Suerte - 2012

Live de Python

#115 - Uma introdução aos padrões de projeto

Obrigado!

apoia.se/livedepython

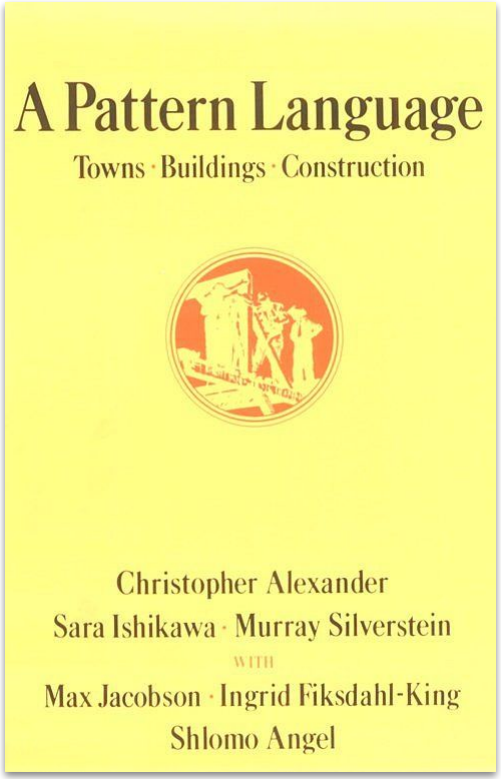
```
dunossauro at bouman in ~/git/apoiase on master*
$ python apoiadores.py
Alexandre Possebom   Alysson Oliveira     Amaziles Carvalho    Andre Machado
Andre Rodrigues      Bernardo Fontes      Bruno Guizi          Carlos Augusto
Cleber Santos        Cleiton Mittmann     David Reis           Dayham Soares
Diego Ubirajara      Edimar Fardim        Eliabe Silva         Eliakim Morais
Elias Soares         Emerson Lara         Eugenio Mazzini      Fabiano Silos
Fabiano Teichmann    Fabiano Gomes        Fernando Furtado     Franklin Silva
Fábio Serrão         Gleison Oliveira     Guilherme Ramos      Hemilio Lauro
Humberto Rocha       Hélio Neto           JOAO COELHO          JONATHAN DOMINGOS
Jean Vetorello       Johnny Tardin        Jonatas Oliveira     Jonatas Simões
João Lugão           Jucélio Silva        Júlia Kastrup        Leon Teixeira
Lucas Nascimento     Magno Malkut         Marcus Salgues       Maria Boladona
Matheus Francisco    Nilo Pereira         Pablo Henrique       Paulo Tadei
Pedro Alves          Rafael Galleani      Regis Santos         Renan Moura
Renato Santos        Rennan Almeida       Rodrigo Ferreira     Rodrigo Vaccari
Sérgio Passos        Thiago Araujo        Tiago Cordeiro       Tyrone Damasceno
Vergil Valverde      Vicente Marcal       Wander Silva         Welington Carlos
Wellington Camargo   Welton Souza         William Oliveira     Willian Gl
Yros Aguiar          Falta você           Falta você           Falta você
```

Roteiro

- Uma visão histórica sobre os patterns
- Nem tudo são flores
- O que são padrões
- Como são classificados
- Um exemplo
- Histórinha para fixação

Origem dos padrões de projeto

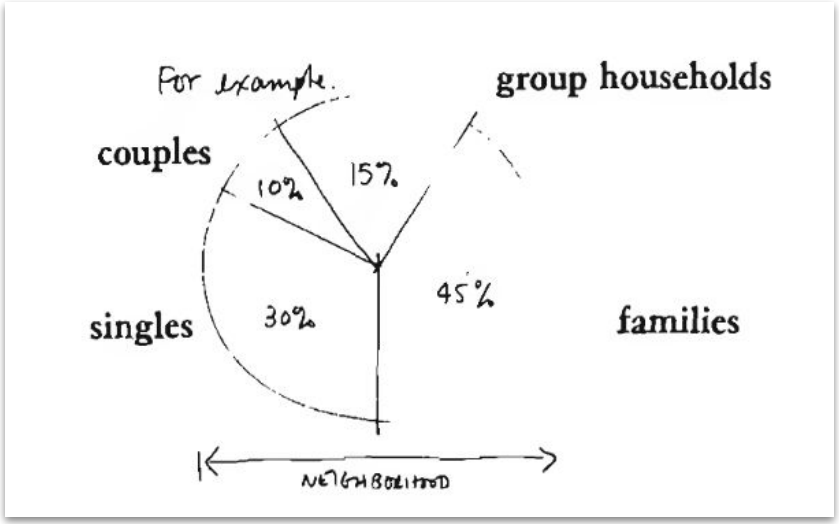
253 padrões para arquitetura e construções de bairros, cidades, áreas e afins...

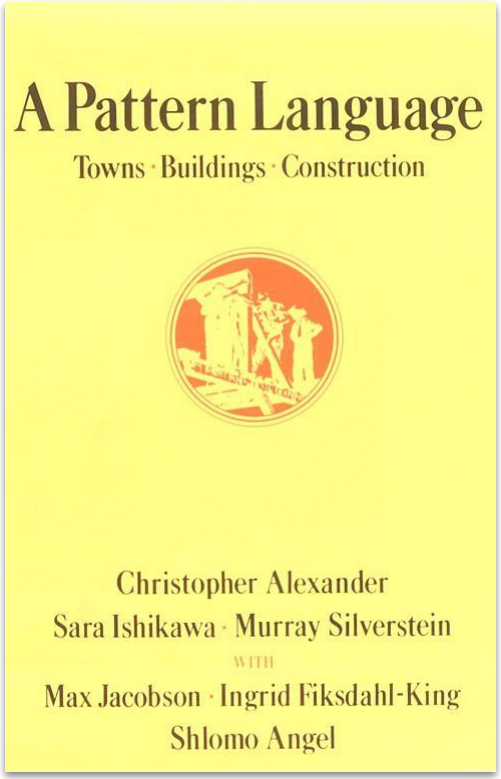


Household mix (padrões)

Que tipo de mistura um bairro bem equilibrado deve conter?

- Life cycle
- House cluster





https://youtu.be/6bSjvmD-XBM

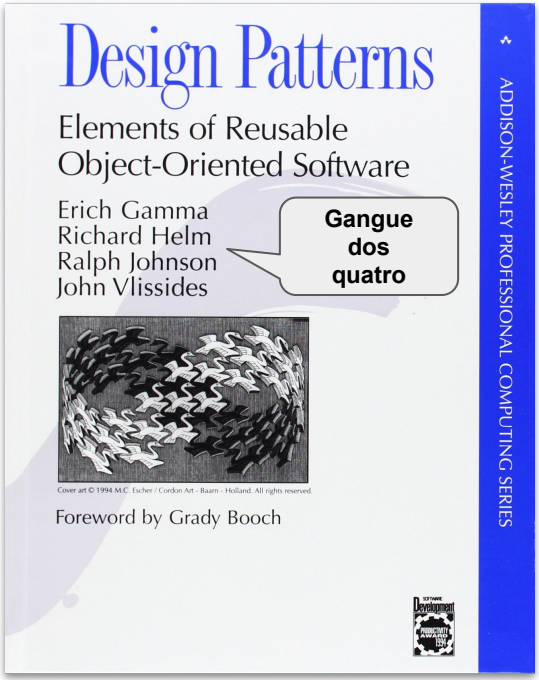
Live de Python

Origem dos padrões de projeto

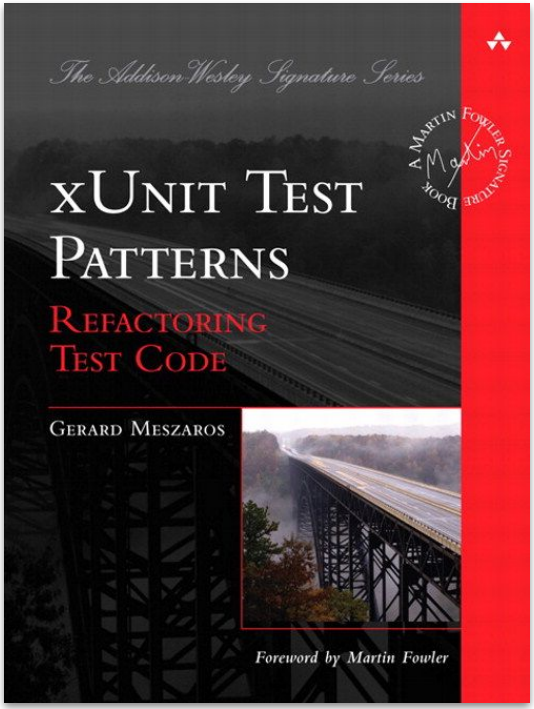
Uma adaptação da linguagem de patterns para o mundo do software.

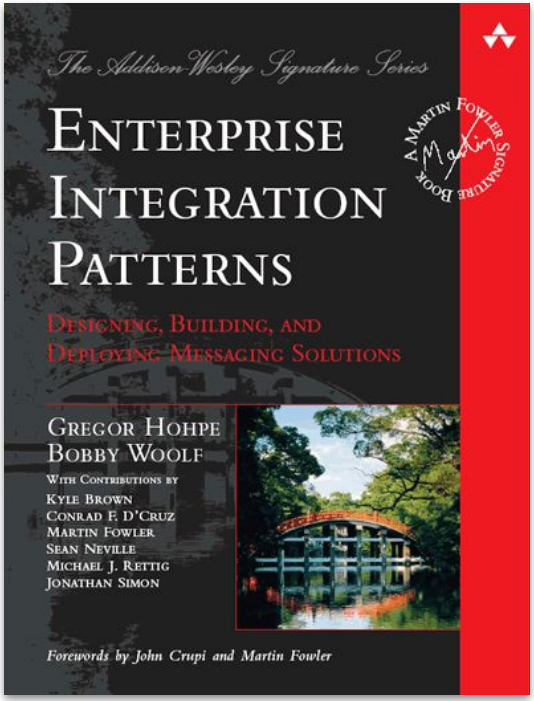
Lançado em 1994 se tornou referência e o livro “clássico” de padrões de projeto. Definem 23 padrões para projetos

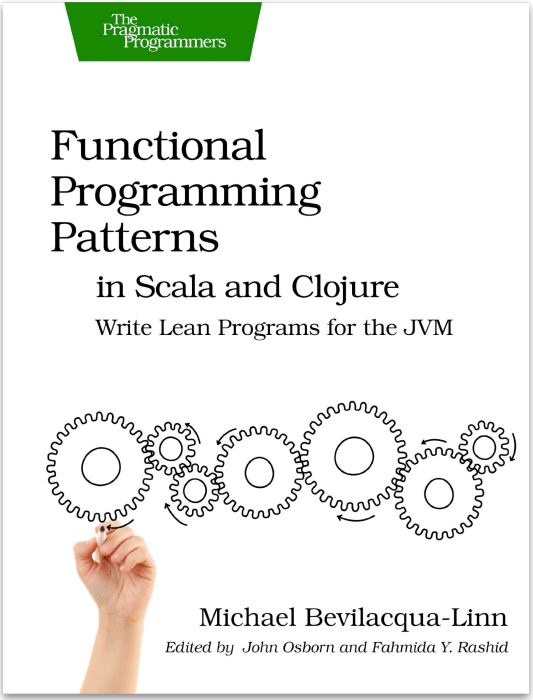
Suas implementações são feitas em C++ (em grande parte)



Isso vai mais além ...

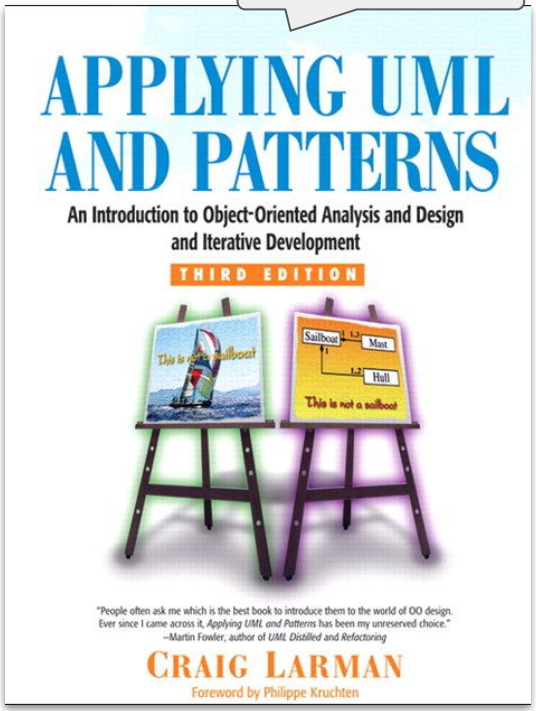




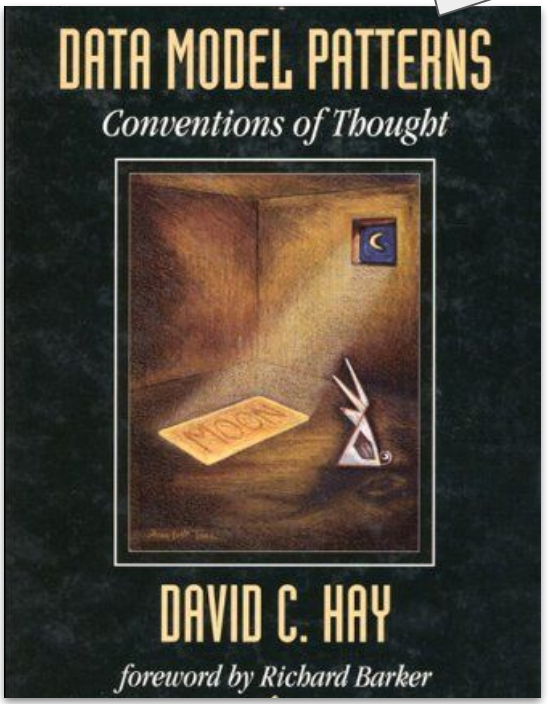


Isso vai mais além ...

GRASP

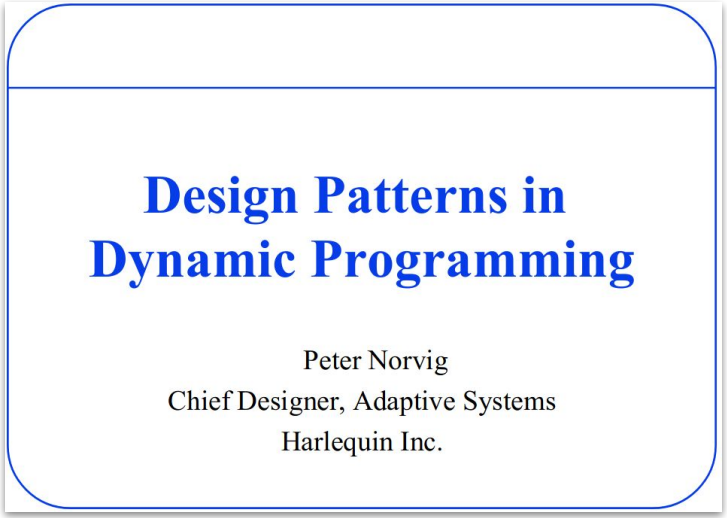


Banco de dados



Live de Python

Pontos de atenção



http://www.norvig.com/design-patterns/design-patterns.pdf

Are Design Patterns Missing Language Features

A list of DesignPatterns and a language feature that (largely) replaces them:

```
VisitorPattern  ..............  GenericFunctions (MultipleDispatch)
FactoryPattern  ..............  MetaClasses, closures
SingletonPattern .............  MetaClasses
IteratorPattern...............  AnonymousFunctions
             (used with HigherOrderFunctions,
              MapFunction, FilterFunction, etc.)
InterpreterPattern............  Macros (extending the language)
              EvalFunction, MetaCircularInterpreter
              Support for parser generation (for differing syntax)
CommandPattern ...............  Closures, LexicalScope,
              AnonymousFunctions, FirstClassFunctions
HandleBodyPattern.............  Delegation, Macros, MetaClasses
RunAndReturnSuccessor.........  TailCallOptimization
Abstract-Factory,
Flyweight,
Factory-Method,
State, Proxy,
Chain-of-Responsibility.......  FirstClass types (Norvig)
Mediator, Observer............  Method combination (Norvig)
BuilderPattern................  Multi Methods (Norvig)
FacadePattern.................  Modules (Norvig)
StrategyPattern...............  higher order functions (Gene Michael
Stover?), ControlTable
AssociationList...............Dictionaries, maps, HashTables
                  (these go by numerous names in different languages)
```

wiki.c2.com/?AreDesignPatternsMissingLanguageFeatures

Tá, mas o que é um padrão de projeto?

O que são padrões de projeto?

"Os padrões de projeto são **soluções típicas** para **problemas comuns** no design de software."

Eles são como modelos pré-fabricados que você pode personalizar para resolver um problema de design recorrente no seu código.

https://refactoring.guru/design-patterns/what-is-pattern

Classificação dos patterns clássicos

Os patterns do livro clássico são divididos em 3 partes grupos:

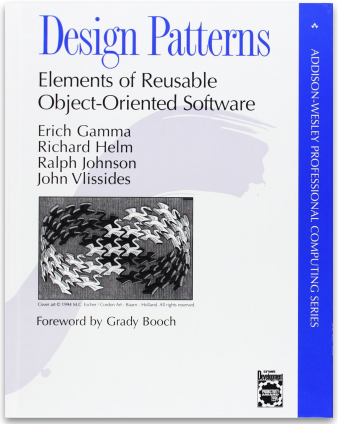
- Criação [creational]
  - fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização do código existente
- Estruturais [structural]
  - explicam como montar objetos e classes em estruturas maiores, mantendo as estruturas flexíveis e eficientes.
- Comportamentais [behavioral]
  - cuidam de uma comunicação eficaz e da atribuição de responsabilidades entre objetos.

Qual a cara de um Padrão?

Um padrão é composto por quatro elementos essenciais:

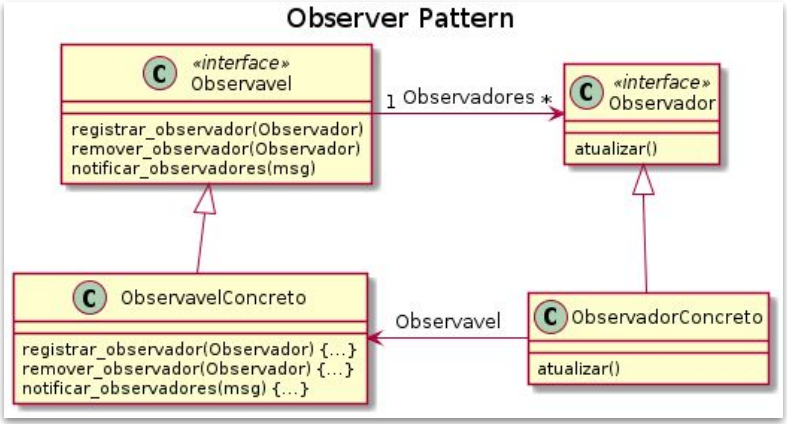
- Intenção
  - Descrição breve do problema e sua solução
- Motivação
  - Explica melhor o problema e as possibilidades do padrão
- Estrutura
  - Mostra as classes e como se relacionam (UML)
- Exemplo de código
  - Geralmente em alguma linguagem popular

Padrão Observer [Observador]



pg 275

- Intenção
  - Permitir que um objeto seja capaz de notificar outro objeto
- Motivação
  - define uma dependência um-para-muitos entre objetos para que, quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente



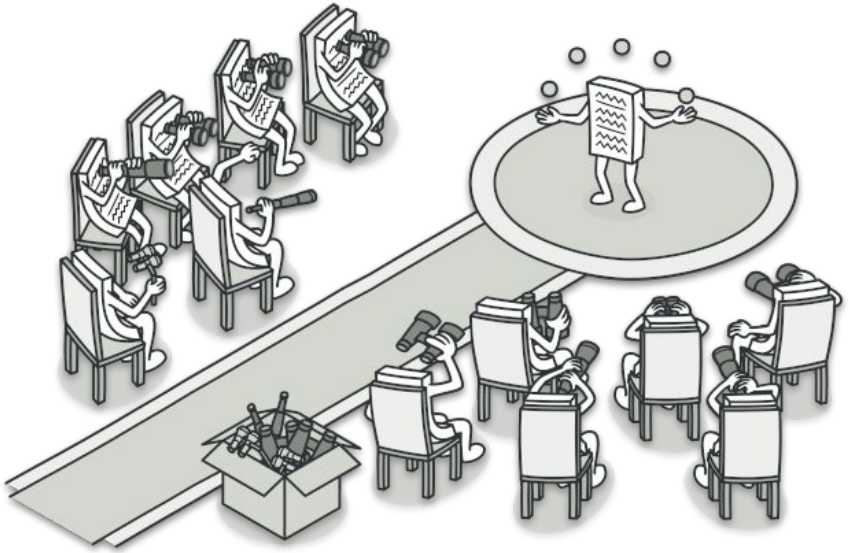
Live de Python

Padrão Observer [Observador]

```python
class Observavel:
    def __init__(self):
        self._observers = []

    def registrar_observador(self, observer):
        self._observers.append(observer)

    def remover_observador(self, observer):
        self._observers.remove(observer)

    def notificar_observadores(self, data):
        for observer in self._observers:
            observer.atualizar(self, data)
```

```python
class Observador:
    def atualizar(self, data):
        ...
```

Padrão Observer [Observador]



pg 336

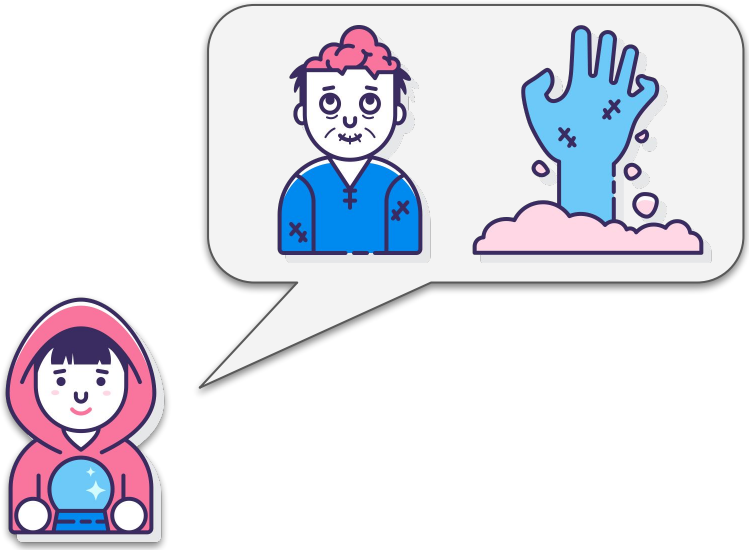
Live de Python

Mostrando o poder dos padrões

(uma historinha legal)

baseada em **Peter Ullrich**

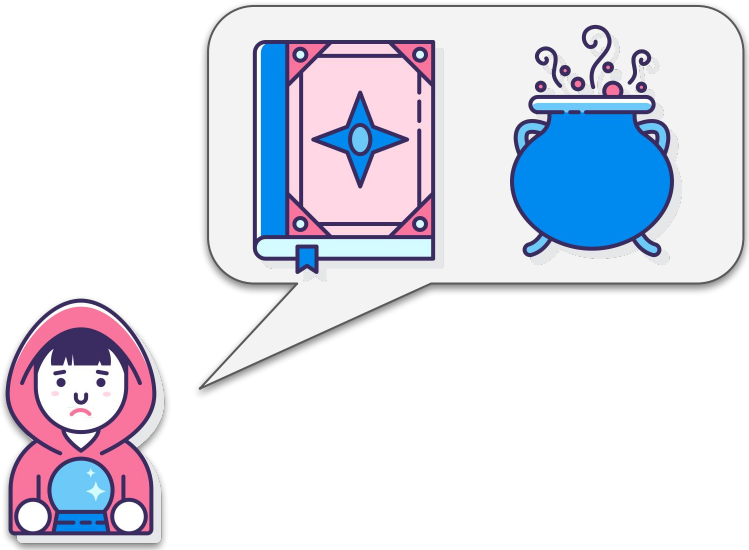
Live de Python



Fausto

Fausto viu em sua bola de cristal que nas próximas semanas os mortos voltariam a vida.

Como gerente de comunicação dos seres mágicos precisava avisar a todos o que estava prestes a acontecer.



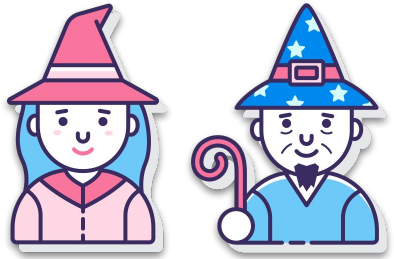
Fausto

Foi nesse momento que ele se lembrou que já fazia um certo tempo que não conversava com seres mágicos e que precisaria passar horas estudando o idioma e a magia de todos os seres para se comunicar

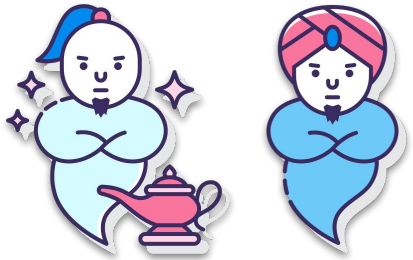
????

Fausto

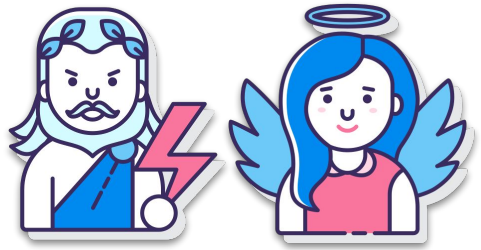
Bruxos



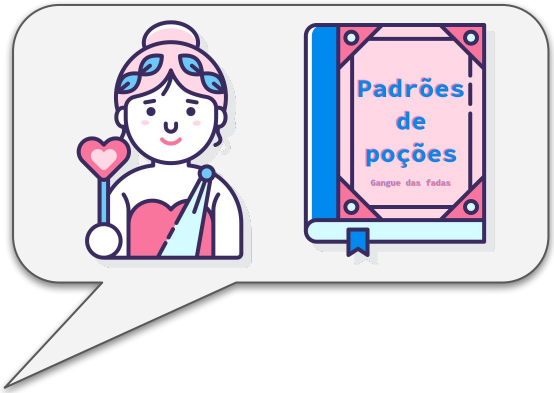
Gênios



Deuses

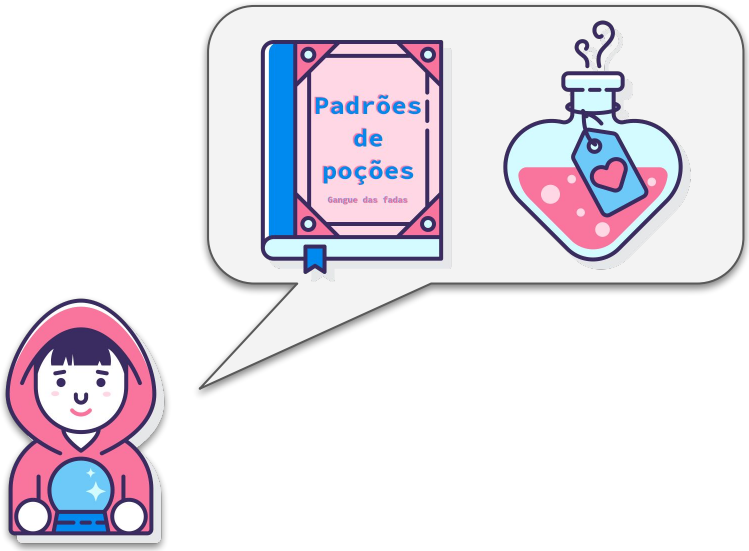


Live de Python



Fausto

De repente ele se lembrou de Bárbara, sua professora de engenharia de poções. Que tinha lhe mostrado o livro “Padrões de poções”



Fausto

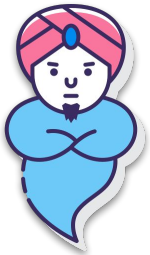
Nesse livro existia uma poção chamada “Adaptador de Linguagem”

Fausto

Então ele jogou no seu caldeirão tudo que queria dizer e fez uma adaptação

Por motivos de efeitos especiais, vamos ter que usar código

Classes de seres e seus idiomas



```
class Bruxo:
    def bruxeis(self):
        print('The dead are coming back to life')


class Genio:
    def genieis(self):
        print('Los muertos vuelven a la vida')


class Deus:
    def deuseis(self):
        print('De doden komen weer tot leven')
```

Live de Python

Solução com tradução direta (sem adaptador)

```
seres = [Bruxo(), Genio(), Deus()]

for ser in seres:
    if isinstance(ser, Bruxo):
        ser.bruxes()
    elif isinstance(ser, Genio):
        ser.genies()
    else:
        ser.deuses()
```

Solução com tradução direta (sem adaptador)

Olha antes para quem vai falar

```
seres = [Bruxo(), Genio(), Deus()]

for ser in seres:
    if isinstance(ser, Bruxo):
        ser.bruxes()
    elif isinstance(ser, Genio):
        ser.genies()
    else:
        ser.deuses()
```

Adaptador de fala

```
class BruxoAdapter:
    def __init__(self, bruxo):
        self.bruxo = bruxo

    def falar(self):
        self.bruxo.bruxeis()
```

```
class GenioAdapter:
    def __init__(self, genio):
        self.genio = genio

    def falar(self):
        self.genio.genieis()
```

```
class DeusAdapter:
    def __init__(self, deus):
        self.deus = deus

    def falar(self):
        self.deus.deuseis()
```

Live de Python

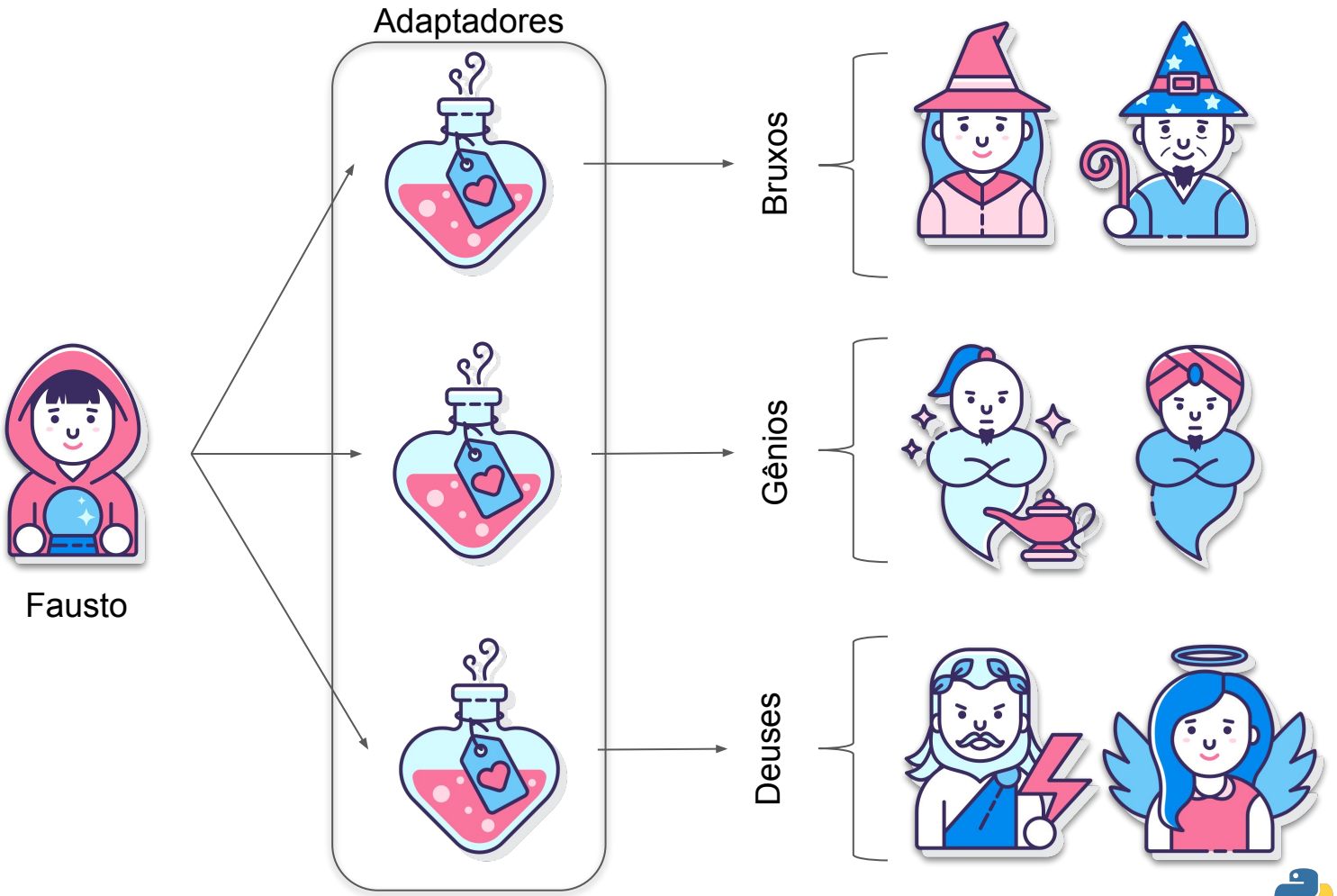
Chamando os adaptadores

```
seres = [
    BruxoAdapter(Bruxo()),
    GenioAdapter(Genio()),
    DeusAdapter(Deus())
]

for ser in seres:
    ser.falar()
```

Adaptador único

```python
class FalaAdapter:

    def __init__(self, ser, *, falar):
        self.ser = ser
        metodo_de_fala = getattr(self.ser, falar)
        self.__setattr__('falar', metodo_de_fala)
```

```python
seres_adapters = [
    FalaAdapter(Bruxo(), falar='bruxeis'),
    FalaAdapter(Genio(), falar='genieis'),
    FalaAdapter(Deus(), falar='deuseis')
]

for adapter in seres_adapters:
    adapter.falar()
```



Obrigado!

apoia.se/livedepython

```
dunossauro at bouman in ~/git/apoiase on master*
$ python apoiadores.py
Alexandre Possebom   Alysson Oliveira     Amaziles Carvalho    Andre Machado
Andre Rodrigues      Bernardo Fontes      Bruno Guizi          Carlos Augusto
Cleber Santos        Cleiton Mittmann     David Reis           Dayham Soares
Diego Ubirajara      Edimar Fardim        Eliabe Silva         Eliakim Morais
Elias Soares         Emerson Lara         Eugenio Mazzini      Fabiano Silos
Fabiano Teichmann    Fabiano Gomes        Fernando Furtado     Franklin Silva
Fábio Serrão         Gleison Oliveira     Guilherme Ramos      Hemilio Lauro
Humberto Rocha       Hélio Neto           JOAO COELHO          JONATHAN DOMINGOS
Jean Vetorello       Johnny Tardin        Jonatas Oliveira     Jonatas Simões
João Lugão           Jucélio Silva        Júlia Kastrup        Leon Teixeira
Lucas Nascimento     Magno Malkut         Marcus Salgues       Maria Boladona
Matheus Francisco    Nilo Pereira         Pablo Henrique       Paulo Tadei
Pedro Alves          Rafael Galleani      Regis Santos         Renan Moura
Renato Santos        Rennan Almeida       Rodrigo Ferreira     Rodrigo Vaccari
Sérgio Passos        Thiago Araujo        Tiago Cordeiro       Tyrone Damasceno
Vergil Valverde      Vicente Marcal       Wander Silva         Welington Carlos
Wellington Camargo   Welton Souza         William Oliveira     Willian Gl
Yros Aguiar          Falta você           Falta você           Falta você
```