

# Linguagem de Programação Comercial 2020/1

FÁBIO CASTRO

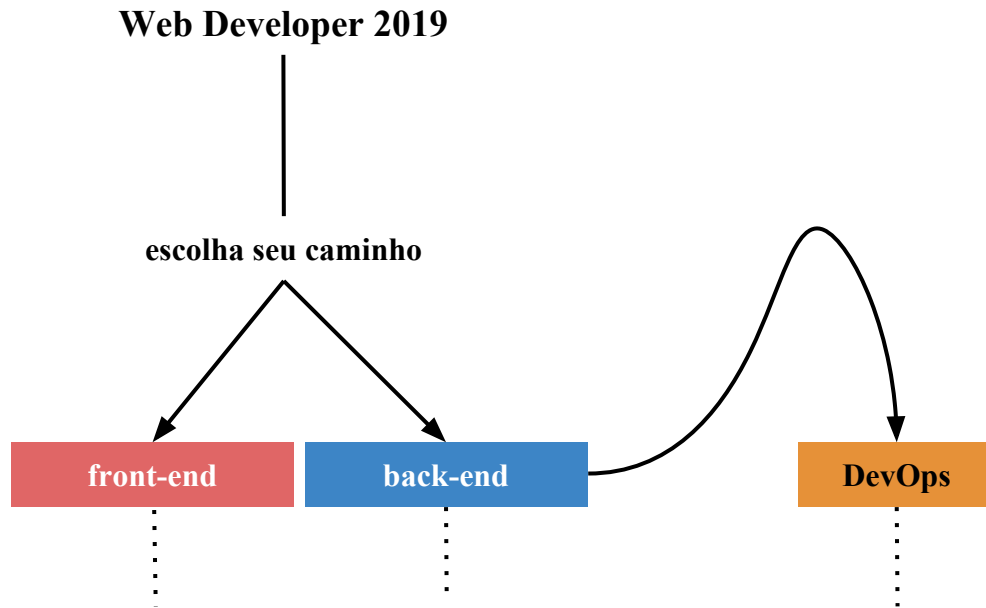
# Programa

---

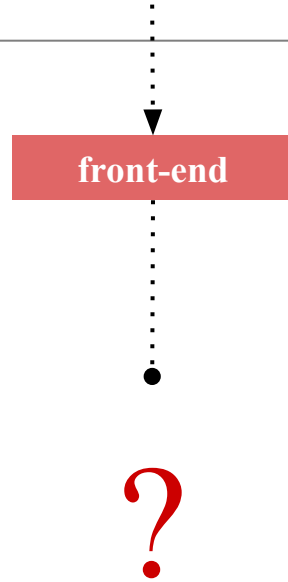
- Fundamentos de Programação Orientada a Objetos em Python
- Controle de versões e Git
- Arquitetura de uma aplicação Web
- Framework Django
- Arquitetura Model View Controller (MVC)
- Projeto e Aplicativo Django: estrutura e comandos
- ORM do Django e Migrations
- URLs e rotas
- Views: funções e views tipadas
- Django Admin
- Conceitos de API HTTP REST
- Implementação de API HTTP REST com Django REST Framework

# Web Developer 2019 - “Roadmap”

POO
Git - controle de versão
Utilização do “Terminal”
SSH
HTTP / HTTPS e APIs
Estrutura de dados
Engenharia de SW
Design Patterns
GitHub / Gitlab / bitbucket

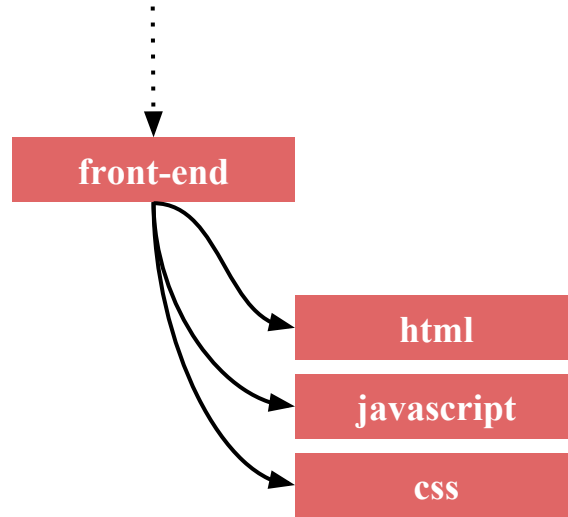


# front-end

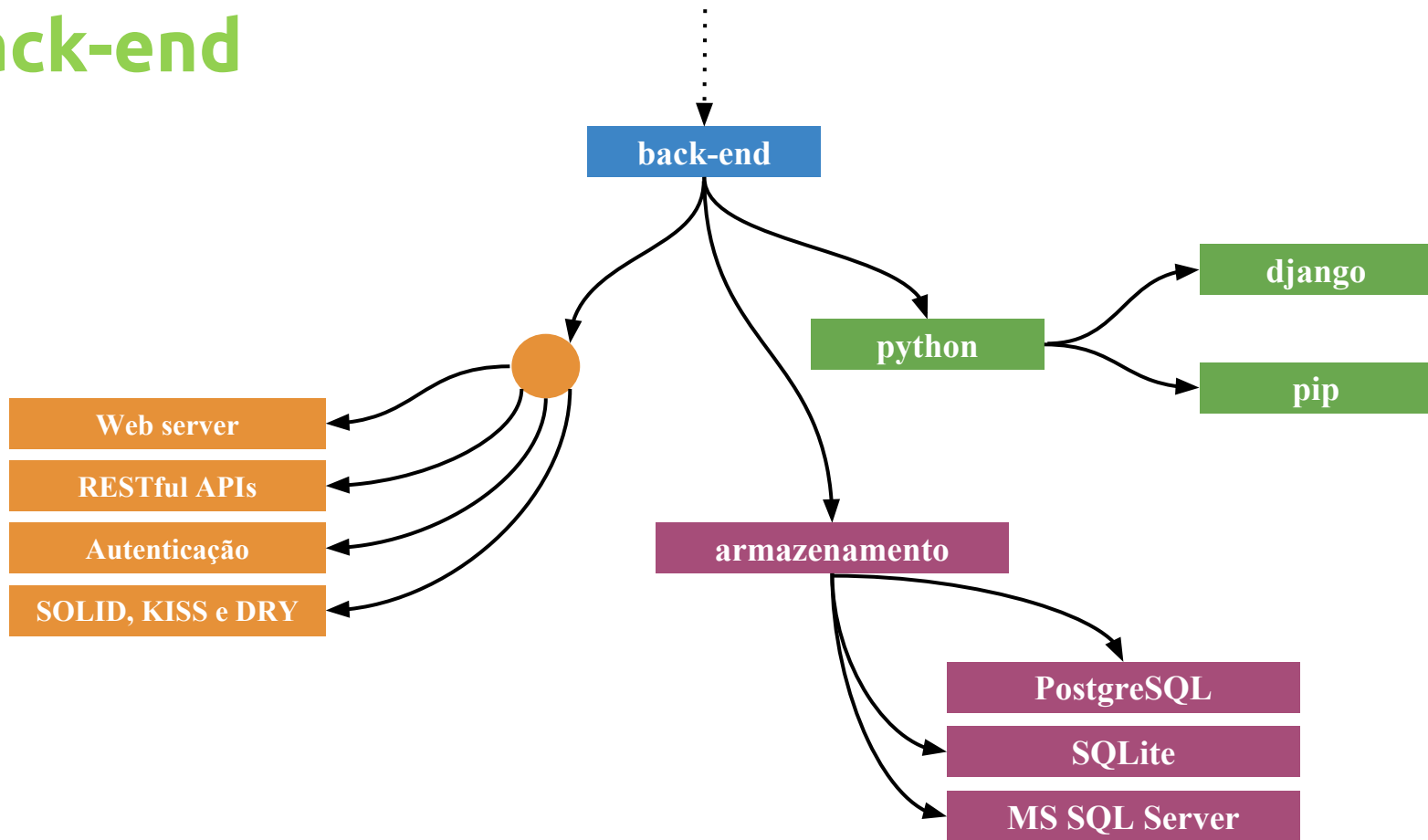


# front-end

---



# back-end



# Git e Github

Guia básico de utilização

# Primeiramente VCS

## Sistema de controle de versões

**Software que gerencia diferentes versões de um documento qualquer.**





---

# Vantagens do controle de versões





- **Criado em 2005 por Linus Torvalds;**
- **Utilizado para o projeto de kernel linux;**
- **VCS mais utilizado;**

# Vocabulário básico

---

**Repositório:** um lugar para armazenar coisas. Com o git, isso significa sua pasta de código;

**head:** um “ponteiro” (apontamento) para o código mais recente em que você estava trabalhando;

**add:** uma ação para pedir ao git para rastrear um arquivo;

**commit:** Uma ação para salvar o estado atual - de modo que alguém possa revisitar esse estado se necessário;

**remote:** um repositório que não é local. Pode estar em outra pasta ou na nuvem;

# Vocabulário básico

---

**pull**: uma ação para obter o código atualizado do repositório remoto;

**push**: uma ação para enviar código atualizado para o repositório remoto;

**merge**: uma ação para combinar duas versões diferentes de código;

**status**: exibe informações sobre o status atual do repositório;

# Antes de começar...

```
git config --global user.name "Nome"
```

```
git config --global user.email "seu_email"
```

# criando um novo repositório

crie uma nova pasta, abra-a no terminal e execute o comando

```
git init .
```

para criar um novo repositório.

# Fluxo de trabalho

Seus repositórios locais consistem em três "árvores" mantidas pelo git. A primeira delas é sua **Working Directory** que contém os arquivos vigentes.

A segunda **Index** que funciona como uma área temporária.

Finalmente a **HEAD** que aponta para o último commit (**confirmação**) que você fez.

# adicionar & confirmar

Você pode propor mudanças (adicioná-las ao Index) usando

```
git add <nome_do_arquivo>
```

Este é o primeiro passo no fluxo de trabalho básico do git. Para realmente confirmar estas mudanças (isto é, fazer um commit), use

```
git commit -m "comentários das alterações"
```

Agora o arquivo é enviado para o HEAD, mas ainda não para o repositório remoto.



# Na prática...



# Github

- **Utiliza o controle de versionamento Git;**
- **Serviço web de hospedagem de projetos;**
- **Rede Social para desenvolvedores;**





**git**



**github**  
SOCIAL CODING

# Concorrentes..



GitLab

# **Criando um perfil no Github**

**<http://github.com>**

---


# Associando uma chave...

- `ssh-keygen -t rsa -C "email\_cadastrado\_no\_github@email.com"`
  - (informe uma senha)
  - `notepad ~/.ssh/id_rsa.pub`
  - Teste a conexão (`ssh -T git@github.com`)
-

# Programação Orientada a Objetos

---

“é um paradigma de programação que fornece um meio de estruturar programas para que propriedades e comportamentos sejam agrupados em objetos individuais.”





---

Por exemplo, um objeto pode representar uma pessoa com uma propriedade de nome, idade, endereço, etc., com comportamentos como caminhar, falar, respirar e correr.

---

# Primeiramente...

Abstração	Encapsulamento	Herança	Polimorfismo
Capacidade de representação de entidades do mundo real;	Proteção de dados e operações internas;	Permite estender características;	Alterar a forma de acordo com a necessidade;

---

# Classes em python

---



As classes são usadas para criar novas estruturas de dados definidas pelo usuário que contêm informações arbitrárias sobre algo.



É importante notar que uma classe apenas fornece estrutura - é um modelo de como algo deve ser definido, mas na verdade não fornece nenhum conteúdo real em si.

# Objetos Python (instâncias)

---

Enquanto a classe é a estrutura, uma instância é uma cópia da classe com valores reais, literalmente, um objeto pertencente a uma classe específica.

# Abstração

```
1 class Pessoa():
2
3     def __init__(self, nome, idade):
4         self.nome = nome
5
6     def __str__(self):
7         return self.nome
8
```

`__init__` é o construtor, ou melhor, o inicializador

`self` é o 1º parâmetro formal em todos os métodos de instância

# Encapsulamento

```
1 class Pessoa():
2     nome = ''
3     data_nascimento = None
4
5     def __init__(self, nome, data_nascimento):
6         self.nome = nome
7         self.data_nascimento = data_nascimento
8
9     def __str__(self):
10         return self.nome
11
```

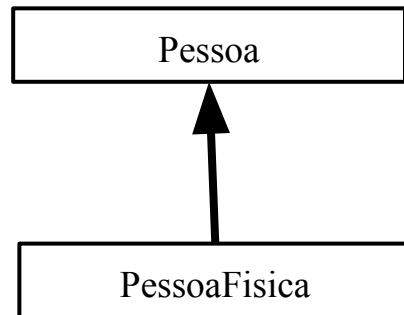
atributos de dados na classe funcionam como valores default para os atributos das instâncias

atributos da instância só podem ser acessados via self

# Herança

```
1 class Pessoa():
2     nome = ''
3     data_nascimento = None
4
5     def __init__(self, nome, data_nascimento):
6         self.nome = nome
7         self.data_nascimento = data_nascimento
8
9     def __str__(self):
10         return self.nome
11
12 class PessoaFisica(Pessoa):
13
14     def __init__(self, cpf):
15         self.cpf = cpf
16
17     def __str__(self):
18         return u'{} - {}'.format(self.cpf, self.nome)
19
```

UML  
Diagrama de Classe

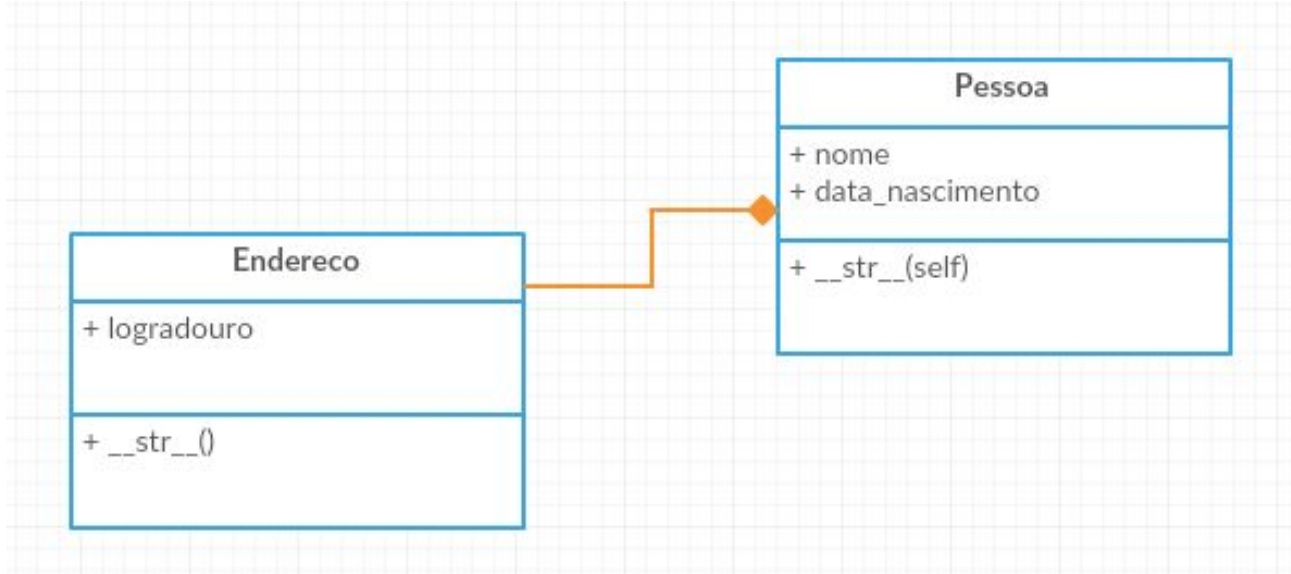


# Polimorfismo

```
1 class Pessoa():
2     nome = ''
3     data_nascimento = None
4
5     def __init__(self, nome, data_nascimento):
6         self.nome = nome
7         self.data_nascimento = data_nascimento
8
9     def __str__(self):
10         return self.nome
11
12 class PessoaFisica(Pessoa):
13
14     def __init__(self, cpf, nome, data_nascimento):
15         Pessoa.__init__(self, nome, data_nascimento)
16         self.cpf = cpf
17
18 class PessoaJuridica(Pessoa):
19
20     def __init__(self, cpf, nome, data_nascimento):
21         Pessoa.__init__(self, nome, data_nascimento)
22         self.CNPJ = CNPJ
```



# e a Composição?



# Composição[...]

```
1 class Endereco(object):
2     logradouro = None
3     def __init__(self, logradouro):
4         self.logradouro = logradouro
5
6 class Pessoa(object):
7     nome = ''
8     data_nascimento = None
9
10    def __init__(self, nome, data_nascimento):
11        self.nome = nome
12        self.data_nascimento = data_nascimento
13        self.endereco = Endereco('rua 1')
14
15    def __str__(self):
16        return self.nome
```

---

# Atividade!

