < 50:
        | print 'foo'
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):
        print 'bar'
    else:
        print 'moo'
```

Operadores booleanos
and e or são palavras

operadores && e ||
não funcionam

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

interpolação strings funciona como em C

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

interpolação strings funciona como em C

```
def foo(param1, param2):  
    res = param1 + param2  
  
    print '%s mais %s é igual a %s' % (param1, param2, res)  
  
    if res < 50:  
        | print 'foo'  
    elif (res >= 50) and ((param1 == 42) or (param2 == 24)):  
        | print 'bar'  
    else:  
        | print 'moo'
```

: após def, for,
while, if, elif, else

```
| return res # Este é um comentário de uma linha  
| '''Esta é uma string multi-linha  
| mas também pode ser um comentário multi-linha.'''
```

```
if __name__ == '__main__':  
|     main()
```


comentários

```
| return res # Este é um comentário de uma linha  
| '''Esta é uma string multi-linha  
| mas também pode ser um comentário multi-linha.'''
```

```
| if __name__ == '__main__':  
|     main()
```

comentários

```
| return res # Este é um comentário de uma linha  
| '''Esta é uma string multi-linha  
| mas também pode ser um comentário multi-linha.'''
```

```
| if __name__ == '__main__':  
|     main()
```

boilerplate
para rodar main



2º CURSO DE PYTHON