



# MAURO DUARTE

# Primeiros

# Passos com

# Web2py

Desenvolvimento Agil para Web

# Primeiros Passos com Web2py

Mauro Duarte



Direitos autorais do texto original 2016

Mauro Duarte

Todos os direitos reservados



Dedico esta obra a minha esposa Fernanda, sem ela não seria nada.



## Lista de Siglas e Abreviaturas

CSS	Cascading Style Sheets, ou Folhas de Estilo em Cascata
DAL	Data Abstraction Layer ou Camada de Abstração de Dados
FLISOL	Festival Latino Americano de Instalação de Software Livre
HTML	HiperText Markup Language
IDE	Integrated Development Environment
MVC	Model, View, Controller
PHP	PHP HiperText Processor
SEO	Search Engine Optimization
SGBD	Sistema Gerenciador de Banco de Dados
URL	Uniform Resource Locator

# Sumário

[Introdução 7](#)

[O que é Web2py 9](#)

[Aplicações Web 9](#)

[Framework 11](#)

[Python 12](#)

[Full-stack 14](#)

[Software Livre 15](#)

[Parte 1 - Iniciando 16](#)

[Instalação 16](#)

[Lógica de programação com Python 18](#)

[Escrever 18](#)

[Variáveis 19](#)

[Operações com variáveis 21](#)

[Exercício 1 - Exercícios com variáveis 28](#)

[Exercício 2 - Exercícios com variáveis 28](#)

[Exercício 3 - Exercícios com variáveis 28](#)

[Vetores 30](#)

[Exercício 4 - Exercícios com Vetores 34](#)

## Exercício 5 - Exercícios com Vetores 34

Instruções condicionais 35

Exercício 6 - Exercícios com Condicionais 42

Exercício 7 - Exercícios com Condicionais 42

Exercício 8 - Exercícios com Condicionais 42

Laços de Repetição 43

Exercício 9 - Exercícios com laços de repetição 46

Exercício 10 - Exercícios com laços de repetição 46

Alguns conceitos 'avançados' 47

Orientação a Objetos 47

MVC 48

Parte 2 – Model 52

DAL 54

Importações padrão 56

Tabela tarefas 57

Validações 64

Parte 3 - Controller 71

Default.py 73

Função index() 74

Função user() 74

Função download() 75

Função call() 75

Novas funções 76

Funções CRUD 82

Formulários sem banco de dados 84

Limitando Acesso a Usuários logados 86

Usuários e Grupos 87

Cada um na sua tarefa 90

Limpando o lixo 93

Imagens para as tarefas 96

View 97

Visões genéricas 97

Visões específicas 99

default/contato.html 99

default/detalhar.html 102

default/editar.html 105

default/nova.html 106

default/tarefas.html 107

[default/index.html 110](#)

[Ajustes da visão Layout 119](#)

[Finalizando 123](#)

[Arquivos estáticos 123](#)

[Internacionalização 125](#)

[Deploy 127](#)

[Conclusão 131](#)



# Introdução

No inicio do ano de 2015 fui organizador do FLISOL<sup>1</sup> em Porto Alegre e neste evento diversas palestras foram apresentadas, as quais tive o privilegio de apreciar, entretanto uma me chamou mais atenção do que as demais, a palestra em questão se tratava de um framework python para desenvolvimento web o Web2py.

Desde que comecei a usar Software Livre a linguagem Python sempre me chamou a atenção, e por vezes comecei a estudar e parei, apesar do python ser fácil de compreender e lembrar ele parecia um pouco distante de fazer algo prático.

Minha última tentativa foi Django, que também é um framework python para desenvolvimento para web, infelizmente também abandonei por não encontrar (na época) documentação clara, muitos códigos não funcionavam na minha versão, então ao estudar um código ou outro e testar não funcionava por diferenças de versão, sistema operacional entre outras coisas.

Finalmente o último empurrão, após passar por PHP, Java, C++, C# e até uma bisbilhotada em ruby, precisava de um framework para demonstrar para meus alunos de Desenvolvimento II na Faculdade Alcides Maya como trabalhar com um framework de desenvolvimento para Web.

Testei uns quatro frameworks de PHP que me apresentaram problemas, pois precisava que funcionasse em Windows e Linux, sem depender de configurações da equipe de suporte.

A solução veio na forma do Web2py, e ao usá-lo percebi o quanto ele vinha de encontro as minhas necessidades de programador para web e de professor de desenvolvimento.

Seu funcionamento demostrou uma estabilidade incrível em multiplataforma, não apresentando nenhum problema de versionamento nem no windows 7 nem no linux Ubuntu, testei em um windows 8 onde o web2py deu mensagens de falhas de segurança, mas creio que pode ser por, na época, ainda ser um sistema bem recente, desafio a testar e ver se já está ok.

O web2py também apresenta um web server próprio sem precisar de instalação ou configuração para o ambiente de desenvolvimento o que na sala de aula foi uma das melhores coisas.

O fato de trazer uma “IDE” própria, e um ambiente de gerenciamento alegrou também meus alunos, eles adoram realce de sintaxe e facilidades de endentação e tudo mais.

Pra finalizar, nem mesmo o SGBD foi preciso instalar ou configurar pois o web2py traz embutido em seu pacote o SGDB Sqlite que também não necessita de instalação e já está pronto para uso no momento que descompactamos o web2py, além de, claro o web2py é capaz de conectar no SGDB de sua preferência, inclusive nos sistemas do google.

Neste estudo quero apresentar o web2py, suas bases e ferramentas bem como minha experiência com esta incrível ferramenta.



# O que é Web2py

De forma reduzida podemos definir o Web2py como um framework para desenvolvimento ágil de aplicações web usando a linguagem Python, vamos detalhar melhor cada uma destes termos.

# Aplicações Web

Segundo Luis Osorio, Consultor SEO[Osório, L] (2013) “Uma aplicação web é algo mais do que uma simples página web. É uma aplicação cliente-servidor”, o grande diferencial comparando sites com aplicações web é a capacidade de “raciocínio”, isto é, um site funciona como um catalogo, um folder, um informativo, onde o usuário pode apenas visualizar informações, já uma aplicação web pode receber informações do usuário e devolver respostas processadas de acordo com essas informações recebidas. Uma loja virtual, rede social, blog, e muitos outros são exemplos de aplicações web.

Também é importante diferenciar aplicações web de aplicações desktop, eventualmente chamadas de programas, os programas são instalados no sistema operacional do usuário e podem funcionar com ou sem internet, como um editor de textos ou um jogo, etc.

Aplicações web dependem de uma conexão com um servidor de requisições HTTP, normalmente este servidor é centralizado prestando serviço a diversos clientes simultâneos, reduzindo o custo de manter um servidor local, mas eventualmente empresas preferem ter seu próprio servidor local, normalmente empresas de porte médio e grande.

Além de um servidor de requisições HTTP normalmente são necessários outros recursos para que a aplicação web funcione corretamente, um SGBD que funciona para que os dados enviados e processados possam ser armazenados para consultas e processamentos futuros.

O que o usuário vê em seu navegador são representações das marcações HTML, isso significa que um aplicativo web deve entregar e receber marcações HTML, para isso o servidor da aplicação web deve rodar uma linguagem de programação que seja capaz de gerar marcações HTML.

A grande vantagem de uma aplicação web em relação a uma aplicação desktop é a possibilidade de acessá-la em a partir de qualquer computador que seja capaz de conectar a este servidor web.

Para o desenvolvedor é possível escrever um código multiplataforma já que rodará no servidor e não no computador cliente, sendo assim, não importa se o usuário está usando um computador Windows, Linux, Mac, ou até mesmo um smartphone, ele terá acesso e resposta da mesma forma, através de um único código fonte.

Já para desenvolver um aplicativo desktop, será necessário escolher qual plataforma rodará, pois ele será instalado no computador cliente, se o desenvolvedor quiser que rode em múltiplas plataformas será preciso fazer um código fonte para cada plataforma, então será quase como fazer outro software para fazer a mesma coisa.

Em resumo temos:

Sites, rodam em servidores web e são exibidos nos navegadores, portanto são multiplataforma, mas sem interatividade, tem um nível de desenvolvimento relativamente fácil.

Programas, são aplicativos desktop, os quais precisamos instalar no PC para funcionar, pode ou não ser conectados em servidores e bancos de dados, carregam o 'problema' de ser necessário criar uma versão para cada plataforma onde precisamos executá-lo.

Aplicativos web, que obrigatoriamente são executados em servidores, mas são multiplataforma, sendo assim mais práticos para o programador, para executá-los é preciso acessar o servidor por um navegador.

Neste trabalho a proposta é trabalharmos somente com aplicativos web, mas vale ressaltar que o web2py já traz todos os elementos necessários para o desenvolvimento e execução do ambiente não sendo

preciso buscar outras fontes para obter servidor HTTP, SGBD ou linguagem de programação.





# Framework

Segundo Nícolas Müller, diretor executivo da Desenvolve Web e fundador da oficina da Net (2008)

“Framework é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação.”[Müller, N.]

Ou seja, cada vez que você faz um novo programa ou desenvolve para web algumas coisas são praticamente iguais, por exemplo: conectar ao banco de dados e demais ações de gerenciamento de dados como inserir, deletar, alterar e consultar.

Então porque escrever tudo novamente? Basta alterar algumas informações e estará tudo pronto.

Existe uma infinidade de frameworks, cada linguagem apresenta três ou quatro opções, recomendo que você teste cada um e escolha a linguagem e o framework ao qual se sente melhor.

Aqui ainda me arrisco chamar a atenção dos fanboys<sup>2</sup>, tecnologia não é time de futebol, não precisa de torcida, e sim de métricas que mostram na prática qual é a mais eficiente, então não fique bitolado a primeira linguagem que aprendeu ou que o professor ensinou na faculdade. Já dei aulas de C# e um dos meus desapontamentos é ter que ficar limitado ao Windows e ao visual studio da Microsoft.

Ainda existe a opção de criar seu próprio framework com as funções mais comuns usadas nos seus projetos, isso pode dar um trabalho mas terás conhecimento total do framework e facilidade de customização. Sem nenhuma curva de aprendizagem do framework já que foi você mesmo que escreveu.

Como disse antes, experimente.





# Python

**Python** é uma linguagem de programação de alto nível, quer dizer que não é preciso de conhecimento de linguagens de máquina, ou binários para programar nesta linguagem, saber as sintaxe básica, isto é os comandos da linguagem já será suficiente para produzir excelentes programas.

Essa linguagem é interpretada e de script o que funciona muito bem para servidores web, imperativa, funcional, de tipagem dinâmica e forte e orientada a objetos de uma forma até radical onde tudo é objeto até mesmo as variáveis, tipos primitivos e classes.

O criador da linguagem python foi Guido van Rossum que lançou a primeira especificação em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

Depois de bem difundida a linguagem python tornou-se uma comunidade forte e unida de usuários, aperfeiçoando-a e expandindo inclusive sua filosofia que é comunmente conhecida como “O zen de Python” declarado pela primeira vez por Tim Peters, segue abaixo:

- Belo é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Plano é melhor que aninhado.
- Esparsa é melhor que denso.

- Legibilidade conta.
- Casos especiais não são especiais o suficiente para violar as regras.
- Embora praticidade vença pureza.
- Erros não devem passar silenciosamente.
- A não ser que sejam explicitamente silenciados.
- Em caso de ambiguidade, resista à tentação de adivinhar.
- Deve haver um - e somente um - jeito óbvio de fazer.
- Embora tal jeito não seja tão óbvio no à primeira vista a não ser que você seja holandês.
- Agora é melhor que nunca.
- Embora nunca é frequentemente melhor que exatamente agora.
- Se a implementação é difícil de explicar, a ideia é ruim.
- Se a implementação é fácil de explicar, talvez a ideia seja boa.
- Espaços de nomes são uma ideia estupenda - vamos fazer mais deles!

Como já falei anteriormente o web2py é um framework python e por isso é necessário conhecer programação na linguagem python, vou detalhar as bases da linguagem mais tarde em outro capítulo, onde ao conhecer a linguagem poderás já testar os códigos.

Então após o capítulo da 'instalação' do web2py vamos também acessar o console python e aprender mais.





# Full-stack

Web2py é um framework full-stack sem dependências, ou seja, quando tu baixas o pacote do web2py tens tudo que precisa para programar sem instalação ou configuração ou dependências, comparando com um framework PHP por exemplo, é necessário instalar um webserver(Apache, IIS, etc.), um SGBD (mysql, Mariadb, etc.) e a própria linguagem (PHP-Server) além de fazer configurações, uma das dificuldades que encontrei foi ativar o mod\_rewrite sem ajuda do suporte nos computadores da escola.

O modo como o web2py foi construído permite “carregar” seus aplicativos, ou melhor, carregar todo seu ambiente de desenvolvimento e executá-lo em qualquer lugar, até mesmo a partir de um pendrive, outra opção é ter no mesmo computador diretórios diferentes rodando web2py diferente sem nenhuma interferência de um com o outro, por exemplo, cada aluno roda seu próprio web2py, sem apagar ou ser influenciado pelo código do colega no mesmo computador.

Tráz também uma IDE baseada na web que roda 'dentro' do próprio web2py e um console python ambos sem instalação, permitindo estudar e testar em qualquer plataforma ou ambiente.





# Software Livre

Todas essas facilidades que o web2py apresenta ainda são cobertas pela licença LGPL que o torna um software livre com suas liberdades fundamentais já conhecidas:

- **Liberdade 0** - Liberdade para o programa para quaisquer propósitos;
- **Liberdade 1** - Liberdade para estudar como o programa trabalha e adaptá-lo às suas necessidades. Ter acesso ao código fonte é essencial para isso.
- **Liberdade 2** - Liberdade de redistribuir cópias de forma que você possa ajudar outras pessoas.
- **Liberdade 3** - Liberdade para melhorar o programa e disponibilizar as melhorias para o público, de forma que toda a comunidade possa se beneficiar disso. Ter acesso ao código fonte é essencial também para isso.

Por isso é encorajado a cada um aprender e ensinar e distribuir o web2py tornando um software melhor e ajudando o mundo a se tornar um mundo melhor.



# **Parte 1 - Iniciando**

Agora vamos aprender os princípios que servirão de base para nosso estudo, se tu já estas familiarizado com programação, python, bancos de dados, etc. alguns destes princípios podem ser repetitivos, então podes pular os testes de código mas seria interessante pelo menos ler este capítulo para relembrar os demais conceitos.

# Instalação

Antes de mais nada é preciso baixar o pacote do web2py, para isso acesse o site oficial do web2py <http://web2py.com> na página de downloads ou através deste link <http://web2py.com/init/default/download>.



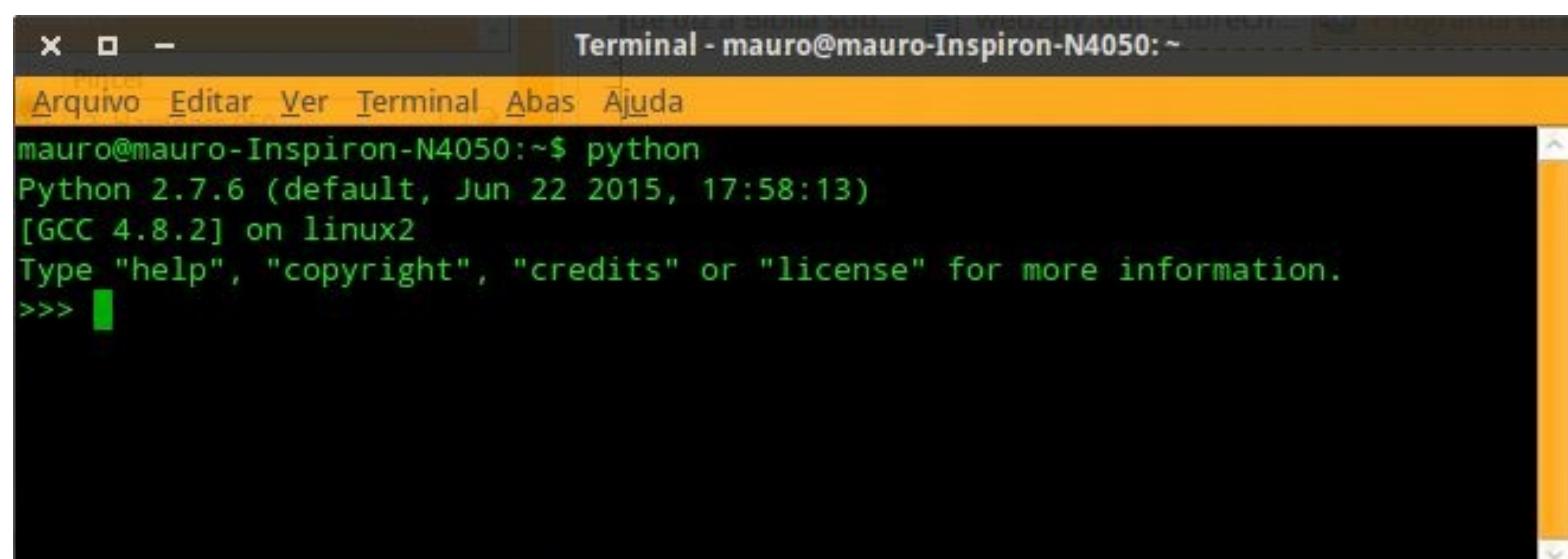
The source code version works on all supported platforms, including Linux, but it requires Python 2.6, or 2.7 (recommended). It runs on Windows and most Unix systems, including Linux and BSD.

*Figura 1: Download do Web2py*

Tu deverás baixar a versão “For Normal Users” caso tu, assim como eu, use linux ou BSD deverás baixar a versão “Source Code”.

Após baixar o próximo passo obvio é descompactar o pacote no seu diretório de preferência, antes de iniciarmos com o web2py propriamente dito precisamos entender alguns conceitos do python, então se tu estás usando windows encontre o arquivo py.exe e clique para abrir o console do python, no linux e no macOS normalmente o python já esta instalado, então abra um console (Terminal) e escreva python (minusculo) e de Enter.

Se tudo der certo terás uma tela como esta:



The screenshot shows a terminal window titled "Terminal - mauro@mauro-Inspiron-N4050: ~". The window has a menu bar with options: Arquivo, Editar, Ver, Terminal, Abas, Ajuda. Below the menu, the command prompt is shown: "mauro@mauro-Inspiron-N4050:~\$ python". The output of the command is displayed, showing the Python version and build information: "Python 2.7.6 (default, Jun 22 2015, 17:58:13) [GCC 4.8.2] on linux2". It also includes a message: "Type "help", "copyright", "credits" or "license" for more information." A green cursor is visible at the bottom left of the terminal window.

Figura 2: Console Python

Agora estamos prontos para aprender um pouquinho do python.





# Lógica de programação com Python

Para programar com web2py é preciso saber programar, este trabalho não é direcionado para aqueles que não sabem programar, mesmo assim vamos rever algumas regras básicas que permitiram aqueles que estão iniciando ter a base necessária para acompanhar o restante do conteúdo.

## Escrever

Uma das bases da programação consiste em mandar mensagens para o usuário, segundo o princípio de que 'simples e melhor que complexo' e 'deve haver um jeito óbvio de fazer isso' temos o comando **print**:

1.     

```
>>> print "olá mundo"
```
2.     

```
olá mundo
```

Caso se queira escrever uma informação que não seja posteriormente exibida para o usuário e nem seja interpretada como um comando podemos usar comentários que sempre começam com o caractere sustenido “#”

1.     

```
>>>#isto é um comentário
```
2.     

```
... print "olá mundo"
```
3.     

```
olá mundo
```

Muitas funções e comandos tem sua saída padrão imprimindo seu retorno na tela, um exemplo que podemos usar são as funções matemáticas simples que nos permitem fazer uma mini calculadora em python de forma muito fácil.

```
1.      >>> #calculadora
```

```
2.      ... 2+2
```

```
3.      4
```

```
4.      >>> #repare que não foi usado o comando print
```

```
5.      ... 2*3
```

```
6.      6
```

```
7.      >>>10-2
```

```
8.      8
```

```
9.      >>>10/2
```

```
10.     5
```

```
11.     >>>10/3
```

```
12.     3
```

```
13.     >>>#divisões de inteiros são arrredondados
```

## Variáveis

Variáveis são espaços reservados na memoria que armazenam informações enquanto o programa está em execução, como o próprio nome diz as informações armazenadas podem modificar durante o tempo de vida da variável, quando o script termina essas informações são perdidas.

Variáveis sempre tem nome, valor e tipo, não confunda valor com número, valor refere-se mais ao conteúdo da variável.

**Valor:** O valor da variável é seu conteúdo, quando usamos o comando print para uma variável podemos exibir seu conteúdo, se no terminal do python escrevermos o nome de uma variável, ela funcionará como um comando print mostrando seu conteúdo mas com algumas diferenças.

```
1.      >>> x = 1
```

```
2.      >>> y = "Olá"
```

```
3.      >>>print x
```

```
4.      1
```

```
5.      >>>print y
```

```
6.      Olá
```

```
7.      >>> y
```

```
8.      'ol\xc3\xal'
```

```
9.      >>>#repare que o acento não foi bem  
interpretado
```

**Nome:** Os nomes das variáveis devem ser significativos, evite chamar as variáveis de 'var1', 'var2', ou 'n1', 'n2' etc. Nomes melhores são 'fone', 'preco', etc.

Nomes de variáveis devem seguir algumas regras, jamais devem começar com números, embora possam contê-los em qualquer outro lugar inclusive terminar com eles, alguns caracteres não são permitidos

como “\$”, “/”, “=” entre outros. O espaço também não é permitido, caso seja necessário usar nomes longos ou compostos o recomendado é usarmos o caractere sublinhado “\_” por exemplo: “primeiro\_nome”, “segundo\_nome”.

Outro fator a ser considerado é que o python é case sensitive, ou seja, letras maiúsculas e minusculas são diferentes, Mauro é diferente de mauro.

**Tipo:** diferente de outras linguagens o python não necessita de declaração para definir o tipo da variável, a variável é criada no momento da atribuição, bem como não necessita de conversores de tipos, se a variável receber um valor de um tipo diferente do seu tipo original ela mudará de tipo para se adaptar ao valor recebido.

Podemos usar o comando type para saber o tipo de uma variável.

- **Int**

Refere-se a números inteiros, positivos ou negativos.

10.     `>>> x = 1`

11.     `>>> y = -25`

12.     `>>> type(x)`

13.     `<type 'int'>`

14.     `>>> type(y)`

15.     `<type 'int'>`

- **Float**

Refere-se a números com casas decimais (observe que usa-se ponto “.” para definir casas decimais)

1.    >>> x = 1.25

2.    >>> y = -2.543234233

3.    >>>pi = 3.1416

4.    >>> type(x)

5.    <type 'float'>

6.    >>> type(y)

7.    <type 'float'>

8.    >>> type(pi)

9.    <type 'float'>

- **Str**

São conjuntos de caracteres, podendo aceitar qualquer valor, valores numéricos serão tratados como caracteres não podendo ser usados em cálculos até serem convertidos. A atribuição de um valor string deve usar aspas.

1.    >>> x = "Olá mundo"

2.    >>> y = "-2"

3.    >>>#a variavel y não poderá ser usada em cálculos

4.    >>> type(x)

```
5.      <type 'str'>
```

```
6.      >>> type(y)
```

```
7.      <type 'str'>
```

- **Bool**

Boolianos são valores binários, isto é, só podem ter dois valores no caso do python os valores são True e False

```
1.      >>> x = True
```

```
2.      >>> y = False
```

```
3.      >>> type(x)
```

```
4.      <type 'bool'>
```

```
5.      >>> type(y)
```

```
6.      <type 'bool'>
```

## Operações com variáveis

- **Matemática**

Qualquer variável numérica, isto é, *int* e *float* pode ser usada para fazer cálculos matemáticos utilizando os seguintes operadores abaixo[Desenvolvo].

+	soma
-	subtração

*	multiplicação
/	divisão
**	potenciação
%	módulo(resto da divisão)

Veja mais alguns exemplos:

### Soma com variáveis

1.    `>>> x = 20`

2.    `>>> y = 12`

3.    `>>> z = x+y`

4.    `>>>print z`

5.    `32`

### Subtração com variáveis

1.    `>>> x = 20`

2.    `>>> y = 12`

3.    `>>> z = x-y`

4.    `>>>print z`

5.

8

## Multiplicação com variáveis

1.    >>> `x = 20`

2.    >>> `y = 12`

3.    >>> `z = x*y`

4.    >>>`print z`

5.    240

## Divisão com variáveis

1.    >>> `x = 20`

2.    >>> `y = 2`

3.    >>> `z = x/y`

4.    >>>`print z`

5.    10

6.    >>>`type(z)`

7.    <type 'int'>

8.    >>>`z = x/3`

9.    >>>`print z`

6

```
11.      >>>type(z)
```

```
12.      <type 'int'>
```

```
13.      >>>x = 20.0
```

```
14.      >>>type(x)
```

```
15.      <type 'float'>
```

```
16.      >>>z = x/3
```

```
17.      >>>print z
```

```
18.      6.66666666667
```

```
19.      >>>type(z)
```

```
20.      <type 'float'>
```

## Potenciação com variáveis

```
1.      >>>x = 20
```

```
2.      >>>y = 2
```

```
3.      >>>z = x**y
```

```
4.      >>>print z
```

```
5.      400
```

## Modulo com variáveis

1.      `>>> x = 20`

2.      `>>> y = 2`

3.      `>>> z = x%y`

4.      `>>>print z`

5.      `0`

6.      `>>>z = x%3`

7.      `>>>print z`

8.      `2`

- **Lógica**

As variáveis do tipo *bool* podem armazenar resultados binários verdadeiro (True) ou falso (False), assim sendo podemos usar operadores lógicos para chegar a conclusões lógicas.

Os operadores lógicos que usamos em python estão na tabela abaixo[Desenvolvo]:

>	Maior que
---	-----------

<	Menor que
$\geq$	Maior ou igual á
$\leq$	Menor ou igual á
$=$	Igual
$\neq$	Diferente

## Operadores e testes lógicos com variáveis

1. 

```
>>> x = 20
```
2. 

```
>>> y = 2
```
3. 

```
>>> z = x>y
```
4. 

```
>>>print z
```
5. 

```
True
```
6. 

```
>>> z = x<y
```
7. 

```
>>>print z
```
8. 

```
False
```
9. 

```
>>> z = x==y
```
10. 

```
>>>#cuidado para não confundir um sinal de igual que representa atribuição, com dois
```

sinais que representam comparação

11. ... print z

12. False

13. >>> z = x != y

14. >>> print z

15. True

16. >>> type(z)

17. <type 'bool'>

Ainda existem outros operadores especiais, **and**, **or** e **not**:

**and**

Nos permite fazer mais de um teste lógico para obter uma única resposta lógica, funciona como um E lógico, isto é, haverá um retorno verdadeiro (True) se e somente se, todas os testes forem verdadeiros.

1. >>> idade = 20

2. >>> altura = 1.72

3. >>> requisito = (idade >= 18) and (altura >= 1.70)

4. >>> print requisito

5. True

```
6.      >>>idade = 22
```

```
7.      >>>altura = 1.64
```

```
8.      >>>requisito = (idade >= 18) and (altura  
>= 1.70)
```

```
9.      >>>print requisito
```

```
10.     False
```

*or*

Funciona da mesma maneira que **and** nos permitindo fazer mais de um teste lógico para obter uma única resposta lógica, funciona como um OU lógico, isto é, haverá um retorno verdadeiro (True) se um, qualquer um dos testes forem verdadeiros.

```
1.      >>>idade = 20
```

```
2.      >>>altura = 1.72
```

```
3.      >>>requisito = (idade >= 18) or (altura  
>= 1.70)
```

```
4.      >>>print requisito
```

```
5.      True
```

```
6.      >>>idade = 22
```

```
7.      >>>altura = 1.64
```

```
8.      >>> requisito = (idade >= 18) or (altura  
>= 1.70)  
  
9.      >>>print requisito  
  
10.     True
```

### ***not***

Este é o mais simples de compreender, ***not*** inverte o valor lógico da resposta, ou seja, a resposta verdadeira(True) se torna falso(False)

```
1.      >>> x = 20  
  
2.      >>> y = 2  
  
3.      >>> z = not (x>y)  
  
4.      >>>print z  
  
5.      False
```

- **Texto**

Ainda temos algumas operações que podemos fazer com textos e variáveis do tipo str, sendo que algumas nós já vimos nos operadores lógicos como == (igual) e != (diferente).

```
1.      >>> x = "teste"
```

```
2.      >>> z = x == "teste"
```

```
3.      >>> print z
```

```
4.      True
```

```
5.      >>>z = x != "teste"
```

```
6.      False
```

A função *len()* permite saber quantos caracteres tem uma variável str ou expressão

```
1.      >>> x = "teste"
```

```
2.      >>> y = len(x)
```

```
3.      >>> print y
```

```
4.      5
```

## Concatenação

Concatenar significa juntar uma ou mais partes, são muito úteis para juntar strings em uma única string maior e complexa.

```
1.      >>> x = "fulano"
```

```
2.      >>> y = "da silva"
```

```
3.      >>> z = x + y
```

```
4.      >>>print z
```

```
5.      fulanoda silva
```

```
6.     >>>#este resultado não ficou bom vamos  
melhorar isto  
  
7.     ...z = x + " " + y  
  
8.     >>>print z
```

Existem outras maneiras de fazer isso e outras funções de String, porem não falaremos disso agora.



## **Exercício 1 - Exercícios com variáveis**

Abra o terminal do python e crie as variáveis abaixo com os seguintes valores e use o comando print para mostrar seu valor na tela e seu próprio nome para comparar o retorno e o comando type para exibir seu tipo, complete a tabela abaixo:

Nome	Valor	Saída	Saída (print)	Tipo(type)
primeiro_nome	Cássio			
sobrenome	Da Silva			
idade	32			
saldo	1520.36			
telefone	85236989			
rua	Travessa da calamidade			
cliente	True			

## **Exercício 2 - Exercícios com variáveis**

Abra o terminal do python e crie as variáveis abaixo.

produto\_1 → 15.50

desconto → 10% (pense bem para resolver isso)

Agora calcule o valor do produto\_1 já com desconto.

## ***Exercício 3 - Exercícios com variáveis***

Corrija os códigos abaixo para que a variável *z* tenha o valor True.

### **Código 1**

1.      >>> *x* = 20

2.      >>> *y* = 2

3.      >>> *z* = *x*>*y*

4.      >>>*print* *z*

5.      *False*

### **Código 2**

1.      >>> *x* = 2

2.      >>> *y* = 2

3.      >>> *z* = *x*

4.      >>>*print* *z*

5.

False

## Código 3

1.      >>> x = 20

2.      >>> y = 2 + (20/4) \*3

3.      >>> z = x > y

4.      >>> print z

5.      False



# Vetores

Vetores são conjuntos de variáveis do mesmo tipo, e tem o mesmo nome, somente os valores são diferentes, então pra acessar os valores são usados indices, que podem ser automáticos ou definidos.

## List

Listas são vetores bem simples com índices automáticos, começando com zero (0) e sequencialmente com números inteiros, são definidos com colchetes e separados os valores com vírgulas.

Os valores nas listas de dados não precisam ser do mesmo tipo.

1.     `>>> a = [2, 3, "abc", "xyz"]`

2.     `>>> print a`

3.     `[2, 3, 'abc', 'xyz']`

4.     `>>> print a[0]`

5.     `2`

6.     `>>> print a[2]`

7.     `abc`

é possível fatiar uma lista usando apenas parte de seus dados, o fatiamento da lista é feito com base em dois parâmetros colocados dentro dos colchetes do nome da lista separados por dois pontos “::” como segue:

*nome[inicio:tamanho]*

O parâmetro `início` é o número do índice ao qual queremos iniciar nosso fatiamento, caso seja omitido o índice é considerado zero, iniciando então do primeiro.

O parâmetro `tamanho` indica quantos valores queremos exibir, incluindo o valor do índice '`inicio`', caso seja omitido o final será o último, exibindo todos os valores a partir do parâmetro `início`.

```
1.      >>> a = [2, 3, "abc", "xyz"]
```

```
2.      >>> print a
```

```
3.      [2, 3, 'abc', 'xyz']
```

```
4.      >>> print a[0:2]
```

```
5.      [2, 3]
```

```
6.      >>> print a[:2]
```

```
7.      [2, 3]
```

```
8.      >>> print a[2:]
```

```
9.      ['abc', 'xyz']
```

Podemos modificar os valores de uma lista usando seus índices ou fatias de dados, veja:

```
1.      >>> a = [2, 3, "abc", "xyz"]
```

```
2.      >>> print a
```

```
3.      [2, 3, 'abc', 'xyz']
```

```
4.      >>> a[0] = 4
```

```
5.      >>>print a
```

```
6.      [4, 3, 'abc', 'xyz']
```

```
7.      >>> a[:2] = [1, 9]
```

```
8.      >>>print a
```

```
9.      [1, 9, 'abc', 'xyz']
```

Ainda temos algumas funções de lista que podemos ver mais tarde depois de conhecermos alguns conceitos como Orientação a Objetos.

## Dict

Dicionários de dados são estruturas mais complexas do que as listas, entretanto são mais poderosos e legíveis.

Dicionários não utilizam índices automáticos, precisamos informar os indices manualmente podendo utilizar inclusive strings como indices, os dicionarios são definidos usando chaves ({}), os índices são separados dos valores por dois pontos (:), e cada conjunto de índice e valor é separado por vírgula (,), o retorno destes valores são semelhantes as listas utilizando colchetes([]) com o nome do indice para retornar seu respectivo valor.

```
1.      >>> debito = {"joao":132, "maria":30, "wesley":40, "ivo":7}
```

```
2.      >>> print debito['joao']
```

```
3.      132
```

```
4.      >>>debito['maria'] = 50
```

```
5.      >>>print debito
```

```
6.      { 'joao':132, 'maria':50, 'wesley':40, 'ivo':7
7.      >>>debito[ 'maria' ] = debito[ 'maria' ]+20
8.      >>>print debito
9.      { 'joao':132, 'maria':70, 'wesley':40, 'ivo':7
```

Lembre-se dicionários não tem ordem e devem ser sempre chamados pelo seu índice.

## Tuple

Tuplas são listas imutáveis, então caso seja preciso criar uma lista de forma que o script não seja capaz de modificar seus dados podemos usar este recurso.

As tuplas são criadas atribuindo seus valores sequencialmente separados com vírgula a uma variável, apesar de não ser obrigatório normalmente usamos parênteses para definir as tuplas

As demais operações podem ser feitas quase da mesma forma que nas listas lembrando que só é possível ler as tuplas, elas não podem ser alteradas.

```
1.      >>> a = (2, 3, 15, 0, 22)
2.      >>> print a
1.      (2, 3, 15, 0, 22)
2.      >>> type(a)
3.      <type 'tuple'>
```

```
4.      >>> print a[0]
```

```
5.      2
```

```
6.      >>>a[0] = 10
```

```
7.      TypeError: 'tuple' object does not support  
item assignment
```

## String

Ok, já falamos do tipo *str* que são cadeias de caracteres, estamos voltando neste assunto porque algumas coisas que aprendemos agora podem ser usadas para manipular os tipos *str* também.

O tipo *str* funciona como uma tupla, onde cada caractere é uma posição do vetor.

```
1.      >>> a = "palavra"
```

```
2.      >>> print a[0]
```

```
3.      p
```

```
4.      >>> print a[2]
```

```
5.      l
```

```
6.      >>> a[0] = "x"
```

```
7.      TypeError: 'str' object does not support  
item assignment
```





## ***Exercício 4 - Exercícios com Vetores***

Crie uma lista com os primeiros 5 números primos e exiba na tela do terminal

## ***Exercício 5 - Exercícios com Vetores***

Crie um dicionário com as siglas dos estados brasileiros como índice e o nome por extenso como valor (no mínimo 10 estados), após exiba na tela.



# Instruções condicionais

Para escrever programas é normal termos que usar estruturas para mostrar dados diferentes dependendo das informações recebidas, a principal delas é a instrução se (if), porém antes de mostrarmos algum exemplo precisamos falar de um princípio importante do python a **endentaçāo**.

## *Endentaçāo*

Endentaçāo é um espaço na margem, do código, é usado na maioria das linguagens de programação para organizar o código, no python, entretanto a endentaçāo serve como “fechador” de código, então quando uma estrutura inicia, ela inicia rente a margem, todo o código que está dentro da estrutura deve estar afastada da margem com **quatro espaços**.

Caso dentro da estrutura em questão haja uma outra estrutura o inicio desta segunda deve estar a **quatro espaços** da margem e seu conteúdo a **oito espaços** e assim sucessivamente.

Quando seu código retornar ao alinhamento anterior a estrutura estará finalizada, algo como um fecha chaves ( } ) em PHP ou java.

Após mais algumas explicações vamos ver uns exemplos, então ficará mais fácil compreender essa característica do python.

## *IF*

A instrução *if* funciona interpretando um teste lógico e caso a resposta do teste seja verdadeiro (True) há um retorno, caso a instrução se falso(False) o retorno é outro. Esse teste lógico pode ser também o conteúdo de uma variável bool.

O inicio da instrução é dado pelo próprio comando *if* seguido da expressão que define o teste lógico ou variável bool.

```
1.      >>> if 12>3:
```

```
2.          ... print "sim"
```

```
3.          ...
```

```
4.      sim
```

```
5.      >>> if 12<3:
```

```
6.          ... print "sim"
```

```
7.          ...
```

```
8.      >>>
```

Se usarmos uma variável numérica, zero será considerado False, qualquer outro valor será considerado True.

Se usarmos uma variável str, vazio será considerado False, qualquer outro valor será considerado True.

```
1.      >>>x = True
```

```
2.      >>>type(x)
```

```
3.      <type 'bool'>
```

```
4.      >>> if x:
```

```
5.          ... print "sim"
```

```
6.          ...
```

```
7.      sim
```

```
8.      >>>x = False
```

```
9.      >>> if x:
```

```
10.         ... print "sim"
```

```
11.         ...
```

```
12.      >>>
```

```
1.      >>>x = 1
```

```
2.      >>>type(x)
```

```
3.      <type 'int'>
```

```
4.      >>> if x:
```

```
5.         ... print "sim"
```

```
6.         ...
```

```
7.      sim
```

```
8.      >>>x = 0
```

```
9.      >>> if x:
```

```
10.         ... print "sim"
```

```
11.         ...
```

```
12.      >>>
```

```
1.      >>>x = "ok"
2.      >>>type(x)
3.      <type 'str'>
4.      >>> if x:
5.          ... print "sim"
6.          ...
7.      sim
8.      >>>x = ""
9.      >>> if x:
10.          ... print "sim"
11.          ...
12.      >>>
```

Até agora vimos a instrução *if* retornando um valor em sua condição True, na condição False a instrução simplesmente não faz nada, entretanto podemos aumentar a funcionalidade da instrução if acrescentando a sub-instrução else.

```
1.      >>>x = False
```

```
2.     >>>type(x)
```

```
3.     <type 'bool'>
```

```
4.     >>> if x:
```

```
5.         ... print "sim"
```

```
6.         ... else:
```

```
7.             ... print "não"
```

```
8.             ...
```

```
9.     não
```

```
1.     >>>x = ""
```

```
2.     >>>type(x)
```

```
3.     <type 'str'>
```

```
4.     >>> if x:
```

```
5.         ... print "Tudo ok"
```

```
6.         ... else:
```

```
7.             ... print "x esta vazio"
```

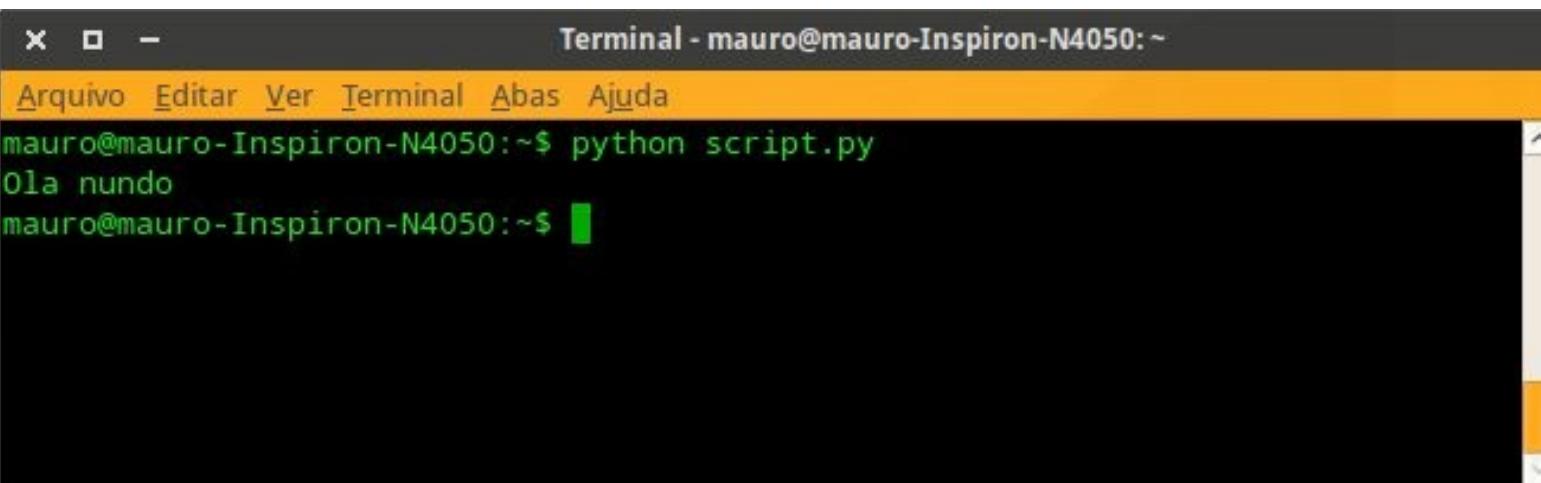
```
8.             ...
```

```
9.     X esta vazio
```

Acredito agora ser um bom momento para vermos 2 novos conceitos, script e entrada de dados:

## Script

Scripts são documentos de texto que contem códigos python, normalmente usamos a extensão .py para ajudar o sistema a interpretar que os dados são códigos do python, mas não é obrigatório, principalmente em sistemas linux, os scripts são executados direto da tela do terminal pelo comando python seguido do nome do script ou caminho até o script, ao fazer isso cada linha do documento será lido como uma linha de terminal python.



```
Terminal - mauro@mauro-Inspiron-N4050: ~
Arquivo Editar Ver Terminal Abas Ajuda
mauro@mauro-Inspiron-N4050:~$ python script.py
Olá mundo
mauro@mauro-Inspiron-N4050:~$
```

Figura 3: Terminal chamando o script

No exemplo acima meu script está na pasta home do usuário logado então foi só executá-lo.

## Entrada de dados

Entrada de dados serve para receber informações do usuário, permitindo assim a interação com o usuário a partir de um mesmo programa trazendo respostas diferentes dependendo da informação dada pelo usuário.

### Input

O comando *input* permite receber dados e armazená-los em variáveis e ao mesmo tempo mandar mensagens ao usuário informando qual tipo de informação deverá ser informada. O comando *input* normalmente é usada para receber números.

### Raw\_input

O comando *input* recebe numéricos, então para receber *str*, textos é preciso usar o comando *raw\_input()* que tem exatamente o mesmo funcionamento com a única diferença no tipo de dado recebido.

Vamos fazer nosso primeiro script com entrada de dados e saídas condicionais, então abra um editor de texto simples ou editor de código de sua preferencia (gedit, kedit, notepad++ ou mesmo o bloco de notas do windows, mas neste tome cuidado na hora de salvar o arquivo<sup>3</sup>) e escreva o seguinte código.

1.     #coding: utf8
- 2.

```
3.     print "Meu primeiro script Python"  
4.     print "Este deverá testar o recebimento e  
leitura de variáveis numéricas"  
5.     x = input("escreva um número: ")  
6.     print "O número informado foi ", x
```

Salve como `script.py` e execute com o comando `python script.py`

Algumas novidades apareceram aqui, na linha 1 temos a seguinte informação “`#coding: utf8`” isto informa ao script o tipo de codificação de caracteres, neste caso utf8 nos permitirá acentuar as palavras como na linha 4.

Já na linha 6 temos uma maneira nova de concatenar usando vírgula em vez de sinal de mais, sinal de mais servirá para concatenar variáveis strings ou somar numéricos a vírgula irá concatenar variáveis de tipos de dados diversos.

Agora vamos usar estruturas condicionais no nosso script, então faça um novo ou altere este para que fique desta forma:

```
1. #coding: utf8  
2.  
3. print "Script com estrutura condicional"
```

```
4.     x = input("escreva um número: ")
5.     if x > 10:
6.         print "O número é maior que 10"
7.     else:
8.         print "O numero é menor que 10"
9.
```

Salve e execute duas vezes, na primeira informe um número maior que dez na segunda vez um menor ou igual a dez, para que possamos testar as duas respostas possíveis.

Acho que deu tudo certo, então, vamos avançar.

Vamos agora fazer um com três possibilidades, seguindo este exemplo podemos fazer quantas possibilidades acharmos conveniente.

```
1. #coding: utf8
2. x = input("Informe a idade: ")
3. if x < 12:
4.     print "Criança"
5. else:
```

```
6.     if x < 18:  
7.         print "Adolescente"  
8.     else:  
9.         print "Adulto"
```

Preste bem a atenção na endentação.

Vamos fazer o mesmo código um pouco melhor substituindo o comando *else*: seguido do comando *if* pelo comando *elif*

### **elif**

Este comando serve para diminuir o código escrito unindo as expressões *else* e *if* o script acima ficaria da seguinte forma:

```
1. #coding: utf8  
2. x = input("Informe a idade: ")  
3. if x < 12:  
4.     print "Criança"  
5. elif x < 18:  
6.     print "Adolescente"
```

```
7.     else:
```

```
8.         print "Adulto"
```

Neste processo economizamos alguns caracteres deixando nosso código um pouco mais limpo e lembre-se 'belo é melhor que feio' e 'legibilidade conta'.

Em qualquer dos casos a instrução será encerrada no momento que uma resposta verdadeira for encontrada, não executando assim os demais testes, a ordem como estes testes são colocados então é muito importante.

Não fizemos isso em nenhum exemplo mas lembre que é possível usar ***and***, ***or*** e ***not*** para testar mais possibilidades que recebam as mesma resposta verdadeira.



## ***Exercício 6 - Exercícios com Condicionais***

Crie um script que receba a nota de um aluno e retorne “aprovado”, caso a nota seja igual ou maior que sete, ou “reprovado”, para a nota menor que 7.

## ***Exercício 7 - Exercícios com Condicionais***

Crie um script que receba a nota de um aluno e retorne “aprovado”, caso a nota seja igual ou maior que sete, ou “reprovado”, caso a nota seja menor que 3 e ainda “em recuperação” caso seja uma nota maior que três e menor que sete.

## ***Exercício 8 - Exercícios com Condicionais***

Crie um script que receba a altura e o peso de uma pessoa e retorne o índice de massa corporal (IMC) retornando as seguintes mensagens de acordo com o resultado obtido.

- Até 18,5 → Subnutrido
- de 18,5 até 25 → Peso ideal
- de 25 até 30 → Levemente acima do peso
- de 30 até 35 → Obesidade
- de 35 até 40 → Obesidade Severa
- Acima de 40 → Obesidade Mórbida





# Laços de Repetição

Eventualmente precisaremos efetuar mais de uma vez os mesmos trechos de códigos, ou até mesmo para chegarmos a uma conclusão a mesma instrução precisa ser repetida sequencialmente, na matemática temos por exemplo a multiplicação que são somas consecutivas e a potenciação que são multiplicações consecutivas.

Existem dois tipos de laços de repetição em python *for* e *while*.

## While

Este é um laço de repetição que tem a propriedade de repetir o conjunto de código quantas vezes forem necessárias, até que as condições não forem mais respeitadas, ou seja quando o teste lógico que segue a abertura da instrução for igual a *False* a instrução *while* finalizará.

Escreva o código abaixo em um script e teste:

```
1. #coding: utf8
2. x = 0
3. while x < 10:
4.     x = input('informe o valor: ')
5.     print x
6.     if x < 10:
7.         print "tente novamente"
```

```
8.     else:
```

```
9.         print "Agora sim"
```

```
10.
```

Neste código instanciamos na linha dois a variável x com antes de iniciar a instrução **while** por que para executar um teste lógico é preciso que a variável exista.

Na linha três temos a abertura do código, propositalmente o valor do teste lógico é True, se fosse False nada mais do código seria executado.

Na linha quatro temos uma parte muito importante, a capacidade de tornar o código False, sem isso teríamos o chamado *loop infinito*, isto é, a incapacidade de finalizar o laço de repetição. Se em alguma situação de erro, entres em um loop infinito podes parar a execução do código com o comando <ctrl+c>.

O restante do código do script nos já conhecíamos anteriormente, então dispensa maiores explicações.

## **For**

Já o laço de repetição **for** é utilizado quando temos um limite de repetições pré definido, normalmente ele é usado para percorrer uma lista ou dicionário definido anteriormente mas também pode ser usado simplesmente para executar um código um certo número de vezes com a ajuda do comando **range()**.

## **Range**

Range é um comando capaz de criar uma sequencia numérica, range tem três parâmetros inicio, fim e passo sendo que o primeiro e o ultimo podem ser omitidos, a sintaxe segue abaixo:

```
1.     range(inicio, fim, passo)
```

**Inicio** → inicio é um número inteiro que como já pode ser obvio para alguns é a partir dele que nossa sequencia numérica vai começar, se for omitido o inicio será zero.

**Fim** → fim é o valor máximo da lista, não pode ser omitido porque se fosse possível mais uma vez teríamos nosso temido loop infinito.

**Passo** → passo define de quanto em quanto nossa sequencia será formada, se omitirmos este valor será considerado 1, isto é, cada item da lista será o próximo inteiro, se usarmos por exemplo 5 teremos um salto de 5 em 5, bem como, se utilizarmos um número negativo como -2 por exemplo a sequencia será decrescente mas é preciso considerar que, neste caso, o valor *fim* deve ser menor que o valor *inicio*.

Monte este script com uma serie de exemplos utilizando *for* e *range()*

```
1.     #coding: utf8
```

```
2.     #for com range() indo do zero até menor
que 10 de um em um
```

```
3.     print "de 0 a 9"
4.     for i in range(0,10,1):
5.         print i
6.     print "de 0 a 9"
7. #a mesma coisa feita com menos código
8.     for i in range(10):
9.         print i
10.    #indo do 5 até 15 de 3 em 3
11.    print "indo do 5 até 15 de 3 em 3"
12.    for i in range(5,15,3):
13.        print i
14.    #contagem regressiva do 10 até maior que 0
15.    print "contagem regressiva"
16.    for i in range(10,0,-1):
17.        print i
```

Nem só de *range* vive um *for* sua principal função é percorrer listas e isso é muito fácil com este comando, veja um exemplo abaixo:

```
1. #coding: utf8
2. a = [2, 3, "abc", "xyz"]
3. for i in a:
4.     print i
```

A mesma maneira funciona para dicionários entretanto vou mostrar uma maneira de ler os valores dos dicionários e também suas chaves o que pode ser importante para seus programas no futuro.

```
1. #coding: utf8
2. mydic = {'prova':7, 'trabalho':3}
3. for key, value in mydic.items() :
4.     print "A chave é ", key, " e o valor é ",  
value
```

Lembre que dicionários não tem ordem.





## ***Exercício 9 - Exercícios com laços de repetição***

Crie um script que leia uma tecla do usuário e imprima na tela, uma a uma, até que ele pressione a tecla M (lembre que o python diferencia maiúsculas e minúsculas).

## ***Exercício 10 - Exercícios com laços de repetição***

Crie um script que receba um número do usuário e após verifique se é menor que 100, se for mostre os números inteiros até 100, se não mande uma mensagem para o usuário avisando que o número deve ser menor que 100 e repita até que o numero seja menos que 100.



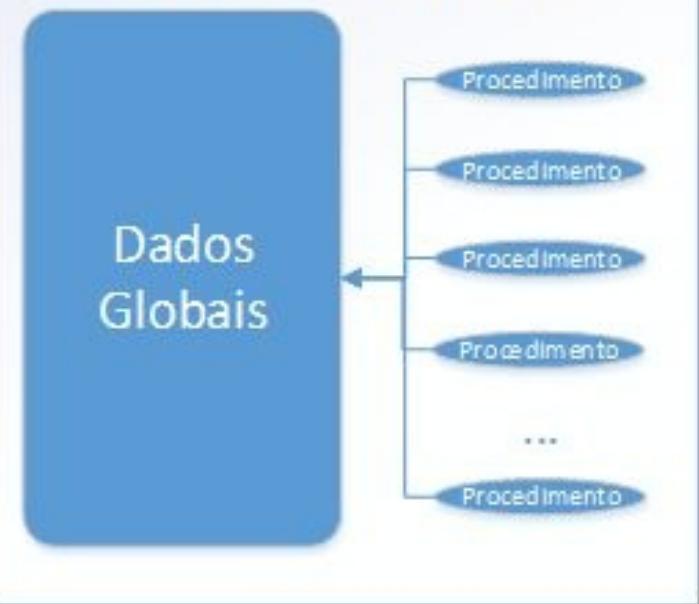
# **Alguns conceitos 'avançados'**

Como último passo da nossa revisão alguns conceitos serão importantes para que o nosso estudo possa seguir sem mais demora, aqui é algo mais teórico então fique tranquilo, leia com atenção e em breve voltaremos aos códigos.

## **Orientação a Objetos**

Orientação a Objetos ou Programação Orientada a Objetos é um paradigma de programação dos mais populares atualmente, diferente a programação estruturada que usa procedimentos ou funções para ler dados globais na Programação Orientada a Objetos os métodos e funções são aplicados dentro dos objetos retornando resultados globais, cada um destes paradigmas tem sua colocação e sua utilidade, no web2py vamos usar orientação a objetos.

## Programação Estruturada



## Programação Orientada a Objetos

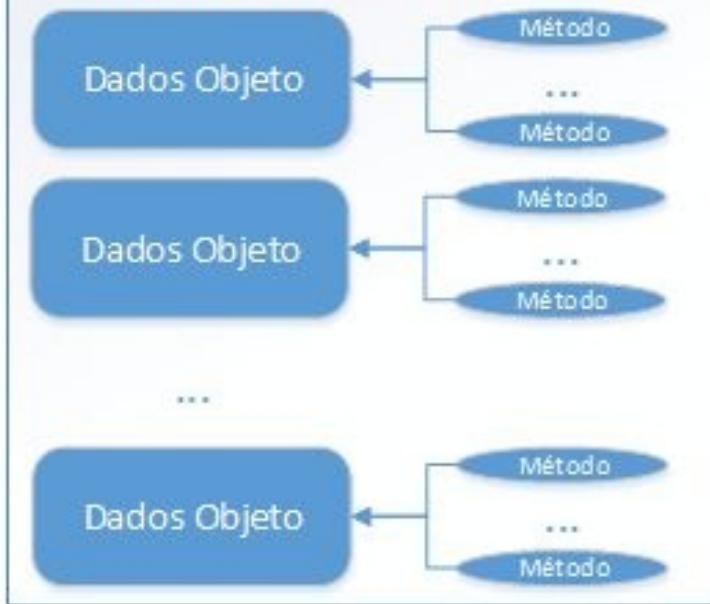


Figura 4: Programação Estruturada x Programação Orientada a Objetos

Na Orientação a Objetos são criados modelos, escopos dos objetos que são chamados de classes, as classes tem tudo que um objeto deve ter quando for instanciado, e a partir dela, da classe, podemos instanciar os objetos que vamos usar, uma classe nunca é usada diretamente, as classes funcionam como uma planta de uma casa, ela serve para construir casas mas não podemos morar em uma planta de casa.

Recomendo a leitura do artigo “*Os quatro pilares da Programação Orientada a Objetos*” do Henrique Machado Gasparotto<sup>4</sup> para entender melhor esses conceitos, não vamos nos ater neste assunto porque daria um livro inteiro só disto, mas é importante conhecer.

## MVC

O MVC ou Model, View, Controller, em português Modelo, Visão, Controlador é um padrão de arquitetura de Software que divide o desenvolvimento do aplicativo em camadas, neste caso três camadas.

A primeira camada é chamada de Model ou Modelos, essa camada é responsável por tratar os dados que serão armazenados no banco ou lidos no banco de dados e exibidos na tela, então qualquer comando, seja leitura, alteração, exclusão, inserção, criação e tudo mais que envolva dos dados armazenados será escrito nesta camada.

A segunda camada é o controler ou Controlador, nesta camada estará a lógica da aplicação, passando dados para as demais camadas, é interessante tentar concentrar o máximo de logica nesta camada, mas sempre teremos lógicas nas outras camadas.

A terceira camada é aquela que 'formata' os dados para o usuário e recebe comandos dele, é nesta camada que teremos a aparência do nosso aplicativo e onde ficará os códigos HTML e CSS mesclado com código python.

Teremos um capítulo para cada uma destas camadas então não vamos nos aprofundar muito nestes conceitos também.

# WEB2PY

Version 2.12.3-stable+timestamp.2015.08.19.00.18.03

Created by Massimo Di Pierro, Copyright 2007-2016

Server IP:

- Local (IPv4) (127.0.0.1)
- Local (IPv6) (::1)
- Public (127.0.1.1)
- Public (0.0.0.0)

Server Port:

Choose Password:



Figura 5: WebServer do Web2py

Vamos dar uma olhada nisso, abra a pasta onde tu descompactou o web2py e rode o web2py.py deve abrir uma janela como esta:

Este é o webserver do web2py é a partir dele que vamos iniciar o desenvolvimento dos nossos aplicativos, o próximo passo é escolher se o server será local (somente na máquina) ou em rede e se usará IP versão 4 ou 6, para nosso desenvolvimento local podemos usar local e IPv4, na dúvida, se tu não faz ideia do que eu estou falando não mude nada.

Em seguida defina uma senha para o administrador, que terá acesso pleno a todos os aplicativos e inicie o servidor.

Ao iniciar o web2py deve chamar o seu navegador de internet padrão abrindo a aplicação welcome que é padrão do web2py.

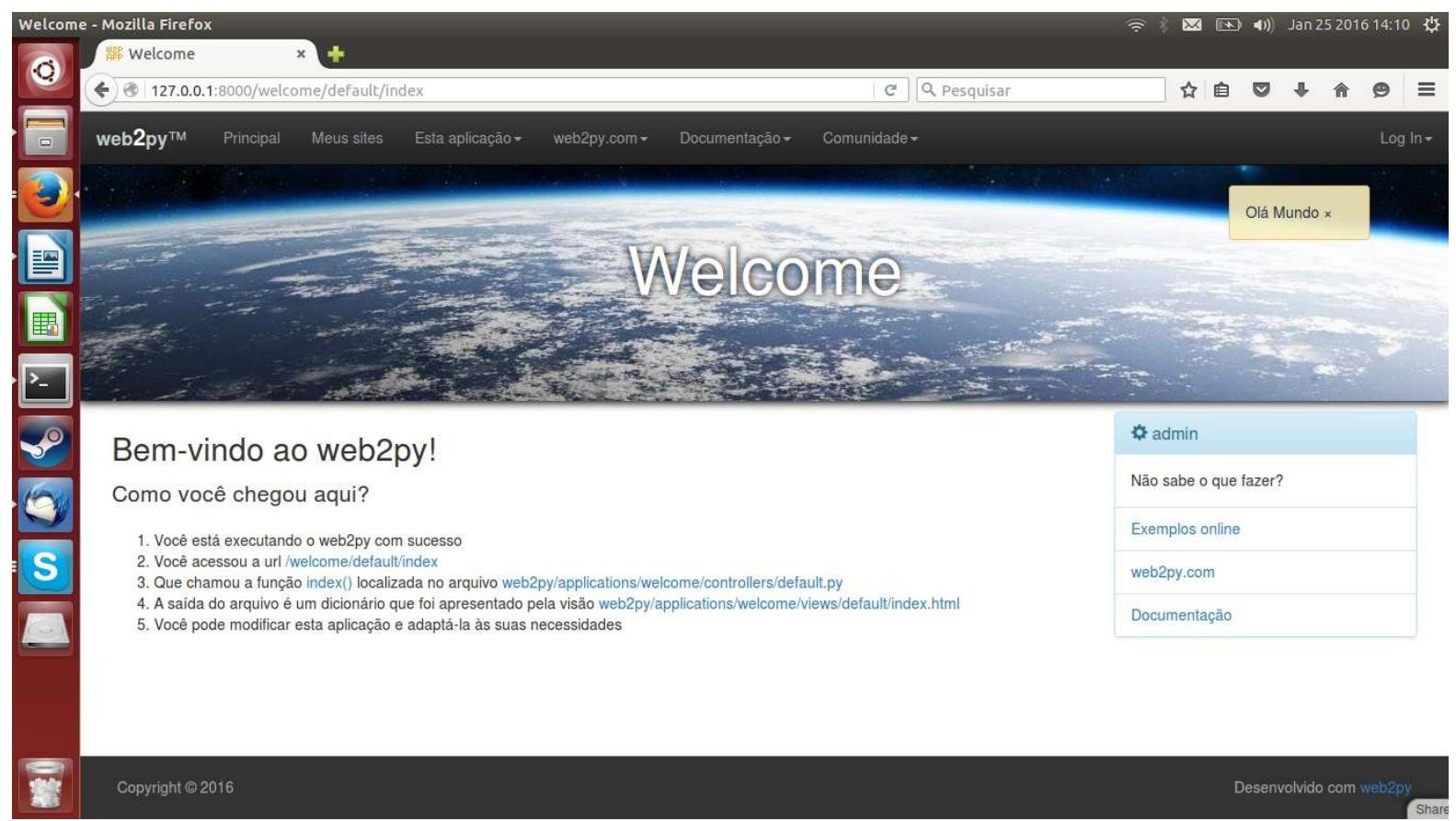


Figura 6: Aplicação Welcome

Após isto encontrarás no lado direito um pequeno menu, com a opção “admin”, ao clicar nesta opção carregará uma tela de login onde deves colocar a mesma senha anteriormente escolhida.

Tendo feito isto a seguinte tela será apresentada:

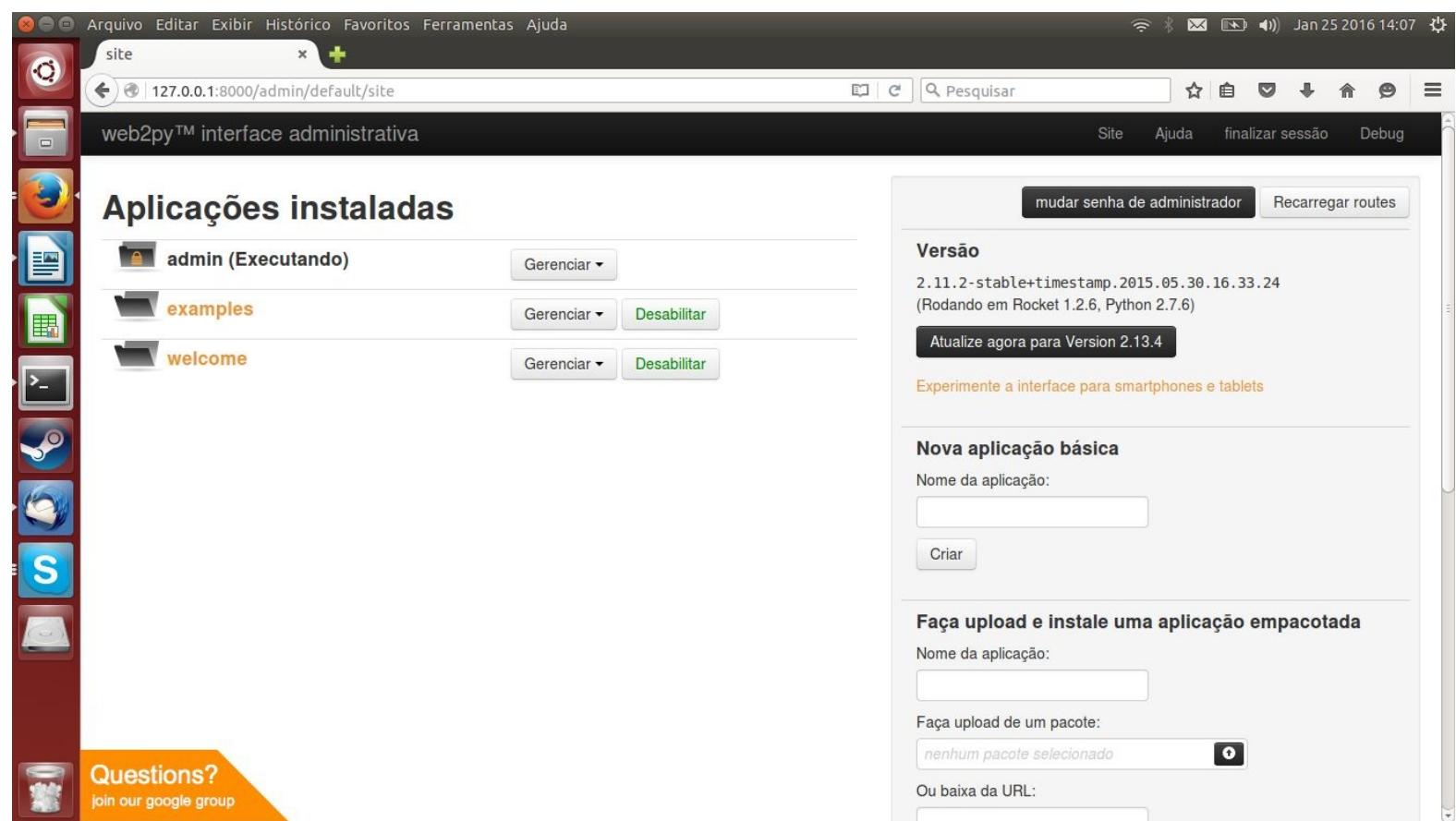


Figura 7: Painel Administrativo

Também no lado direito temos a opção “Nova Aplicação Básica”, no campo Nome da Aplicação escreva “tarefa” e clique criar.

Como é esperado será criada uma nova aplicação neste servidor, esta aplicação será um clone exato da aplicação welcome, e já será aberto a interface de controle desta aplicação.

Como visto abaixo:

design tarefa - Mozilla Firefox

design tarefa

127.0.0.1:8000/admin/default/design/tarefa

Pesquisar

web2py™ interface administrativa

Site Editar sobre Erros Versionamento Ajuda finalizar sessão Debug

## Editar aplicação "tarefa"

Modelos

administração do banco de dados sql.log graph model

db.py  
menu.py

Criar

## Controladores

Terminal testar crontab

appadmin.py expõe bg\_graph\_model, ccache, csv, download, graph\_model, hooks, index, insert, manage, select, state, update  
default.py expõe call, download, index, user

Criar

## Visões

Download layouts from repository

init\_.py  
appadmin.html estende layout.html

This screenshot captures the web2py administrative interface for the 'tarefa' application. At the top, the browser title bar reads 'design tarefa - Mozilla Firefox' and the address bar shows '127.0.0.1:8000/admin/default/design/tarefa'. The main header includes links for Site, Editar, sobre, Erros, Versionamento, Ajuda, finalizar sessão, and Debug. Below the header, the title 'Editar aplicação "tarefa"' is displayed. The interface is organized into three main sections: 'Modelos', 'Controladores', and 'Visões'. The 'Modelos' section contains tabs for 'administração do banco de dados', 'sql.log', and 'graph model', with links to edit 'db.py' and 'menu.py'. The 'Controladores' section contains tabs for 'Terminal', 'testar', and 'crontab', listing 'appadmin.py' and 'default.py' with their exposed functions. The 'Visões' section contains a link to 'Download layouts from repository' and lists 'init\_.py' and 'appadmin.html' under 'estende layout.html'.

Figura 8: Interface Administrativa da Aplicação tarefa

Como é possível ver nesta tela temos fácil acesso aos Modelos, aos Controladores e as Visões, um pouco mais abaixo temos as traduções, arquivos Estáticos (normalmente usados para estilos) e ainda módulos e plugins que são próprios do web2py mas não vamos entrar neste ponto agora.

Agora vamos falar dos Modelos.



# Parte 2 – Model

Segundo Thiago Belem[O que é MVC]

“O model é a camada que **representa os seus dados**, provendo meios de acesso (leitura e escrita) à esses dados.

A regra é simples: tudo que diz respeito à **escrita, validação e leitura** dos dados está dentro da **camada model**, não necessariamente dentro do model em si, mas dentro da camada model.”

No contexto do web2py nos temos um diretório/pasta chamada **model** dentro do diretório de cada aplicação, ou seja, no nosso exemplo teremos a seguinte estrutura de diretórios:

```
_teste  
  \_model  
  \_view  
  \_default  
  \_controler  
  \_databases
```

Obviamente temos outras pastas e arquivos mas para nosso contexto MVC são essas que temos que ver agora.

Então sempre que construirmos algum código ou arquivo que manipule dados do banco de dados devemos deixá-lo dentro do diretório model, não que o código não funcionaria, mas se existe a estrutura é melhor respeitá-la.

Ao criar nossa aplicação básica será criado além dos diretórios uma série de arquivos que deixarão o sistema plenamente funcional, claro que é uma funcionalidade genérica mas esse será nosso ponto de partida.

Analisando o diretório dos modelos, temos dois arquivos recém-criados:

***db.py***

***menu.py***

Cada arquivo criado dentro do diretório model será lido e consequentemente executado seu código, e isso é muito importante, a ordem da leitura dos arquivos será alfabética crescente, ou seja de A a Z de 0 a 9 e sucessivamente. Já dentro do arquivo a leitura é feita linha por linha de cima para baixo como qualquer outro script do python, lembre-se, web2py ainda é python.

No caso de não criarmos mais nenhum outro model o arquivo ***db.py*** será lido primeiro e ***menu.py*** será lido e executado depois.

Se utilizarmos o que o web2py nos apresenta teremos os principais comandos do model no arquivo db.py vamos ver algumas linhas que são importantes, então clique no botão editar no lado esquerdo do nome do arquivo.





# DAL

Logo no começo do arquivo encontramos o comando DAL algo semelhante a isso:

```
db  
=  
DAL('sqlite://storage.db')
```

Vamos entender, o comando DAL remete a uma sigla para **Data Abstraction Layer** ou **Camada de Abstração de Dados**, ou seja, não será necessário conhecer os comandos de um sistema gerenciador de banco de dados específico, basta conectar o web2py ao banco de dados e usar os mesmos comandos que o web2py vai interpretar corretamente para o sistema gerenciador de banco de dados.

Isso é muito prático porque não será necessário aprender banco de dados use o banco de dados que tu já sabe, não sabe? Infelizmente não ensinarei aqui, aprenda e vamos continuar.

Para que possas fazer isso segue abaixo uma tabela com exemplos de conexão em diversos bancos de dados adaptado do livro do Massimo Di Pierro (que tu precisas ler).

**SQLite**

sqlite://storage.db

**MySQL**

mysql://usuario:senha@servidor/ban

**PostgreSQL**

postgres://usuario:senha@servidor/

**MSSQL (legacy)**

mssql://usuario:senha@servidor/ban

**MSSQL (>=2005)**

mssql3://usuario:senha@servidor/ba

**MSSQL (>=2012)**

mssql4://usuario:senha@servidor/ba

**FireBird**

firebird://usuario:senha@servidor/

**Oracle**

oracle://usuario/senha@banco

**DB2**

db2://usuario:senha@banco

**Ingres**

ingres://usuario:senha@servidor/ba

**Sybase**

sybase://usuario:senha@servidor/ba

**Informix**

informix://usuario:senha@banco

**Teradata**

teradata://DSN=dsn;UID=usuario;PWD

**Cubrid**

cubrid://usuario:senha@servidor/ba

**SAPDB**

sapdb://usuario:senha@servidor/ban

**IMAP**

imap://usuario:senha@servidor:port

**MongoDB**

mongodb://usuario:senha@servidor/b

**Google/SQL**

google:sql://project:instance/data

**Google/NoSQL**

google:datastore

**Google/NoSQL/NDB**

google:datastore+ndb

Se tu não tiveres certeza do que fazer aqui simplesmente não faça nada, use a configuração padrão vai funcionar, pelo menos para nível de estudo.



# Importações padrão

Mais abaixo um pouquinho encontraremos a importação de algumas funções que podemos ou não usar no nosso projeto, veja:

1. 

```
from gluon.tools import Auth, Service,  
PluginManager
```
- 2.
3. 

```
auth = Auth(db)
```
4. 

```
service = Service()
```
5. 

```
plugins = PluginManager()
```

Essas importações servem respectivamente para utilizar as funções de autenticações de usuários (auth) serviços (services) e complementos (plugins), vamos usar nestes nossos exemplos as autenticações do web2py e todo o resto então não mexa nessas linhas.

As demais linhas logo abaixo são configurações dos comandos acima, inclusive funções de envio de e-mail para recuperação de senha e tudo mais.





# Tabela tarefas

Na ultima linha vamos começar e escrever nosso códigos para criar um pequeno aplicativo que norteará nosso estudo, por falar nisso nosso exemplo será um pequeno agenda de tarefas, essa ideia veio do livro *Desenvolvimento Web com PHP e MySql* de *Evaldo Junior Bento*.

Segundo a modelagem de dados apresentada teremos a tabela que segue abaixo, lembrando que graças a DAL não é preciso especificar qual Sistema Gerenciador de Bancos de Dados, em SQL/MySql a representação ficaria assim.

Tarefa	
Id	Integer
Nome	Varchar
Descricao	Text
Prazo	Date
Prioridade	Integer
Concluida	Boolean

Em web2py temos maneiras um pouquinho diferente de representar cada uma desses dados, e por padrão o web2py já cria um campo id em cada tabela que não precisa ser explicitado, este campo é integer, e autoincrement, então traduzindo a tabela acima teremos:

## Tarefa

Nome	string
Descricao	Test
Prazo	Date
Prioridade	Integer
Concluida	Boolean

Todos os tipos interpretados pelo web2py são os seguintes:

- string → para campos de texto curto;
- text → para campos de texto longo;
- password → para campos de texto oculto;
- integer → para números inteiros;
- double → para números com casas decimais;
- boolean → para valores binários, verdadeiro ou falso;
- date → para datas;
- time → para horas, minutos, segundos;
- datetime → para campos de data e hora;

- blob → dados binários codificados em base64;
- upload → para envio de arquivos;
- reference TABLENAME → para chaves estrangeiras;

Sabendo o que temos que fazer só nos resta saber como, e isso também é bem fácil o comando para criar um tabela vem do objeto DAL que usamos acima e se tu te lembras bem, instanciamos um objeto DAL chamado 'db' então vamos usar este objeto, veja um exemplo.

```
1. db.define_table('tabela',
2.     Field('campo1', 'string'),
3.     Field('campo2', 'integer')
4. )
```

Na primeira linha do nosso exemplo temos o comando ***db.define\_table()*** este comando tem, de modo simplista, três opções, caso a tabela, cujo o nome é o primeiro campo entre as aspas, não exista, será criada a tabela no banco de dados com os campos citados, caso a tabela exista e os campos definidos no código sejam diferentes dos que já existam a tabela será atualizada de acordo com o código, e se a tabela exista e os campos sejam exatamente iguais aos que já citados o comando não fará nada.

Abaixo temos os subcomandos Field que servem para criar os campos das tabelas, a primeira palavra entre aspas será o nome do campo a segunda palavra define o tipo que o campo será.

Sendo assim o código completo para criar a tabela tarefas será o que segue abaixo, abra o arquivo db.py, clicando no botão editar ao lado esquerdo do nome do arquivo no painel administrativo da aplicação teste escreva este código na ultima linha do arquivo db.py.

```
1. db.define_table('tarefa',  
2.     Field('nome', 'string'),  
3.     Field('descricao', 'text'),  
4.     Field('prazo', 'date'),  
5.     Field('prioridade', 'integer'),  
6.     Field('concluida', 'boolean')  
7. )
```

Salve e já está pronto nosso banco de dados, volte ao painel administrativo clicando no menu editar na barra superior.

Para visualizar o modelo gráfico do nosso banco podemos voltar ao painel administrativo e clicar no botão 'graph model', mas no meu caso (ubuntu 14.04) foi preciso instalar o pygraphviz com o comando:

```
sudo apt-get install python-pygraphviz
```

Depois disso tive o seguinte resultado:

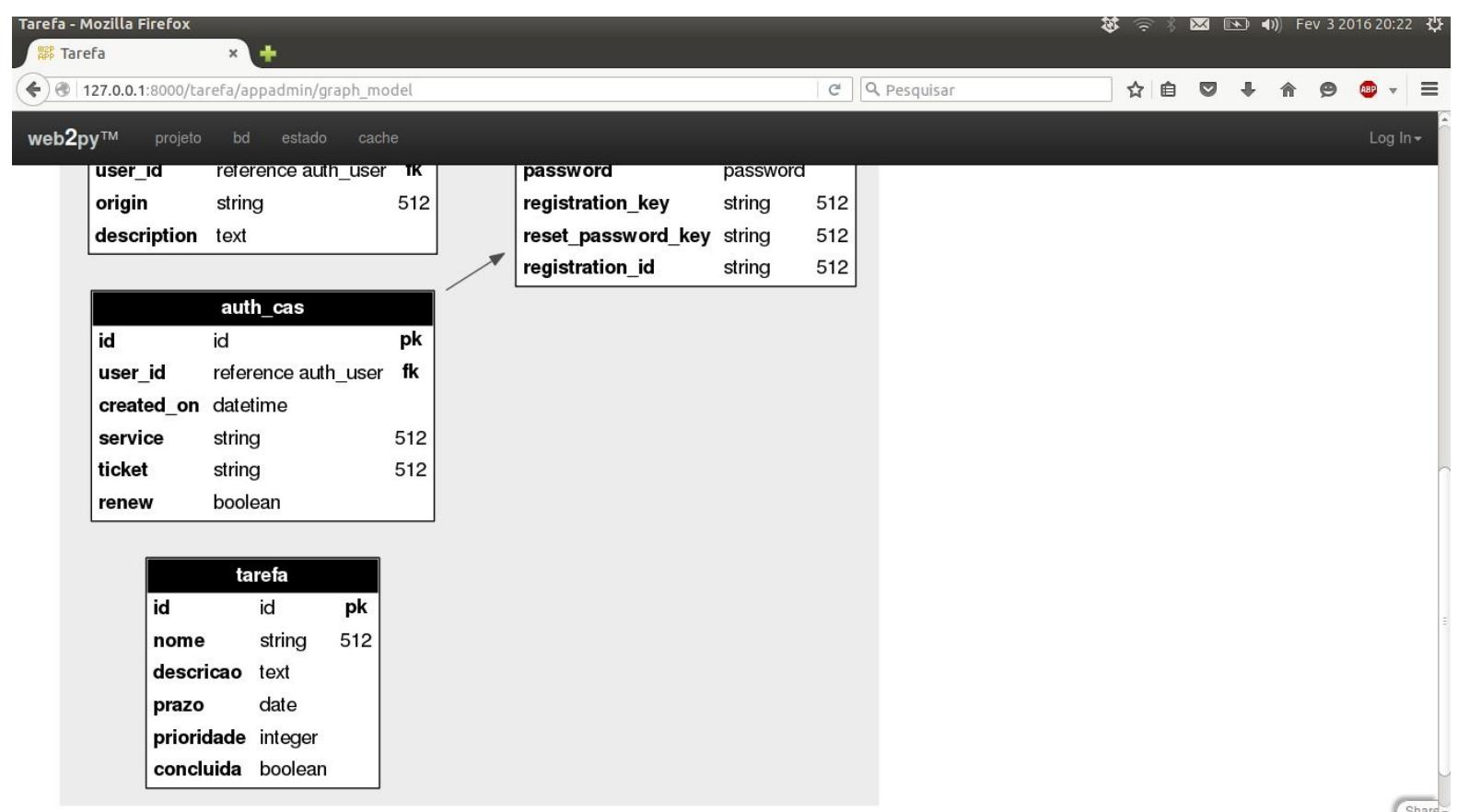
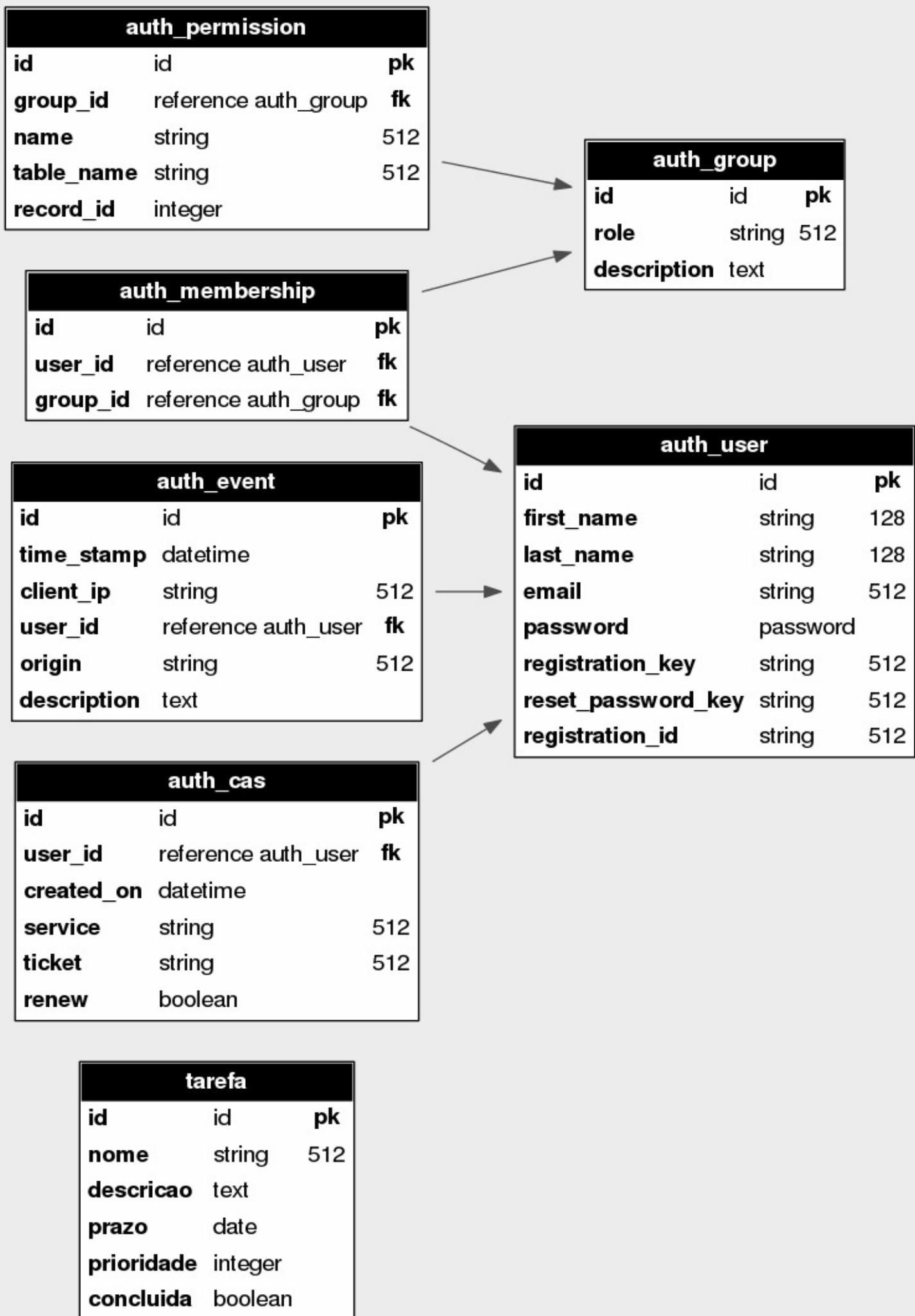


Figura 9: *Modelo Gráfico*

E ainda podemos fazer o download deste modelo, eu baixei em em png e obtive o seguinte resultado.



## tarefa



## *Figura 10: Modelo Gráfico Exportado*

Podemos testar nosso banco de dados e inclusive inserir valores na administração de banco de dados, clique no botão administração de banco de dados dentro da sessão modelos.

Aqui podemos ver uma serie de tabelas criadas, a maioria delas deve-se ao uso da função auth para gerenciamento de usuários, mas se tudo deu certo veremos a tabela 'tarefa' entre as tabelas listadas como mostra na figura.

Este - Mozilla Firefox

127.0.0.1:8000/teste/appadmin/index

web2py™ projeto bd estado cache Log In ▾

## Bancos de dados e tabelas disponíveis

Tables Hooks

<a href="#">db.auth_user</a>	<a href="#">Novo Registro</a>
<a href="#">db.auth_group</a>	<a href="#">Novo Registro</a>
<a href="#">db.auth_membership</a>	<a href="#">Novo Registro</a>
<a href="#">db.auth_permission</a>	<a href="#">Novo Registro</a>
<a href="#">db.auth_event</a>	<a href="#">Novo Registro</a>
<a href="#">db.auth_cas</a>	<a href="#">Novo Registro</a>
<a href="#">db.tarefa</a>	<a href="#">Novo Registro</a>

Copyright © 2016 Desenvolvido com web2py Share

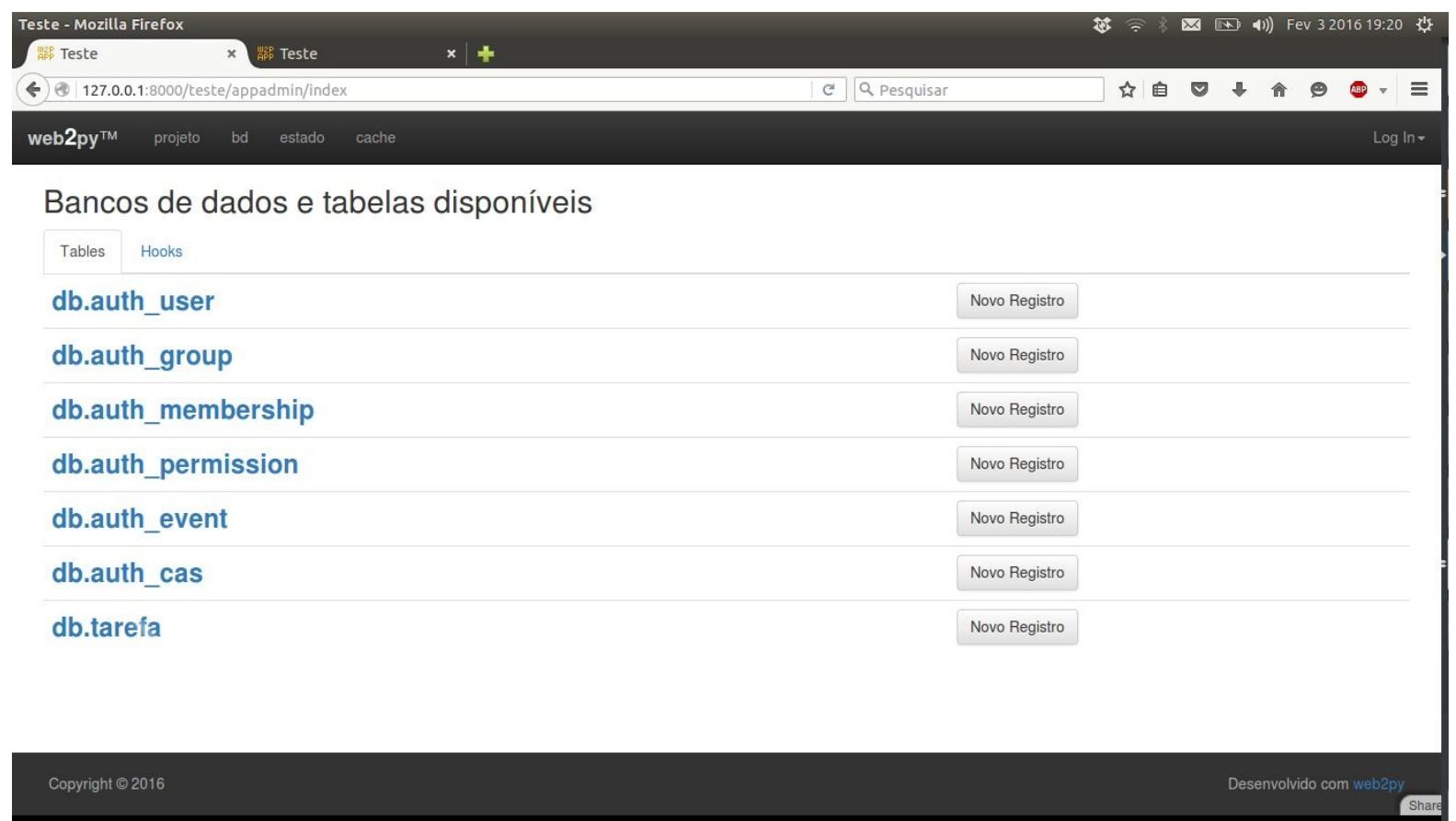


Figura 11: Administração de Banco de Dados

Clicando no nome da tabela, neste caso db.tarefas teremos a listagem do conteúdo, e algumas funções como, inserir novos dados, baixar ou importar informações de arquivos CSV entre outros. Veja:

The screenshot shows a Mozilla Firefox browser window titled 'Tarefa - Mozilla Firefox'. The address bar displays the URL '127.0.0.1:8000/tarefa/appadmin/select/db?query=db.tarefa.id>0'. The page content is titled 'Selecionar banco de dados db'. It includes a 'Novo Registro' button, a section for 'Linhos na tabela' with input fields for 'Consulta' (containing 'db.tarefa.id>0'), 'Atualizar' (checkbox), and 'Apagar' (checkbox), followed by an 'enviar' button. A note explains SQL query syntax. Below this, it says '0 selecionado'. There's also an 'Importar/Exportar' section with 'exportar como um arquivo csv' and 'ou importar de um arquivo csv' buttons.

*Figura 12: Administração de Banco de Dados*

Para testar se está tudo ok, vamos acrescentar algumas tarefas, então clique no botão 'Novo Registro'. Note que o formulário de inserção dos dados foi criado na ordem e com os nomes dos campos que criamos no arquivo db.py, o campo id não aparece pois é auto preenchido pelo próprio web2py.

Algumas validações já são feitas automaticamente pelo tipo de dado escolhido como a data e a prioridade que não aceitam valores diferentes dos que foram definidos. Faça este processo umas duas ou três vezes. E retorne para a administração de banco de dados da tabela tarefa.

Tarefa - Mozilla Firefox

127.0.0.1:8000/tarefa/appadmin/insert/db/tarefa

web2py™ projeto bd estado cache Log In

banco de dados db Tabela tarefa

Novo Registro

Nome	Estudar Web2py
Descrição	Usar o livro do professor Mauro Duarte para aprender mais sobre web2py.
Prazo	03-02-2016
Prioridade	1
<input checked="" type="checkbox"/> Concluída	
<input type="button" value="Submit"/>	

Share

Figura 13: Administração de Banco de Dados

Ao preencher e retornar a administração de banco de dados já nos dará uma consulta retornando todos os itens listados no banco, e cada

item poderá ser editado clicando no seu 'id'.

Já temos o esqueleto do nosso aplicativo pronto, inclusive alguns testes pode ser feitos para identificar se será preciso modificar o aplicativo logo no seu inicio, retornando a criação das tabelas se for necessário, mas, tome cuidado com isso pois inconsistências nos campos podem ocasionar perda de dados, então quanto antes encontrares a forma perfeita melhor.

Por exemplo, nossa tabela está com o campo prioridade numérico, entretanto acho que fica melhor se o usuários puder escolher se a prioridade da tarefa é baixa, média ou alta, então vamos fazer isso utilizando validações.



# Validações

Só o fato de definirmos tipos de dados já cria algumas validações automáticas mas muitas outras situações precisaremos especificar o que se quer. A seguinte alteração no código deve ser feita para permitir que valores de texto possam ser inseridos no campo prioridade em vez de numéricos como feito anteriormente.

```
1. db.define_table('tarefa',  
2.     Field('nome', 'string'),  
3.     Field('descricao', 'text'),  
4.     Field('prazo', 'date'),  
5.     Field('prioridade', 'string'),  
6.     Field('concluida', 'boolean')  
7. )
```

Isso somente permitirá escrever textos no campo prioridade, mas o que realmente queremos é escolher em uma lista, para fazer isso vamos usar uma validação, esta validação pode ser feita diretamente dentro do comando **Field** mas fica um pouco mais organizado se fizermos separadamente, fazendo as seguintes alterações e acrescentando a seguinte linha:

```
1. Tarefa = db.define_table('tarefa',  
2.     Field('nome', 'string'),  
3.     Field('descricao', 'text'),
```

```
4.     Field('prazo', 'date'),  
5.     Field('prioridade', 'string'),  
6.     Field('concluida', 'boolean')  
7. )  
8. Tarefa.prioridade.requires=IS_IN_SET( ('Bai
```

Nesta modificação primeiro atribuímos a criação da tabela a uma variável, fazendo isso qualquer ação que envolva a tabela pode ser feita pela variável sem precisar chamar o comando db, a diferença é pouca.

Já na última linha temos, a variável que representa a tabela do banco, seguido do nome do campo que será validado, a rotina 'requires' é que chama a validação neste exemplo temos 'IS\_IN\_SET' que significa algo como 'esteja nesta lista' que recebe uma lista ou vetor.

Este código obriga o usuário a usar um dos itens da lista para preencher o campo, volte a administração de banco de dados e insira um novo valor para ver como as alterações estão se comportando.

Alguns detalhes do validador 'IS\_IN\_SET' por padrão não será aceito valores nulos ou vazios. É possível preencher o campo com um valor informativo, não válido acrescentando o comando zero='valor' onde *valor* é uma string, veja:

```
1. Tarefa.prioridade.requires=IS_IN_SET( ('Bai
```

Abaixo seguem alguns validadores possíveis:

CLEANUP

Remove caracteres especiais.

**Tabela.campo.requires=CLEANUP()**

	Codifica o campo, por padrão, com o algoritmo criptográfico definido no campo <code>CRYPT</code> .
<code>CRYPT</code>	<b>Tabela.campo.requires=CRYPT()</b>
<code>IS_ALPHANUMERIC</code>	Permite apenas valores letras, números e espaço.
	<b>Tabela.campo.requires=IS_ALPHANERIC()</b>
<code>IS_DATE_IN_RANGE</code>	Permite datas dentro de um determinado intervalo.
	<b>Tabela.campo.requires=IS_DATE_IN_RANGE()</b>
<code>IS_DATE</code>	Permite somente data (O campo tipo date).
	<b>Tabela.campo.requires=IS_DATE()</b>
<code>IS_DATETIME_IN_RANGE</code>	Permite datas com horas e minutos dentro de um determinado intervalo.
	<b>Tabela.campo.requires=IS_DATETIME_IN_RANGE()</b> <code>maximum=datetime.datetime(2009,12,31,23,59)</code>
<code>IS_DATETIME,</code>	Permite datas com horas e minutos. O formato é YYYY-MM-DD HH:MM:SS.
	<b>Tabela.campo.requires=IS_DATETIME()</b>
<code>IS_EMAIL,</code>	Permite texto formatado como e-mail, podendo conter múltiplas linhas.
	<b>Tabela.campo.requires=IS_EMAIL()</b>
	<b>Tabela.campo.requires=IS_EMAIL(formato)</b>
	<b>Tabela.campo.requires=IS_EMAIL(base)</b>

IS_EMPTY_OR,	Essa validação é uma das mais versáteis e pode ser usada para validar se um campo é vazio ou não.
	<b>Tabela.campo.requires=IS_EMPTY_O</b>
IS_EXPR,	Esta validação permite criar uma expressão que pode ser usada para validar o campo.
	<b>Tabela.campo.requires=IS_EXPR('5&lt;id&lt;10')</b>
IS_FLOAT_IN_RANGE,	Permite valores numéricos com casas decimais dentro de um intervalo definido.
	<b>Tabela.campo.requires=IS_FLOAT_IN_RANGE(5.0, 10.0)</b>
IS_IMAGE,	Esta validação é usada normalmente juntamente com a validação IS_TYPE, para validar se uma imagem é do tipo correto.
	<b>Tabela.campo.requires=IS_IMAGE()</b>
IS_IN_DB,	Esta validação serve para ser usada em relacionamentos para validar se o valor existe em uma determinada tabela.
	<b>Tabela.campo.requires=IS_IN_DB(db, tabela, coluna)</b>
IS_IN_SET,	Esta validação aceita valores em uma lista definida.
	<b>Tabela.campo.requires=IS_IN_SET(('a', 'b', 'c'))</b>
IS_INT_IN_RANGE,	Permite valores numéricos sem casas decimais dentro de um intervalo definido.
	<b>Tabela.campo.requires=IS_INT_IN_RANGE(5, 10)</b>
IS_IPV4,	Permite valores do tipo IPv4 ou seja 4 colunas de números entre 0 e 255, com máscara para limitar as opções.
	<b>Tabela.campo.requires=IS_IPV4()</b>

	Valida o comprimento da informação em limites.
IS_LENGTH,	<b>Tabela.campo.requires=IS_LENGTH()</b>
	<b>Tabela.campo.requires=IS_LENGTH(i)</b>
IS_LOWER,	Esta validação torna o conteúdo inserido em minúsculas. Caso contrário, elas serão modificadas para tornarem-se minúsculas.
	<b>Tabela.campo.requires=IS_LOWER()</b>
IS_MATCH,	Esta validação permite usar expressões regulares para validar se o valor corresponde ao padrão desejado.
	<b>Tabela.campo.requires=IS_MATCH('')</b>
IS_EQUAL_TO,	Esta validação permite validar se o valor é igual ao especificado.
	<b>Tabela.campo.requires=IS_EQUAL_T</b>
IS_NOT_EMPTY,	Esta validação verifica se o valor não é um espaço em branco.
	<b>Tabela.campo.requires=IS_NOT_EM</b>
IS_NOT_IN_DB,	Esta validação verifica se o valor não está presente em uma determinada tabela.
	<b>Tabela.campo.requires=IS_NOT_IN_D</b>
IS_NULL_OR,	Esta validação funciona semelhante a IS_NOT_IN_DB, exceto que aceita valores nulos.
	<b>Tabela.campo.requires=IS_NULL_OR</b>
IS_STRONG,	Exige que o valor seja 'forte' no sentido de que deve ser resistente a ataques de força bruta.

## Tabela.campo.requires=IS\_\_STRONG

IS\_TIME,

Permite somente tempo em horas e minutos.

## Tabela.campo.requires=IS\_TIME()

IS\_UPLOAD\_FILENAME,

Esta validação filtra os nomes dos arquivos.

## Tabela.campo.requires=IS\_UPLOAD\_

IS\_UPPER,

Esta validação torna o conteúdo inserido maiúsculo, elas serão modificadas para tornarem-se maiúsculas.

## Tabela.campo.requires=IS\_UPPER()

IS\_URL

Valida se o campo corresponde a um endereço de URL.

## Tabela.campo.requires=IS\_URL()

Alguns validadores aceitam outros critérios além dos citados acima, um deles é o *format* para aceitar formatos de data e hora, exemplo:

`format="%m/%d/%Y"`

Segue abaixo os códigos de formato com alguns exemplos:

Formato	Descrição	Exemplo
<code>%Y</code>	Ano com quatro dígitos.	'1963'

%y	Ano com dois dígitos.	'63'
%d	Dia com dois dígitos.	'28'
%m	Mês com dois dígitos.	'08'
%b	Mês abreviado com três letras, em inglês, por padrão.	'Aug'
%B	Mês por extenso, em inglês, por padrão.	'August'
%H	Horas no formato de 24 horas.	'14'
%I	Horas no formato de 12 horas.	'02'
%M	Minutos com dois dígitos.	'30'
%S	Segundos com dois dígitos.	'59'

Vamos ajustar ainda mais nossa tabela com algumas validações das que vimos acima , então modifique o código para que fique desta forma:

1. Tarefa = db.define\_table('tarefa',
2. Field('nome', 'string'),
3. Field('descricao', 'text'),
4. Field('prazo', 'date'),

```
5.     Field('prioridade', 'string'),  
6.     Field('concluida', 'boolean')  
7. )  
8. Tarefa.prioridade.requires=IS_IN_SET ('Baixa', 'Alta')  
9. Tarefa.nome.requires=IS_NOT_EMPTY()  
10. Tarefa.prazo.requires=IS_EMPTY_OR(IS_DATE(
```

Existe mais um ajuste que quero fazer nesta tabela, e veja que estamos fazendo todas estas coisas no model porque isso irá propagar cada vez que usarmos referencia a tabela nos models, nos controllers e views.

As próximas alterações são as seguintes, quando vamos testar nossa tabela na administração de banco de dados as indicações dos campos estão exatamente iguais aos nomes dos campos na tabela da base de dados. Porem se prestarmos atenção isso mostrará na tela erros de português, isso pode ser resolvido de duas formas através do sistema de tradução ou através de etiquetas, que é o que vamos fazer agora.

```
1. Tarefa = db.define_table('tarefa',  
2.     Field('nome', 'string'),  
3.             Field('descricao',  
4.                 'text', label='Descrição'),  
5.     Field('prazo', 'date'),  
6.     Field('prioridade', 'string'),  
7.     Field('concluida', 'boolean'))
```

```
5.     Field('prioridade', 'string'),  
6.             Field('concluida',  
'boolean', label='Concluída')  
7.         )  
8.     Tarefa.prioridade.requires=IS_IN_SET(['Bai:  
9.     Tarefa.nome.requires=IS_NOT_EMPTY()  
10.    Tarefa.nome.requires=IS_NOT_IN_DB(db,  
Tarefa.nome)  
11.    Tarefa.prazo.requires=IS_EMPTY_OR(IS_DATE(:
```

Já temos aqui uma grande noção dos modelos, já podemos falar dos controles, mas é claro, voltaremos a falar nisso no futuro, neste mesmo livro.



# Parte 3 - Controller

Os controles são a parte principal de uma aplicação web2py, a partir deles que são chamados os modelos e as visões. Cada url digitada é interpretada a partir do controller e do sistema de mapeamento de url do web2py que funciona por padrão assim.

`http://servidor/aplicação/controle/função/argumentos?variáveis`

Ou seja, dentro de cada aplicação deve existir ao menos um controle, e este controle deve ter ao menos uma função, os argumentos podem ou não existir dependendo da função.

A seguinte url:

`http://meusite.com/blog/default/inicio.html`

Seria interpretada desta forma:

- Servidor → meusite.com
- aplicação → blog
- controle → default
- função do controle → inicio
- formato da resposta para o cliente → html

As formas de apresentação serão detalhadas no capítulo sobre as visões.

Alguns padrões são usados caso informações sejam omitidas a aplicação padrão é 'welcome', o controle padrão é 'default', a função padrão é 'index' e a apresentação padrão é 'html'.

Lembrando que a ordem da url precisa ser respeitada para omissões, ou seja, é preciso omitir sempre as ultimas informações, se omitir o controle sem omitir a função por exemplo, a função será interpretada como um controle e causará um erro.

Veja nosso exemplo anterior, se omitirmos o controle, apesar de ser o controle default isso retornará uma erro, veja:

<http://meusite.com/blog/inicio.html>

Seria interpretada desta forma:

- Servidor → meusite.com
- aplicação → blog
- controle → inicio

A partir daqui temos um erro, pois, não existe controle inicio na nossa aplicação, por isso preste bastante atenção.





# Default.py

Quando criamos uma nova aplicação básica com web2py, da mesma forma que são criados alguns modelos padrão, também são criados alguns controles e funções padrão dentro deles, no caso temos estes dois controles.

## appadmin.py

## default.py

Sendo que o controle appadmin.py serve para gerenciar nossa aplicação, temos usado muitas das funções escritas aqui para gerenciar nossa aplicação, **porem o aquivo que o usuário que visita a aplicação acessa normalmente é o default.py.**

Como já sabemos, ao criar uma nova aplicação básica ela será basicamente **uma cópia da aplicação welcome**, então é claro que o controle **default.py** já está populado com algumas funções, vamos dar uma olhada.



# Função index()

Logo no inicio encontramos a função index, que é uma das principais funções, já que se omitirmos todas as informações depois do nome do servidor e da aplicação **será chamado o controle default e a função index.**

Aqui já temos algumas informações como o return que basicamente mostrará uma pequena mensagem de título, se quiser ver o que já temos aqui basta acessar o seu servidor e a aplicação tarefa, de uma olhada acesse

<http://127.0.0.1:8000/tarefa/default/index>

Deves ter um resultado como este:

Bem-vindo ao web2py!

Como você chegou aqui?

1. Você está executando o web2py com sucesso
2. Você acessou a url /tarefa/default/index
3. Que chamou a função index() localizada no arquivo web2py/applications/tarefa/controllers/default.py
4. A saída do arquivo é um dicionário que foi apresentado pela visão web2py/applications/tarefa/views/default/index.html
5. Você pode modificar esta aplicação e adaptá-la às suas necessidades

Olá Mundo ×

Não sabe o que fazer?

Exemplos online

web2py.com

Documentação

Copyright © 2016

Desenvolvido com web2py

Figura 14: Função Index

As demais informações que vê, na maioria vem das visões, aguarde para falarmos disso mais tarde.

Outra coisa que veremos mais tarde é a ferramenta de tradução que já está funcionando, lá no controle lemos “Hello Word”, aqui como meu firefox está configurado para português brasil vemos “Olá mundo”.



# **Função user()**

Logo abaixo vemos a função user() esta função é responsável por chamar as funções do *auth*, ou seja, login, logout, perfil, troca de senha e todo o gerenciamento de usuários e grupos necessário.

# **Função download()**

Esta função permite ativar as funções de download e upload de arquivos, é preciso deixar ativada caso usemos o campo upload nos modelos.

# **Função call()**

Esta função ativa uma serie de serviços como suporte a XML, json csv e muitos outros.



# Novas funções

Vamos criar uma função para que o nosso usuário possa ver e gerenciar a tabela tarefas que acabamos de criar, vamos fazer isso da maneira mais obvia e simples possível, não que dizer que seja a melhor maneira mas com certeza é a mais simples.

Então na ultima linha deste arquivo acrescente o seguinte código, não esqueça da endentação, aqui ela é muito importante.

1.     def tarefas ():
2.         grid=SQLFORM.grid(Tarefa)
3.         return locals()

Para visualizar o resultado, como visto pelo usuário visitante, basta acessar no navegador o controle default e a função tarefas, como não será preciso usar nenhum parâmetro depois do nome da função, então podemos acessar pela seguinte url:

<http://127.0.0.1:8000/tarefa/default/tarefas>

O resultado será semelhante ao que vemos abaixo:

Tarefas

		Search	Clear	4 records found	
<b>Id</b>	<b>Nome</b>	<b>Descrição</b>	<b>Prazo</b>	<b>Prioridade</b>	<b>concluída</b>
1	Estudar Web2py	Usar o livro do p...	03/Feb/2016	1	<input checked="" type="checkbox"/> <a href="#">Visualização</a>
2	Estudar Python	Preciso aprender ...	None	2	<input checked="" type="checkbox"/> <a href="#">Visualização</a>
5	DSADSA		None	Alta	<input type="checkbox"/> <a href="#">Visualização</a>
6	NOVA TAREFA	teste	None	Baixa	<input checked="" type="checkbox"/> <a href="#">Visualização</a>

Export: [CSV](#) [CSV \(hidden cols\)](#) [HTML](#) [JSON](#) [TSV \(Spreadsheets\)](#) [TSV \(Spreadsheets, hidden cols\)](#) [XML](#)

[design](#) [request](#) [response](#) [session](#) [db tables](#) [db stats](#)

Copyright © 2016 Desenvolvido com [web2py](#) Share

Figura 15: Função Tarefas Controle Default

E caso já esteja cadastrado e logado a visualização será algo como isto:

Tarefas

Id	Nome	Descrição	Prazo	Prioridade	concluída	
1	Estudar Web2py	Usar o livro do p...	03/Feb/2016	1	<input checked="" type="checkbox"/>	
2	Estudar Python	Preciso aprender ...	None	2	<input checked="" type="checkbox"/>	
5	DSADSA		None	Alta	<input type="checkbox"/>	
6	NOVA TAREFA	teste	None	Baixa	<input checked="" type="checkbox"/>	

Export: [CSV](#) [CSV \(hidden cols\)](#) [HTML](#) [JSON](#) [TSV \(Spreadsheets\)](#) [TSV \(Spreadsheets, hidden cols\)](#) [XML](#)

[design](#) [request](#) [response](#) [session](#) [db tables](#) [db stats](#)

Copyright © 2016 Desenvolvido com [web2py](#) Share

Figura 16: Função Tarefas Controle Default

Este código acima utiliza um dos muitos atalhos, funções prontas que facilitam muitas coisas neste caso utiliza a API *grid* que já cria a listagem dos itens da tabela do banco, paginação, opções de baixar os dados, pesquisar, detalhar reorganizar e muitas outras, até mesmo um botão para inserir novos dados é criado.

Como não existe uma visão para a função *tarefas* do controle *default* é usada uma visão genérica que expande o layout básico, calma já vamos falar das visões.

Apesar de podermos tranquilamente usar esta API para as consultas dos bancos de dados vou mostrar como podemos passar parâmetros e interpretá-los depois na visão.

Então vamos modificar nossa função para que fique desta maneira:

```
1. def tarefas():
2.     tar = db(Tarefa).select()
3.     return dict(tar=tar)
```

Neste caso estamos fazendo uma consulta ao banco de dados e retornando todo seu conteúdo, nossa visão genérica já dá uma ajeitada mas fica bem mais feio do que o *grid* depois faremos uma visão específica para esses dados que vai enfeitar um pouco mais.

Veja como ficou o resultado da nossa consulta:

tarefa.id	tarefa.nome	tarefa.descricao	tarefa.prazo	tarefa.prioridade	tarefa.concluida
1	Estudar Web2py	Usar o livro ...	2016-02-03	1	True
2	Estudar Python	Preciso apren...	None	2	True
5	DSADSA	None	Alta	False	
6	NOVA TAREFA	teste	None	Baixa	True

design request response session db tables db stats

Copyright © 2016 Desenvolvido com web2py Share

Figura 17: Consulta Tabela Tarefas

Perdemos também a capacidade de gerenciamento e detalhamento que o *grid* nos proporciona mas podemos recuperar estas funções.

Vamos fazer uma outra função no controle para detalhar cada tarefa, para isso vamos usar o id da tarefa para fazer um select único, além de GET e request para ler as informações e dar uma retorno mais detalhado.

Abaixo da função criada anteriormente escreva esta:

```
1. def detalhar_tarefa():
2.     tarefa = db(Tarefa.id
request.vars.id).select()
3.     return dict(tarefa=tarefa)
```

Essa função fará o mesmo que a função anterior mas com uma tarefa apenas, veja na segunda linha do código acima o que acontece é o seguinte:

tarefa é uma variável, esta variável recebe o resultado de um select no banco, a diferença é que o resultado do select está limitado as linhas onde o id é igual ao id que está vindo pelo GET ou pelo POST, através do comando request, todas as informações enviadas são lidas pelo request.

Para ver o resultado tu já sabes, basta acessar o controle default e a função detalhar\_tarefa, mas é preciso mandar qual id de qual tarefa será exibido, se essa informação for omitida ou o id for nulo ou inexistente nenhum valor será exibido.

[http://127.0.0.1:8000/tarefa/default/detalhar\\_tarefa?id=1](http://127.0.0.1:8000/tarefa/default/detalhar_tarefa?id=1)

O resultado deve estar assim:

127.0.0.1:8000/tarefa/default/detalhar\_tarefa?id=6

Pesquisar

web2py™ Principal Meus sites Esta aplicação web2py.com Documentação Comunidade Log In

## Detalhar Tarefa

tarefa.idtarefa	tarefa.nome	tarefa.descricao	tarefa.prazotarefa	tarefa.prioridadetarefa	tarefa.concluida
6	NOVA TAREFA	teste	None	Baixa	True

design request response session db tables db stats

Copyright © 2016 Desenvolvido com web2py Share

Figura 18: Detalhar Tarefa

Caso o id seja nulo ou inválido o resultado será como este:

[http://127.0.0.1:8000/tarefa/default/detalhar\\_tarefa](http://127.0.0.1:8000/tarefa/default/detalhar_tarefa)

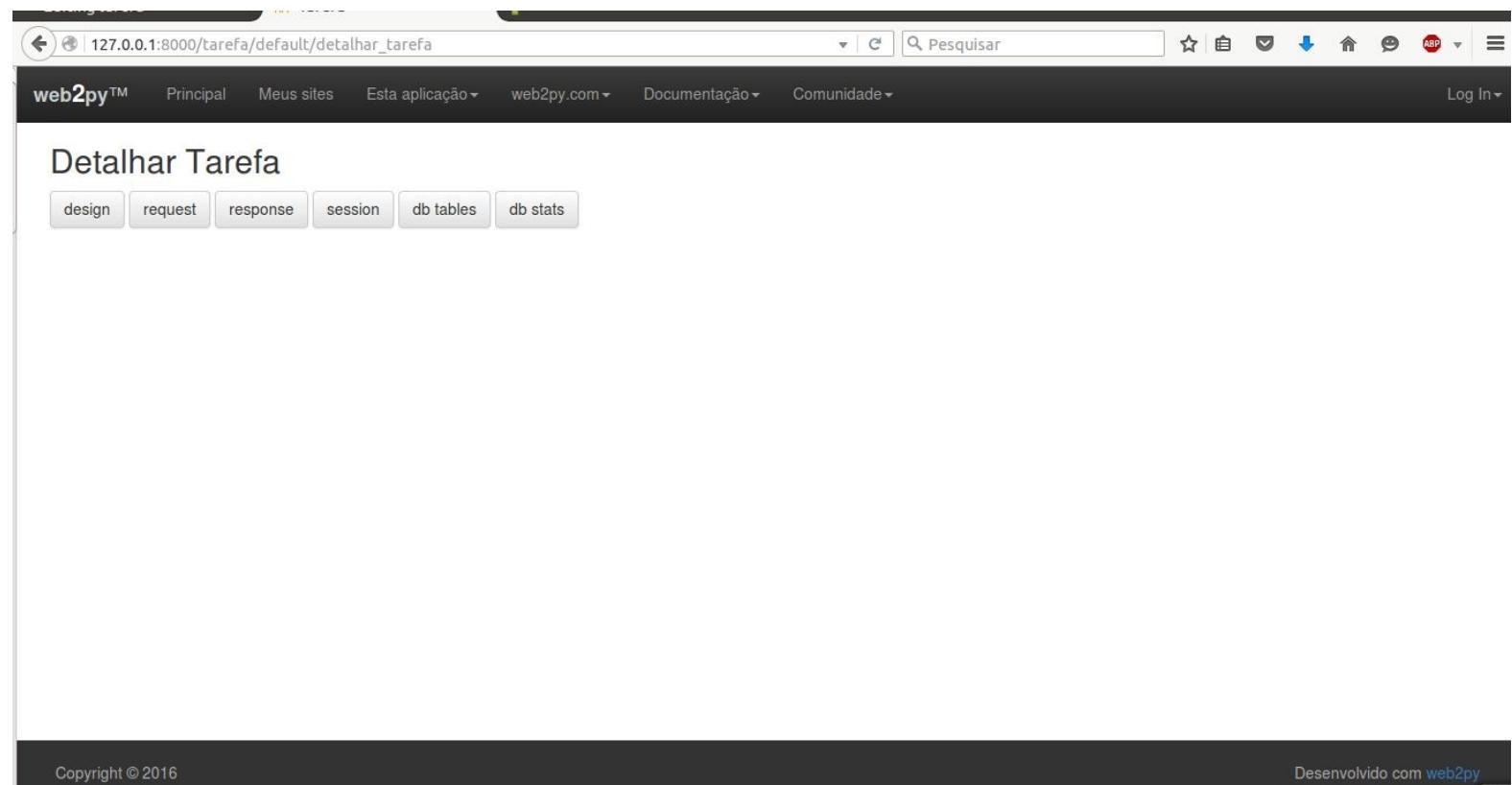


Figura 19: Detalhar Tarefa Nulo

O web2py gera tickets de erro somente se houverem erros no código, neste caso não há nenhum erro, o que mandamos o sistema

fazer, ele fez, mostrar todas as tarefas com o id informado, como não foi informado nenhum id não há nenhum retorno.

Ainda precisamos criar a possibilidade de inserir novas tarefas para isso vamos criar a seguinte função dentro do controle default.

Então na última linha escreva o seguinte código:

```
1.     def nova_tarefa():
2.         form = SQLFORM(Tarefa)
3.         return dict(form=form)
```

Vamos testar, basta acessar nossa função da forma tradicional

[http://127.0.0.1:8000/tarefa/default/nova\\_tarefa](http://127.0.0.1:8000/tarefa/default/nova_tarefa)

O resultado deve ser algo assim:

Nova Tarefa

Nome

Descrição

Prazo

Prioridade

concluída

Submit

design request response session db tables db stats Share

Figura 20: Nova Tarefa

Ainda tem um pequeno ajuste, quando o formulário é enviado seria interessante mandar uma mensagem de ok ou de erro para o usuário saber o que aconteceu então altere o código acima para ficar assim, cuidado com a endentação:

```
1. def nova_tarefa():
2.     form = SQLFORM(Tarefa)
```

```
3.     if form.process().accepted:  
4.         session.flash = 'Tarefa cadastrada!'  
5.     elif form.errors:  
6.         response.flash = 'Erros!'  
7.     else:  
8.         response.flash = 'Preencha!'  
9.     return dict(form=form)
```

Fazendo isso já temos uns recadinhos para quando o usuário trabalhar nosso formulário.

O método *process* permite verificar se o formulário foi validado ou não, utilizando *form.process().accepted* teremos um retorno True caso tenha dado tudo certo, False se não.

Já os erros ficam armazenadas em *form.errors* que retorna True caso tenha algum valor, esses erros vem dos validadores que nos criamos nos modelos, então quando algum erro surgir vamos exibir uma mensagem

Já que fizemos exibir mensagens quando tudo deu certo, quando algo deu errado nossa ultima mensagem refere-se a quando nada aconteceu ainda.



# Funções CRUD

CRUD é um acrônimo para Create, Read, Update e Delete, são as quatro funções básicas de interação com o banco de dados, no web2py temos uma família de funções para essa interação que facilitaria ainda mais o que já temos, incluindo as mensagens de erro , acerto e etc.

Veja esta pequena função abaixo:

```
1. from gluon.tools import Crud  
2. crud = Crud(db)  
3.  
4. def manage():  
5.     table=Tarefa  
6.     form = crud.update(table, request.args(0))  
7.     table.id.represent = lambda id,  
row:A('editar:',id,_href=URL(args=(id)))  
8.     search, rows = crud.search(table)  
9.     return dict(form=form, search=search, rows=rows)
```

Esta função tem um pequeno resumo de todas as funções crud, criando um pequeno painel de gerenciamento semelhante ao SQLFORM.grid() mas com menos estilo dando liberdade para fazer o design que quiseres.

Na primeira linha temos o comando que faz a importação da família de comandos CRUD, sem ele os demais comandos não vão funcionar.

Ainda temos que vincular os comandos crud ao banco de dados isso é feito na segunda linha, vinculando ao comando db assim temos acesso a todas as tabelas, essas linhas são uma preparação para usarmos os comandos crud.

Na linha 4 começamos a função propriamente dita.

Na linha 5 criei uma variável para armazenar a tabela que vamos manipular, como só temos uma tabela isso é meio tranquilo, caso tenhamos mais de uma tabela podemos usar get ou request para capturar o nome da tabela que vamos manipular.

Na sexta linha criamos uma variável form que recebe um dos comandos crud, o crud.update que é responsável por fazer atualizações na tabela, esse comando recebe dois argumentos o nome da tabela que será atualizada e o id do registro que será modificado estamos passando esses parâmetros pela variável que tem o nome da tabela e o id pelo request.

Lembra da ordem que os elementos são mandados pela URL certo? Não? Veja:

<http://servidor/aplicação/controle/função/argumentos?variáveis>

Os argumentos podem ser lidos pelo request sendo que o primeiro argumento é o número 0, e os demais separador por “/” barra seguem a ordem de números inteiros crescentes.

Então para editar o primeiro registro da tabela tarefas temos que acessar o seguinte URL:

<http://127.0.0.1:8000/tarefa/default/manage/1>

Sendo interpretada desta forma:

- Servidor → 127.0.0.1:8000
- aplicação → tarefa
- controle → default
- função do controle → manage
- Argumento → 1
- formato da resposta para o cliente → html

Outra consideração importante, caso não seja enviado nenhum argumento para o id do crud.update() ele funcionará mas não da mesma forma, em vez de fazer um update ele fará um insert, adicionando um novo registro na tabela, então é uma maneira meio preguiçosa de fazer as duas coisas ao mesmo tempo.

Na linha 7 temos um trecho de programação funcional utilizando um construtor para gerar links que após feita uma consulta nos permitirá selecionar qual tarefa desejamos editar.

Na linha 8 usamos o comando crud.search() para criar um 'formulário' de pesquisa no banco de dados, novamente informando qual tabela será pesquisada, retornando linhas de links.

Finalmente retornamos as variáveis para a visão.





# Formulários sem banco de dados

Existem algumas situações onde não precisamos guardar as informações no banco de dados, então podemos gerar formulários que são processados e depois as informações são desprezadas pelo web2py e talvez aproveitadas de outra forma.

Vou utilizar o exemplo que vi no curso da Julia Rizza<sup>5</sup> levemente adaptado, é um formulário de contato, então vamos acrescentar as seguintes linhas no nosso controle para criar um formulário sem tabela.

```
1. def contato():
2.     form = SQLFORM.factory(
3.         Field('nome', requires=IS_NOT_EMPTY()),
4.         Field('email', requires=IS_EMAIL(),
5.               label='E-mail'),
6.         Field('mensagem', 'text',
7.               requires=IS_NOT_EMPTY()))
8.     if form.process().accepted:
9.         mail.send(
```

to=['meu@email.com'],

```
10.     subject='Contato pelo site por %s' %  
form.vars.nome,  
  
11.    reply_to = form.vars.email,  
  
12.    message=form.vars.mensagem  
  
13. )  
  
14. elif form.errors:  
  
15.     response.flash = 'Erros no formulário!'  
  
16. return dict(form=form)
```

Logo na linha 2 temos o código que fará toda a mágica, neste exemplo SQLFORM.factory(). O resto é muito semelhante a maneira como criamos tabelas no modelo utilizando Field para criar campos e requires() para valida-los.

Abaixo ainda temos o código necessário para enviar os e-mails, claro que ainda precisa de ajustes para funcionar, mas já podemos testar o formulário.

Teremos algo como visto abaixo ao acessar o controle:

<http://127.0.0.1:8000/tarefa/default/contato>

127.0.0.1:8000/tarefa/default/contato

Pesquisar

web2py™ Principal Meus sites Esta aplicação web2py.com Documentação Comunidade Log In

## Contato

Nome

E-mail

Mensagem

design request response session db tables db stats

Copyright © 2016 Desenvolvido com web2py

The screenshot shows a web browser displaying a web2py application at the URL 127.0.0.1:8000/tarefa/default/contato. The page title is "Contato". It features a form with three input fields: "Nome" (Name), "E-mail" (Email), and "Mensagem" (Message). Below the message field is a "Submit" button. At the bottom of the page, there is a footer with links for "design", "request", "response", "session", "db tables", and "db stats". The overall layout is clean and functional, typical of a web-based application.

Figura 21: Contato

Este é apenas um exemplo, muitas outras aplicações são possíveis com este código SQLFORM.factory().



# Limitando Acesso a Usuários logados

Nosso aplicativo, apesar de ser um gerenciador de coisas a fazer pode ser acessado e modificado por qualquer um, obviamente existem algumas situações em que precisamos limitar este acesso, por exemplo ver as tarefas pode até ser feito por qualquer um mas modificá-las deveria ser uma tarefa só de um administrador ou o mínimo de um usuário logado, para fazer estas limitações basta usar algumas decorações, são códigos que inserimos antes da função no controle, veja:

Para impedir um usuários não logado de acessar uma determinada função usaremos antes da função em questão o seguinte código

```
@auth.requires_login()
```

Veja um exemplo:

1.

```
@auth.requires_login()
```

2.     def nova\_tarefa():

3.         form = SQLFORM(Tarefa)

4.             return dict(form=form)

Assim somente usuários cadastrados e autenticados podem acessar esta função do controle.





# Usuários e Grupos

Nos também podemos limitar o acesso as funções do controle de usuários cadastrados e logados mas que não pertencem a um grupo específico, para isso também usamos uma decoração, neste caso:

```
@auth.requires_membership('admin')
```

No exemplo acima somente usuários cadastrados, logados e que pertençam ao grupo 'admin' podem acessar a função, segue um exemplo de código

1.

```
@auth.requires_membership('admin')
```

2.     def nova\_tarefa():

3.         form = SQLFORM(Tarefa)

4.             return dict(form=form)

Mas ainda precisamos ver como gerenciar usuários e grupos e isso pode ser feito com códigos ou com o painel de administração na parte da gestão de banco de dados, então vamos acessá-lo:

The screenshot shows the Mozilla Firefox browser window with two tabs open: 'Teste' and 'Teste'. The main content area displays the 'appadmin' interface for a web2py application. The title bar reads 'Bancos de dados e tabelas disponíveis'. Below this, there are two tabs: 'Tables' (selected) and 'Hooks'. A list of tables is shown, each with a 'Novo Registro' button:

- db.auth\_user
- db.auth\_group
- db.auth\_membership
- db.auth\_permission
- db.auth\_event
- db.auth\_cas
- db.tarefa

At the bottom of the page, there are copyright and development information:

Copyright © 2016 Desenvolvido com web2py Share

Figura 22: Administração de Banco de Dados

Na tabela auth\_user temos a lista de todos os usuários cadastrados, na tabela auth\_group temos todos os grupos cadastrados e finalmente na

tabela auth\_membership são feitas as relações entre os usuários e os grupos.

Vamos criar um novo grupo clicando em novo registro na tabela auth\_group

The screenshot shows a web2py application interface for creating a new group. The top navigation bar includes links for 'web2py™', 'projeto', 'bd', 'estado', 'cache', and 'Log In ▾'. The main title is 'banco de dados db Tabela auth\_group' with a subtitle 'Novo Registro'. The form has two fields: 'Papel' (with value 'admin') and 'Descrição' (with value 'Grupo de acesso maior que o normal'). A 'Submit' button is at the bottom left of the form area. The footer contains copyright information ('Copyright © 2016') and developer credits ('Desenvolvido com web2py Share').

Papel	admin
Descrição	Grupo de acesso maior que o normal

Submit

Copyright © 2016 Desenvolvido com web2py Share

Figura 23: Novo Registro Grupo

Criei o grupo admin para ficar igual ao exemplo mostrado acima, agora vamos vincular o usuário ao grupo, eu já tenho o usuário criado e vou fazer a vinculação, caso tu não tenhas podes criá-lo através de um novo registro na tabela auth\_user ou até mesmo pela própria aplicação, no canto superior direito temos um link para login e registro de novos usuários.

Para vincular os usuários aos grupos vamos usar a tabela auth\_membership, então clicando em novo registro da tabela auth\_membership temos a seguinte tela:

127.0.0.1:8000/carrera/appadmin/insert/do/auth\_membership

web2py™ projeto bd estado cache Log In ▾

banco de dados db Tabela auth\_membership

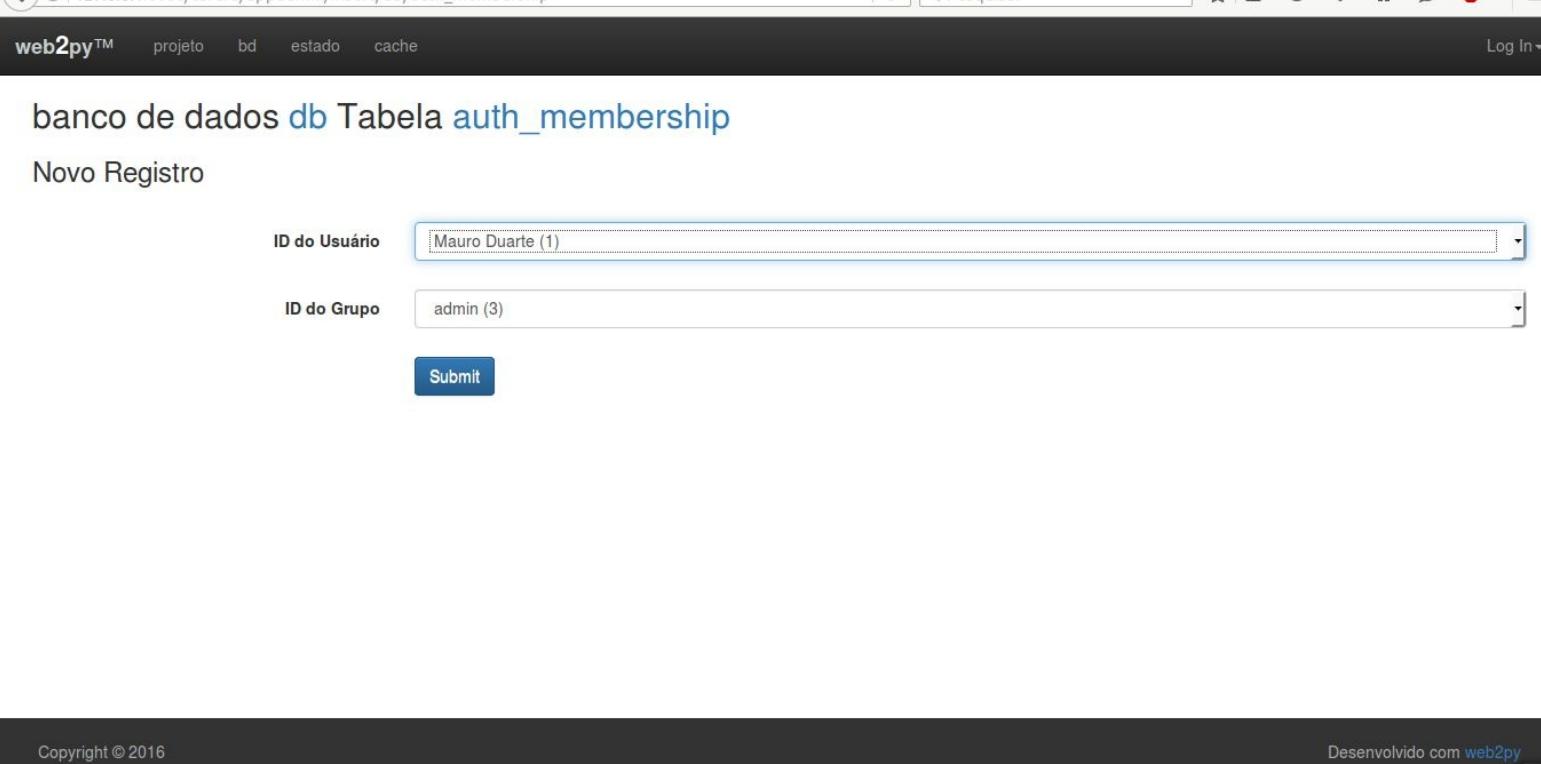
Novo Registro

ID do Usuário: Mauro Duarte (1)

ID do Grupo: admin (3)

Submit

Copyright © 2016 Desenvolvido com web2py Share



*Figura 24: Novo Registro Membership*

Nesta tela podemos selecionar um usuário e inseri-lo em um grupo usando os seletores, como visto acima.

Agora nosso usuário já pode acessar a função limitada aos pertencentes ao grupo 'admin' isso pode ser feito quantas vezes forem necessárias criando grupos e vinculando os usuários a eles.



# Cada um na sua tarefa

Ainda temos uma última coisinha que pode ser interessante usar, até agora temos uma lista única de tarefas, todos podem editá-la, mas em sites com múltiplos usuários pode ser interessantes limitar cada usuário as suas próprias tarefas.

Para fazer isso precisamos que na tabela tarefas fique gravado quem criou a tarefa em questão.

Vamos modificar nosso modelo para cumprir também esta função.

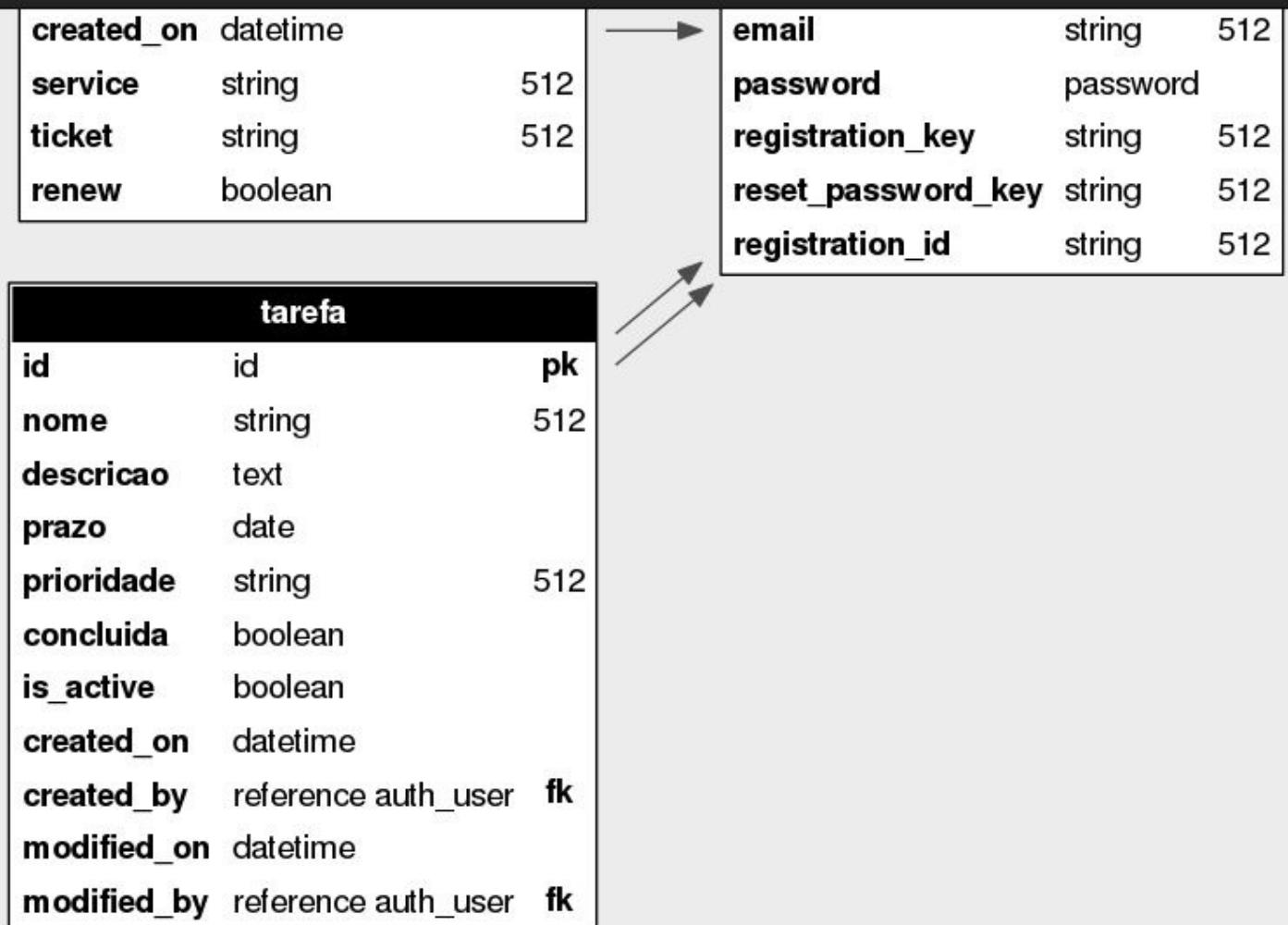
Abra o arquivo do modelo db.py e modifique a tabela tarefas para que ela fique da seguinte forma:

```
1. Tarefa = db.define_table('tarefa',  
2.     Field('nome', 'string'),  
3.     Field('descricao', 'text'),  
4.     Field('prazo', 'date'),  
5.     Field('prioridade', 'string'),  
6.     Field('concluida', 'boolean'),  
7.     auth.signature  
8. )
```

Bem simples, agora temos um vínculo com a tabela auth\_user e a tabela 'tarefa' ganhará alguns campos extras como:

- created\_on → datetime → Armazena quando a tarefa foi criada;
- created\_by → reference → Cria uma chave estrangeira para a tabela auth\_user armazenando quem criou a tarefa;
- modified\_on → datetime → Armazena quando a tarefa foi modificada pela última vez;
- modified\_by → reference → Cria uma chave estrangeira para a tabela auth\_user armazenando quem modificou a tarefa pela última vez;

Podemos ver no modelo gráfico como a tabela 'tarefas' está agora.



Copyright © 2016

Figura 25: Modelo Gráfico Tarefa Modificado

Finalmente precisamos modificar nosso controle para mostrar apenas as tarefas do usuário que está logado, criamos então este controle.

1.

```
@auth.requires_login()  
2. def tarefas_do_usuario():  
3.     user_id = auth.user.id  
4.     tar      = db(Tarefa.created_by ==  
user_id).select()  
5.     return dict(tar=tar)
```

Usamos a decoração `@auth.requires_login()` porque é óbvio que é preciso que o usuário esteja logado, usamos `user_id = auth.user.id` para capturar o id do usuário e guardas a informação em uma variável, com esta informação passamos para um select que retornará uma lista de tarefas do usuário logado para cada usuário individualmente.

Veja:

web2py™ Principal Meus sites Esta aplicação▼ web2py.com▼ Documentação▼ Comunidade▼ Bem-vindo Mauro ▾

## Tarefas Do Usuario

	tarefa.idtarefa.nome	tarefa.descricaotarefa.prazotarefa.prioridadetarefa.concluidotarefa.is_activetarefa.created_on	tarefa.created_bytarefa.modified_on	tarefa.modified_by						
9	Cadastrar tar...	Já estou fazendo	None	Alta	False	True	2016-02-26 11:26:11	Mauro Duarte	2016-02-26 11:26:11	Mauro Duarte
10	mais uma tarefa		None	Baixa	False	True	2016-02-26 11:27:26	Mauro Duarte	2016-02-26 11:27:26	Mauro Duarte

design request response session db tables db stats

Copyright © 2016 Desenvolvido com web2py Share

Figura 26: Tarefas do Usuário



# Limpando o lixo

Acho que já temos uma boa noção dos modelos, mas neste ciclo de aprendizagem fizemos e desfizemos muitas coisas deixando uma grande sujeira, esta sujeira vai nos atrapalhar quando formos fazer as visões, então vamos corrigir nosso modelo para que fique desta forma, excluindo todas as funções que criamos ate agora.

1. ##### estes códigos vão no final abaixo da função call() e devem substituir os que foram feitos até agora.
2. ## Cadastrar uma nova tarefa
3. @auth.requires\_login()
4. def nova():
5. form = SQLFORM(Tarefa)
6. if form.process().accepted:
7. session.flash = 'Formulário aceito!'
8. elif form.errors:
9. response.flash = 'Erros no formulário!'
10. else:
11. response.flash = 'Preencha o formulário!'

```
12.     return dict(form=form)
13.
14.     ## Listar as tarefas do usuário cadastrado
15.     @auth.requires_login()
16.     def tarefas():
17.         user_id = auth.user.id
18.         tar      = db(Tarefa.created_by == user_id).select()
19.         return dict(tar=tar)
20.
21.     ## Ver os detalhes de uma tarefa
22.     @auth.requires_login()
23.     def detalhar():
24.         tarefa      = db(Tarefa.id == request.args(0)).select()
25.         return dict(tarefa=tarefa)
26.
27.     ## Editar os detalhes de uma tarefa
28.     @auth.requires_login()
```

```
29.     def editar():
30.         form = SQLFORM(Tarefa, request.args(0))
31.         if form.process().accepted:
32.             session.flash = 'Tarefa atualizada: %s' % form.vars.nome
33.             redirect(URL('tarefas'))
34.         elif form.errors:
35.             response.flash = 'Erros no formulário!'
36.         else:
37.             if not response.flash:
38.                 response.flash = 'Preencha o formulário!'
39.         return dict(form=form)
40.
41.     ## Formulário de contato
42.     def contato():
43.         form = SQLFORM.factory(
44.             Field('nome', requires=IS_NOT_EMPTY()),
45.             Field('email', requires=IS_EMAIL(),
label='E-mail'))
```

```

46.             Field('mensagem',
        'text',
        requires=IS_NOT_EMPTY())
47.     )
48.     if form.process().accepted:
49.         mail.send(
50.             to=['meu@email.com.br'],
51.             subject='Contato pelo site por %s' %
        form.vars.nome,
52.             reply_to = form.vars.email,
53.             message=form.vars.mensagem
54.     )
55.     elif form.errors:
56.         response.flash = 'Erros no formulário!'
57.     return dict(form=form)

```

Agora nosso controle tem as seguintes funções:

- nova → para cadastrar uma nova tarefa;
- tarefas → para listar todas as tarefas do usuário logado;

- detalhar → para ver mais especificamente as informações de uma tarefa específica;
- editar → para modificar uma tarefa;
- contato → para receber e-mails da audiência não logada

Além destas, e das demais funções padrão que deixamos uma muito importante que usaremos depois é a função index que define nossa página inicial, mas ela ficará como está até vermos o assunto referente as visões.



# Imagens para as tarefas

Um ultimo detalhe, vamos fazer só mais uma modificação para que nossas tarefas fiquem ainda mais interessantes, vamos ilustrá-las com imagens, para isso vamos atualizar nosso modelo para receber as imagens que ilustrarão nossa tarefa.

Modifique o modelo para que fique assim:

```
1. Tarefa = db.define_table('tarefa',  
2.     Field('nome', 'string'),  
3.     Field('descricao', 'text',  
label='Descrição'),  
4.     Field('prazo', 'date'),  
5.     Field('prioridade', 'string'),  
6.     Field('concluida', 'boolean',  
label='concluída'),  
7.     Field('imagem', 'upload'),  
8.     auth.signature  
9. )
```

E acrescente também a seguinte validação no model:

```
1.     Tarefa.imagem.requires=IS_IMAGE()
```

Tudo pronto, vamos as visões.





# View

As visões, como o próprio nome diz, é o que vemos na tela, no caso do web2py que é um framework para web, então o que vemos é visto no navegador, e os navegadores sempre leem códigos HTML, desta forma temos duas maneiras de fazer as visões:

1. Usar o python do web2py para gerar códigos HTML;
2. Criar códigos HTML e mesclar com informações vindas do python;

Como minha carreira no desenvolvimento começou no HTML eu acho na maioria das vezes mais fácil mesclar, ou seja, crio códigos HTML e mesclo com informações vindas do python. O bom é que aprendas as duas maneiras e faça a que for melhor para ti.

# Visões genéricas

Da mesma forma que os modelos e os controles, já temos muitas visões criadas e prontas para usar

- \_\_init\_\_.py
- appadmin.html
- default/
  - index.html
  - user.html
- generic.html
- generic.ics
- generic.json
- generic.jsonp
- generic.load
- generic.map
- generic.pdf
- generic.rss
- generic.xml
- layout.html
- web2py\_ajax.html

Algumas são muito importantes, cada uma das que tem o nome de generic é utilizada quando nenhuma visão específica da função do controle é chamada, a visão generic que será chamada depende da extensão.

Por exemplo em `http://servidor/aplicação/controle/função.xml` vai chamar a visão generic.xml no caso de não haver nenhuma visão específica com a extenção .xml, já `http://servidor/aplicação/controle/função.pdf` vai chamar a visão generic.pdf e caso não seja usado nenhuma extensão então será chamado generic.html.

Lembrando que apenas se não houver nenhuma visão específica.

Outra visão importante é layout.html que basicamente define todo o estilo do nosso aplicativo, o design original é feito com base no bootstrap, que é um framework frontend para desenvolvimento responsivo mobile first, mas pode ser editado na própria visão layout, nas visões específicas para cada função do controle ou nos arquivos estáticos que veremos mais a frente.





# Visões específicas

Uma regra importante para criar visões específicas, é que é preciso criar uma pasta com o nome exato do controle onde a função está escrita, como podes ver nos temos já uma pasta chamada default e dentro dela já temos duas visões index.html e user.html.

Vamos criar novas visões para as funções que criamos anteriormente, todas dentro do diretório default já que nossas funções estão no diretório default.

- ***default/contato.html*** → para receber e-mails da audiência não logada
- ***default/detalhar.html*** → para ver mais especificamente as informações de uma tarefa específica;
- ***default/editar.html*** → para modificar uma tarefa;
- ***default/nova.html*** → para cadastrar uma nova tarefa;
- ***default/tarefas.html*** → para listar todas as tarefas do usuário logado;

Depois vamos editar a visão ***default/index.html*** que é a página inicial do nosso aplicativo.

## **default/contato.html**

Vamos criar a visão contato.html, tenha certeza de que a visão foi criada dentro do diretório default se não o mapeamento não vai chamar a visão corretamente.

Tu podes criar a nova visão clicando no botão criar logo abaixo da lista de visões, para criar dentro de default escreva 'default/contato.html'.

Agora que já está criado podemos abri-lo em modo de edição, já temos algum código que foi criado automaticamente veja:

1. { {extend 'layout.html' } }
2. <h1>Este é o template  
default/contato.html</h1>
3. { {=BEAUTIFY (response.\_vars) } }

Na linha 1 temos o código que chama o layout.html criando a maior parte da página, cabeçalho, rodapé, barras laterais e tudo mais.

Na linha 2 temos um código HTML puro que podemos editar para que fique da seguinte forma:

1. { {extend 'layout.html' } }
2. <h1>Entre em Contato conosco</h1>
3. { {=BEAUTIFY (response.\_vars) } }

Na terceira linha temos um comando genérico para exibir, da melhor maneira, e mais simples possível os returns da função do controle.

Aqui já podemos observar um padrão da visões do web2py os códigos HTML são escritos normalmente, já os códigos python são escritos entre chaves duplas {{ }}.

Quando trabalhamos em visões, não precisamos respeitar a edentação sendo que quando não é obvio fecharemos os laços de repetição e as estruturas condicionais com o comando {{pass}}.

Vamos fazer mais alguns ajustes no nosso código, basicamente o que vamos fazer é organizar o formulário em uma coluna e deixar espaço para inserirmos novas informações estáticas como endereço e telefone. Veja como fica:

```
1.     {{extend 'layout.html'}}
```

```
2.     <h1>Entre em contato conosco</h1>
```

```
3.     <div class="col-md-6">
```

```
4.     {{=BEAUTIFY(response._vars.values()[0])}}
```

```
5.     </div>
```

```
6.     <div class="col-md-5">
```

```
7.         <div class="row">
```

```
8.             <h4>Rua da Bolhas Estouradas, 1000</h4>
```

```
9.      <h4>Porto Alegre, RS.</h4>
10.     <h4>
11.     <span class="glyphicon glyphicon-
earphone"></span> (51) 8866.5544
12.     </h4>
13.   </div>
14.   <div class="row" style="padding-
top:20px;">
15.     <div id="googleMap"
style="height:230px;width:100%"></div>
16.   <!-- Adicionando Google Maps com
JavaScript -->
17.   <script
src="http://maps.googleapis.com/maps/api/js">
</script>
18.   <script>
19.     var myCenter = new
google.maps.LatLng(-30.0279622, -51.2298482);
20.
21.   function initialize() {
22.     var mapProp = {
23.       center:myCenter,
```

```
24.     zoom:12,  
25.     scrollwheel:true,  
26.     draggable:true,  
27.     mapTypeId:google.maps.MapTypeId.ROADMAP  
28.   } ;  
29.  
30.   var map = new  
31.       google.maps.Map(document.getElementById("goog:  
32.   var marker = new google.maps.Marker({  
33.     position:myCenter,  
34.   }) ;  
35.   marker.setMap(map) ;  
36. }  
37. google.maps.event.addListener(window,  
38.   'load', initialize);  
39. </script>  
40. </div>
```

Podes ver que a parte python do web2py ficou praticamente a mesma, a única modificação foi no comando `{=BEAUTIFY(response._vars.values()[0])}` onde limitamos a apenas o primeiro valor assim não é exibida o nome do indice do dicionário que veio do return.

Os demais códigos são HTML, JavaScript e classes do Bootstrap, e um pouquinho de estilo embutido em algumas tags.

O resultado final deve ser este aqui:

## Entre em contato conosco

Nome

Rua da Bolhas Estouradas, 1000

Porto Alegre, RS.

E-mail

📞 (51) 8866.5544

Mensagem



Submit

Figura 27: Visão contato

Ainda temos a herança do layout que vamos modificar depois.



## default/detalhar.html

Ao criar a visão detalhar.html temos o mesmo código original da visão contato que criamos antes, da mesma forma vamos modificar estes código só que desta vez o return da função no controle não é um formulário e sim uma consulta SQL, então vamos usar um laço de repetição para percorrer a matriz criada pela consulta.

Veja o return da função é uma variável chamada 'tarefa' com todo o conteúdo da consulta, o python do web2py consegue entender os cabeçalhos das colunas no banco como funções de um objeto (POO), na prática é bem simples.

Veja o código completo da nossa visão detalhar.html:

```
1.      {{extend 'layout.html'}}
```

```
2.      {{for i in tarefa:}}
```

```
3.          <h1>Tarefa</h1>
```

```
4.          <p>
```

```
5.              <a href="{{=URL('tarefas')}}" class="btn btn-default">Voltar</a>
```

```
6.              <a href="{{=URL('editar', args=i.id)}}" class="btn btn-default">Editar Tarefa</a>
```

```
7.          </p>
```

```
8.          <h2>
```

```
9.     Nome: { {=i.nome} }
```

```
10.    </h2>
```

```
11.    
```

```
12.    <h4>
```

```
13.    Descrição:
```

```
14.    </h4>
```

```
15.    <p>{ {=i.descricao} }</p>
```

```
16.    <p>
```

```
17.    { {if i.concluida:} }
```

```
18.    <span class="label label-success">Tarefa  
concluída</span>
```

```
19.    { {else:} }
```

```
20.    <span class="label label-danger">Tarefa  
pendente</span>
```

```
21.    { {pass} }
```

```
22.    - Prazo Final <strong>  
{ {=i.prazo.strftime("%d/%m/%Y") } }</strong>
```

```
23.    </p>
```

```
24.      <p>
25.
26.      { {for y in db(db.auth_user.id ==
27.          i.created_by).select() : } }
27.      Autor: { {=y.first_name} } { {=y.last_name} }
28.      { {pass} }
29.      </p>
30.
31.      { {pass} }
```

Apenas algumas linhas merecem uma explicação especial:

Na linha 2 começamos nosso laço de repetição que percorre o select ele vai do inicio ao fim da nossa visão, este laço inicia a variável 'i' que usaremos para chamar cada informação da nossa consulta, sempre seguido do nome do campo no modelo da tabela como na linha 6 onde usamos dentro da função helper URL para vincular nossa tarefa detalhada a tarefa editada usando i.id.

Na linha 22 temos uma pequena função para ajudar na exibição da nossa data formatada.

Finalmente na linha 26 iniciamos um novo laço de repetição for que inicia a variável 'y' para percorrer uma nova consulta no banco de dados, esta consulta usará o campo created\_by da tabela 'tarefas' para buscar da tabela 'auth\_user' o nome do usuário que criou a tarefa detalhada.

Veja como ficou a nossa visão depois de pronta:

The screenshot shows a web browser displaying a web2py application. The header bar includes links for 'web2py™', 'Principal', 'Meus sites', 'Esta aplicação', 'web2py.com', 'Documentação', 'Comunidade', and 'Bem-vindo Mauro'. The main content area has a title 'Tarefa' and two buttons: 'Voltar' and 'Editar Tarefa'. Below this, the task details are shown: 'Nome: Estudar Python' with a Python logo icon; 'Descrição: Preciso estudar python cad vez mais e nunca parar para evoluir cada vez mais.'; a status message 'Tarefa pendente - Prazo Final 12/12/2016'; and the author 'Autor: Mauro Duarte'. At the bottom, there is a footer with 'Copyright © 2016' and 'Desenvolvido com web2py'.

Figura 28: Visão Detalhar

Talvez aches que precise de mais ajustes, isso é contigo, por enquanto vamos deixar assim mesmo.



## default/editar.html

Esta visão não precisa de muitos ajustes, o layout já prove a maior parte do que precisamos, vamos apenas mudar o título e filtrar o retorno para exibir apenas o formulário. Nosso código ficará desta forma:

```
1.    { {extend 'layout.html'} }  
2.    <h1>Editar tarefa</h1>  
3.    <div class="col-md-6">  
4.        <a href="{ {=URL('tarefas') } }" class="btn  
        btn-default">Voltar</a>  
5.    { {=BEAUTIFY(response._vars.values()[0])} }  
6.    </div>
```

E a visão aparecerá assim:

## Editar tarefa

Preencha o formulário!

**Id** 11

**Nome** tarefa com imagem

**Descrição** teste da tarefa com upload de imagem

**Prazo** 12/12/2016

**Prioridade** Baixa

concluída

**Imagen**  Nenhum arquivo selecionado.

**Submit**



Figura 29: Visão Editar



## default/nova.html

Semelhante a anterior está visão também receberá os mesmos ajustes.

```
1.    {{extend 'layout.html'}}
```

```
2.    <h1>Nova tarefa</h1>
```

```
3.    <div class="col-md-6">
```

```
4.        <a href="{{=URL('tarefas')}}" class="btn btn-default">Voltar</a>
```

```
5.    {{=BEAUTIFY(response._vars.values()[0])}}
```

```
6.    </div>
```

Ficando desta forma:

## Nova tarefa

Preencha o formulário! x

**Nome****Descrição****Prazo****Prioridade** Escolha ▾ concluída**Imagem**

Nenhum arquivo selecionado.

Figura 30: Visão Nova



## default/tarefas.html

Diferente das visões anteriores que usam formulários e por isso são mais simples está será uma das mais complexas por exibir muitos dados os quais queremos que tenha uma apresentação muito boa.

Para isso vamos exibir apenas parte dos dados dentro de div formatadas com CSS do Bootstrap nosso código de estar assim:

```
1. {{extend 'layout.html'}}
```

```
2. <h1>Lista das tarefas</h1>
```

```
3. <a href="{{=URL('nova')}}" class="btn btn-default">Nova Tarefa</a>
```

```
4. <div class="row">
```

```
5.
```

```
6. {{for i in tar:}}
```

```
7.
```

```
8. <div class="col-sm-2">
```

```
9. <a href="{{=URL('detalhar', args=i.id)}}>
```

```
10. <h4>
```

```
11. {{=i.nome}}
```

```
12. </h4>
```

```
13.      <div style="width:150px;height:150px;">
14.          
17.      </div>
18.      <p>
19.          <strong>Prioridade:</strong>
20.          {{if i.prioridade == 'Baixa':}}
21.          <span class="label label-success">Baixa</span>
22.          {{elif i.prioridade == 'Média':}}
23.          <span class="label label-info">Média</span>
24.          {{else:}}
25.          <span class="label label-danger">Alta</span>
26.
27.          {{pass}}
28.      </p><p>
```

```
29.    {{if i.prazo != None:}}
```

```
30.      Prazo Final <strong>
{ {=i.prazo.strftime("%d/%m/%Y") } }</strong>
```

```
31.    {{else:}}
```

```
32.      <p>
```

```
33.      &nbsp;
```

```
34.      </p>
```

```
35.    {{pass}}
```

```
36.      </p>
```

```
37.      <p>
```

```
38.      <a href="{{=URL('detalhar', args=i.id)}}>Ver mais</a>
```

```
39.
```

```
40.      </p>
```

```
41.
```

```
42.      </div>
```

```
43.    {{pass}}
```

```
44.      </div>
```

Novamente apenas algumas linhas precisam de comentários:

Logo na linha 3 iniciamos um link que vai para o detalhamento de cada tarefa ao clicar no título ou na imagem, que se repete no final junto a expressão “ver mais”.

Os demais códigos são iguais aos que já tínhamos visto, só resta explicar que na linha 29 fizemos um teste lógico para ver se o prazo possui algum valor, se possuir será exibido formatado, se não será criado um paragrafo com um espaço para manter a formatação.

Depois nossa visão deve estar como esta:

## Lista das tarefas

[Nova Tarefa](#)[Cadastrar tarefa](#)[mais uma tarefa](#)[tarefa com imagem](#)**Prioridade:** Alta**Prioridade:** Baixa[Estudar Python](#)[Terminar o livro de web2py](#)[Ver mais](#)[Ver mais](#)**Prioridade:** Baixa

Prazo Final 12/12/2016

[Ver mais](#)**Prioridade:** Baixa

Prazo Final 12/12/2016

[Ver mais](#)**Prioridade:** Média

Prazo Final 01/03/2016

[Ver mais](#)

Figura 31: Visão Tarefas



## default/index.html

Index é a página inicial do nosso aplicativo, é a função que é chamada por padrão caso seja omitida na URL qual função e de qual controle, que é o que sempre acontece quando entramos pela primeira vez no servidor.

Como ela é só ilustrativa vou fazer uma edição suave, aqui tudo praticamente depende só de HTML e CSS, vamos só entender o que já temos.

Não será preciso criá-la ela já existe então abra a visão index.html em modo de edição, lembre que ela é uma 'cópia' da index do welcome:

```
1.    {{left_sidebar_enabled,right_sidebar_enabled}}  
     ('message' in globals() ) } }  
  
2.    {{extend 'layout.html'}}}  
  
3.  
  
4.    {{block header}}}  
  
5.    <header class="container-fluid  
background">  
  
6.    <div class="jumbotron text-center">  
  
7.    {{if response.title:}}}  
  
8.    <h1>{{=response.title}}</h1>
```

```
9.      <small>{{=response.subtitle or ''}}</small></h1>
10.     {{pass}}
11.   </div>
12.   </header>
13.   {{end}}
14.
15.   {{if 'message' in globals():}}
16.   <h2>{{=message}}</h2>
17.   <p class="lead">{{=T('How did you get here?')}}</p>
18.   <ol>
19.     <li>{{=T('You are successfully running web2py')}}</li>
20.     <li>{{=XML(T('You visited the url %s', A(request.env.path_info, _href=request.env.path)))}}</li>
21.     <li>{{=XML(T('Which called the function %s located in the file %s', A(request.function+'()', _href='#'), A('web2py/applications/%(application)s/controllers/%(controller)s.py'))}}</li>
22.   (A(request.function+'()', _href='#'),
```

```
% request,
```

24.        \_href=URL('admin','default','peek', args=  
          (request.application,'controllers',request.cor  
          </li>

25.        <li>{ {=XML(T('The output of the file is a  
          dictionary that was rendered by the view %s',

26.        A('web2py/applications/%  
          (application)s/views/%  
          (controller)s/index.html' % request,

27.        \_href=URL('admin','default','peek',args=  
          (request.application,'views',request.controller  
          </li>

28.        <li>{ {=T('You can modify this application  
          and adapt it to your needs')} }</li>

29.        </ol>

30.        {{elif 'content' in globals():}}

31.        { {=content} }

32.        {{else:}}

33.        { {=BEAUTIFY(response.\_vars)} }

34.        {{pass}}

35.

36.        {{block right\_sidebar}}

```
37.      <div class="panel panel-info">
38.          <div class="panel-heading"><h3
39.              class="panel-title"><a class="btn-block"
40.                  href="
41. { {=URL ('admin', 'default', 'index') } }">
42.          <i class="glyphicon glyphicon-cog"></i>
43.          { {=T ("admin") } }
44.      </a></h3></div>
45.      <div class="panel-body">
46.          { {=T ("Don't know what to do?") } }
47.          <ul class="list-group">
48.              <li class="list-group-item">{ {=A(T("Online
49. examples")),
50. _href=URL('examples', 'default', 'index')) } }</li>
51.          <li class="list-group-item"><a
52. href="http://web2py.com">web2py.com</a></li>
53.          <li class="list-group-item"><a
54. href="http://web2py.com/book">
55. { {=T ('Documentation') } }</a></li>
56.      </ul>
```

51. </div>

52. { {end} }

O resultado vocês já viram antes:

Bem-vindo ao web2py!

Como você chegou aqui?

1. Você está executando o web2py com sucesso  
2. Você acessou a url /tarefa/default/index  
3. Que chamou a função index() localizada no arquivo web2py/applications/tarefa/controllers/default.py  
4. A saída do arquivo é um dicionário que foi apresentado pela visão web2py/applications/tarefa/views/default/index.html  
5. Você pode modificar esta aplicação e adaptá-la às suas necessidades

Olá Mundo x

admin

Não sabe o que fazer?

Exemplos online

web2py.com

Documentação

Copyright © 2016

Desenvolvido com web2py

Figura 32: Index Original

Logo na primeira linha do arqui temos o seguinte trecho:

```
1. {{left_sidebar_enabled,right_sidebar_enabled}}  
('message' in globals())}}
```

Essa linha habilita a barra lateral e desabilita a barra direita importando o conteúdo do globals chamado 'message':

Para habilitar as duas barras podemos usaremos:

```
1.    {{left_sidebar_enabled=False,  
right_sidebar_enabled=False, ('message' in  
globals())}}}
```

Ficando desta forma:

Olá Mundo ×

# Tarefa

Left Sidebar

## Bem-vindo ao web2py!

Como você chegou aqui?

1. Você está executando o web2py com sucesso
2. Você acessou a url [/tarefa/default/index](#)
3. Que chamou a função `index()` localizada no arquivo `web2py/applications/tarefa/controllers/default.py`
4. A saída do arquivo é um dicionário que foi apresentado pela visão `web2py/applications/tarefa/views/default/index.html`
5. Você pode modificar esta aplicação e adaptá-la às suas necessidades

 admin
Não sabe o que fazer?
Exemplos online
<a href="#">web2py.com</a>
Documentação

Copyright © 2016

Desenvolvido com [web2py](#)

Share

Figura 33: Index com duas barras

Ou para usar somente a barra da esquerda e do centro usamos:

```
1. {{left_sidebar_enabled=False, ('message' in
globals()))}}
```

Ficando desta forma:

Olá Mundo ×

# Tarefa

Left Sidebar

## Bem-vindo ao web2py!

Como você chegou aqui?

1. Você está executando o web2py com sucesso
2. Você acessou a url [/tarefa/default/index](#)
3. Que chamou a função `index()` localizada no arquivo [web2py/applications/tarefa/controllers/default.py](#)
4. A saída do arquivo é um dicionário que foi apresentado pela visão [web2py/applications/tarefa/views/default/index.html](#)
5. Você pode modificar esta aplicação e adaptá-la às suas necessidades

E para desabilitar as duas faremos:

```
1. {{{('message' in globals())}}}
```

Ficando desta forma:

Olá Mundo ×

# Tarefa

Bem-vindo ao web2py!

Como você chegou aqui?

1. Você está executando o web2py com sucesso
2. Você acessou a url </tarefa/default/index>
3. Que chamou a função `index()` localizada no arquivo `web2py/applications/tarefa/controllers/default.py`
4. A saída do arquivo é um dicionário que foi apresentado pela visão `web2py/applications/tarefa/views/default/index.html`
5. Você pode modificar esta aplicação e adaptá-la às suas necessidades

Copyright © 2016

Desenvolvido com [web2py](#)

Shar

Um pouco mais abaixo temos uma novidade o comando `block` que usa herança de classe das visões substituindo o conteúdo do `layout.htm` ou deixando caso não haja substituições, os blocos começam com o comando `{{block nome}}` e terminam com o comando `{{end}}`

Nosso primeiro bloco é o `{{block header}}` que define o topo do site, aqui mais uma vez temos praticamente códigos HTML e CSS do Bootstrap, não vem ao caso perder tempo explicando isso é conteúdo de outra matéria.

Lembra da nossa primeira linha, nela estamos importando a variável 'message' de `globals`, esta variável é testada logo depois do bloco `header` se existir uma série de mensagens explicando o web2py aparecerá, não precisamos de nada disso então podemos apagar tudo e substituir pelo nosso conteúdo, e já que não precisamos da variável 'message' podemos também apagar o trecho da linha 1, esta parte é uma das principais do `index` nela é que vai ficar o site em si.

O próximo bloco é o `right_sidebar` nele definimos o conteúdo da barra lateral direita, podemos até aproveitar parte desta estrutura para fazer nosso próprio menu lateral, vamos fazer isso sem muitas funcionalidade só como um exemplo mesmo.

Nosso código modificado completo ficou assim:

```
1.      {{left_sidebar_enabled,right_sidebar_enabled}}  
2.      ('message' in globals()) } }  
3.  
4.      {{block header}}}  
5.      <header class="container-fluid  
background">  
6.      <div class="jumbotron text-center">  
7.      <h1>Gerenciador de Tarefas</h1>  
8.      </div>  
9.      </header>  
10.     { {end} } }  
11.     <h1>  
12.     O Melhor Gerenciador de Tarefas do Mundo  
13.     </h1>  
14.     <div class="row text-center">  
15.     <div class="img-thumbnail"
```

```
16.     style="
17.         max-width:250px;
18.         margin:10px;
19.         border: 1px solid #939393">
20.     <h4>
21.     Seja mais organizado
22.     </h4>
23.     
26.     </div>
27.     <div class="img-thumbnail"
28.         style="
29.             max-width:250px;
30.             margin:10px;
31.             border: 1px solid #939393">
32.     <h4>
33.     Produza mais
34.     </h4>
```

```
34.        
42.      <h4>  
43.        Controle suas pendências  
44.      </h4>  
45.      
```

```
53.        <div class="panel-heading">
```

```
54.            <h3 class="panel-title">
```

```
55.                <i class="glyphicon glyphicon-ok"></i>
```

```
56.            { {=T ("Menu") } }
```

```
57.        </a></h3></div>
```

```
58.        <ul class="list-group">
```

```
59.            <li class="list-group-item"><a href="#">Tarefas</a></li>
```

```
60.            <li class="list-group-item"><a href="#">Como usar</a></li>
```

```
61.            <li class="list-group-item"><a href="#">Compartilhe</a></li>
```

```
62.        </ul>
```

```
63.    </div>
```

```
64.    {{end}}
```

Depois disso teremos uma visão index.html semelhante a esta:

# Gerenciador de Tarefas

Olá Mundo ×

## O Melhor Gerenciador de Tarefas do Mundo

Seja mais organizado



Produza mais



Controle suas pendências



✓ Menu

Tarefas

Como usar

Compartilhe

Que ainda pode ser melhorada com a sua imaginação.

Veja que usamos algumas url's para arquivos estáticos que veremos logo mais.



# Ajustes da visão Layout

Já terminamos de criar nossas visões específicas mas ainda precisamos de alguns ajustes para que o nosso sistema fique bem legal.

Primeiramente vamos ajustar nosso menu superior, nos temos na verdade dois menus o menu administrativo que nos dá acesso a várias funções de desenvolvimento e o menu da aplicação que é o que nos importa mesmo.

Os menus são gerenciados no modelo menu.py, então abra-o em modo de edição e encontre a linha começando em:

```
response.menu = [
```

Esta linha inicia o menu da nossa aplicação, cada elemento dentro dos colchetes será um item do menu.

Logo abaixo edite a linha para desabilitar o menu administrativo ficando assim:

```
1. DEVELOPMENT_MENU = False
```

Desta forma só teremos o menu da aplicação que deve estar assim:

Antes:

Depois:

Vamos adicionar novos itens no nosso menu, edite as linhas de cima da que editamos anteriormente para que fiquem assim:

```
1.     response.menu = [  
2.         (T('Home'), False, URL('default',  
'index'), []),  
3.         (T('Tarefas'), False, URL('default',  
'tarefas'), []),  
4.         (T('Fale Conosco'), False, URL('default',  
'contato'), [])  
5.     ]
```

Desta forma já adicionamos novos itens ao nosso menu direcionando para as funções do nosso controle:

Temos então agora o seguinte menu:

The screenshot shows a dark-themed browser window. At the top is a navigation bar with the 'web2py™' logo on the left, followed by three menu items: 'Principal', 'Tarefas', and 'Fale Conosco'. On the far right is a 'Log In' button with a dropdown arrow. Below the navigation bar, there is a large, empty white area which is likely the main content area of the web application.

Deves ter percebido que cada item do menu tem um colchete no final, este colchete serve para criarmos submenus cada item de menu que estiver dentro de um colchete será um submenu.

Vamos editar o código anterior para criar submenus, deixe-o desta forma:

```
1.     response.menu = [  
2.         (T('Home'), False, URL('default',  
3.             'index'), []),  
4.         (T('Tarefas'), False, URL('default',  
5.             'tarefas'), [  
6.                 (T('Listar'), False, URL('default',  
7.                     'tarefas')),  
8.                 (T('Nova Tarefa'), False, URL('default',  
9.                     'nova'))  
10.            ]),  
11.            (T('Fale Conosco'), False, URL('default',  
12.                'contato'), [])]
```

Feito isto teremos:



Agora que já sabemos como manipular o menu recomendo dar uma lida nas primeiras linhas onde é possível manipular algumas funções da tag head do HTML como `<meta author="">` `<meta description="">` e muitas outras, além de gerenciar o titulo e subtítulo da aplicação, o logo e outros itens do cabeçalho.

Agora vamos dar uma olhada na visão layout.html:

Temos um código já preparado para não precisar ser editado muito pra funcionar, acho que só o rodapé precisa de uma atenção, então encontre este trecho:

```
1.     {{block footer}} <!-- this is default
footer -->
```

```
2.     <footer class="footer">
```

```
3.     <div class="container-fluid">
```

```
4.      <div class="copyright pull-left">  
 { {=T('Copyright')} } } &#169;  
 { {=request.now.year} }</div>  
  
5.      <div id="poweredBy" class="pull-right">  
  
6.          { {=T('Powered by') } }  
  
7.      <a  
 href="http://www.web2py.com/">web2py</a>  
  
8.      </div>  
  
9.      </div>  
  
10.     </footer>  
  
11.     { {end} }
```

Vamos editar a parte do desenvolvimento para fazer uma '*merchan*' da minha própria agência de design então edite o código pra que fique assim:

```
1. { {block footer} } <!-- this is default footer  
-->  
  
2. <footer class="footer">  
  
3. <div class="container-fluid">
```

```
4. <div class="copyright pull-left">  
    {{=T('Copyright')}} } } &#169;  
    {{=request.now.year}}</div>  
  
5. <div id="poweredBy" class="pull-right">  
  
6. Desenvolvido por  
  
7. <a href="http://www.designholder.net">Design  
Holder</a>  
  
8. com  
  
9. <a href="http://www.web2py.com/">web2py</a>  
  
10. </div>  
  
11. </div>  
  
12. </footer>  
  
13. {{end}}
```

Assim fica nossa versão final do layout e do index, lembrando que todas as mudanças no layout são herdadas pelas visões que os estendem:

# Gerenciador de Tarefas

Olá Mundo ✕

## O Melhor Gerenciador de Tarefas do Mundo

Seja mais organizado



Produza mais



Controle suas pendências



✓ Menu

Tarefas

Como usar

Compartilhe

127.0.0.1:8000/tarefa/default/index

web2py™ Principal Tarefas Fale Conosco Log In

# Gerenciador de Tarefas

Olá Mundo x

O Melhor Gerenciador de Tarefas do Mundo

Seja mais organizado  
  
KEEP CALM AND FOCA NA TAREFA

Produza mais  


Controle suas pendências  


Menu  
Tarefas  
Como usar  
Compartilhe

Copyright © 2016 Desenvolvido por Design Holder com web2py

Figura 34: Versão final

Muito bem, acabamos nosso aplicativo de demonstração creio que já temos uma boa base pra programar em web2py vamos lá, mãos a massa.



# **Finalizando**

## **Arquivos estáticos**

O web2py tem uma seção própria para manter arquivos que não são fundamentais para o funcionamento do web2py ou da logica da aplicação, ou seja arquivos que não fazem parte nem dos modelos, nem dos controles, nem das visões.

Podemos acessá-los pela sessão específica do web2py que fica logo abaixo das visões:

- [Editor](#) [tr.py](#) ( Plural-Forms: are not used )
- [Editor](#) [uk.py](#) ( Plural-Forms: [Editor](#) plural-uk.py )
- [Editor](#) [zh.py](#) ( Plural-Forms: are not used )
- [Editor](#) [zh-cn.py](#) ( Plural-Forms: are not used )
- [Editor](#) [zh-tw.py](#) ( Plural-Forms: are not used )

[Criar](#)

## Static

- [Editor](#) [403.html](#)
  - [Editor](#) [404.html](#)
  - [Editor](#) [500.html](#)
- [css/](#)
  - [fonts/](#)
  - [images/](#)
  - [js/](#)

[Create/Upload](#)

## Módulos

- [Editor](#) [\\_\\_init\\_\\_.py](#)

[Create/Upload](#)

## Private files

### Figura 35: Arquivos Estáticos

Temos neste diretório três arquivos html que correspondem aos tipos de erros de servidor mais comuns 403, 404 e 500 e quatro diretórios para armazenar informações específicas, respeite estes diretórios:

- css → para folhas de estilo
- font → para arquivos de fontes (tipos de letras)
- images → para imagens
- js → para arquivos JavaScript

Faça o fazer o upload de uma imagem para os arquivos estáticos.

Esta imagem pode ser recuperada em um visão a partir do helper URL com no exemplo do curso da Julia Rizza:

```
1.      <html>
2.      <head>
3.          <title>Teste</title>
4.          <link rel="stylesheet" type="text/css"
 href="{{=URL('static', 'css/style.css')}}" />
5.          <script src="{{=URL('static',
 'js/script.js')}}" type="text/javascript">
</script>
6.      </head>
7.      <body>
8.          
9.      </body>
10.     </html>
```



# Internacionalização

O web2py tem um sistema incrível de internacionalização que percebe o idioma do sistema e traduz para o idioma do usuário, mas é claro que nem todas as expressões são traduzidas, é preciso que a expressão esteja dentro do operador 'T' e preferencialmente escrita em inglês, após ainda é preciso configurar os idiomas para entender as traduções, caso não haja tradução nada será feito.

Ou seja, se na nossa visão colocarmos:

1. `<h1>What is your name?</h1>`

Simplesmente aparecerá na tela a frase dentro de um título.

Mas se colocarmos desta forma

1.

```
<h1>{ { =T ("What  
is your name?") } }</h1>
```

A frase aparecerá traduzida para cada idioma. Tem algumas coisinhas no sôsso aplicativo que ainda precisa de tradução, como por

exemplo 'Submit' dos formulários, vamos arrumar isso na configuração de idioma então encontre pt-br.py e clique em editar, uma tela como esta aparecerá:

web2py™ interface administrativa

Site Editar sobre Erros Versionamento Ajuda finalizar sessão Debug

## Editando arquivo de idioma "tarefa/languages/pt-br.py"

Hide/Show Translated strings

### Original/Tradução

!=

!langcode!

!langname!

"update" is an optional expression like "field1='newvalue'". You cannot update or delete the results of a JOIN

"update" é uma expressão opcional como "campo1='novovalor'". Você não pode atualizar ou apagar os resultados de um JOIN

apagar

apagar

apagar

Save Changes

Figura 36: Painel de tradução

Cada elemento que está precisando de tradução aparecerá em destaque amarelo, para traduzir basta escrever no campo logo abaixo o significado e clicar em salvar.

Veja:

Stylesheet

Folha de estilo  apagar

submit

enviar  apagar

Submit

Enviar  apagar

Success!

Feito!  apagar

Support

Suporte  apagar

Sure you want to delete this object?

Está certo(a) que deseja apagar este objeto?

Save Changes

Figura 37: Painel de tradução

Será que deu certo? Vamos testar em uma visão vou abrir a visão nova para ver como ficou.

Nome

Preencha o formulário! ×

Descrição

Prazo

Prioridade

 concluída

Imagem

Nenhum arquivo selecionado.

Figura 38: Visão Nova Traduzida

Vale a pena dedicar um tempinho para traduzir nosso aplicativo.



# Deploy

Fazer o upload do nosso aplicativo para um servidor é uma tarefa tranquila principalmente porque temos alguns recursos prontos para isso, o mais simples deles é o [pythonanywhere.com](http://pythonanywhere.com).

É bem simples basta acessar o site e criar uma conta, pode ser a conta gratis mesmo, não tem problema, o ruim é que não poderas usar um dominio próprio mas terás um dominio como `seu_usuario.pythonanywhere.com`.

Então na aba 'web' clique em 'Add a new app', 'next', escolha 'web2py' e 'next', escolha o caminho e uma senha, da mesma forma que fazemos no desktop, já está pronto. Quando acessares `seu_usuario.pythonanywhere.com` terás a mesma interface do seu web2py de desktop.

Bem simples não? Agora vou mostrar como pegar seu aplicativo que está pronto no desktop e mandar para o servidor web.

Abra o painel administrativo do web2py do seu desktop, não o painel da aplicação o painel do web2py mesmo, este aqui:

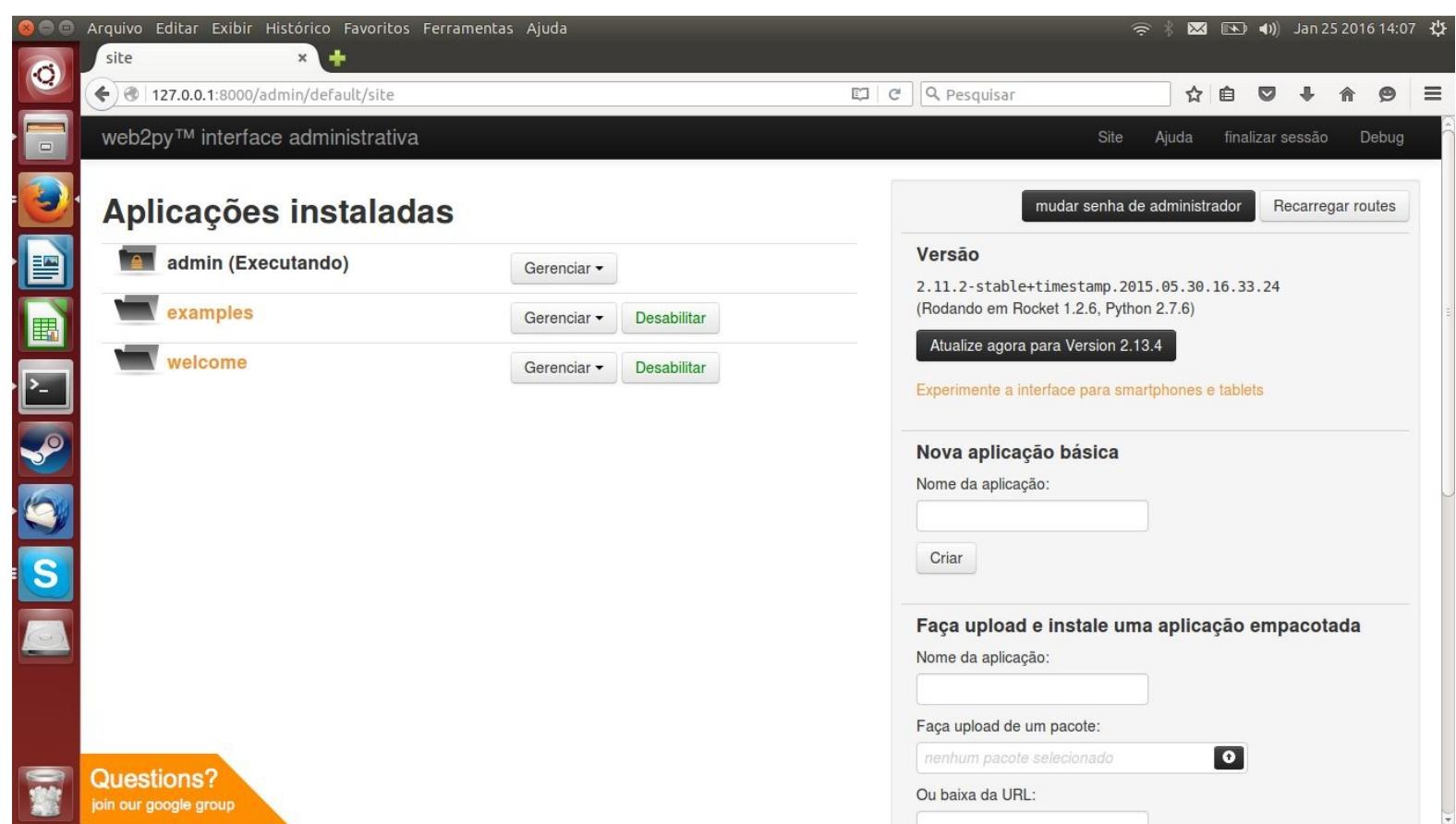


Figura 39: Painel do Web2py

Na sua aplicação, clique em gerenciar e depois em criar pacote,

## Aplicações instaladas

admin (Executando)	Gerenciar
examples	Gerenciar Desabilitar
tarefa	Gerenciar Desabilitar
welcome	Gerenciar Desabilitar

**Questions?**  
Join our google group  
[127.0.0.1:8000/admin/default/compile/app/tarefa](http://127.0.0.1:8000/admin/default/compile/app/tarefa)

[mudar senha de administrador](#) [Recarregar routes](#)

**Versão**  
2.11.2-stable+timestamp.2015.05.30.16.33.24  
(Rodando em Rocket 1.2.6, Python 2.7.6)  
[Verificar se existem atualizações](#)

Experimente a interface para smartphones e tablets

---

**Nova aplicação básica**  
Nome da aplicação:  
  
[Criar](#)

---

**Faça upload e instale uma aplicação empacotada**  
Nome da aplicação:  
  
Faça upload de um pacote:  
 [Upload](#)  
Ou baixa da URL:

Figura 40: Criar Pacote

Será feito o download de um do pacote da sua aplicação no formato .w2p está é a nossa aplicação que será mandada para o web server.

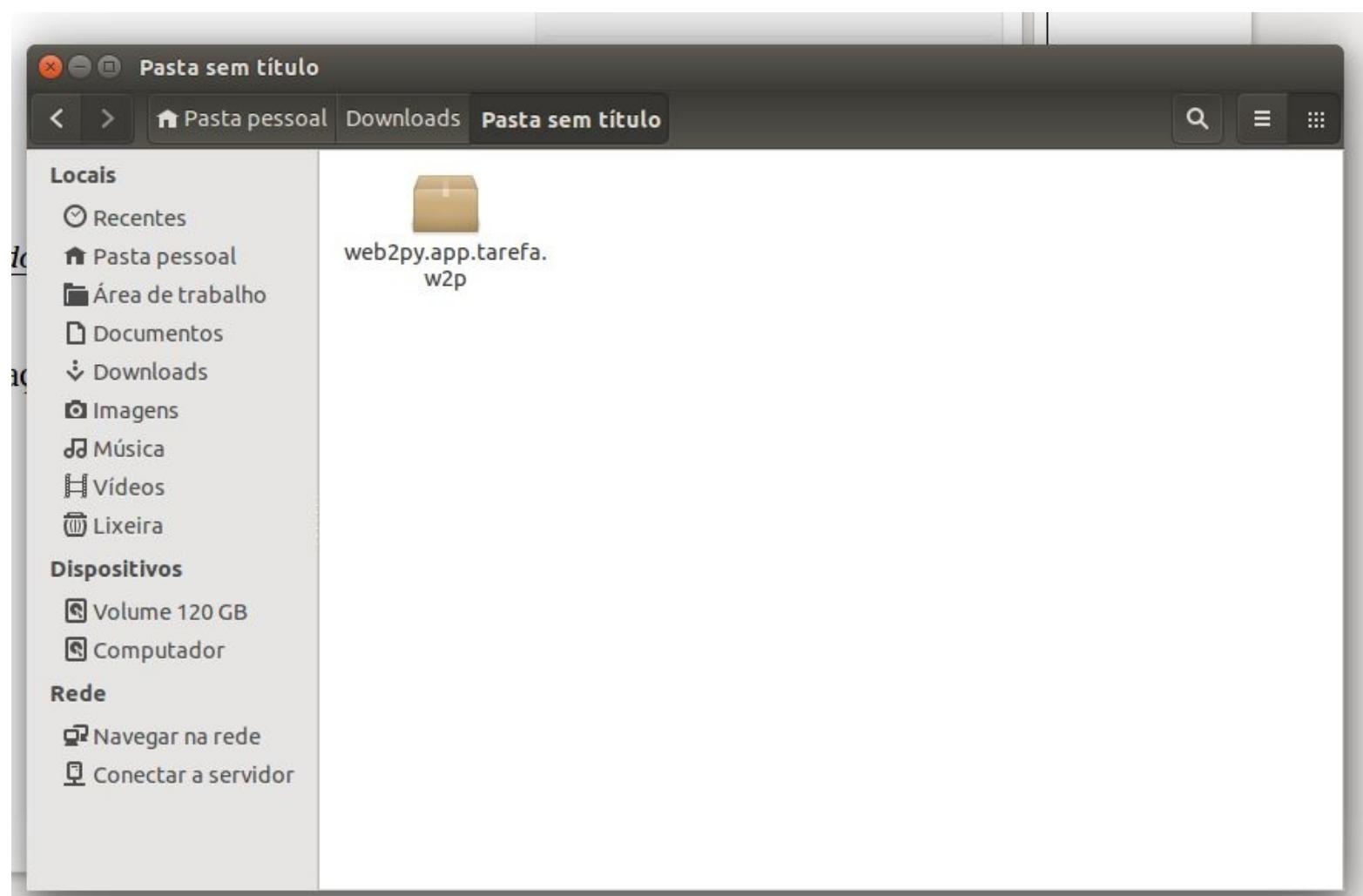


Figura 41: Pacote w2p

Agora vamos para o web server e acesse a interface administrativa do web2py de lá, na barra lateral direita temos a opção 'Faça upload e instale uma aplicação empacotada'.

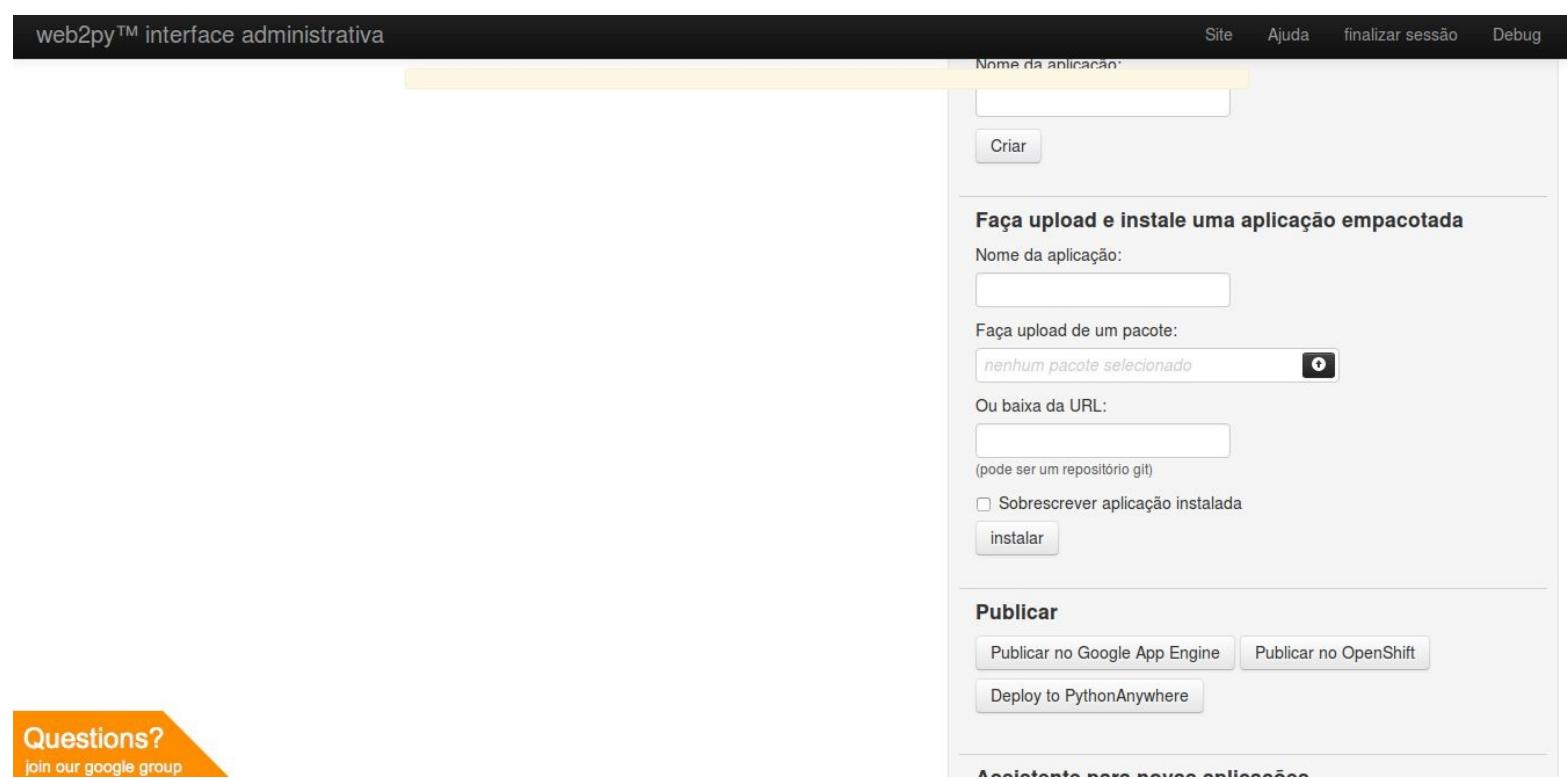


Figura 42: Upload

Em nome da aplicação escreva o nome que queres que a aplicação tenha, ela será renomeada caso o nome seja diferente, depois escolha o pacote criado em 'Faça upload de um pacote' e clique em instalar.

Prontinho, já temos nossa aplicação online inclusive com as informações dos bancos de dados, ou seja podes usar o e-mail para logar, e ver as mesmas tarefas cadastradas offline, elas já estão online.

Repare na URL com as coisas estão ok já (estou usando a conta da minha esposa nesse print).

https://fernandabmd.pythonanywhere.com/tarefas/default/tarefas

web2py™ Principal Tarefas▼ Fale Conosco Bem-vindo Mauro ▾

## Lista das tarefas

Nova Tarefa

Cadastrar tarefas mais uma tarefa tarefa com imagem Estudar Python Terminar o livro de web2py

Prioridade: Alta Prioridade: Baixa Prioridade: Baixa Prioridade: Baixa Prioridade: Média

Prazo Final 12/12/2016 Prazo Final 12/12/2016 Prazo Final 01/03/2016

Ver mais Ver mais Ver mais

Copyright © 2016 Desenvolvido por Design Holder com web2py

Figura 43: Aplicação Online no pythonanywhere.com

Isto é bem interessante já que não perdemos o trabalho feito offline quando vamos para o online, e também bem prático.

Tente você também.





# Conclusão

Este livro 'Primeiros Passos com Web2py' foi escrito com a proposta de mostrar o caminho que percorri para conhecer mais sobre esta tecnologia que achei incrível.

Passamos por diversas etapas, conhecemos um pouco da história e da filosofia por trás destas tecnologias, bem como, da orientação a objetos e do padrão modelo, visão, controlador e detalhamos os modelos, visões e controles do web2py, e ainda alguns outros recursos como a tradução e o deploy na aplicação.

Muito ainda falta para vermos tudo que podemos ver, mas isso ficará para outros trabalhos futuros.

Certamente ainda não sei tudo que preciso saber e a cada dia aprendo um pouco mais, inclusive ao escrever fiz uma serie de pesquisas e aprendi bastante, sei que temos, blogs, e-books, tutoriais e principalmente o livro do próprio Massimo como material de consulta.

Ainda continuo estudando python, web2py e muitas outras tecnologias e linguagens de programação, recomendo que faça o mesmo e pratique, porque programar só se aprende praticando muito.

Minha sugestão é que invente um motivo qualquer para programar e programe, seja em web2py ou qualquer outra linguagem e tecnologia.

Boa sorte, fique com Deus e até a próxima.

# Índice de figuras

Figura 1: Download do Web2py 16

Figura 2: Console Python 17

Figura 3: Terminal chamando o script 38

Figura 4: Programação Estruturada x Programação Orientada a Objetos 48

Figura 5: WebServer do Web2py 49

Figura 6: Aplicação Welcome 50

Figura 7: Painel Administrativo 51

Figura 8: Interface Administrativa da Aplicação tarefa 52

Figura 9: Modelo Gráfico 59

Figura 10: Modelo Gráfico Exportado 60

Figura 11: Administração de Banco de Dados 61

Figura 12: Administração de Banco de Dados 62

Figura 13: Administração de Banco de Dados 63

Figura 14: Função Index 74

Figura 15: Função Tarefas Controle Default 76

Figura 16: Função Tarefas Controle Default 77

Figura 17: Consulta Tabela Tarefas 78

Figura 18: Detalhar Tarefa 79

Figura 19: Detalhar Tarefa Nulo 79

Figura 20: Nova Tarefa 80

Figura 21: Contato 4

Figura 22: Administração de Banco de Dados 6

Figura 23: Novo Registro Grupo 7

Figura 24: Novo Registro Membership 8

Figura 25: Modelo Grafico Tarefa Modificado 10

Figura 26: Tarefas do Usuário 11

Figura 27: Visão contato 21

Figura 28: Visão Detalhar 24

Figura 29: Visão Editar 25

Figura 30: Visão Nova 26

Figura 31: Visão Tarefas 29

Figura 32: Index Original 33

Figura 33: Index com duas barras 34

Figura 34: Versão final 44

Figura 35: Arquivos Estáticos 45

Figura 36: Painel de tradução 47

Figura 37: Painel de tradução 48

Figura 38: Visão Nova Traduzida 48

Figura 39: Painel do Web2py 50

Figura 40: Criar Pacote 51

Figura 41: Pacote w2p 51

Figura 42: Upload 52

Figura 43: Aplicação Online no pythonanywhere.com 53



# Bibliografia

Acesse [http://web2pybrasil.com.br/livro\\_passos](http://web2pybrasil.com.br/livro_passos)

1Festival Latino Americano de Instalação de Software Livre

2Pessoa que por motivos diversos se torna um fiel fanático de uma determinada marca ou tecnologia, por exemplos: Apple, Sony, Linux, etc.

3O bloco de notas do windows salva os arquivos com extenção automatica .txt para que isso não aconteça é preciso no momento de “salvar como” escolher “todos os arquivos” na opção “tipo de arquivo”, acho que é isso.

4<http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>

5Editora Casa do Código

6Você precisa estudar expressões regulares. :)

7<https://www.juliarizza.com/>

53 De 138