

Tem Uma Cobra Na Minha Bota !

Criando um Protótipo com Python

print("Vamos lá! Vem com o Py!")





Universidade
de Fortaleza

Este e-Book foi idealizado por:

Leandro Soares Coelho
Lana Praça Oliveira
Eduardo Dourado Sales de Oliveira
Éderson Façanha Souza
Jully Emerson Ribeiro Costa

Prof. Me. M. Miqueias Mala

Não tenha medo de errar, um acerto é o reflexo de
vários erros, várias tentativas, até acertar! Nisso, o
aprendizado vai sendo forjado.

Codar é sobre errar, até aprender, e errar achando que
já aprendeu. Faz parte, não tema o erro, já que dos
professores, ele é um dos melhores!

"Você pode encarar um erro como
uma besteira a ser esquecida ou
como um resultado que aponta
uma nova direção."

- Steve Jobs

Sumário

1. Bem Vindo ao Tutorial	4
1.1 Olá, Sai do WhatsApp e Pega um Café Que Vamos Começar!	4
1.2 O Que Vamos Construir?	4
1.2.1 E o que esse Projeto vai fazer?	5
1.2.2 Onde Vamos Usar Isso Aí?	5
1.3 Entendi, Mas Por Que?	5
1.4 E Para Isso Vamos Precisar... ..	6
2. Python	6
2.1 Origem e por que "Python"?	6
2.2 Features do Python	7
2.3 Cenários de Uso	8
2.4 Um canivete suíço cheio de features	9
2.4.1 Frameworks	10
2.4.2 Bibliotecas / Libs	11
2.5 Python no Projeto!	12
3. Todas as Tecnologias do Nosso Projetinho	13
3.1 IDE (Visual Studio Code)	13
3.1.1 Extensões: Um mundo de Customização	14
3.2 HTML	15
3.3 CSS	16
3.4 Datetime	17
3.5 Dictionary	18
3.6 Tudo Junto e Misturado	19
4. Sistema de Estacionamento: saindo do papel	19
4.1 Código Fonte em Python	20
5. Versionamento	24
5.1 O Que é Isso?	24
5.2 Pra Que Serve?	24
5.3 Comandos Básicos	25
5.4 Branches e Gitflow	26
5.5 NÃO! Git NÃO é Github	28
5.5.1 Git	28
5.5.2 Github	29
6. Muito Obrigado!	30
Extras!	31

1. BEM VINDO AO TUTORIAL!

1.1 Olá, Sai do WhatsApp e Pega um Café Que Vamos Começar!

Seja muito bem-vindo(a) ao nosso livro!

Ajeita essa postura, eu sei que você está aí que nem um camarão nessa cadeira. A ideia de criar este eBook veio do desejo de juntar a prática de codar, aprender passo a passo e misturar tudo isso em algo real, palpável, saindo de fazer somas no terminal do seu editor de código favorito (ou aquele que seu professor(a) estiver usando!) para algo que conseguimos ver ou imaginar acontecendo no dia a dia. Aqui, eu confesso que vamos ver essa integração de uma maneira um tanto superficial, mas você já consegue imaginar o que dá pra fazer de fato. Mais do que ensinar, quero mostrar isso: potencial.

1.2 O Que Vamos Construir?

Bom, quero te mostrar aqui o que fazer com tanto código e tanta dor de cabeça por causa de uma vírgula mal colocada (e nem mesmo colocada, acontece, relaxa). Vamos criar um sistema de estacionamento!

Imagine um estacionamento, o vai e vem de carros, precisamos identificar estes carros, guardar quanto tempo eles ficaram estacionados, até para que então possamos cobrar esse povo (de graça nem água é mais! Cocem os bolsos aí!).

Antes, o pessoal fazia isso com um caderno, uma caneta e muito foco; obviamente, isso abria margem para muito erro e mesmo questionamentos por parte dos clientes.



1.2.1 E o que esse Projeto vai fazer?

Nosso sistema de gestão, controle e cobrança de veículos estacionados (ou só Sistema de Estacionamento, por praticidade, né?!) vai ter as seguintes funções... Ah! Normalmente, aqui na T.I., falamos “features”. Chique, né?! Já até damos um passinho no inglês pro LinkedIn! Então, teremos as seguintes features:

- Registro de Entrada e Saída de veículos;
- Identificação do veículo por Placa;
- Identificação do Estado de origem do veículo;
- Valor a Pagar mediante período estacionado;
- Isenção de cobrança mediante permanência mínima.



1.2.2 Onde Vamos Usar Isso Aí?

Como eu disse lá em cima, é um sistema de estacionamento, então dá pra usar em quase todo tipo de estacionamento, seja ele grande ou pequeno.

Porém, acredito que nosso tímido projeto brilhará mais caso seja implementado em estacionamentos de pequeno e médio porte (porque aqueles estacionamentos enormes de shoppings e aeroportos fazem até automatização, pagamentos integrados... calma, gente, ainda estamos saindo do “Olá, Mundo!”).

1.3 Entendi, Mas Por Que?

Programar, ser programador, é principalmente sobre resolver problemas, gente — sanar as dores que nossos usuários estejam tendo. Uma coisa que acredito ser importante para cada um de nós é, além de entender os códigos, passar por cada loop, fazer a indentação correta e tudo o mais, termos também essa percepção de entender a dor do outro. E, com essa compreensão, vamos criar uma solução (até porque, toda solução de um problema normalmente é paga — e muito bem paga. Abrete, carteira!).

No caso, vamos criar um sistema de estacionamento simplificado para ajudar pequenos negócios a gerenciarem melhor seu espaço.



1.4 E Para Isso Vamos Precisar...

Certo, falei o que vamos fazer, falei do motivo... até aí, tudo bem. Agora, pra fazer esse sonho dar certo, vamos usar:

- Python & Cia 🐍

É o carro-chefe do nosso projeto. Com Python, a gente consegue fazer desde a armazenagem dos carros, o tempo que eles ficaram estacionados e as contas do valor a pagar referente ao tempo. Aqui, dizemos que nossa lógica no back-end foi codada em Python.

Python tem algumas features dele, que chamamos de bibliotecas, ou libraries, ou somente lib, pros íntimos. Vamos usar, inclusive, algumas libs do Python em nosso projeto — calma que ainda vamos ver sobre elas!

- HTML e CSS 🌐

Usados para criar nossa interface de usuário. Normalmente, chamamos essa parte de GUI (Graphical User Interface), que é onde nosso usuário vai interagir com o código do sistema (aquele lá em Python).

Lembra dos conceitos básicos de Web Design? O HTML dá a estrutura/corpinho e o CSS, o estilo do mesmo. “Nunca um sem o outro”.

2. Python

2.1 Origem e por que “Python”?

Então, Python foi criada lá em 1991 (pra você que achava que era uma linguagem recente, não é! É mais velha inclusive que o Java, que nasceu em 95), por um programador holandês chamado Guido van Rossum.


Nosso Guido aí queria uma linguagem que fosse fácil de ler e escrever, altamente produtiva (dá pra fazer um monte de coisas!). A ideia do Guido é que Python fosse possível encaixar desde automação à desenvolvimento de sites (nem só de Javascript vive a internet).



Apesar de o nome e até mesmo o símbolo do nosso Python serem associados a uma cobra, na verdade o Guido era muito fã de um grupo chamado "Monty Python's Flying Circus" e acabou escolhendo "Python" porque queria algo que fosse curto, único e um tanto divertido, já que o grupo, do qual Guido era (e ainda é!) fã, era um grupo britânico de comédia, e as linguagens da época tinham nomes um tanto sérios.

2.2 Features do Python

Aqui eu vou te passar algumas características do Python, a ficha técnica mesmo, pra mostrar por que escolhemos essa linguagem pro nosso projeto!

- Linguagem de Alto Nível 

Significa que a escrita mais se aproxima da linguagem humana, abstraindo os detalhes que o hardware dá. Isso quer dizer que mexer com bits (0 e 1), fazer manipulação direta na memória, especificar registradores do processador e esses detalhes, o Python faz de forma automática (coisa que normalmente você faz com Assembly ou C as vezes).

- Linguagem Interpretada 

O código é executado linha por linha, direto do interpretador (no caso o interpretador de Python se chama CPython). É como se fosse uma tradução simultânea, você escreve o código e ele já é compilado. Isso deixa o código mais rápido para testar, desenvolver, debugar (corrigir erros).

- Tipagem Dinâmica 

As variáveis são definidas automaticamente na hora que você executa o código. Por exemplo, se você coda `print(5+4)`, vai ter como saída o número 9, porque o Python entende que você quer uma soma numérica sem precisar declarar nada.

```
python
x = 5      # Python entende que x é int
x = "Oi"   # Agora x vira uma string
```



- Tipagem Forte 

Só acontece interação entre tipos iguais de dados (String com String, int com int) se você quiser somar um número à uma String, deve fazer um processo que chamamos de casting. Então, Python literalmente não se mistura mesmo não.

O Python não mistura tipos automaticamente. Se quiser somar número com string, precisa converter antes.

```
x = "5"
y = 2
print(x + y) → ✗ erro
print(int(x) + y) → ✔
7
```

É como um garçom rígido: ele não mistura café com pimenta só porque você pediu "dois líquidos". Você tem que dizer exatamente o que quer.

- Multiparadigma 

É a capacidade do Python ser utilizado de várias formas, seja na programação procedural (faz isso, agora faz aquilo, retorna pra mim isso e aquilo junto), orientação à objetos (nosso querido POO) e o chamado paradigma funcional. Ou seja, Python é aquele canivete suíço, o que você precisa, ele tem!

Change
or
Adapt?



2.3 Cenários de Uso

Aqui, realmente vou dar só uma pincelada, senão esse livro teria mais de 100 páginas fácil.

Como toda linguagem que é multiparadigma, você pode usar Python para basicamente tudo — ou quase tudo. Aqui vou colocar uma tabela para a gente ver o potencial que podemos ter com Python.

NEXT PAGE

Área de Aplicação	Descrição	Exemplos Reais
Desenvolvimento Web	Criação de sites e APIs com frameworks como Flask e Django.	Sites dinâmicos, blogs, sistemas de login, REST APIs
Análise de Dados	Manipulação, visualização e análise estatística de dados.	Relatórios, gráficos, dashboards em empresas
Ciência de Dados / IA	Machine Learning, Deep Learning e inteligência artificial.	Reconhecimento facial, sistemas de recomendação
Automação de Tarefas	Scripts para automatizar processos repetitivos.	Renomear arquivos, enviar e-mails automáticos
Desenvolvimento de Jogos	Criação de jogos simples com bibliotecas específicas.	Jogos 2D com Pygame, protótipos educativos
Aplicações Desktop	Softwares com interface gráfica (GUI) para desktop.	Calculadoras, gerenciadores de tarefas, apps simples
Robótica e IoT	Controle de sensores e dispositivos embarcados.	Projetos com Raspberry Pi, automação residencial
Educação e Ensino	Linguagem simples para aprender lógica de programação.	Usado em escolas, universidades e cursos online
Cibersegurança	Scripts para testes de invasão e análise de vulnerabilidades.	Ferramentas como sniffers, brute-force, scanners
APIs e Integrações	Comunicação entre sistemas, criação de bots e webhooks.	Chatbots, integração com redes sociais e plataformas

2.4 Um canivete suíço cheio de features

Pela tabela anterior, tivemos um spoiler das ferramentas que o Python tem, né? Vão desde frameworks — que seriam algo como uma forma para um bolo: você põe os ingredientes (o framework chama o código e tudo que ele tem), mas é a forma que vai definir o resultado.



Já as bibliotecas (pode chamar de lib, eu deixo) são exatamente isso: um livro que você pega para usar quando precisa. Se você precisa de um livro, você o pega e, inclusive, só usa o capítulo que precisa desse livro:

python

```
import math  
print(math.sqrt(9)) # Só chamei o que quis: a raiz quadrada
```




2.4.1 Frameworks

Falando mais a fundo, um framework é toda uma estrutura de código pronta, em vez de você precisar criar toda uma estrutura pra algo, o framework já traz isso, como um prédio onde a fundação e base já estão feitas. Alguns frameworks que você já deve ter ouvido são:

- Django 

Facilita a criação de aplicações grandes, com muitos recursos prontos para você moldar como autenticação, administração e banco de dados. Sabe quem usa Django? Spotify, YouTube e Instagram!

- Flask 

Chamam esse aqui de microframework, porque ele tem menos coisas pré-prontas, é melhor para projetos menores ou API mais simples. Porém simplicidade não é sinônimo pra nome pequenos, Netflix, Airbnb e o nosso sempre positivo LinkedIn.

- PyTorch 

É bem usado para criar estruturas de pesquisa, banco de dados e até a tão falada Inteligência Artificial. Ah! Esse rapazinho foi criado pelo Facebook, mas também é usado pela Tesla (!!), Uber (para análise e machine learning) e também por um nome conhecido por nós gamers: NVIDIA (aqui o PyTorch é focado no Deep Learning).



2.4.2 Bibliotecas / Libs

Bom, por definição, é um conjunto de funções e recursos prontos que você importa e usa quando precisa, pra, desse jeito, você não ter que escrever tudo sempre ou criar algo do zero. Ela facilita teu trabalho.

Por exemplo: digamos que na sua aula de Lógica de Programação você precise criar um código para calcular a raiz quadrada de 25 (pois é, contas para exemplo são maravilhosas).

Sem a Lib, no seco mesmo:

```
numero = 25
resultado = numero ** 0.5
print(resultado) # Saída: 5.0
```

Agora, com a lib, lembrando sempre: é como um livro — você precisa pegar essa lib e dizer “É você, vem cá!”. Para isso, usamos sempre o import lá em cima do código. Olha só como muda com a biblioteca.

```
import math

resultado = math.sqrt(25)
print(resultado) # Saída: 5.0
```

Vamos encarar como um livro: você chama o livro math, mas usa somente um capítulo desse livro, que é o “sqrt”. Por isso, fazemos math.sqrt(25), sendo 25 o valor que vamos calcular.

Parece pouco? Bom, imagina aí um código com umas 900 linhas, vários processos diferentes. Algumas bibliotecas famosas que você já deve ter ouvido: pandas (tabelas, queridinho do Power BI), numpy (contas, contas e mais contas), requests (faz requisição HTTP, basicamente com isso você acessa — ou não — um site).



2.5 Python no Projeto!

Em nosso projeto do estacionamento, vamos usar Python por conta do seu lado multitarefa: para fazer cálculos, para usar na Web e para guardar e tratar dados. Além disso, a sintaxe do Python é realmente mais tranquila de ler, o que torna tudo mais fácil.

Em suma, nosso projetinho aqui terá Python por três grandes motivos, onde ele é muito bom: banco de dados, cálculos e web.



3. Todas as Tecnologias do Nosso Projéto

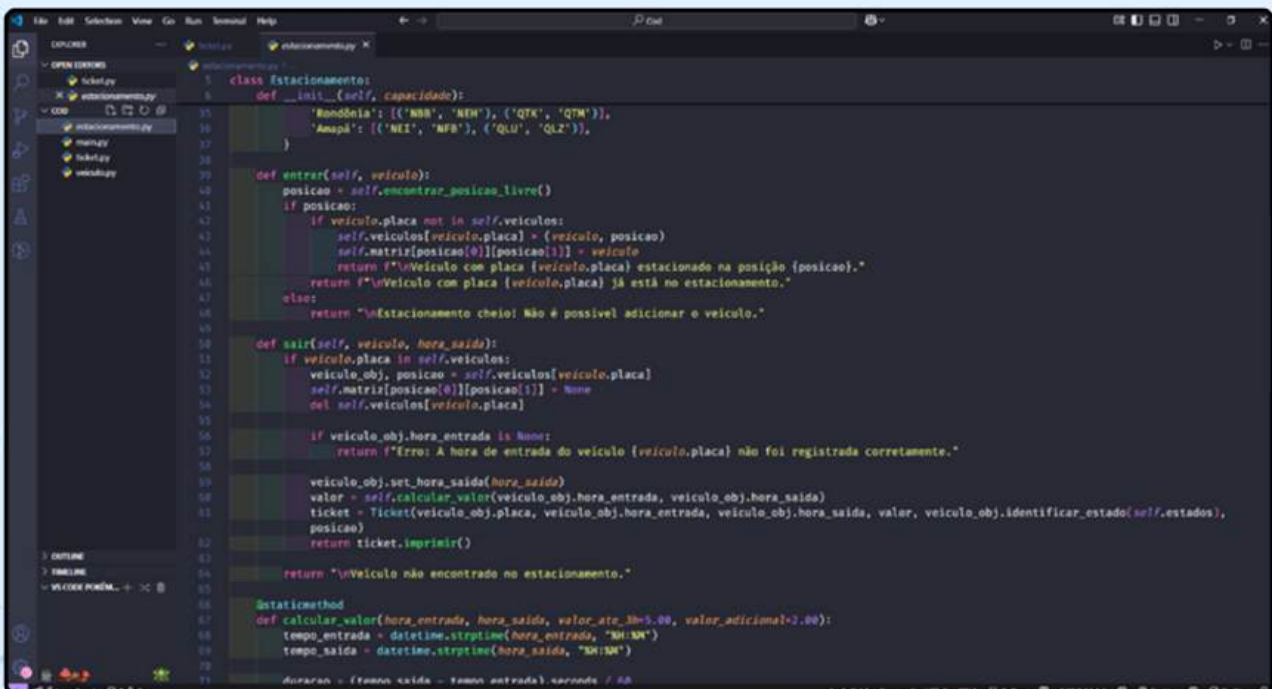
Aqui vou discorrer um pouco sobre o que vamos usar em nosso projeto do estacionamento, mas também são tecnologias que você pode — e muito provavelmente vai — usar, sabe? Pega aí outra caneca de café que lá vamos nós!

3.1 IDE (Visual Studio Code)

Vamos por definição: IDE significa Integrated Development Environment, em português: Ambiente de Desenvolvimento Integrado.

Ou seja, ele reúne tudo o que é necessário para o desenvolvimento (integra linguagens, frameworks, etc.) sob um editor de código. Imagine aqui um Bloco de Notas bastante turbinado.

O VS Code será nossa IDE de escolha, mas existem vários por aí, viu? Depende da linguagem que você quer desenvolver.



```
1 class Estacionamento:
2     def __init__(self, capacidade):
3         'Rodobela': [('NBB', 'NEH'), ('QTK', 'QTM')],
4         'Amopá': [('NEI', 'NFB'), ('QLU', 'QLZ')],
5     }
6
7     def entrar(self, veiculo):
8         posicao = self.encontrar_posicao_livre()
9         if posicao:
10             if veiculo.placa not in self.veiculos:
11                 self.veiculos[veiculo.placa] = (veiculo, posicao)
12                 self.matriz[posicao[0]][posicao[1]] = veiculo
13                 return f"Veículo com placa {veiculo.placa} estacionado na posição {posicao}."
14                 return f"Veículo com placa {veiculo.placa} já está no estacionamento."
15             else:
16                 return "Estacionamento cheio! Não é possível adicionar o veículo."
17
18     def sair(self, veiculo, hora_saida):
19         if veiculo.placa in self.veiculos:
20             veiculo_obj, posicao = self.veiculos[veiculo.placa]
21             self.matriz[posicao[0]][posicao[1]] = None
22             del self.veiculos[veiculo.placa]
23
24         if veiculo_obj.hora_entrada is None:
25             return f"Erro: A hora de entrada do veículo {veiculo.placa} não foi registrada corretamente."
26
27         veiculo_obj.set_hora_saida(hora_saida)
28         valor = self.calcular_valor(veiculo_obj.hora_entrada, veiculo_obj.hora_saida)
29         ticket = Ticket(veiculo_obj.placa, veiculo_obj.hora_entrada, veiculo_obj.hora_saida, valor, veiculo_obj.identificar_estado(self.estados),
30                         posicao)
31         return ticket.imprimir()
32
33     def encontrar_posicao_livre(self):
34         return "Veículo não encontrado no estacionamento."
35
36     @staticmethod
37     def calcular_valor(hora_entrada, hora_saida, valor_atc_3h=5.00, valor_adicional=2.00):
38         tempo_entrada = datetime.strptime(hora_entrada, "%H:%M")
39         tempo_saida = datetime.strptime(hora_saida, "%H:%M")
40         duracao = (tempo_saida - tempo_entrada).seconds / 60
```

O que mais marca no VS Code é a palavra “integrado”, sério mesmo. Ele reúne tudo e mais um pouco! Inclusive, existem várias coisas que você pode instalar nele para deixá-lo com a sua cara — desde coisas divertidas (como a extensão de Pokémon que uso ali no print, sim, é print do MEU VS Code) até ferramentas que melhoram sua produtividade. Vou dar umas sugestões a seguir:

3.1.1 Extensões: Um mundo de Customização

Prettier

É um formatador de código, sabe? Tipo um corretor ortográfico, só que melhor e sem te constranger trocando palavras. O Prettier formata quebras de linha, ponto, vírgula, aspas simples e duplas. Você pode até criar suas próprias regras de formatação.

ESLint

Assim como o Prettier, ele trabalha no teu código em si. Esse rapaz aqui detecta erros de código, más práticas e até códigos mortos (códigos que não servem para a aplicação — tua aplicação roda, mas não lê esses códigos). Normalmente, o ESLint é usado junto com o Prettier, aí você tem um código formatado e “saudável”.

Indent Rainbow

Um dos meus favoritos: essa extensão aqui coloca umas cores para cada nível de indentação do código, assim você consegue ver rapidinho onde cada coisa fica. Recomendo especialmente para quem gosta de Desenvolvimento Web, viu!

Live Server

Ele cria um servidor no teu computador mesmo e recarrega os arquivos HTML, CSS, JS (Javascript). Ele é legal porque no próprio VS Code você consegue ver o site que está fazendo, sem precisar ficar dando aquele F5 direto, você faz uma coisinha e puf! Aparece na hora!

Auto Rename Tag



Muito bom para quem mexe com HTML, sabe? Ele fecha as tags automaticamente e, caso você acabe renomeando alguma, já renomeia lá no fim também. É ótima para quem, às vezes, é meio esquecido.

Material Icon Theme



Certo, esse aqui, diferente dos outros, não age diretamente no teu código, mas deixa o teu VS Code mais bonito e organizado, com aquela sensação de faxina recente. Ele muda os ícones do teu VS Code (arquivos Java, Python, Ruby, Perl — gente, tem muito, viu!).

Dracula



Isso mesmo! Nosso morceguão aqui no VS Code é uma extensão que chamamos de skin. Ela muda as cores de todo o VS Code para uma paleta que é bem confortável para a gente poder codar por horas. Sério, depois de 5 horas olhando a tela, você vai agradecer. Inclusive, o print inicial é com a skin Dracula!



3.2 HTML

O nome completo do HTML é HyperText Markup Language, uma “língua” falada na internet. Nós a usamos para estruturar os conteúdos das páginas na web. Esse HyperText (HiperTexto) quer dizer que o texto contém links para outros textos, permitindo que você, cyber marujo, navegue para outras páginas, documentos, seções... Sabe aquele menu que você clica para ir para outra parte? O HyperText é isso. Repara só:

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>É aí eu sou um teste</title>
7 </head>
8 <body>
9   <h1>Olá Mundo! Aqui é o HTML!</h1>
10
11   <p>Deixa eu te mostrar um gatinho fofo</p>
12   <span>
13     <a href="https://hips.hearstapps.com/hmg-prod/images/sacred-birma-cat-in-interior-royalty-free-image-1718202055.jpg?crop=0.672xw:1.00xb;0.163xh,0resize=980:*"> É só clicar aqui!
14   </a>
15 </span>
16 </p>
17
18 </body>
19 </html>
```

Olá Mundo! Aqui é o HTML!

Deixa eu te mostrar um gatinho fofo [É só clicar aqui!](https://hips.hearstapps.com/hmg-prod/images/sacred-birma-cat-in-interior-royalty-free-image-1718202055.jpg?crop=0.672xw:1.00xb;0.163xh,0resize=980:*)

Viu como dentro do HTML tem um link ali para levar você a outro conteúdo? E sim, fiz de propósito para mostrar o Indent Rainbow em ação, pra você ver. Essas cores do código, juntamente com o fundo, são do nosso morceção Dracula!

Não, HTML não é uma linguagem de programação. Por quê? Bem, para ser uma linguagem de programação, é necessário que se possa criar uma lógica (aquele if/else, os laços de repetição...), além de manipular dados e executar instruções. O HTML não faz nada disso, tá? Ele apenas descreve como vai ser a estrutura do conteúdo, não decide nada. É como um pedreiro: ele levanta a parede, que fica lá, porém sem nada ainda...

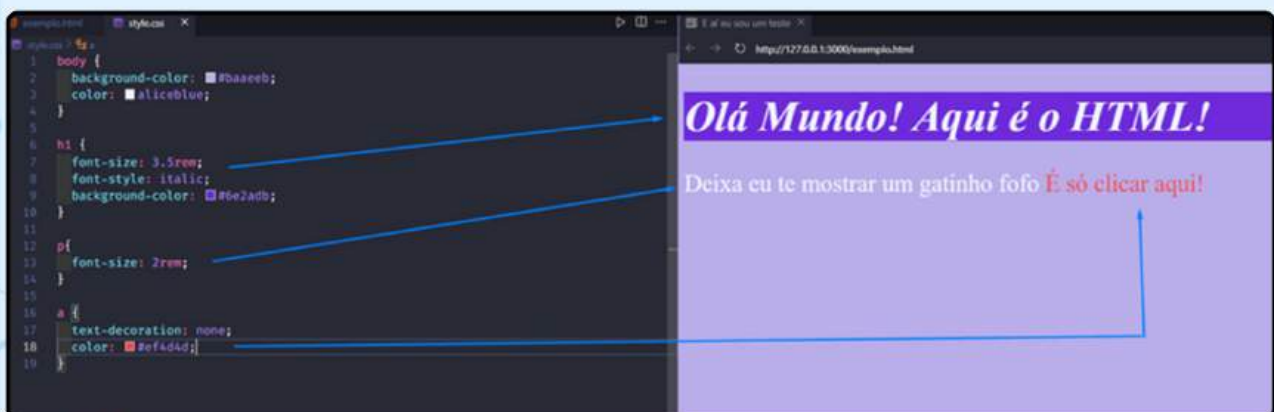
Nosso projeto precisa ser visto e acessado por quem está trabalhando no estacionamento, então precisamos organizar nosso código em HTML para que possa ser acessado facilmente.

3.3 CSS



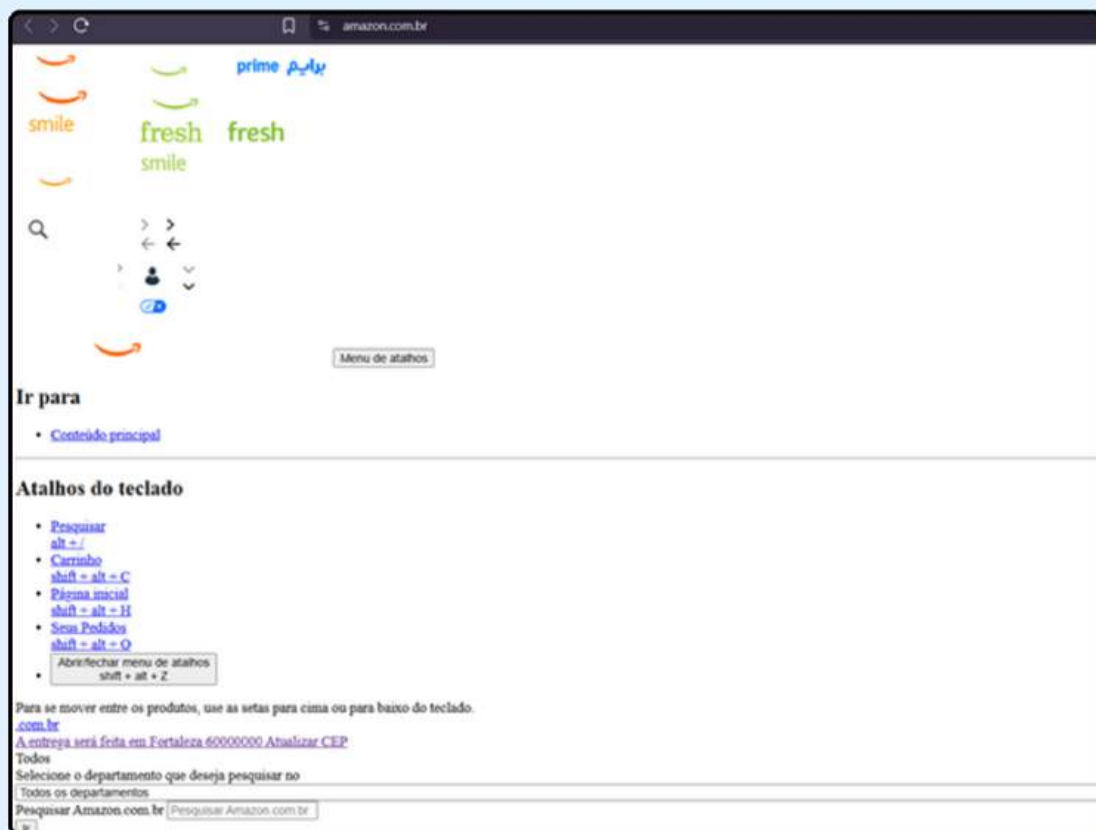
Assim como o HTML, o CSS também tem um nome completo: Cascading Style Sheets (Folhas de Estilo em Cascata). Esse rapaz aqui é usado para estilizar o que o HTML constrói. Se o HTML é o pedreiro, o CSS é o arquiteto.

O “Cascading” (Cascata) no nome vem porque o CSS aplica os estilos numa ordem de prioridade. Explicando melhor: se você quer aplicar uma cor para o fundo inteiro, o CSS faz isso. Mas, se você especificar que só um cantinho do fundo deve ter outra cor, o CSS vai obedecer essa especificação, deixando aquele cantinho com a cor diferente. Imagine uma cascata: quanto mais específico, mais prioridade o estilo tem.



Aqui vemos que o CSS aplica a cor #baaeb para todo o corpo (body) do nosso HTML, e a cor aliceblue para tudo que for escrito nele. Porém, nosso cabeçalho (<h1>) e nossos links (<a>) têm seus próprios estilos, que acabam tendo prioridade na cascata. Com o CSS, você pode estilizar e até reordenar alguns elementos do HTML.

Nosso projeto não pode ser só uma tela branca com botões jogados aleatoriamente, né? Ele precisa ser bonito visualmente e, ao mesmo tempo, fácil e intuitivo de usar. Pra isso, temos o CSS! E, sério, faz uma diferença imensa. Dá uma olhada em como fica um site sem CSS:



Olha como é o site da Amazon “de cara limpa”.



3.4 Datetime

Lembra quando falei das Bibliotecas (lib, somos íntimos agora)? Pois bem, Datetime é uma lib do Python que nos permite trabalhar com datas e horas. Com ela, conseguimos pegar a data e hora atuais, criar datas específicas, formatar essas datas do jeito que quisermos, somar ou subtrair períodos de tempo e até lidar com fusos horários!

Essa biblioteca será fundamental para nosso projeto, especialmente para controlar o tempo que cada carro fica estacionado e calcular o valor a pagar baseado nesse tempo.



Datetime - datas, horas, você controla tudoooo!

Em nosso projeto será crucial que saibamos o dia e a hora que os veículos entram e saem! Então, o Datetime é uma peça-chave para não flopar na hora de controlar o tempo de estacionamento e calcular corretamente o valor a pagar.

3.5 Dictionary



Aqui temos outra Lib do Python, o Dictionary, ou Dict mesmo, que faz bastante coisa, viu?! Ele é uma estrutura de dados, o que significa que organiza as informações usando algo chamado "Key". Imagine um guarda-roupas onde cada coisa sua tem uma chave única — assim fica muito mais fácil saber onde está cada coisa, né?



Disctionary - Tudo no seu lugar!

Em nosso sistema, o poderoso dict será chamado para adicionar os carros ao sistema, assim como para buscar e retirar os veículos.

```
python

estacionamento = {}

# Entrada de um veículo
placa = "ABC1234"
entrada = "10:30"
estacionamento[placa] = entrada
```

Aqui, o carro da placa ABC1234 entrou no estacionamento às 10:30.

3.6 Tudo Junto e Misturado

Vamos lá! Como um quebra-cabeça, vamos juntando todas as peças para formar aquela imagem que temos em mente: o sistema de estacionamento. Python, nossa estrela, será onde criaremos o código, com a ajuda das libs Datetime e Dictionary. Por fim, com nosso HTML e CSS, será criada uma interface bonita para o usuário poder interagir.

O quê? Terminal!? Para o usuário!? Vá pegar um café, vá!

4. Sistema de Estacionamento: saindo do papel

Certo, primeiramente precisamos criar nosso sistema, o famoso Código Fonte. Lembre-se: nosso código aqui é apenas uma demonstração, para mostrar a ideia geral.





4.1 Código Fonte em Python

1. Inicialmente fazemos a chamada das 2 libs: datetime e dictionary.
2. Criamos 2 variáveis constantes (que não mudam nem podem ser atribuídas a novos valores):
 - precoDaHora, um float com valor 12.00
 - registros, um dicionário que armazena a placa do veículo.
3. Em seguida, temos a função registrarEntrada, que recebe a placa do carro (a key), registra o horário usando nosso datetime.now() e armazena no dicionário registros os dados de entrada.
4. Temos aqui a função registrarSaida, onde verificamos se a placa informada está registrada no sistema (usando uma lógica if). Caso esteja, calculamos o tempo de permanência do veículo com base no horário atual e no horário de entrada registrado.
5. Convertendo o tempo de permanência em horas: Usamos a diferença entre o horário atual e o horário de entrada para calcular o tempo total. Esse tempo, originalmente em segundos, é então convertido para horas utilizando `.total_seconds() / 3600`.
6. Usamos o `math.ceil()`: Essa função do Python arredonda qualquer número decimal (float) para o inteiro imediatamente acima. Isso garante que mesmo que o veículo fique, por exemplo, 1 hora e meia, o sistema cobre 2 horas — resultando em R\$24,00 se o valor por hora for R\$12,00.



Agora que recebemos esse detonado, espero que ajude a entender o código abaixo, se não conseguir, relaxa, tá tudo bem.



Código Python Utilizado

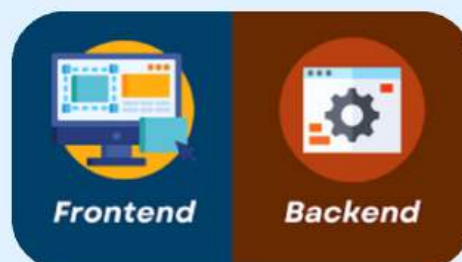
```
from datetime import datetime
import math

precoDaHora = 12.00
registros = {}

def registrarEntrada(placa):
    entrada = datetime.now()
    registros[placa] = {"entrada": entrada, "saida": None, "valor": None}
    print(f"Entrada: {placa} às {entrada.strftime('%H:%M:%S')}")

def registrarSaida(placa):
    if placa in registros:
        entrada = registros[placa]["entrada"]
        saida = datetime.now()
        tempo = saida - entrada
        horas = math.ceil(tempo.total_seconds() / 3600)
        valor = horas * precoDaHora
        registros[placa]["saida"] = saida
        registros[placa]["valor"] = valor
        print(f"Saída: {placa} às {saida.strftime('%H:%M:%S')}")
        print(f"Tempo: {horas} hora(s) | Valor: R$ {valor:.2f}")
        del registros[placa]
```

💡 Esse código em Python ficaria no nosso Back-end, já que a interface do usuário (HTML e CSS) não consegue processar código Python diretamente. Para isso, seria necessário usar um framework como o Django ou o Flask, o que deixaria todo esse processo de criação um pouco mais complexo.



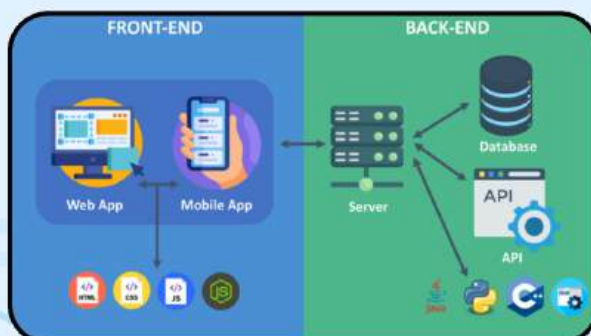
Nosso formulário teria como modelo, um HTML assim:

```
10 <body>
11 <div class="container">
12 <div class="python-logo"></div>
13 <h1>Nosso Estacionamento de Taubaté</h1>
14
15 <form>
16 <div class="field-group">
17 <label for="placa">Placa do carro:</label>
18 <input type="text" id="placa" name="placa" placeholder="ABC1234">
19 </div>
20
21 <div class="field-group">
22 <label for="entrada">Horário de entrada:</label>
23 <input type="text" id="entrada" name="entrada" placeholder="14:30">
24 </div>
25
26 <div class="field-group">
27 <label for="saida">Horário de saída:</label>
28 <input type="text" id="saida" name="saida" placeholder="16:45">
29 </div>
30
31 <div class="field-group">
32 <label for="valor">Valor calculado:</label>
33 <input type="text" id="valor" name="valor" placeholder="R$ 24,00" readonly>
34 </div>
35
36 <button type="button" disabled>Calcular</button>
37 </form>
38
39 <div class="note">
40 <p><em>Esse formulário é apenas uma visualização. Os dados seriam processados no código Python.</em></p>
41 </div>
42 </div>
43 </body>
```

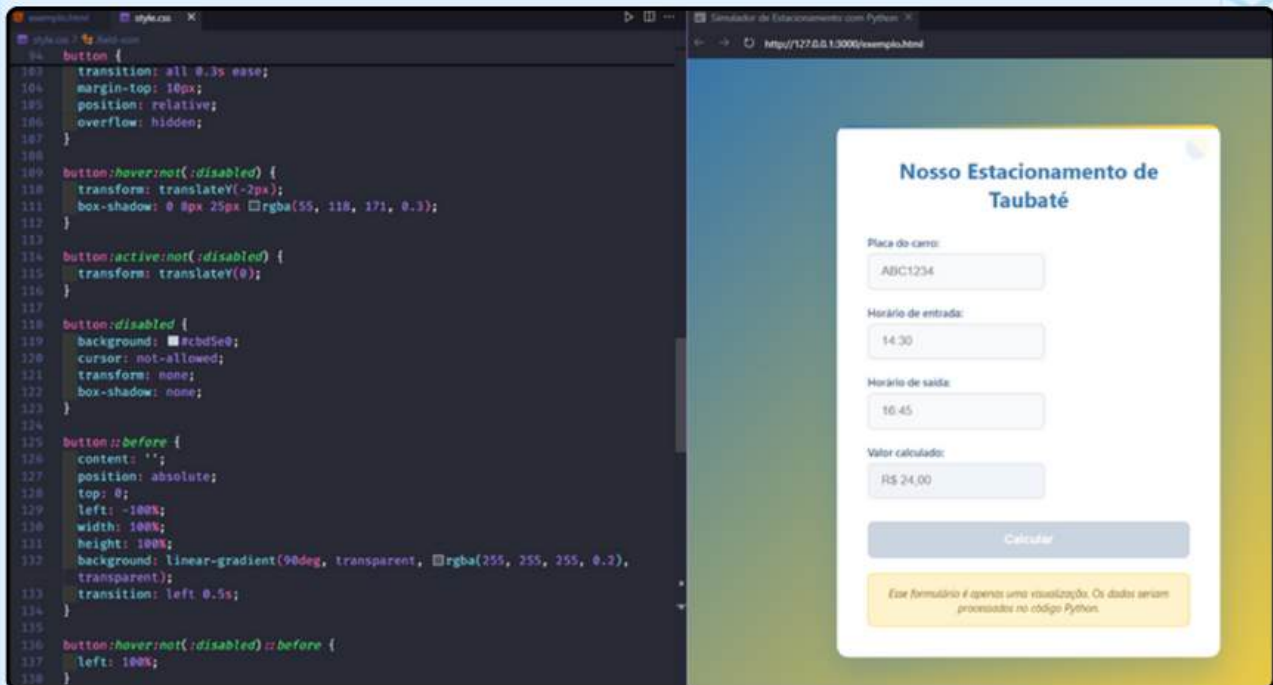
Na essência, temos aqui um formulário simples, com a placa do carro, entrada, saída e, finalmente, o preço que seria calculado. Estes dados seriam enviados ao nosso Back-end em Python, para que, através daquele código, ele nos devolvesse o valor a pagar!

Bom, vamos rever o que aconteceu até agora: nosso queridinho Python fica em nosso Back-end, que no caso seriam os “bastidores” do código, a parte que você vê, o famoso Front-end, é esse HTML, é o site que você acessa, põe as informações necessárias, o Front envia para o Back que então as guarda. Isso é uma comunicação comum entre Front e Back.

Certo, temos nosso Código Fonte, temos nosso <form> para colocar as informações, então:



Após um tapa no visual com um banho de loja CSS, nosso Formulário ficaria assim:



Nosso formulário ficou um luxo vai! 📈

Repare que o CSS não somente deixa o formulário bonito, como também organiza as informações. Isso faz com que tudo fique mais intuitivo pro usuário e até mais fácil de enxergar! Um bom visual ajuda muito na experiência de quem vai usar o sistema, então nada de negligenciar o estilo, viu?

Ufa! Mesmo sendo algo puramente demonstrativo, dá pra imaginar a trabalhadeira, não é?! Não é o tipo de coisa que se faz em 1 dia. Precisamos inclusive acompanhar e guardar todo esse código, especialmente se mais de uma pessoa estiver envolvida.

Então, para isso, usamos outra peça fundamental...



5. Versionamento

5.1 O Que é Isso?



O versionamento é um sistema que permite registrar o desenvolvimento da aplicação em que você está trabalhando.

Nele, você pode salvar todas as alterações feitas ao longo do tempo, como se fosse um "arquivo de salvamento" de um jogo, mas com a vantagem de mostrar exatamente o que foi alterado em cada etapa.

Imagine que você salva em outro slot, criando dois pontos de salvamento. Isso te dá segurança: se um der problema, você pode voltar ao outro.

Guarde essa referência, pois vamos usá-la daqui a pouquinho!

5.2 Pra Que Serve?

Versionar teu código dá um controle maior e organiza todas as mudanças que você e outras pessoas vão fazendo nesse projeto, exatamente como o save de um jogo.

E melhor: mantendo esse histórico de versionamento, você consegue fazer o seguinte:




- Voltar a uma versão anterior (um save anterior) se algo der ruim!
- Testar novas coisas, pois com vários slots você pode experimentar à vontade, sem perder o original (main).
- Trabalhar com muitas pessoas ao mesmo tempo, sem que uma sobreponha o trabalho da outra.
- Acompanhar o histórico e a evolução do projeto.
- Revisar mudanças específicas para permitir ou não que elas entrem no seu save (famosa PR ou Pull Request, que vamos ver em breve).



5.3 Comandos Básicos

Vou ser sincero aqui: existem mais de 150 comandos oficiais do Git e esse livro está ficando grandinho, então pedi para fazer uma tabela com os principais comandos. Sabendo esses, você já consegue fazer bastante coisa, viu!

Fluxo Inicial do Git e GitHub: Passo a Passo

Ordem	Comando Git	O que faz
1	<code>git config --global user.name "Seu Nome"</code>	Configura seu nome (só precisa fazer 1x por máquina)
2	<code>git config --global user.email "seu@email.com"</code>	Configura seu e-mail
3	<code>git init</code>	Inicia um repositório local (caso esteja criando um projeto do zero)
 ou	<code>git clone <URL></code>	Clona um repositório existente (caso esteja baixando de um GitHub)
4	<code>git status</code>	Verifica o que mudou
5	<code>git add .</code>	Adiciona todos os arquivos modificados ao staging
6	<code>git commit -m "mensagem"</code>	Salva um ponto no histórico com uma mensagem
7	<code>git branch -M main</code>	(Opcional) Renomeia para a branch <code>main</code> padrão
8	<code>git remote add origin <URL-do-GitHub></code>	Conecta o repositório local ao GitHub
9	<code>git push -u origin main</code>	Envia seu projeto local para o GitHub pela primeira vez
	<code>git pull</code>	Baixa mudanças do GitHub (antes de editar algo)
	<code>git push</code>	Envia seus commits para o GitHub

Sim, está na ordem que você precisa seguir pra subir seu “Olá Mundo” por GitHub

5.4 Branches e Gitflow

Branches



Imagina aí que o seu projeto está numa linha principal do versionamento, chamada Main. Você quer desenvolver uma nova função, mas não quer arriscar “quebrar” sua aplicação com algo novo, então cria outro “save” da aplicação, onde pode testar à vontade, pois a Main está intacta.

Esse é o conceito de branch (galho ou ramo). O versionamento é uma árvore: o tronco é onde fica seu projeto principal (Main) e os muitos galhos são ramificações que você faz para introduzir novas coisas, tentar novas coisas!

Gitflow



Vamos por nosso inglês em dia: flow significa ritmo ou fluxo, então estamos falando do fluxo do Git, que é uma estratégia de usar as branches de forma mais organizada, padronizada, sabe?

Imagina umas 20 pessoas trabalhando em cima da mesma coisa... dá pra sentir o caos da bagunça, não é mesmo?

Deploy



Vamos lá, imagina junto comigo: você e sua equipe trabalharam duro para construir um sistema de estacionamento incrível (sua aplicação). Ele está montado na garagem (seu código no GitHub), todas as peças no lugar, testadas, e o manual (seu Gitflow) foi seguido à risca. O deploy é o momento de botar esse sistema pra jogo! Isso mesmo, botar ele na pista para que o pessoal possa utilizar e registrar os carros!

Na TI, o deploy é o processo de pegar todo o código que você desenvolveu, testou e versionou no GitHub, e enviá-lo para um servidor ou ambiente de hospedagem (tipo aquelas amostras grátis das lojas sabe?) onde ele pode ser executado 24 horas por dia, 7 dias por semana. É quando seu site fica online, seu aplicativo mobile se torna disponível nas lojas, ou seu sistema de backend começa a processar requisições (os carros entrando no sistema!). E claro, dinheiro na conta, se o código não pegar fogo...

Antes do Deploy, precisamos garantir que tudo esteja certinho, e como garantimos isso?! Com organização! Então segue aquele Gitflow, aliás, deixa eu te mostrar

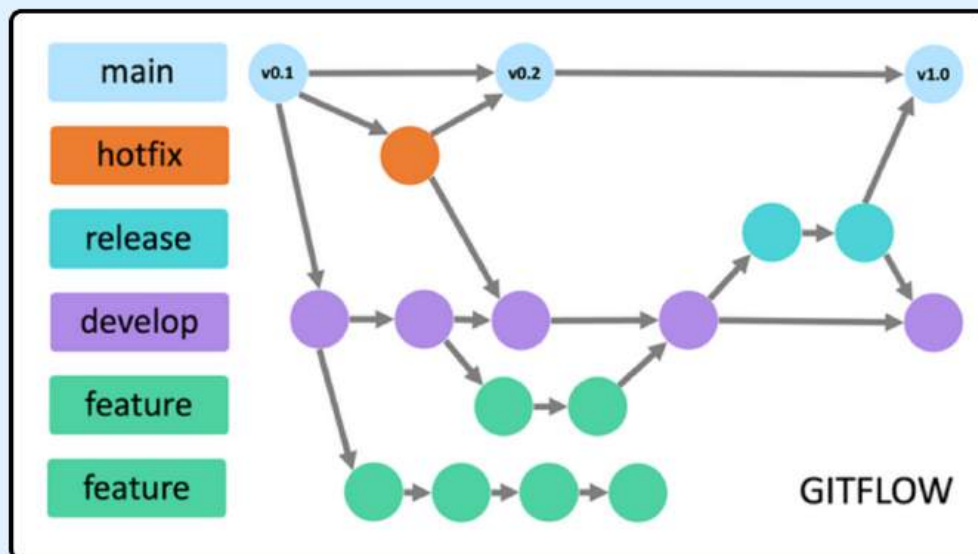
Para mostrar melhor esse Gitflow, vou apresentar uma planilha com os principais termos (sim, mais uma planilha — gosto de planilhas!).

Termo	Descrição	Exemplo / Uso
Branch	Linha paralela de desenvolvimento para trabalhar isoladamente sem afetar a principal.	<code>feature/login</code> , <code>bugfix/ajustar-bug</code>
Main (ou Master)	Branch principal que contém o código estável e pronto para produção.	Código finalizado e testado
Develop	Branch onde todas as funcionalidades em desenvolvimento são integradas antes de ir para o main.	Junta várias features antes de lançar versão
Feature Branch	Branch criada para desenvolver uma funcionalidade específica a partir do <code>develop</code> .	<code>feature/cadastro-usuario</code>
Release Branch	Branch criada para preparar uma nova versão, corrigir bugs e ajustar detalhes antes da entrega.	<code>release/v1.2</code>
Hotfix Branch	Branch criada para corrigir rapidamente bugs críticos na versão estável (main).	<code>hotfix/corrigir-falha-login</code>
Merge	Processo de unir mudanças de uma branch para outra.	Unir <code>feature/login</code> ao <code>develop</code>
Gitflow	Estrutura de branches que organiza o fluxo de desenvolvimento com as branches acima.	Workflow padrão para equipes de desenvolvimento

Muitos termos e comandos, não é mesmo?! Mas ajudam a manter o fluxo da coisa.

Uma coisa importante: normalmente, quando acontece o Merge entre branches, a branch usada para se fundir à outra costuma ser apagada depois (até porque, ela já fez o que era pra fazer).

Na prática gente, um Gitflow segue mais ou menos esse caminho aqui oh:



Cada branch evolui paralela às outras, somente se encontrando quando damos o Merge.

Nessa hora, elas deixam de existir, podendo “nascer” outra depois — mas não tem nada a ver com a anterior, viu!

5.5 NÃO! Git NÃO é Github

Apesar do nome ser parecido, não, pequeno gafanhoto, git não é uma abreviação carinhosa de GitHub.

Vamos ver melhor o que é cada um:

5.5.1 Git

É o sistema de versionamento do código, é ele que cria toda aquela árvore que falei antes, lembra? Funciona no nosso computador, que chamamos de Local.



5.5.2 Github

Aqui estamos falando do site, que é onde você hospeda aquele versionamento que criou, que chamamos de repositório Git.

É como se fosse o “slot online” do seu jogo — você salva localmente (com o Git), mas também envia uma cópia pra nuvem (no GitHub), pra não perder, compartilhar ou continuar de onde parou em qualquer lugar.

Uma vez no GitHub, você pode clonar os repositórios (os públicos ou os que você tem acesso, tá?). Você pode inclusive opinar sobre outros repositórios. É como se, além de um lugar para hospedar seus repositórios (que aqui no GitHub chamamos de remote ou remoto), você também interage com outros programadores — quase como uma rede social, só que com muito código.



Ah, a mascote do GitHub, aquele gato-polvo, se chama Octocat!

A ideia por trás dele foi criar uma figura simpática, e aqueles tentáculos representam justamente isso: conexão entre pessoas, projetos e branches. E sobre ser um gato? Bom... quem é que não gosta de gatos, né?!

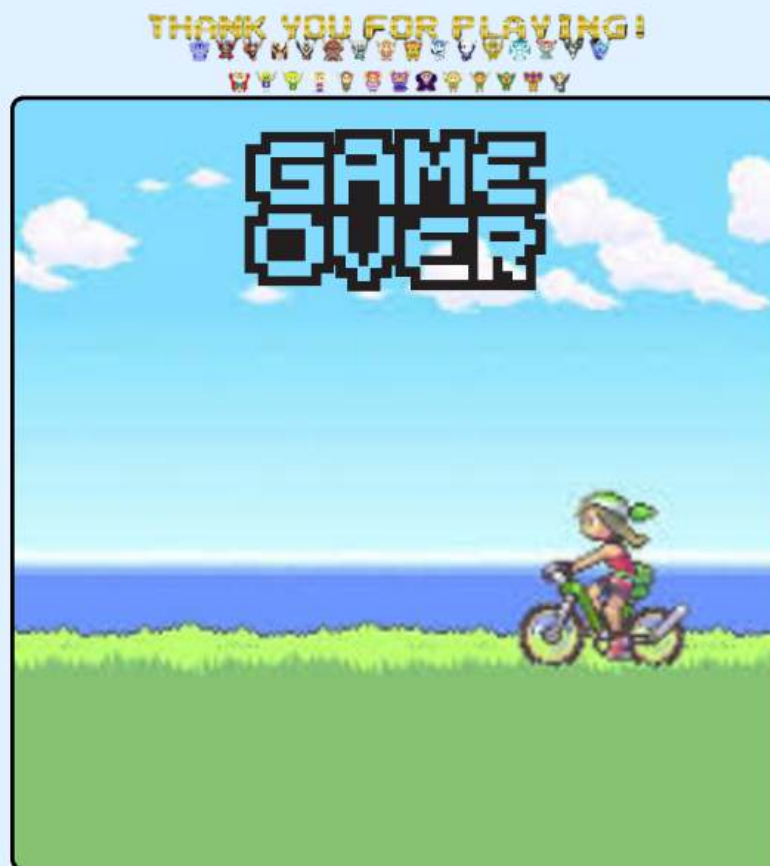


6. Muito Obrigado!

Montar esse eBook foi uma tarefa viu! Ao mesmo tempo instigante e desafiadora, porque ao passo que falava sobre alguns conceitos, mais coisas vinham a mente, olha, escrever algo é uma tarefa difícil viu, então espero que o esforço tenha valido a pena e ajudado você!

Sobre o projeto, bom, ele tem um potencial enorme de poder ser aumentado, desenvolvido (chamamos isso de escalabilidade). Pode-se usar Django para criar uma interface Web com o Python, um banco de dados como Supabase para receber todos os dados de entrada, saída, pagamentos e até mesmo hospedar todo o sistema em um servidor remoto como a conhecida AWS, da Amazon. Pois é, um mundo de possibilidades!

Programação é isso, sabe: ver, imaginar, pensar, moldar, recomeçar tudo e expandir! Divirta-se e permita-se encantar!





Extras!

Aqui eu vou deixar o link pra alguns vídeos e um presentinho pra você que chegou até aqui!

Me valoriza hein! Deu trabalho escrever tudo isso, haja café!

Prettier + ESLint, configura aí pra nunca mais errar!



Repositório com o projeto real do Sistema de Estacionamento

