

# PostgreSQL + Python/Django – Uma Parceria de Sucesso

Marcos Thomaz da Silva



# Perfil do Instrutor

- Graduação em Sistemas de Informação
- Especialização em Bancos de Dados
- Analista de Tecnologia da Informação da Universidade Federal do Acre
- Desenvolvedor Clipper, Delphi, PHP e Python
- Participante e Moderador da Lista Django Brasil;
- Entusiasta PostgreSQL



# Python



- Criada por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação de Amsterdam;
- Nome originado da série britânica Monty Python Flying Circus;
- Lançada em 1991;
- Linguagem de alto nível;
- Interpretada;
- Gera *bytecodes* (*pyc*, *pyo*);



# Python

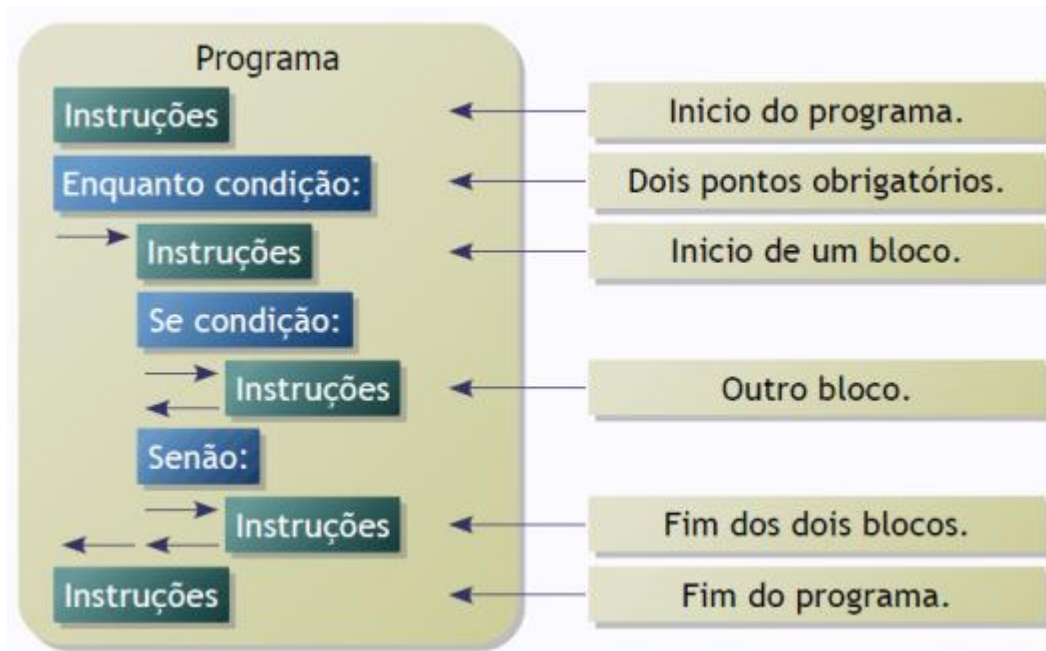
- Multiplataforma;



# Python



- Endentação como delimitador de blocos;



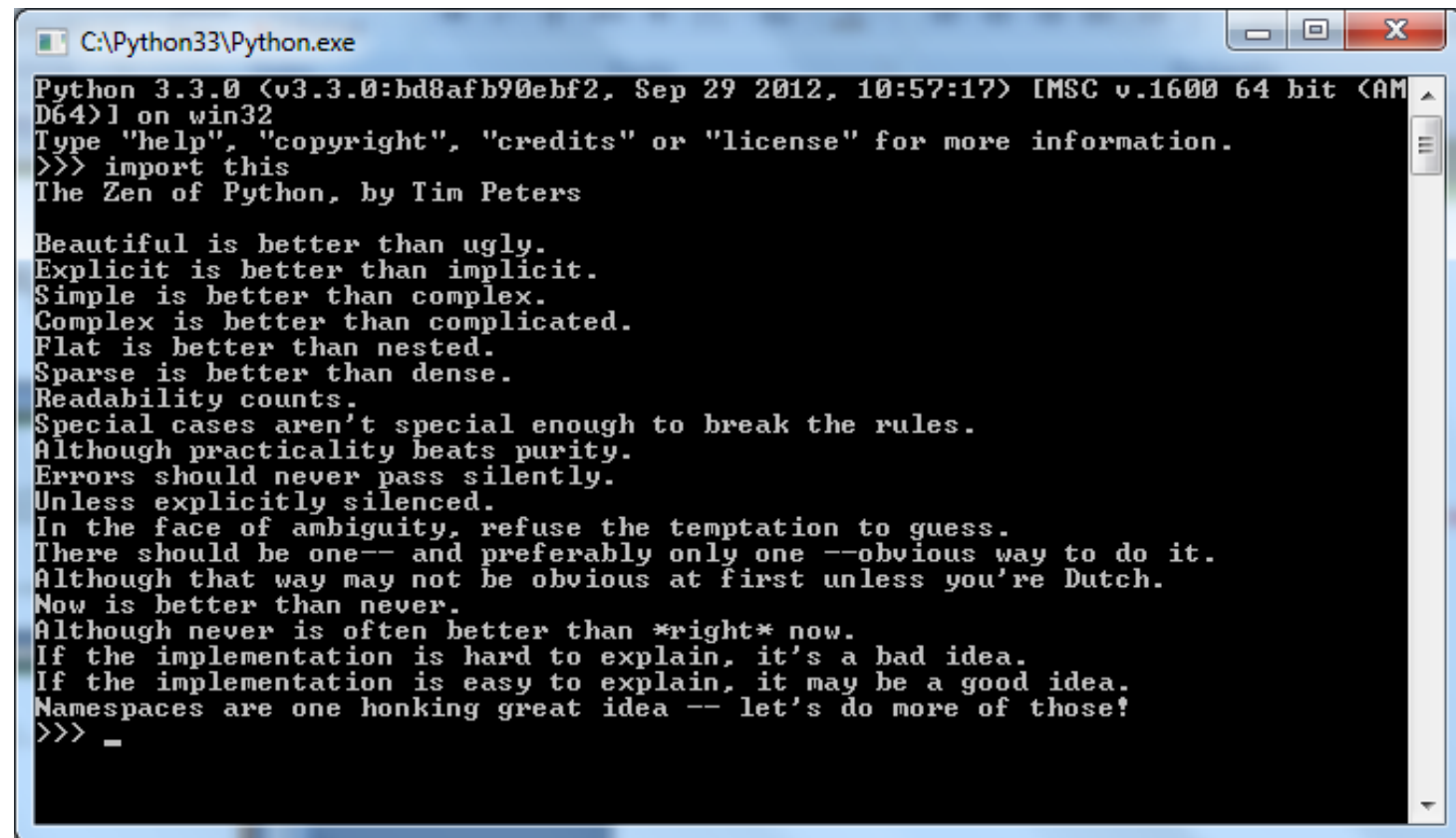
```
1 print("Imprimindo tabuada do 8")
2 for i in range(1, 11):
3     j = i * 8
4     print("8 x ", i, " = ", j)
5 print("Simples assim")
```

← **Início do bloco**

# Python



- Possui Interpretador Interativo;

A screenshot of a Windows command prompt window titled "C:\Python33\Python.exe". The window shows the Python 3.3.0 interactive shell. The prompt is "Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit (AMD64)] on win32". The user has entered the command ">>> import this", and the shell has displayed the "Zen of Python" text. The text is as follows:

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 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!
>>> _
```

# Python

- Tipagem dinâmica e forte;



```
>>>
>>> a = 1
>>> b = "Olá"
>>> c = True
>>> a = "Oi Mundo"
>>> b = 2
>>> c = False
>>> d = 4
>>> print( d )
4
>>> print( c )
False
>>> print( b )
2
>>> print( a )
Oi Mundo
>>> a
'Oi Mundo'
>>> b
2
>>> c
False
>>> d
4
>>> b + d
6
>>> b + a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> _
```



# Python

- Orientada a Objetos;
  - Herança (inclusive herança múltipla);
  - Polimorfismo;
  - Suporte a metaclasses;
- Tratamento de Exceções;
- Trabalha nativamente com tipos “primitivos” e estruturas complexas (listas, dicionários, tuplas);
- Introspecção (Reflection);



```
>>> class A:
    def f(self):
        return 'F in A'
    def f1(self):
        return 'F1'

>>> class B:
    def f(self):
        return 'F in B'
    def f2(self):
        return 'F2'

>>> class C(A,B):
    pass

>>> class D(B,A):
    pass

>>> dir(A)
['__doc__', '__module__', 'f', 'f1']
>>> dir(B)
['__doc__', '__module__', 'f', 'f2']
>>> dir(C)
['__doc__', '__module__', 'f', 'f1', 'f2']
>>> dir(D)
['__doc__', '__module__', 'f', 'f1', 'f2']
>>> c, d = C(), D()
>>> c.f()
'F in A'
>>> d.f()
'F in B'
```

# Python



```
>>> a = 'oi mundo p0stgresql!'
>>> a.lower()
'oi mundo postgresql!'
>>> a.upper()
'OI MUNDO POSTGRESQL!'
>>> a.capitalize()
'Oi mundo postgresql!'
>>> a.count("o")
2
>>> a.count("O")
1
>>> a.upper().count("O")
3
```

```
>>> a.center(30)
'      oi mundo p0stgresql      '
>>> a.center(30,'=')
'====oi mundo p0stgresql===='
>>> a.title()
'Oi Mundo Postgresql!'
>>> a.isupper()
False
>>> 'OI'.isupper()
True
>>> a.swapcase()
'OI MUNDO PoSTGRESQL!'
```

```
>>> a.rjust(30,'=')
'=====oi mundo p0stgresql!'
>>> a.ljust(30,'=')
'oi mundo p0stgresql!===== '
>>> b = a + ' '
>>> b
'oi mundo p0stgresql! '
>>> a.strip()
'oi mundo p0stgresql!'
>>> a.split()
['oi', 'mundo', 'p0stgresql!']
>>> c = '<->'.join( a.split() )
>>> c
'oi<->mundo<->p0stgresql!'
>>> a.replace('!', '!!!!')
'oi mundo p0stgresql !!!!!'
```

# Python



- Sobrecarga de operadores;

```
>>> class MyStr(str):
    def __add__(self, v1):
        if isinstance(v1, int):
            return self + str(v1)
        elif isinstance(v1, str) and v1.count('database')>0:
            return str(self) + v1.replace('database', 'database (PostgreSQL is better)')
        else:
            return super(MyStr, self).__add__(v1)

>>> a = MyStr('Hello World!')
>>> a + 127
'Hello World!127'
>>> a
'Hello World!'
>>> a + 'Choice your database:'
'Hello World!Choice your database (PostgreSQL is better):'
>>>
```

```
>>> class Aluno:
    def __init__(self, nome=None):
        self.nome = nome if nome else ''
    def __str__(self):
        return self.nome.title()
    @property
    def primeiro_nome(self):
        if self.nome.count(' ')>0:
            return self.nome.split(' ')[0]
        return self.nome
    def __repr__(self):
        return '<Aluno: '+self.nome+'>'

>>> class Turma:
    def __init__(self, *args):
        self.alunos = []
        for i in args:
            if isinstance(i, Aluno):
                self = self + i
    def matricular(self, aluno):
        if aluno in self.alunos:
            print("Aluno ja matriculado - %s"%aluno)
        else:
            self.alunos.append( aluno )
            print("Matricula do aluno %s efetuada com sucesso"%aluno)
    def __add__(self, v):
        if isinstance(v, Aluno):
            self.matricular(v)
        return self
    def __repr__(self):
        return '<Turma: [' + ', '.join([str(a) for a in self.alunos]) + ']>'
```

```
>>> joao = Aluno('joao silva')
>>> marcos = Aluno('marcos thomaz')
>>> alfredo = Aluno('alfredo marques')
>>> joao
<Aluno: joao silva>
>>> turma = Turma()
>>> turma
<Turma: []>
>>> turma.matricular(joao)
Matricula do aluno Joao Silva efetuada com sucesso
>>> turma + marcos
Matricula do aluno Marcos Thomaz efetuada com sucesso
<Turma: [Joao Silva, Marcos Thomaz]>
>>> turma + joao
Aluno ja matriculado - Joao Silva
<Turma: [Joao Silva, Marcos Thomaz]>
>>> turma.matricular(marcos)
Aluno ja matriculado - Marcos Thomaz
>>> turma
<Turma: [Joao Silva, Marcos Thomaz]>
>>> turma = Turma(joao, marcos, joao, alfredo)
Matricula do aluno Joao Silva efetuada com sucesso
Matricula do aluno Marcos Thomaz efetuada com sucesso
Aluno ja matriculado - Joao Silva
Matricula do aluno Alfredo Marques efetuada com sucesso
>>> turma
<Turma: [Joao Silva, Marcos Thomaz, Alfredo Marques]>
```



# Python - Interoperabilidade



- Jython
- CPython
- IronPython
- PyPy
- PyObjC (Mac OSX middleware)
- Python for Delphi
- Brython

# Onde / quando usar python



- Geração de scripts;
  - Suporte a administradores de redes;
  - Apoio a outros sistemas, etc;
- Acesso a bancos de dados;
  - Suporte a praticamente todos os bancos de dados – excelente com PostgreSQL (psycopg2);
- Desenvolvimento web;
  - Existência de diversos frameworks: Django, Pyramid, Web2Py;
- Ferramenta de Gerenciamento de Conteúdo (Plone);

# Onde / quando usar python



- Desenvolvimento Desktop;
  - Uso de GTK, QT, wxWindow, etc;
- Diversos Editores de Código;
  - Pagos: Pycharm, Sublime Text, Komodo, WingIDE, Ninja IDE;
  - Livres: PyDev (Eclipse), Eric, Pida, Boa-Constructor, Notepad++, Vim;
- Desenvolvimento de ERP's:
  - OpenERP, Stoq;
- Desenvolvimento de Jogos:
  - PyGame, PyOpenGL

# Onde / quando usar python



- Área Científica: Scipy, NumPy;
- Aplicações Geo: Mapproxy (com Postgis);
- Inteligência Artificial;
- Animações 3D (Blender);
- Aplicativos Móveis ;





# Python – Quem usa?



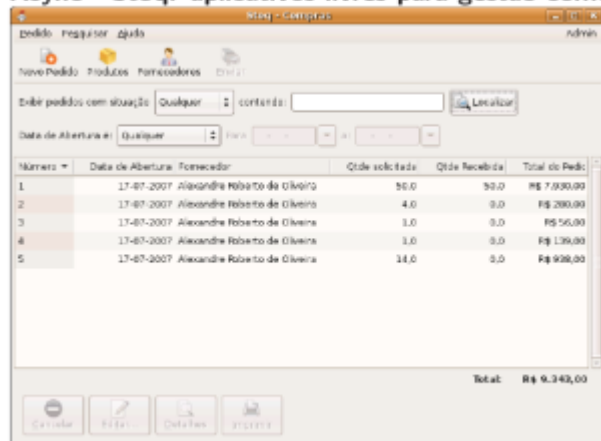
# Python – Quem Usa



INdT - Instituto Nokia de Tecnologia



Async - Stoq: aplicativos livres para gestão comercial

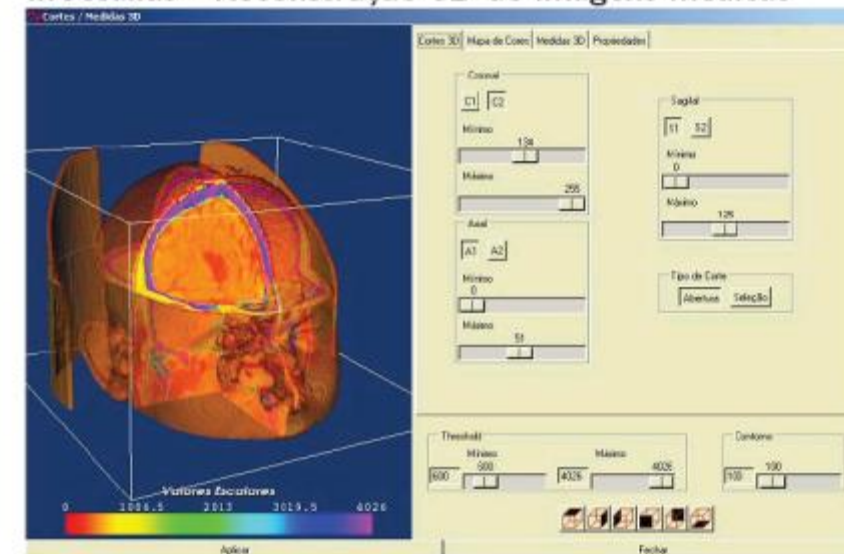


<http://www.async.com.br>

LZT - AutoSystem: automação de postos de combustível



InVesalius - Reconstrução 3D de imagens médicas






# Python – Quem Usa



# Python – site www.python.org



[Advanced Search](#)

ABOUT >>

NEWS >>

DOCUMENTATION >>

DOWNLOAD >>

下载 >>

COMMUNITY >>

FOUNDATION >>

CORE DEVELOPMENT >>

Help

Package Index

Quick Links (2.7.5)

- >> Documentation
- >> Windows Installer
- >> Source Distribution

Quick Links (3.3.2)

- >> Documentation
- >> Windows Installer
- >> Source Distribution

Python Jobs

Python Merchandise

Python Wiki

## Python Programming Language – Official Website

**Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.**

Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.

Python is free to use, even for commercial products, because of its OSI-approved [open source license](#).

New to Python or choosing between Python 2 and Python 3? Read [Python 2 or Python 3](#).

The [Python Software Foundation](#) holds the intellectual property rights behind Python, underwrites the [PyCon conference](#), and funds many other projects in the Python community.

[Read more](#), -or- [download Python now](#)

» **EuroPython 2014/2015 Conference Team Call for Site Proposals**  
The EuroPython Society announced [the Call for Proposals for EuroPython](#), to collect proposals from teams volunteering to organize the EuroPython conference in 2014-2015.  
Published: Thu, 13 June 2013, 11:43 -0400

» **Python 3.2.5 and 3.3.2 have been released**  
[Python 3.2.5](#) and [Python 3.3.2](#) regression fix releases have been released.  
Published: Wed, 15 May 2013, 22:30 +0100

» **Python 2.7.5 released**

Support the Python Community

Help the Python community by becoming an [associate member](#) or making a [one-time donation](#).

Python 3 Poll


I wish there was Python 3 support in

(enter [PyPI package name](#))

[Results](#)

Carmanah lights the way with Python

Welcome



Introducing the EverGEN 1500.3 Parking Lot Light. All the power of a 1500-watt generator.

... joining users such as [Packages](#), [Industrial Light and](#)

# Python – site: [www.python.org.br](http://www.python.org.br)



Logado como [\(Entrar\)](#)

[associação](#) [pythonbrasil\[9\]](#) [django](#) [zope/plone](#) [planet](#)



Títulos

Texto

## Impressione-se »

Python pode ser usada em diversos tipos de desenvolvimento.

## Inicie-se »

Python pode ser usada livremente por qualquer desenvolvedor.

## Aprenda mais »

Python tem uma sintaxe muito simples e também é fácil de aprender.

## Envolve-se »

Participe da comunidade brasileira de desenvolvedores Python.

## PythonBrasil

Brasília

30 de setembro a 6 de outubro

A comunidade Python Brasil reúne grupos de usuários em todo o Brasil interessados em difundir e divulgar a linguagem de programação [Python](#), abrigando em seu domínio todo o material editado sobre esta linguagem em nosso idioma (português do Brasil). Ele utiliza o software [MoinMoin](#) que é um [Wiki](#) inteiramente escrito em Python. Esse sistema foi escolhido por sua facilidade de uso e por ser uma ferramenta bastante democrática, permitindo que se desenvolva conteúdo rapidamente.

## Notícias da PythonBrasil



### Veja também:

- [PythonBrasil](#)
- [MudançasRecentes](#)
- [IndiceDeTítulos](#)
- [DocumentacaoPython](#)
- [CookBook](#)
- [OutrasSecoes](#)

[Planet PythonBrasil](#)

[Mais...](#)



**Psycopg / Psycopg2**

# Psycopg2



- É o mais popular adaptador Python para PostgreSQL;
- Oferece acesso a praticamente todos os recursos do PostgreSQL;
- Liberado sob os termos LGPL;
- Atualmente na versão 2.5.1;
- Multiplataforma;
- Instalação:
  - Windows: Binário;
  - Linux / Mac: Pacote;

# Psycopg2



- Escrito a maior parte em C;
- Faz uso da libpq (objetivando melhor desempenho e segurança);
- Suporte a todas as versões do Python atuais;
- Suporte a todas as versões do PostgreSQL (acima da 7);
- Implementação totalmente compatível com as especificações de adaptadores de Bancos de Dados Python;
- Thread-safe: as threads podem compartilhar a mesma conexão ou usar conexões diferentes;
- Possui adaptação de diversos tipos de dados python;



# Psycopg2



- Possui typecasters que convertem tipos do PostgreSQL para objetos python;
- Possui cursores Server-Side;
- Suporte ao comando COPY;
- Pode enviar e receber notificações assíncronas;
- Suporta commit em duas fases;
- Permite a conversão de tipos decimal / numeric (PostgreSQL) para tipos float ou Decimal (Python);

# Psycopg2 – Alguns Exemplos



- Conectando a uma base de dados

```
>>> import psycopg2
>>> conn=psycopg2.connect(database="teste_pgbr", user="pgbr", password="pgbr")
>>>
>>> conn2=psycopg2.connect("dbname=teste_pgbr user=pgbr password=pgbr host=localhost")
>>>
```

- Desconectando de uma base de dados

```
>>>
>>> conn.close()
>>>
```

# Psycopg2 – Alguns Exemplos



- Criando uma tabela e inserindo dados

```
>>>
>>> import psycopg2
>>> conexao=psycopg2.connect(database="teste_pgbr", user="pgbr", password="pgbr")
>>> cursor=conexao.cursor()
>>> cursor.execute("CREATE TABLE teste (id serial PRIMARY KEY, num integer, dados varchar(50));")
>>> cursor.execute("INSERT INTO teste (num, dados) VALUES (%s, %s)",
                    (15, "Dia de Inicio do PGBR - Show d'hora" ) )
>>> cursor.execute("SELECT * FROM teste;")
>>> linha = cursor.fetchone()
>>> linha
(1, 15, "Dia de Inicio do PGBR - Show d'hora")
>>> conexao.commit()
>>> cursor.close() #fecha o cursor
>>> conexao.close()
>>>
```

# Psycopg2 – Alguns Exemplos



- Alterando e excluindo registros de uma tabela

```
>>> conexao=psycopg2.connect(database="teste_pgbr", user="pgbr", password="pgbr")
>>> cursor=conexao.cursor()
>>> cursor.execute("UPDATE teste SET num=17 WHERE id=1")
>>> conexao.commit()
```

```
>>> conexao=psycopg2.connect(database="teste_pgbr", user="pgbr", password="pgbr")
>>> cursor=conexao.cursor()
>>> cursor.execute("DELETE FROM teste WHERE id=%s", (1,))
>>> conexao.commit()
```

# Psycopg2 – Alguns Exemplos



- Conversão de tipo automática e setar o encoding no cliente:

```
>>>
>>> conexao.set_client_encoding('LATIN1')
>>>

>>> cursor.execute("SELECT %s, %s, %s, %s", (None, True, False, Decimal("20.56"), ))
>>> cursor.fetchone()
(None, True, False, Decimal('20.56'))
>>> cursor.mogrify("SELECT %s, %s, %s, %s", (None, True, False, Decimal("20.56"), ))
'SELECT NULL, true, false, 20.56'
>>> |
```



# django

# Django

- Framework Web;
- Criado por Adrian Holovaty em 2005;
- Inicialmente seria apenas um gerenciador de notícias;
- Licença BSD;
- Nome inspirado no músico de jazz Django Reinhardt
- Escrito em Python
- Utiliza padrão MVC
- Conceito DRY

# Django - Recursos

- Possui ORM próprio;
  - Acesso a diversos bancos;
    - PostgreSQL;
    - MySQL;
    - Oracle;
    - DB2;
    - SQLServer;
  - Permite multi-bancos;
  - Permite herança;
  - Classes Abstratas

```
from django.db import models

class Palestra(models.Model):

    nome = models.TextField('Nome da Palestra')
    autor = models.CharField('Autor', max_length=80)
    observacoes = models.TextField('Obs', null=True, blank=True)

    class Meta:
        db_table = 'palestras'
        unique_together = (('nome', 'autor'),)
        verbose_name = 'Palestra'
        verbose_name_plural = 'Palestras'

class Evento(models.Model):

    nome = models.CharField('Nome do Evento', max_length=100, unique=True)
    palestras = models.ManyToManyField(Palestra)
    inicio = models.DateField('Data de Início')

class Participante(models.Model):

    nome = models.CharField('Nome do Participante', max_length=100, unique=True)
    evento = models.ForeignKey(Evento, verbose_name='Evento')
```



# Django - Recursos

- Possui ORM próprio;
  - Chaves estrangeiras;
  - Relações Many-To-Many;
  - Indicação de índices únicos;
  - Indicação de chaves primárias;
  - Trabalha com arquivos;
  - Permite métodos adicionais;

# Django - Recursos

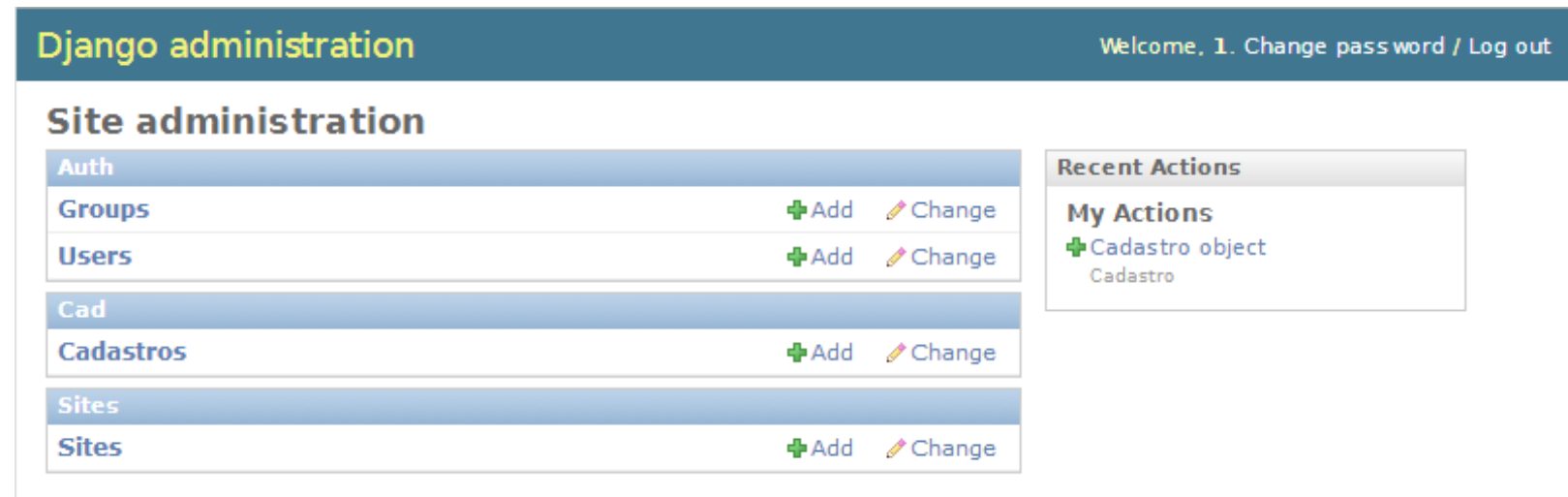
- Possui Sistema próprio de Templates
  - Permite herança de templates;
  - Permite condicionais: if;
  - Permite iterações: for;
  - Possui filtros especiais;
  - Proteção contra tags especiais (html);
  - Possui formatação de data;
  - Internacionalização;
  - Cache;

```
<!DOCTYPE>
<html>
<head>
  <script src="{{ MEDIA_URL }}jquery.js"></script>
  <title>{{ titulo }}</title>
</head>
<body>
  Sistema de Inscrição em Eventos
  {% block conteudo %}{% endblock %}
  {% if mostraFim %}Valeu Pessoal{% endif %}
</body>
</html>
```

```
{% extends 'base.html' %}
{% block conteudo %}
  {{ block.super }}
  <div>Novas informações</div>
{% endblock %}
```

# Django - Recursos

- Interface Administrativa muito poderosa e flexível;
  - Permite o cadastro de usuários e grupos (especificando o acesso);
  - Permite a criação de listagens com busca, filtros (inclusive por data) e sintaxes adicionais;



# Django - Recursos

- Integração com o PostGis (inclusive na Interface Administrativa);
  - GeoDjango, MapProxy e Leaflet;



# Django - Recursos

- Interface Administrativa
  - CRUD Básico /Intermediário;
  - Personalização de templates;
  - Adição de recursos;
  - Gerar modelos mestre-detalle;
  - Geração de Novos Widgets;
  - Adição de Rotinas e Módulos (grapelly, django-admin-shortcut);
  - Geração de ações e sistemas rápidos;

# Django – Outros Recursos

- Internacionalização;
- Cache por página, view ou consulta ;
  - Memcached;
  - Memória;
  - Arquivos;
- Flatpages;
- Feeds;
- Coleta de Arquivos Estáticos;
- Envio de Email;
- Validação de URL's de Recuperação de Email;

# Django – Outros Recursos



- Geração de Log's;
- Aplicações Plugáveis;
- Deploy Simplificado (FastCGI, WSGI);
- Compatibilidade com Servidores Web (Apache, nginx, lighttpd, cherokee, etc);
- Middleware;
- Multibanco;
- Class Based Views;
- E o mais importante... Simplicidade!

Tudo muito legal mas....

E o PostgreSQL com isso??



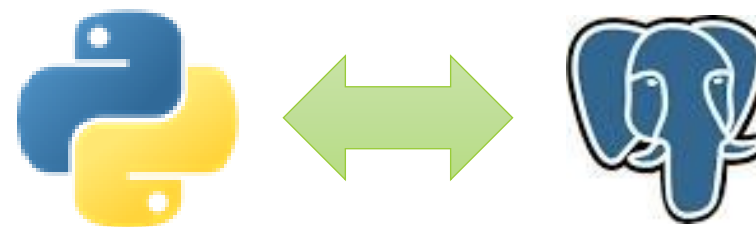
**django**



## Eis a razão....

- Conector python compatível com a maioria dos recursos do PostgreSQL;
- Conexão nativa / homologada com o ORM do Django;
- Possibilidade de execução segura de transações (incluindo em duas fases);
- Proteção contra SQL Injection;
- Proteção contra CSRF;
- Proteção contra XSS;

## Um pouco mais...



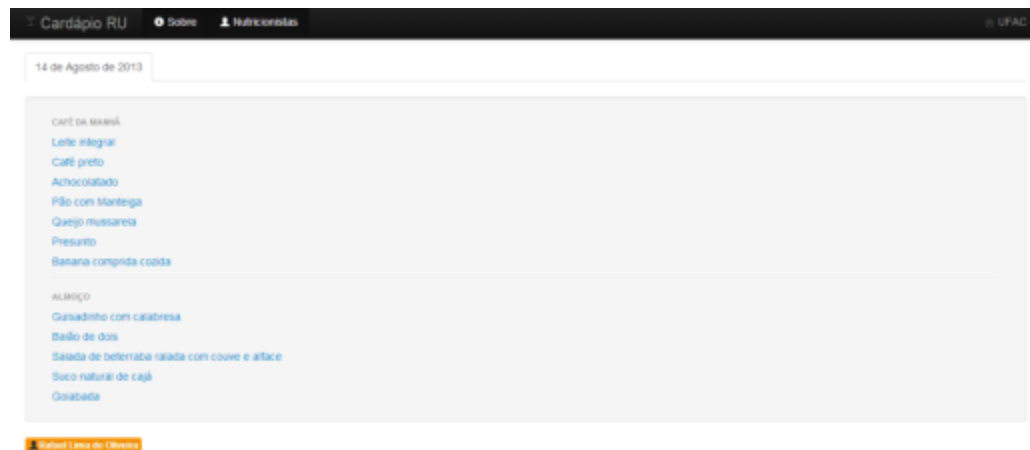
- Conversão precisa e automática entre tipos do PostgreSQL para os tipos do Python e vice-versa;
- Uso de Tablespaces;
- Uso de schemas (com aplicações plugáveis);
- Ampla documentação;
- Facilidade no Deploy;
- Simplicidade no Desenvolvimento;

## E ainda mais...

- Excelente desempenho;
- Geração de *migrations*;
- Geração de modelos a partir de bancos legados;
- Integração nativa com o Postgis;
- Interface administrativa com diversas funções;
- Integração da interface administrativa com o postgis;
- Soluções de suporte e complemento ao postgis (MapProxy, Mapnik);
- Ambiente Livre;
- Integração com JQuery, bootstrap, angularJS, Hightcarts;

# Onde foi utilizado – Aplicações Pequenas

- Cardápio RU (UFAC);
- Plano de Metas – 100 dias (UFAC);



# Onde foi utilizado – Aplicações Pequenas

- Sistema de Busca de Documentos – Finder – (UFAC);
- Sistema de Pedidos - [www.youtube.com/watch?v=emUXmvlGP0E](http://www.youtube.com/watch?v=emUXmvlGP0E)
- Sistema Simulador de Financiamentos Imobiliários - CET
- Sistema de Protocolo – Ufac (Integração com DB2);
- SUAP – Usado por Institutos Federais (des. IFRN)



## Onde foi utilizado – Aplicações de Médio Porte

- Sistema de Gestão Imobiliária – Cohab Acre;
- aQui Alli – Sistema Gerador de HotSites e catálogo online;
- Sistema de Registro de Projetos – UFAC;
- Sistema de Gestão de Processos Seletivos (gestão de 250 mil candidatos registrados) – UFAC
- Sistema de Gestão Escolar – Ensino Médio – CAp/UFAC
- Diversos sites: <http://www.djangosites.org/>

**Obrigado!**

**Dúvidas??**

**Marcos Thomaz**  
**marcosthomazs@gmail.com**