Módulos e Pacotes

Python

Módulos

- import <modulo1>, <modulo2>, ..., <moduloN>
- from <modulo> import <def1>, <def2>, ..., <defN>
- from <modulo> import (<def1>, <def2>, ..., <defN>)
- ...

```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import math, core
>>>
```

Módulos

```
1  from math import sqrt, cos, sin
2
3  class Robot2D():
4      counter = 0
5      def __init__(self, x=0.0, y=0.0, dx=1.0, dy=0.0):
6          self.name = "mybot"
7          self.pos = [x, y]
8          self.dir = [dx, dy]
9          self._normalize_dir()
10         self.live = True
11         Robot2D.counter += 1
```

Módulos

```
12
13      def move(self, d):
14          x = self.pos[0] + self.dir[0] * d
15          y = self.pos[1] + self.dir[1] * d
16          self.pos[0] = x
17          self.pos[1] = y
18
19      def rotate(self, ang):
20          x = self.dir[0] * cos(ang) + self.dir[1] * sin(ang)
21          y = -self.dir[0] * sin(ang) + self.dir[1] * cos(ang)
22          self.dir[0] = x
23          self.dir[1] = y
24          self._normalize_dir()
```

...

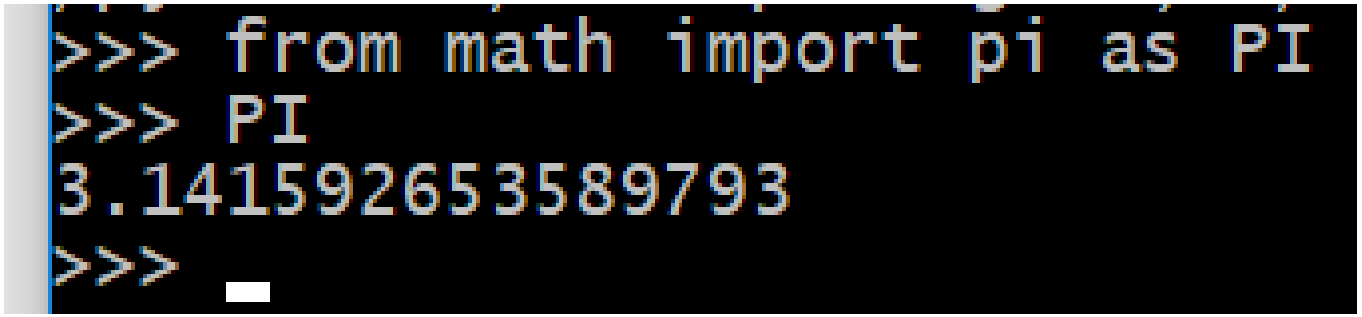
Módulos

- Qualquer arquivo terminado com extensão .py e que contenha código python válido pode ser considerado um módulo e será reconhecido como tal se estiver dentro de um caminho de busca (*path*) do python:
  - Diretório de trabalho atual
  - Caminhos indicados pela variável sys.path:

```
>>> import sys
>>> sys.path
['', '/usr/lib/python3.4', '/usr/lib/python3.4/plat-x86_64-linux-gnu',
/python3.4/dist-packages', '/usr/lib/python3/dist-packages']
>>>
```

Módulos

- Modificando nomes importados localmente

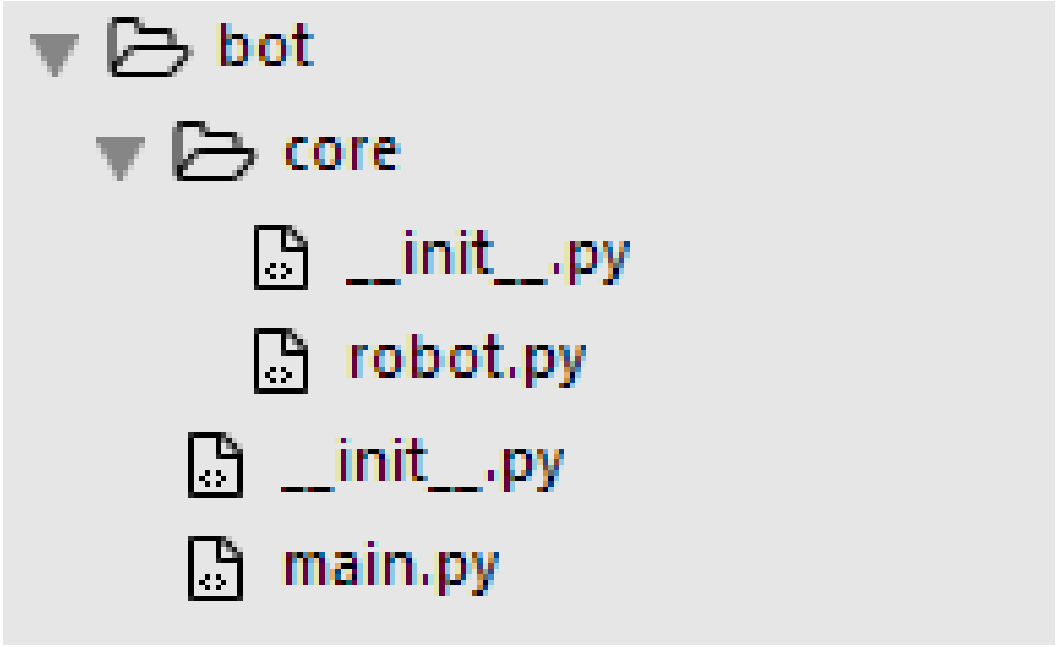


Pacotes

- O que é: agrupamento de módulos!
- Objetivo: lidar com a complexidade de uma arquitetura com muitos módulos.
- Na prática: um diretório com um arquivo __init__.py é tratado como um pacote.

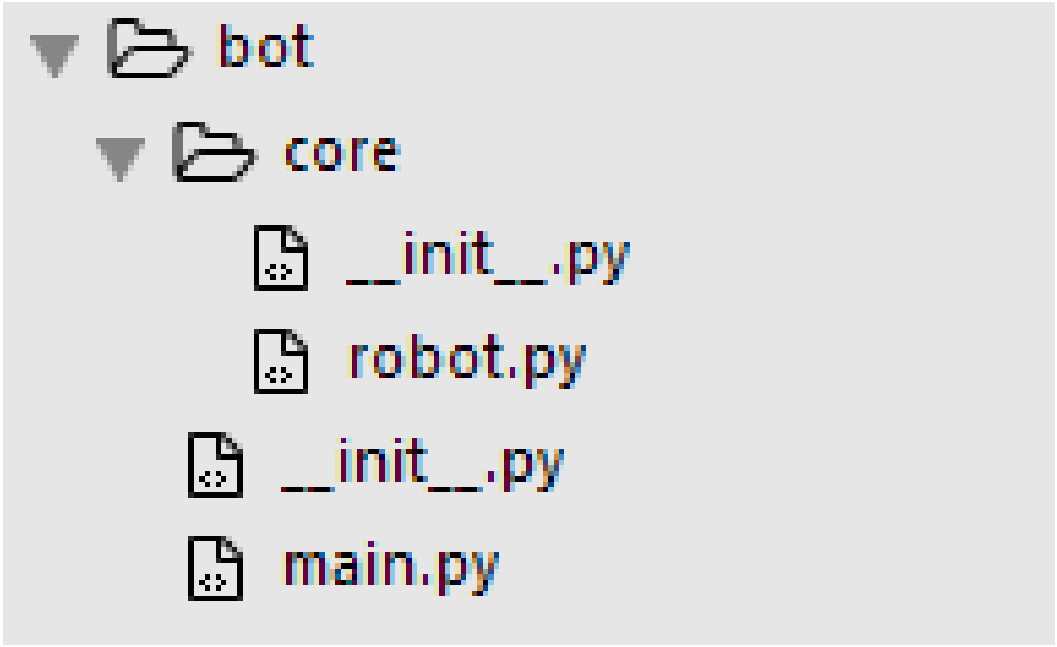
Pacotes

- Exemplo: um aplicativo para controlar um robô virtual que se move em um plano bidimensional.



Pacotes

- Exemplo: dentro do módulo robot.py, temos uma classe Robot2D
- Para usar essa classe em main.py, como o arquivo bot/core/__init__.py vazio, podemos fazer:
  - **from core.robot import Robot2D**



Pacotes

- Exemplo: arquivo main.py

```
1  from core.robot import Robot2D
2  from math import pi
3
4  if __name__ == '__main__':
5      rbt = Robot2D()
6      rbt.move(0.2)
7      rbt.rotate(pi/4.0)
8      rbt.move(1.0)
9      print(rbt)
```

Pacotes

- Quando importamos um pacote, todo o código dentro do arquivo __init__.py do pacote é executado.
- No exemplo anterior, observe que, para importar a classe Robot2D, foi necessário importar o pacote core.robot. Se quisermos agrupar todos os módulos de pacote em um único nome para simplificarmos a forma de importação, podemos utilizar o arquivo __init__.py do pacote para fazermos o controle fino de quais nomes são exportados quando o pacote for carregado.
- Lembre-se: quando o pacote é carregado, o código no __init__.py do pacote é executado.

Expondo nomes e definições em módulos para clientes do pacote

- Para expor os nomes de módulos sob o nome do pacote, dentro do __init__.py do pacote, importamos os módulos e definições que queremos expor:
  - No exemplo anterior, queremos expor a class Robot2D de tal modo que para a utilizarmos no módulo main.py, tenhamos simplesmente que fazer:

Para isso, em __init_.py de core, fazemos:

from core.robot import *

```
1  from core import Robot2D
2  from math import pi
3
4  if __name__ == '__main__':
5      rbt = Robot2D()
6      rbt.move(0.2)
7      rbt.rotate(pi/4.0)
8      rbt.move(1.0)
9      print(rbt)
```

Expondo nomes e definições em módulos para clientes do pacote

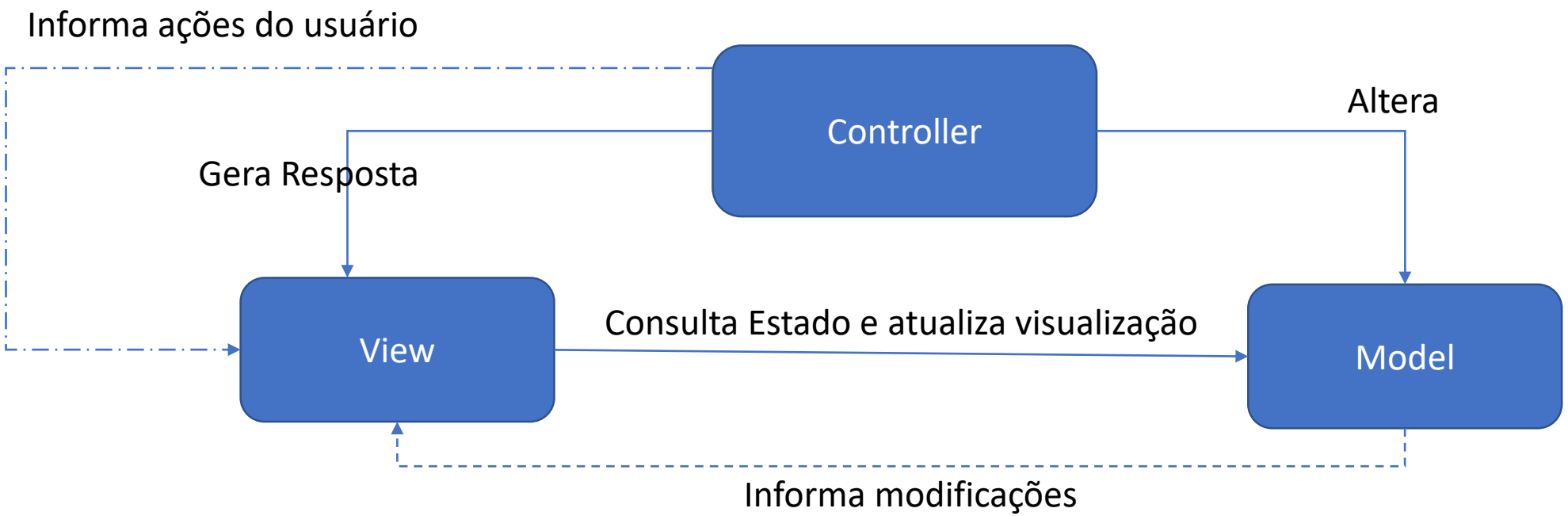
- Suponha que no arquivo robot.py tenhamos duas classes: Robot2D e Room2D.
- Com o comando **from core.robot import *** em core/__init__.py, as duas classes ficam expostas aos clientes ao código cliente do pacote core.
- Se quisermos expor apenas uma classe, Robot2D, por exemplo, fazemos: **from core.robot import Robot2D**. Nesse caso, a classe Room2D somente fica visível no código cliente (main.py, por exemplo), se fizermos a importação completa em main.py: **from core.robot import Room2D**.

Expondo nomes e definições em módulos para clientes do pacote

- Outra forma de controle de exposição de nomes é pela definição da variável __all__.
- Em relação ao __init__.py de um pacote:
  - Quando não declaramos a variável __all__, todos os módulos e definições importadas são expostas no nome do pacote.
  - Quando definirmos a variável __all__ como uma lista, apenas os nomes contidos na lista serão expostos
  - Exemplo: no __init__.py do pacote core:

```
from core.robot import *
__all__ = [“Robot2D”]
```

Arquitetura MVC no Python



Atividade 1

- O ideal no padrão arquitetural MVC é que o **Controller**, a **View** e o **Model** sejam classes abstratas, de forma a isolarmos o uso da arquitetura de implementações específicas. Contudo, em Python, podemos utilizar outra abordagem para implementar o padrão MVC. Esta outra abordagem está relaciona à forma como os tipos são tratados em Python. Qual seria essa outra abordagem?

Atividade 2

- Baixe o repositório https://github.com/professorgilzamir/pythoncourse
- Observe a estrutura da aplicação e veja como a aplicação foi construída com base no padrão MVC. Para isso, os conceitos utilizados nesta aula foram utilizados.
- Contudo, observe no pacote **control**, o módulo controller.py importa o módulo console diretamente. Este módulo é uma implementação de **view**. Se criarmos uma nova **view** em um módulo diferente, teremos que alterar o módulo controller.py, especificamente, teremos que alterar o **import** de modo que a aplicação tenha acesso ao módulo específico da nova **view**. Como poderíamos evitar que o controller.py seja alterado toda vez que criarmos um novo módulo de **view**? Altere a aplicação para se resolver este problema. (DICA: no **__init__.py** do pacote **view** podemos controlar a exposição do que quem está fora de **view** pode acessar pelo nome **view**).

Atividades 3

- Abra o terminal no diretório **src**
- Execute a aplicação com o comando **python3 main.py**
- Observe que as opções 3 e 4 mostram uma mensagem dizendo que as respectivas funcionalidades não foram implementadas.
- Tente encontrar aonde estas funcionalidades deveriam ser implementadas e as implemente.
- Qual dificuldade de encontrar a implementação específica? Este seria um efeito colateral negativo da exposição de nomes/definições nos pacotes?

Atividade 4

- Implemente uma nova **View** em um módulo chamado report.py. Esta nova **view** funciona como a anterior, contudo, no lugar de mostrar os dados na tela, salvo os dados resultantes em um arquivo **html** e de forma estruturada dentro de uma tabela com duas colunas: o nome da unidade de saúde e o seu endereço (concatene os campos do endereço de forma que os dados de endereço caibam em uma única célula da tabela).