Plano Detalhado do Curso de Desenvolvimento com Mac, iPad e Ferramentas Modernas

Visão Geral e Objetivo do Curso

Este curso prático de desenvolvimento tem como objetivo principal capacitar você a criar e implementar aplicações de forma eficiente, aproveitando recursos modernos (como contêineres Docker e assistentes de código com IA) e utilizando tanto um Mac quanto um iPad no processo. O foco está em **aprender fazendo**: cada conceito será explicado em teoria e imediatamente aplicado na prática, garantindo compreensão profunda do *como* e *porquê* de cada passo. O curso será conduzido em língua **portuguesa**, porém adotaremos convenções internacionais de programação – ou seja, nomes de variáveis, funções e outros identificadores em código serão em **inglês**, conforme as boas práticas da área[1]. Assim, unimos o melhor dos dois mundos: você recebe explicações claras em português, enquanto escreve códigos legíveis globalmente.

Ao final do curso, você estará apto a configurar seu ambiente de desenvolvimento tanto localmente (no Mac) quanto na nuvem (via iPad com GitHub Codespaces), utilizando Docker para garantir consistência, versionando seu código com Git/GitHub e até aproveitando ferramentas de Inteligência Artificial (como o GitHub Copilot) para aumentar sua produtividade. Tudo isso **sem abrir mão dos fundamentos** – cada técnica moderna será ensinada lado a lado com seus princípios básicos subjacentes.

Ambiente de Desenvolvimento: Mac, iPad e Ubuntu na Nuvem

Dispositivos e Plataformas: Você dispõe de um Mac e um iPad, o que abre duas possibilidades de ambiente de desenvolvimento: **local** e **na nuvem**. No Mac, podemos trabalhar localmente, instalando as ferramentas necessárias diretamente no macOS. Já no iPad, aproveitaremos o **GitHub Codespaces**, que permite rodar um ambiente de desenvolvimento completo via internet. O Codespaces nada mais é que um ambiente baseado em VS Code rodando em um servidor remoto – essencialmente um contêiner Docker Linux hospedado na nuvem, vinculado a um repositório GitHub[2]. Isso significa que ao usar o iPad você estará, via Safari ou aplicativo VS Code, codificando em uma máquina virtual Linux remota com todas as ferramentas instaladas, mas com a interface de um editor de código no tablet. Essa abordagem torna possível programar no iPad quase como se estivesse em um desktop, desde que haja boa conexão de internet. Teremos uma parte inicial do curso dedicada a configurar e **familiarizar você com o Codespaces no iPad**, incluindo dicas de ergonomia (por exemplo, usar teclado e mouse/trackpad com o iPad, abrir o Codespace em tela cheia pelo Safari etc., para uma experiência mais próxima de um notebook).

Sistema Operacional e Configuração: Você mencionou não ter preferência por sistemas operacionais e já ter ouvido falar do Ubuntu. Vamos então adotar o que for mais adequado para cada situação: - No Mac, utilizaremos o próprio **macOS**, que é Unix-like e compatível com praticamente todas as ferramentas de desenvolvimento (terminal Bash/Zsh, compilers, etc.). Grande parte dos comandos e softwares Linux também funcionam no macOS ou possuem equivalentes, então sua experiência local será próxima de um ambiente Ubuntu. - Nos Codespaces (iPad), o ambiente padrão é um **Linux (Ubuntu)** containerizado. Iremos configurar um *dev container* específico para o curso – em outras palavras, definir uma imagem Docker com todas as dependências que o nosso projeto precisa. Com isso, **independentemente de você estar no Mac ou no iPad**, poderemos usar o mesmo ambiente Linux padronizado dentro de um contêiner. Essa padronização é extremamente benéfica: elimina o problema do "na minha máquina funcionou" garantindo que o projeto rode exatamente igual em qualquer sistema, seja no Mac local ou no servidor na nuvem[3]. Caso precisemos de ferramentas adicionais durante o curso (por exemplo, banco de dados para testes, ou uma versão específica de alguma biblioteca), nós **"criaremos o resto conforme necessidade"**, o que pode significar rodar contêineres auxiliares via Docker no Mac ou adicionar serviços na configuração do Codespace. Você mencionou que gosta do Docker, então ficará à vontade nesse ambiente – usaremos contêineres tanto para uniformizar o dev (via Codespaces/devcontainer) quanto possivelmente para rodar serviços isolados (por exemplo, subir um container de banco de dados para testar uma aplicação).

Em resumo, **nosso plano de ambiente** será: - **Mac local:** Instalar um editor de código (VS Code), Docker Desktop e ferramentas de linha de comando necessárias (Git, Python, etc.). Assim, o Mac servirá para desenvolvimento offline/tradicional. - **iPad:** Configurar o GitHub Codespaces com nosso repositório do curso. Vamos garantir que você saiba iniciar um Codespace e conectar via navegador do iPad. O Codespace fornecerá um shell Linux e VS Code Web com o mesmo conteúdo do seu repositório. Como Codespaces é pago após certa cota, verificaremos as opções (conta estudantil ou alternativas) – mas no início do curso daremos todas as instruções para habilitar e usar essa ferramenta (habilitar Codespaces no repo, setar limite de uso, etc.[4]). Uma vez ativo, você terá um ambiente Ubuntu 20.04/22.04 (ou outra versão apropriada) rodando nossas ferramentas – efetivamente um *Docker container em VM* que o GitHub preparou para você[2].

- **Ubuntu/Linux (opcional local):** Se em algum momento quisermos simular o ambiente Linux do Codespace *direto no Mac*, podemos usar o Docker Desktop no Mac para rodar um contêiner Ubuntu. Por exemplo, poderíamos montar seu diretório de projeto dentro de um container Ubuntu para rodar algo exatamente como no servidor. Isso não será obrigatório graças ao Codespace, mas é uma alternativa. Outra opção seria uma máquina virtual Linux no Mac (via VirtualBox/UTM) caso prefira, mas acreditamos que não será necessário.

Nos primeiros módulos, dedicaremos tempo para **instalação e configuração**: preparando o Mac (instalando Homebrew, Git, VS Code, Docker), criando sua conta GitHub (se ainda não tiver) e habilitando Codespaces, testando acesso via iPad. Ao final dessa etapa inicial, seu ambiente estará pronto: seja no Mac ou iPad, você conseguirá acessar o mesmo projeto. Aproveitaremos para explicar conceitos básicos de sistema operacional e linha de comando durante essa preparação – por exemplo, navegaremos pelo terminal, mostraremos comandos Linux essenciais (ls, cd, mkdir, etc.) para que você se sinta confortável tanto no Terminal do macOS quanto no console do Codespace (Ubuntu). Assim, nivelamos seu conhecimento de **CLI (Command Line Interface)**, já que trabalhar com código frequentemente envolve usar a linha de comando.

Ferramentas e Tecnologias que Iremos Utilizar

Listando as principais ferramentas/tecnologias do curso, temos:

- **Linguagem de Programação:** Vamos utilizar **Python** como linguagem principal de ensino e prática. Python é altamente recomendada para iniciantes por ter sintaxe simples (quase pseudocódigo) e ampla aplicação prática – de desenvolvimento web a automação e inteligência artificial[5]. Isso permitirá focar nos conceitos e lógica sem esbarrar em complexidades desnecessárias da linguagem. Além disso, muitas das ferramentas modernas de IA (como bibliotecas de Machine Learning, APIs de LLMs) possuem suporte excelente em Python, o que se alinha com partes avançadas do curso.
- **Editor/IDE: Visual Studio Code** será nosso ambiente de edição de código. No Mac, você pode instalar o VS Code Desktop; no iPad, usará o VS Code via browser no Codespaces (a interface é praticamente idêntica). O VS Code é leve, extensível e já tem integração nativa com GitHub e Docker, o que agilizará nosso trabalho.
- **Git e GitHub:** Usaremos **Git** para controle de versão do código e o **GitHub** para hospedar nosso repositório de código do curso. Logo no começo, você aprenderá a iniciar um repositório, salvar versões do seu código (*commits*), e enviar para o GitHub (*push*). Como o Codespaces depende de um repositório, isso já estará inserido no fluxo. Explicaremos não só o *como* usar Git, mas também *por que* controle de versão é importante (histórico de alterações, colaboração, backup etc.). Com o tempo, introduziremos conceitos como *branch*, *pull request* e mesmo *Issues* no GitHub – por exemplo, mostraremos como abrir um Pull Request para suas próprias alterações (mesmo que você estude sozinho, é bom praticar o fluxo) e como mencionar issues ou outros colaboradores. Essa familiaridade com Git/GitHub será útil inclusive para aproveitar plenamente o Codespaces e futuramente colaborar em projetos open-source.
- **Docker:** O Docker será uma peça-chave no nosso curso. Na prática, ele aparecerá de duas formas: (1) de forma *transparente* dentro do Codespace

(que utiliza um contêiner Docker para seu ambiente de desenvolvimento); e (2) de forma *explícita* quando ensinarmos você a criar e usar seus próprios contêineres. Após você se ambientar com programação básica, teremos um módulo dedicado ao **Docker** em si – onde explicaremos conceitos de imagem, contêiner, Dockerfile, e vamos construir um ambiente isolado para sua aplicação. A ideia é que você veja na prática como *"empacotar"* uma aplicação com todas as dependências em uma imagem Docker e executá-la em qualquer lugar. Isso reforça o benefício da portabilidade: com Docker, garantimos que seu projeto funciona da mesma maneira em qualquer ambiente, seja no Windows de outra pessoa, num Linux de servidor ou na nuvem[3]. Vamos demonstrar, por exemplo, como criar uma imagem Docker do seu projeto final e rodá-la localmente ou no Codespace, enfatizando como o container traz consigo tudo o que é necessário para rodar a aplicação (evitando problemas de "versões incompatíveis" ou "falta algo instalado").

- **GitHub Codespaces:** Já mencionado, é nosso ambiente cloud. Vale lembrar que ele nos dá uma máquina Linux virtual sob medida para nosso projeto. Iremos configurar arquivos **DevContainer** no repositório para preinstalar dependências, garantindo que ao abrir o Codespace você tenha todas as ferramentas do curso prontas (Python, bibliotecas, etc.). *Dica:* Como estudante você pode ter horas gratuitas de Codespaces (o GitHub oferece uma cota para alunos verificados[6]); vamos verificar isso para evitar custos. Alternativamente, ensinarei também a usar o VS Code remotamente conectado ao Mac ou outras soluções se necessário, mas o foco será Codespaces por ser mais simples e integrado.
- **Assistentes de Código com IA:** Um diferencial deste curso é introduzir gradualmente ferramentas de Inteligência Artificial que auxiliam no desenvolvimento. Em particular, usaremos o **GitHub Copilot** (especialmente o Copilot Chat) e possivelmente outras ferramentas como um **CLI de LLM**. O GitHub Copilot é um assistente de código integrado ao VS Code que sugere trechos de código ou mesmo explica código em linguagem natural. Já o **Copilot Chat** é como ter um "chat" dentro do VS Code onde você pode perguntar sobre o código, pedir correções, etc., e ele responde com base na análise do seu projeto. Paralelamente, exploraremos um **CLI AI (Linha de comando para IA)** que chamamos de "Gemini CLI" no plano – basicamente, uma ferramenta de terminal para interagir com um modelo de linguagem. Isso pode permitir, por exemplo, gerar código ou texto automatizadamente via linha de comando, integrar em scripts ou fluxos de trabalho. Esses recursos de IA serão introduzidos **depois** que você aprender a fazer as coisas manualmente. Assim, primeiro você entenderá e implementará soluções por conta própria; em seguida, verá como a IA pode acelerar certas tarefas. Dessa forma, você colhe os benefícios (produtividade) sem ficar dependente – sempre entendendo o que está acontecendo por trás. Veremos casos de uso do Copilot, mas também discutiremos limitações e a importância de revisar o código sugerido. Em módulos avançados, abordaremos até mesmo **Agentes**

de IA – ou seja, scripts que usam modelos de IA para automatizar fluxos maiores (por exemplo, ler uma descrição de problema e automaticamente gerar código, testes, etc.). Isso está na fronteira da tecnologia atual, e daremos um gostinho de como funciona, em um contexto seguro, para você já vislumbrar tendências futuras.

Resumindo as ferramentas: você terá um ambiente integrado onde codificará em Python usando VS Code, versionará tudo no GitHub, rodará e testará em contêineres Docker, e ainda contará com ajuda de IA (Copilot) para agilizar, tudo isso em português (nas explicações) e com padrões profissionais (no código). O curso ensinará não apenas a usar essas ferramentas, mas a compreender **por que cada ferramenta existe** e **quando utilizá-la ou não**, formando um desenvolvedor completo e consciente.

Metodologia de Ensino: Teoria aliada à Prática

Você ressaltou que possui conhecimento **"básico do básico" em teoria** e gostaria que cada passo prático venha acompanhado da explicação teórica do como e porquê. Essa será exatamente nossa abordagem. Cada módulo ou aula seguirá este formato geral:

1. **Conceito Teórico Inicial:** Apresentamos o tópico do dia, definindo conceitos e motivando sua importância. Por exemplo, antes de abrir o Codespace, explicaremos o que é computação em nuvem e por que ambientes de desenvolvimento remoto podem ser úteis; antes de usar Git, discutiremos os problemas de se não versionar código; antes de usar Docker, entenderemos o cenário de incompatibilidades de ambiente que ele resolve; e assim por diante. Tudo em uma linguagem simples, com analogias se possível, para conectar com a teoria que você já conhece.
2. **Demonstração Prática Guiada:** Em seguida, **fazemos na prática**. Eu (como instrutor) vou executar passos no ambiente, falando em voz alta (ou escrevendo, no caso do material escrito) o que estou fazendo e relacionando com a teoria. Por exemplo: "*Agora vou criar um contêiner Docker para nossa aplicação – lembre que falamos que contêiner é como uma 'caixinha' isolada, então vamos ver isso acontecendo...*". Esse formato imita um pensamento em voz alta para que nada passe despercebido. **Cada "coisinha" terá sua justificativa** apresentada. O objetivo é que você nunca execute um comando ou escreva uma linha de código sem saber **exatamente** o motivo.
3. **Atividade Prática do Aluno:** Após a demonstração, é a sua vez. Você será convidado a repetir o que foi feito (ou resolver um pequeno desafio) para fixar o conhecimento. Por exemplo, após mostrarmos como fazer um commit no Git, podemos pedir: "*Agora faça você mesmo uma alteração simples em um arquivo e realize um commit descrito, push para o GitHub, e veja no site se atualizou.*" Estarei acompanhando e pronto para tirar dúvidas durante esse hands-on. Essa alternância garante envolvimento ativo – **teoria e prática**

andam juntas, pois é praticando logo após aprender que consolidamos de fato o conhecimento[7].

4. **Recapitulação e Esclarecimentos:** No fim de cada módulo, faremos um breve resumo dos pontos-chave vistos e abriremos espaço para perguntas. Qualquer detalhe que tenha ficado confuso será revisitado. Essa recapitulação também reforça a teoria subjacente: por exemplo, ao fim do módulo de Docker, você deverá ser capaz de explicar com suas palavras o que é uma imagem e um container, não apenas seguir receitinhas.

Adotaremos também uma filosofia de aprendizado **iterativo**: alguns conceitos aparecerão de forma simples primeiro e depois serão aprofundados. Por exemplo, você pode ter um primeiro contato com "funções" em Python bem no início (apenas para abstrair comandos repetitivos), mas voltaremos a falar de funções com mais detalhes conforme os projetos ficarem mais complexos, abordando parâmetros, retorno, boas práticas, etc. Isso permite que você **veja o panorama geral** cedo, sem ficar parado muito tempo só na teoria, e depois enriqueça a compreensão conforme já estiver mais confortável.

Além disso, o curso incentivará que você **anote e comente** seu entendimento. Como será em português, você pode escrever comentários no código (ou em um caderno) resumindo o que cada parte faz, em português mesmo, inicialmente. Com o tempo, quando fizer sentido, podemos passar esses comentários para inglês técnico ou remover quando o entendimento já for pleno. O importante é você criar seu próprio *significado* do código.

Por fim, em termos de ritmo: começaremos **do básico absoluto**, assumindo apenas o mínimo (ex.: você sabe usar o computador e tem vaga noção de lógica). Se em algum momento você já dominar bem algo, poderemos acelerar um pouquinho, mas pelo seu feedback vamos prezar por **explicar até o trivial**, sem presumir demais. É melhor arriscar explicar algo que você já saiba do que pular algo necessário. Com o avançar das aulas, a dificuldade aumenta gradualmente, mas sempre construída sobre os pilares já consolidados anteriormente.

Estrutura de Conteúdo e Cronograma Proposto

A seguir, está o esboço dos módulos do curso, incorporando todas as preferências e requisitos discutidos:

Módulo 1: Preparação do Ambiente de Desenvolvimento

- **Descrição:** Nesta primeira etapa, vamos deixar tudo pronto para você começar a desenvolver tanto no Mac quanto no iPad.
- **Conteúdo Teórico:** Visão geral de ambientes de desenvolvimento. Diferenças entre ambiente local e remoto. Conceito de máquina virtual vs. contêiner (explicação leve, já introduzindo Docker conceitualmente). Importância de ter um ambiente consistente configurado.

- **Conteúdo Prático:**
- *No Mac:* Instalação do VS Code; configuração do terminal (oh-my-zsh, por exemplo, para melhorar a usabilidade); instalação do Git e Docker Desktop. Teste: rodar `git --version`, `docker --version` no terminal para validar.
- *No iPad:* Acesso ao GitHub Codespaces. Criação de um repositório **do curso** no GitHub (por exemplo, um repositório template que preparamos com README inicial). Ativação do Codespace nesse repo (clicando em **Code > Codespaces** no GitHub). Tutorial de uso básico do VS Code Web: abrir terminal integrado, abrir arquivos, editar e salvar.
- Sincronização: garantir que o que for feito no Codespace aparece no Git (commit/push) e pode ser puxado no Mac, e vice-versa. Faremos um teste criando um arquivo "hello.txt" no Codespace, commitando e vendo no Mac.
- **Destaques:** Dicas de ergonomia no iPad (uso de teclado físico, atalho para Escape, como usar toque vs. mouse). Apresentação da interface do VS Code (explorer, editor, terminal, etc.). Explicação do conceito de devcontainer (mencionamos que há um container Ubuntu rodando – podemos mostrar rapidamente um `uname -a` no Codespace para ver que é Linux).
- **Atividade:** Você repetirá as etapas principais: clonar o repo no Mac via Git, fazer uma alteração e mandar de volta, e abrir/atualizar no iPad. Isso fixa tanto o uso do Git básico quanto a familiaridade com os dois ambientes. Teremos certeza de que ao final deste módulo você consegue navegar e editar código em ambos os dispositivos.

Módulo 2: Fundamentos de Git e Controle de Versão

- **Descrição:** Introdução aprofundada ao controle de versão usando Git, integrado ao seu fluxo de trabalho local/nuvem.
- **Conteúdo Teórico:** Problemas de não versionar código (risco de perder trabalho, difícil colaborar, etc.). Como o Git resolve isso armazenando históricos de alterações. Conceitos de repositório, commit (snapshot), stage, branch, merge. Conceito de remoto vs local. Também introduzimos o GitHub como uma plataforma de colaboração – explicamos termos como *pull request* (solicitação de mudança), *issues* (para rastrear tarefas/bugs), e a ideia de que seu código no GitHub serve de portfólio público.
- **Conteúdo Prático:**
- Configurar nome e email do Git (identidade). Criar um repositório do zero (já fizemos no Módulo 1 para Codespaces, mas aqui podemos criar outro ou usar o mesmo para history). Fazer commits atômicos (cada pequena mudança com mensagem significativa).
- Exemplo: criar um arquivo `hello.py` simples, adicionar ao stage e fazer commit com mensagem `"Add hello world script"`. Editar o arquivo para imprimir outra coisa, fazer novo commit `"Update message"` e usar `git log` para ver histórico.

- Uso do GitHub: empurrar (*push*) os commits para o GitHub. Visualizar no site. Editar um arquivo pelo site do GitHub para mostrar que também é possível (apesar de não ser principal forma) e fazer pull no local para sincronizar.
- Branching: criar uma branch nova (ex.: dev), fazer uma alteração e abrir um Pull Request no GitHub via interface web para mesclar, só para ilustrar o fluxo. Como é um curso solo, podemos fazer você mesmo revisar e mesclar, explicando o conceito de revisão de código (mesmo sozinho, você pode usar isso para experimentar mudanças sem bagunçar a principal).
- Issues e documentação: registrar talvez uma *Issue* no GitHub descrevendo uma tarefa (ex: "melhorar saída do hello.py") e depois fechar a issue com um commit (mostrando o uso de # referência no commit message). Não aprofundaremos muito, só para familiarizar com termos – isso também prepara terreno para quando formos usar o Copilot Chat, que pode referenciar issues e PRs.
- **Destaques:** Boas práticas de commits (mensagens no imperativo, curtas porém descritivas). Importância de .gitignore (mostrar superficialmente). Explicar que GitHub é amplamente usado no mercado (vale a pena dominar).
- **Atividade:** Você irá criar um pequeno projeto de exemplo, por exemplo um "Livro de Receitas" em texto: vários arquivos .md com receitas. Vai versionar esse projeto: adicionar arquivos, remover um, renomear, com commits adequados. Simular colaboração: forçar uma situação de conflito (edição no Mac e no iPad no mesmo arquivo) para aprender a resolver merge conflicts de forma simples. Tudo acompanhado, claro.
- **Teoria junto da prática:** A cada comando novo de Git, relembramos o conceito. Ex: ao ensinar `git commit -m "Mensagem"`, reforçamos "um commit é um ponto na história do projeto, que guarda as mudanças feitas...". Isso reforça seu entendimento conceitual.

Módulo 3: Introdução à Programação em Python (Fundamentos)

- **Descrição:** Aqui começa nossa jornada de programação de verdade. Vamos introduzir a linguagem Python e cobrir os pilares básicos: variáveis, tipos de dados, operadores, estruturas de controle (condicionais e loops), funções e estruturas de dados principais (listas, dicionários).
- **Conteúdo Teórico:** Iniciaremos com lógica de programação – como pensar em termos de passos lógicos para resolver problemas. Falaremos sobre algoritmo simples e como isso traduz para código. Em seguida, para cada tópico (variáveis, condicionais, etc.), faremos uma explicação do conceito: por exemplo, *variáveis* como caixas que guardam valores (com exemplos do cotidiano), *loops* como uma forma de repetir tarefas automaticamente, e assim por diante. Se necessário, relembramos conceitos matemáticos básicos (operadores aritméticos, precedência) e a ideia de booleanos (verdadeiro/falso) para condicionais.
- **Conteúdo Prático:**

- Ambiente: Usaremos tanto o terminal Python interativo (REPL) quanto escrever scripts `.py` no editor. Isso porque a experiência de ver resultado imediato no REPL ajuda na compreensão de instruções simples. Ao mesmo tempo, é bom aprender a escrever programas completos em arquivos.
- Primeiro programa: o clássico **"Hello, World!"** em Python, executado no Mac (via `python3 hello.py`) e no Codespace (mesmo comando). Ver output no terminal. Explicar a sintaxe mínima.
- Em seguida, exemplos incrementais:
    - Declaração de variáveis e impressão: criar variáveis de vários tipos (`int`, `float`, `str`) e usar `print()`. Mostrar como Python é dinamicamente tipado (você não declara tipo, ele infere).
    - Operações: fazer alguns cálculos simples, concatenar strings, etc., demonstrando operadores.
    - Condicional: escrever um pequeno script que lê um número e diz se é par ou ímpar, por exemplo. Ensinar uso de `if/elif/else` com indentação do Python. Executar e testar diversos inputs.
    - Loop: talvez criar uma repetição `for` para somar números, ou um `while` simples. Mostramos ambos e explicamos quando usar cada.
    - Função: pegar algum código escrito e transformar em uma função reutilizável. Exemplo, se fizemos um cálculo várias vezes, encapsular em `def minha_func(x): ... return ...`. Mostrar chamada da função e resultado.
    - Estruturas de dados: criar uma lista de elementos (ex: lista de nomes), mostrar como iterar sobre ela, adicionar/remover elementos. Criar um dicionário representando, por exemplo, uma ficha de cadastro com campos nome, idade, etc., e acessar valores.
- Tudo isso entremeado com exercícios rápidos: a cada sub-tópico, você tenta algo parecido. Por exemplo, depois de eu demonstrar um `if` para número par, peço você para escrever um `if` que verifica se um texto contém certa letra, ou algo do tipo – adaptando o conceito para outro caso, confirmando que entendeu.
- Integração com Git: Continuaremos versionando tudo que escrevemos. Você vai criar um diretório `src/` no repositório para guardar seus códigos Python e fará commits a cada funcionalidade adicionada. Isso pratica Git continuamente e cria um histórico útil para revisar depois.
- **Destaques:** Como o curso é *bilingue em código*, apontaremos que **palavras-chave** da linguagem (if, for, def, etc.) e funções built-in são em inglês, o que justifica por que convencionamos escrever identificadores em inglês também – evita misturar idiomas no código, mantendo consistência[1]. Mostraremos casos estranhos se mistura (por exemplo, nomear variável "idades" e depois usar método `.append()` em inglês – claramente não fica coeso).

- **Atividade:** Ao final deste módulo de fundamentos, proporemos um pequeno projeto console: por exemplo, um **jogo de adivinhação** no terminal (o programa sorteia um número e você tenta adivinhar, com dicas de mais/menos). Esse projeto envolve ler entrada do usuário, usar loop para múltiplas tentativas e condição para verificar acerto – consolidando variáveis, loop, condicional e talvez uso de funções se encapsularmos lógica. Você vai desenvolvendo passo a passo com nossa orientação e depois refatoramos juntos para melhorar (por exemplo, ver se dá para tornar alguma parte uma função). Essa atividade amarra os conceitos de forma divertida e gera um resultado que você pode mostrar (e até versionar no GitHub Pages se quisermos fazer output bonitinho, mas provavelmente ficará apenas no terminal por simplicidade).

Nota: Até este ponto, cobrimos as bases da programação. Embora você já tivesse noção teórica, agora terá visto tudo isso funcionando na prática e aprendido boas práticas (nomes claros, código indentado corretamente, comentários quando necessário explicando intenção, etc.). Você também terá um repositório no GitHub com seus primeiros códigos – todos documentados por commits e possivelmente um README resumindo o projeto do jogo, o que já te dá um pequeno portfólio inicial.

Módulo 4: Desenvolvimento Orientado a Projetos & Introdução ao Docker

- **Descrição:** Com os fundamentos dominados, vamos caminhar para projetos maiores e introduzir contêineres Docker explicitamente. Este módulo serve de transição entre a base e assuntos mais avançados, começando a construir algo mais completo enquanto mostra como o Docker pode ajudar na configuração do ambiente desse projeto.
- **Conteúdo Teórico:** Revisamos brevemente o conceito de projetos de software – como deixar de escrever scripts avulsos e organizar um sistema com múltiplos arquivos, talvez módulos, etc. Falamos sobre dependências de projetos (bibliotecas externas) e o problema que isso acarreta: *"funciona na minha máquina mas não na do colega"* devido a configurações diferentes[8][3]. Daqui engatamos no **conceito de Docker**: recapitular que já mencionamos contêineres e agora veremos na prática. Explicar o que é imagem Docker e container de forma didática (muitas vezes se usa analogia com container de navio que leva tudo dentro). Enfatizar que Docker permite replicar o ambiente de produção localmente, isolar dependências e facilitar o deploy.
- **Conteúdo Prático:**
- Escolheremos um projeto levemente mais complexo para desenvolver. Pode ser, por exemplo, uma **aplicação web simples** usando Python (um pequeno webservice Flask ou FastAPI) *ou* uma **aplicação de análise de dados** que carrega um dataset, faz processamento e gera um resultado (gráfico ou

relatório). A decisão aqui pode se basear no seu interesse: se você quiser algo web, vamos por esse caminho; se preferir data science, adaptamos. *(Pelo escopo do curso proposto originalmente, talvez algo envolvendo IA – podemos também pensar em um projeto de chatbot simples, mas isso vai entrar mais nos módulos seguintes com IA. Então, por ora, vamos supor um mini-sistema web ou análise.)*

- **Configurando o projeto:** Estruturar pastas, criar um virtual environment (caso não usemos direto containers para deps) e instalar dependências. Exemplo: "Vamos criar um projeto Flask: precisamos instalar flask (`pip install flask`), escrever um app.py que define rotas, etc." Essa é oportunidade para introduzir **Gerenciamento de Pacotes (pip)** e o conceito de requirements.txt para listar dependências.
- **Dockerizando o projeto:** Escrever um **Dockerfile** para o projeto. Passo a passo, explicando cada instrução: `FROM python:3.X` (imagem base), `WORKDIR` (diretório de trabalho no container), `COPY` (copiar código), `RUN pip install -r requirements.txt` (instalar deps), `CMD python app.py` (comando para rodar). Construir a imagem (`docker build -t meuapp .`) e executar (`docker run -p 5000:5000 meuapp` se for web, por exemplo). Mostrar que o app roda dentro do container. Testar acessando o serviço (abrir localhost no Mac, ou via port forwarding no Codespace se estivermos lá).
- **Ciclo Dev com Docker:** Ensinar como iterar: se mudar código, precisa rebuildar imagem ou montar volume para desenvolvimento; apresentar soluções rápidas como Docker Compose se relevante (talvez overkill se projeto é simples). Porém, dado que você gosta de Docker, podemos mostrar o Compose para, por exemplo, subir múltiplos containers se for caso (um para app, um para banco de dados Postgres, etc., dependendo do projeto escolhido).
- **Integração com Codespaces:** Ressaltar que no Codespace, possivelmente já configuramos o devcontainer para ter Docker pronto (o Codespace permite Docker-in-Docker). Então você pode também construir e rodar o container no próprio Codespace. Isso valida que o container funciona igual em qualquer lugar.
- Durante todo esse processo, cada comando do Docker será explicado (por que usar determinada base de imagem, importância de não deixar imagem muito grande, etc., apenas toques de boas práticas).
- **Destaques:** Vamos citar que container não é máquina virtual completa – por isso inicia rápido e compartilha kernel do host, conforme teoria Docker[9]. Também mostrar comandos úteis: `docker ps` (ver containers rodando), `docker images` (ver imagens), `docker stop/rm` (parar e remover containers), para você já começar a se familiarizar em gerenciar containers.
- **Atividade:** Você escreverá seu próprio Dockerfile (podemos dar um exercício de dockerizar uma pequena aplicação diferente, por exemplo dockerizar o script de jogo de adivinhação anterior – embora não faça muito sentido um

container para algo interativo de console, poderíamos transformá-lo num serviço API e dockerizar; mas pode ser mais simples: damos um programa Python que depende de alguma biblioteca e pedimos para você containerizar). Faremos revisão juntos se o Dockerfile atende, e então você mesma executará a imagem para ver se roda. Essa atividade reforça a fixação dos passos Docker.

- **Resultado do Módulo:** Ao final, você terá **um projeto completo rodando em container**. Seja um pequeno site ou aplicação de análise, o importante é que agora você domina tanto a lógica do programa em si quanto a forma de empacotá-lo para rodar em qualquer lugar. Seu repositório GitHub agora terá inclusive um Dockerfile, e podemos configurar uma ação simples de CI (GitHub Actions) para, por exemplo, **construir automaticamente a imagem Docker** a cada push – isso já introduz o conceito de **Integração Contínua (CI)** de maneira prática. (Por exemplo, mostraremos um arquivo YAML de Actions que faz build e possivelmente teste, se tivermos testes.)

Módulo 5: Desenvolvimento Assistido por IA – GitHub Copilot e Ferramentas de IA

- **Descrição:** Neste módulo, exploramos como Inteligência Artificial pode auxiliar no desenvolvimento. Mostraremos primeiro o GitHub Copilot em ação e depois o Copilot Chat, ensinando a usar efetivamente essas ferramentas dentro do VS Code. Também apresentaremos um utilitário de linha de comando para interagir com modelos de linguagem (denominado aqui de *Gemini CLI*), para ampliar sua produtividade em tarefas além do código.
- **Conteúdo Teórico:** Introdução aos *Code Assistants*. Explicar de forma simples o que é um modelo de linguagem treinado em código (ex: GPT-x, Codex) e como consegue sugerir códigos. Discutir benefícios (acelera escrita de código repetitivo, lembra sintaxe, etc.) e riscos (pode sugerir algo errado ou desatualizado, exige validação do desenvolvedor). Conceitos de contexto e prompt – porque às vezes o Copilot acerta, outras erra, dependendo do comentário/código em volta. Também mencionaremos brevemente questões de segurança (ele pode vazar código licenciado, etc., mas Copilot filtra um pouco) e éticas (o desenvolvedor continua responsável).
- **Conteúdo Prático:**
- **Configuração:** Certificar que você tem acesso ao GitHub Copilot (é uma ferramenta paga/assinatura, mas talvez com acesso educacional gratuito por um período – veremos seu status; se não tiver, podemos usar a versão ChatGPT Code Interpreter via browser como alternativa para demonstrar conceitos de IA assistindo no código). Supondo que tenha Copilot, habilitamos no VS Code (extensão, login).
- **Copilot básico:** Abrir um arquivo Python e começar a digitar uma função com um comentário em inglês, tipo `# Função para calcular fatorial` e ver o Copilot sugerir implementação. Mostrar como aceitar sugestão (Tab) ou ver

alternativas. Testar a sugestão e validar se está correta. Tentar exemplos: escrever docstring e deixar Copilot completar corpo da função, ou escrever o nome de uma função bem descritivo e ver ele gerar todo o código. Experimentar com casos do seu projeto existente – por exemplo, em nosso projeto web, pedir Copilot para escrever uma nova rota ou um teste unitário.

- **Copilot Chat:** Abrir a janela de chat do Copilot. Demonstrar usos: "*Encontrar bug no meu código*" (colar trecho com bug e perguntar no chat), "*Explicar o que esse função faz*" e ver a explicação, "*Sugerir melhorias*" etc. Mostrar também que podemos referenciar arquivos ("@") ou PRs/issues ("#") no prompt para dar contexto, e usar comandos ("/") como `/test` ou `/fix` se habilitado – isso conforme as capacidades do Copilot Chat[10][11]. Tudo será feito de forma guiada para que você aprenda a formular boas perguntas/pedidos à ferramenta.
- **Gemini CLI (IA via Terminal):** Introduzir a ferramenta CLI (pode ser algo como usar o OpenAI API via linha de comando ou um script custom). Por exemplo, ensinar você a usar o `curl` ou uma CLI específica para fazer perguntas a um modelo fora do VS Code – útil para quando se está no terminal puro (como no iPad, podemos abrir um terminal no Codespace e usar essa CLI). Simularemos: "*Gerar texto de README*" – você fornece um prompt e o modelo devolve um texto, que você pode revisar e colocar no README.md do projeto. Ou "*Escrever script para X*" e ele gera um arquivo que depois testamos. Essa parte mostra que a IA não está presente só em IDE, mas pode ser integrada em fluxos diversos (e.g., automatizar tarefas repetitivas).
- **Prática Orientada:** Vamos aplicar o Copilot em um mini-desafio: por exemplo, escrever uma função complexa (mas que você já saberia a lógica básica). Primeiro, você codará sem Copilot, apenas com sua lógica, para relembrar a importância de saber o fundamento. Depois, usaremos o Copilot para ver como ele faria, e comparar resultados, talvez até melhorando o seu código. Isso reforça que a IA é uma assistente, não um substituto do conhecimento – serve para ganhar velocidade quando você já entende o que precisa ser feito.
- Também faremos um exercício de *leitura crítica* de sugestão da IA: provocaremos um erro de propósito (por exemplo, perguntar algo ambíguo para o Copilot) e ele pode sugerir código incorreto; identificaremos o erro e corrigiremos. Assim, você pratica a habilidade de validar as saídas da IA em vez de confiar cegamente.
- **Destaques:** A linguagem do curso continua em português nas explicações, mas ao usar ferramentas de IA, provavelmente interagiremos em inglês, porque esses modelos respondem melhor a prompts em inglês. Vamos combinar isso da melhor forma – talvez escrever o comentário do código em inglês para guiar o Copilot, mas te explicar em português o que estamos fazendo. Isso também ajuda você a se familiarizar com alguns termos técnicos em inglês. Não se preocupe, sempre traduziremos e explicaremos cada resposta que a IA der, se necessário.

- **Atividade:** Desafio: Resolver um problema usando *Pair Programming* com Copilot. Por exemplo: "*Crie uma função que leia um arquivo CSV de contatos e filtre apenas os emails de um certo domínio*." Você vai tentar implementar e usar o Copilot Chat para ajuda. Vai perguntar no chat se travar, vai pedir testes unitários gerados por ele para validar sua função, etc. O resultado será entregue e avaliado em conjunto – a ideia é exercitar a colaboração humano+IA.
- **Extra:** Mencionar outras IA tools – ex: Codeium (alternativa open), ChatGPT direto via web para dúvidas conceituais (como "o que significa tal erro?"). Ressaltar que a IA pode também explicar teoria se perguntada – podemos testar perguntando "*Explique o conceito X*" e validar se está correto com o que ensinamos. Assim, você ganha mais uma fonte de aprendizado (sempre com olhar crítico).

Módulo 6: Projeto Integrador e Tópicos Avançados (Agentes, CI/CD)

- **Descrição:** Este é o módulo de fechamento, onde consolidamos tudo em um projeto integrador e exploramos brevemente tópicos avançados como agentes de IA autônomos e pipelines de CI/CD. O objetivo é você aplicar as habilidades adquiridas em um desafio mais abrangente que simula um cenário real de desenvolvimento.
- **Projeto Final:** Vamos definir um projeto para você desenvolver praticamente do zero, aplicando o processo completo:
- **Exemplo de Projeto:** *"Dashboard de Análise de Dados com Assistente IA"*. Suposição: você construirá uma pequena aplicação que lê um conjunto de dados (pode ser um dataset público simples, ex: dados meteorológicos ou de vendas), realiza algumas análises ou treinamentos de modelo, e disponibiliza um **dashboard web** com gráficos (usando, digamos, Streamlit ou Flask com charts). Além disso, incluirá um componente de chat onde o usuário pode fazer perguntas sobre os dados e uma IA responde (usando uma API ou modelo local simples).
- Esse projeto é só uma sugestão combinando vários elementos: envolve programação Python (ETL dos dados, cálculo de métricas), envolve possivelmente treinamento de um modelo ou uso de IA (poderíamos integrar um modelo pré-treinado de NLP para responder perguntas ou simplesmente filtrar dados no chat), envolve construir uma interface (dashboard web), e finalmente, empacotar tudo em Docker para rodar facilmente. O iPad via Codespace rodará o servidor web e você poderá visualizar no Safari via port forwarding.
- **Planejamento:** Antes de codar, faremos juntos o planejamento do projeto – escreveremos um README com objetivos, dividiremos em tarefas (por exemplo: 1. script de análise dos dados, 2. criar API ou função para responder pergunta, 3. interface Streamlit, 4. integrar componentes, 5. dockerizar, 6.

testes). Essa etapa de planejamento e divisão em issues/tasks será feita para simular a organização de um projeto real.

- **Conteúdo Teórico:** À medida que o projeto final avança, vamos tocando em tópicos avançados conforme a necessidade:
- **Agentes de IA:** Suponha que para o chat de perguntas sobre dados, usemos um agente que faz *RAG (Retrieval-Augmented Generation)*: busca informações nos dados e formula resposta. Explicaremos conceituamente o que é um agente (um programa que utiliza um LLM para decidir próximas ações como buscar dados, executar código, etc.). Não entraremos na implementação complexa, mas talvez usemos uma biblioteca existente ou pseudo-código para demonstrar. A ideia é mostrar que existem *pipelines* onde uma IA não apenas gera código, mas *controla fluxo* (por exemplo, lê a pergunta, decide qual função de busca chamar, obtém resultado e responde). Isso conecta com a vanguarda do desenvolvimento de software com IA. (No repositório do curso, eles chamaram de "Agents & Model Orchestration" – podemos tocar nisso alinhado com o que você vai ver lá).
- **CI/CD (Integração Contínua/Entrega Contínua):** Introduziremos a noção de que podemos automatizar testes e implantação. Se já colocamos um GitHub Action para buildar Docker, agora podemos expandir: adicionar um **teste automatizado** (ensinaremos você a escrever um ou dois testes unitários com `unittest` or `pytest` para partes críticas do projeto). Configurar no GitHub Actions para rodar esses testes a cada push. Mostrar um *badge* no README indicando o status da build CI. Conceitos: pipeline, esteira de deploy contínuo, importância de testes no desenvolvimento profissional.
- **Melhores Práticas de Código:** Conforme refatoramos o projeto final, discutiremos organização de código em módulos, docstrings e documentação, tratamento de erros/exceções, logs, etc. Coisas que diferenciam um script improvisado de um programa bem estruturado. Também falaremos de desempenho se relevante (por exemplo, se o dataset for grande, mostrar noção de complexidade ou uso eficiente de bibliotecas).
- **Conteúdo Prático:** O projeto final será desenvolvido iterativamente:
- **Configuração Inicial:** Criar repositório (ou reutilizar o existente, mas provavelmente um novo repo para o projeto final), configurar devcontainer se necessário, e scaffold básico (pastas, requirements).
- **Implementação em Partes:** Desenvolver cada funcionalidade por etapas. Você irá codar, com meu suporte e usando Copilot quando útil. Após cada parte, rodamos e testamos. Por exemplo, primeiro fazer a análise de dados funcionar offline, imprimir resultados no console; depois integrar a resposta via chat; depois criar o dashboard web que mostra gráficos; etc.
- **Testes:** Escrever testes básicos para assegurar que certas funções dão output correto. Executar `pytest` local ou via CI.

- **Containerização:** Escrever Dockerfile para essa aplicação completa, possivelmente usando multi-stage if image is heavy, etc. Construir e rodar localmente e no Codespace.
- **Deploy (Opcional):** Se houver interesse e tempo, mostrar como rodar esse container na nuvem (por ex., deploy em alguma plataforma free like Heroku or fly.io, or simply instruct that one could run it on a server). O foco não é implantar produto final comercialmente, mas entender o caminho de entregar um software.
- **Acompanhamento:** Durante esse desenvolvimento, cada decisão será tomada discutindo o porquê: exemplo "*Vamos usar Streamlit para o dashboard porque é alto nível e rápido para prototipar, ao invés de construir HTML do zero...*" ou "*Vamos usar biblioteca X para IA porque treinar do zero seria inviável*". Isso amarra todo conhecimento com critérios de engenharia de software.
- **Atividade/Desafio Final:** O projeto final em si é o grande desafio. Porém, podemos incrementar: após pronto, sugerir uma pequena mudança ou nova feature para você tentar implementar sozinho e depois revisamos junto. Isso simula a situação de manter e evoluir um projeto.
- **Conclusão do Projeto:** Finalizamos rodando a aplicação completa, testando todas as funcionalidades (por exemplo, abrir o dashboard, ver gráficos, fazer perguntas no chat e obter respostas). Paramos para apreciar o quanto foi construído a partir do zero até aqui – reforçando cada componente aprendido: "*Veja, ao fazer uma pergunta no chat, por trás dos panos um agente de IA está consultando os dados (que você manipulou com Python), possivelmente usando as funções que você escreveu, e devolvendo resposta. Tudo isso roda em um container Docker consistente, podendo ser executado tanto local quanto no Codespace. E cada alteração que você faz passa pelos testes e é registrada no Git.*". É o momento de conectar todos os pontos e cimentar a compreensão holística.

Módulo 7: Encerramento e Próximos Passos

(Módulo mais breve, de fechamento.)

- **Revisão Geral:** Recapitulação dos principais aprendizados do curso, revisitando cada módulo em poucos minutos para reforço final. Você terá a oportunidade de esclarecer quaisquer dúvidas remanescentes.
- **Avaliação:** Se for do seu interesse, podemos aplicar uma **avaliação** prática ou um quiz teórico. Por exemplo, um questionário sobre conceitos (como “Qual a diferença entre container e VM?” ou “O que é um commit no Git?”) e/ou solicitar pequenas implementações sem consultar as notas, para avaliar sua retenção. A correção seria comentada em detalhe – o foco não é uma nota, mas sim garantir que você está confortável com tudo. (Você havia indicado

"Sim, com certeza" quando perguntado se queria avaliações, então presumimos que sim, acha válido ter alguma forma de avaliar o progresso).

- **Feedback:** Também importante, você dará feedback sobre o curso – o que funcionou melhor, se o ritmo atendeu, etc., para ajustarmos qualquer necessidade futura. Afinal, construímos este plano com bastante flexibilidade para adaptarmos conforme "o que se propõe" originalmente.
- **Certificado/Conclusão:** Caso seja um curso formal, aqui entregaríamos certificado de conclusão. Mas mesmo que não seja, marcamos essa conquista: agora você é capaz de conduzir projetos de desenvolvimento do início ao fim usando ferramentas profissionais.
- **Próximos Passos:** Orientação sobre como continuar evoluindo. Sugerir, por exemplo, aprofundar em algoritmos e estruturas de dados (para fortalecer base teórica), ou em frameworks específicos conforme seu interesse (desenvolvimento web com Django/React, ciência de dados com Pandas/NumPy, etc.). Também indicar comunidades e recursos (fóruns, open source projects, documentação) para você praticar e se manter atualizado.
- **Encerramento:** Finalizamos motivando você a **continuar praticando** sempre – a chave para dominar programação é consistência e curiosidade. Como você viu, teoria e prática caminham juntas[7] e agora você tem um ótimo embasamento nos dois; a partir daqui, cada pequeno projeto ou desafio que assumir reforçará e expandirá seu conhecimento.

Considerações Finais

Com este plano detalhado, cobrimos **todos os pontos levantados** e os alinhamos às metas do curso: - Aproveitamos que você tem Mac e iPad, usando ambos de forma complementar (local vs cloud) para máxima conveniência. - Escolhemos tecnologias adequadas (Ubuntu/Linux via contêiner para padronizar ambiente, Python como linguagem inicial) sem preferência pessoal, apenas focando no que atende melhor aos objetivos. - Integramos o Docker, que você já aprecia, como ferramenta fundamental no fluxo de trabalho. - Garantimos abordagem **teórico-prática** equilibrada, nunca perdendo de vista a explicação do “por quê” enquanto fazemos o “como”[7]. - Mantivemos a **bilinguagem** apropriada: o curso é ministrado em português claro, enquanto o código segue padrões em inglês[1], para você já se habituar ao estilo profissional. - Partimos do **básico absoluto** e evoluímos até tópicos de ponta, sempre construindo sobre fundamentos sólidos. Mesmo conteúdos avançados (IA, CI/CD) serão ensinados de forma acessível, relacionando com princípios básicos aprendidos (por exemplo, entender um agente de IA ainda requer lógica, que você terá desenvolvido nos módulos iniciais). - Oferecemos opções “ambas” em vez de escolher um só caminho: você experimentará tanto o ambiente local quanto o na nuvem, tanto a codificação manual quanto a assistida por IA, tanto projetos de linha de comando quanto aplicações web, conforme for adequado em cada contexto do curso. Essa versatilidade atende à sua solicitação de explorar **ambos os lados** quando perguntado sobre preferências em diversos

tópicos. - Por fim, o curso se mantém focado no que foi proposto inicialmente (que entendemos ser formar você em desenvolvimento moderno, pleno, utilizando ferramentas como Codespaces e Copilot, possivelmente orientado a um contexto de IA). Todos os recursos e módulos planejados convergem para esse propósito.

Esperamos que este plano atenda (e exceda) suas expectativas, proporcionando uma jornada de aprendizagem enriquecedora. Você terminará o curso não apenas com conhecimento, mas com experiências práticas vivenciadas e projetos tangíveis construídos por você. Vamos em frente com confiança – qualquer ajuste fino que seja necessário faremos ao longo do caminho, mas a estrutura acima nos dá um roteiro claro para chegar lá. **Bom estudo e bom desenvolvimento!**

Referências Utilizadas: Para embasar nosso planejamento, consideramos recomendações de boas práticas amplamente aceitas, como a padronização do código em inglês[1], os benefícios do Docker na padronização de ambientes[3], a efetividade de combinar teoria e prática no aprendizado de programação[7], além de informações oficiais sobre o funcionamento do GitHub Codespaces[2] e relatos de uso em iPad[4]. Esses princípios e dados reforçam as escolhas feitas, assegurando que o curso esteja alinhado com o estado da arte em desenvolvimento de software. Boa jornada no curso!

---

[1] Programar em português ou inglês? That's the question! | carlos schults / blog

https://carlosschults.net/pt/programar-portugues-ou-ingles/

[2] [6] Using GitHub Codespaces – Johns Hopkins Engineering

https://support.cmts.jhu.edu/hc/en-us/articles/31239703506701-Using-GitHub-Codespaces

[3] [8] [9] Docker: a solução para ambientes de desenvolvimento

https://hub.asimov.academy/blog/docker-como-usar/

[4] Using GitHub Codespaces on iPad - DEV Community

https://dev.to/cubikca/using-github-codespaces-on-ipad-5412

[5] [7] Programação para iniciantes: por onde começar e como evoluir | Growdev

https://growdev.com.br/blog/programacao-para-iniciantes/

[10] [11] README.md

https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/README.md

INOVAÇÃO SURPREENDENTE AVANÇADA

1. Módulo de Desenvolvimento Multimodal com IA

- **Proposta:** Ensinar como criar aplicações que combinam texto, imagem, áudio e até vídeo usando APIs de IA multimodal (ex: OpenAI GPT-4o, Gemini, Stable Diffusion, Whisper).
- **Prática:** Desenvolver um app Python que recebe comandos por voz (usando Whisper), gera imagens (Stable Diffusion), responde perguntas sobre imagens (GPT-4o), e integra tudo em uma interface web.
- **Diferencial:** Pouquíssimos cursos ensinam integração multimodal real, especialmente usando iPad/Mac e Codespaces.

2. Laboratório de Automação de Ambientes com Infraestrutura como Código

- **Proposta:** Introduzir ferramentas como Terraform ou Ansible para automatizar a criação de ambientes de desenvolvimento na nuvem, inclusive provisionando Codespaces, containers, bancos de dados e até servidores web.
- **Prática:** O aluno cria scripts que, com um comando, montam todo o ambiente do projeto, incluindo permissões, rede e dependências.
- **Diferencial:** Isso aproxima o curso do DevOps moderno, algo raro em cursos para iniciantes e até intermediários.

3. Desafio de Programação com Agentes Autônomos

- **Proposta:** Criar um módulo onde o aluno constrói e treina um agente de IA que navega pelo próprio repositório, sugere melhorias, abre issues automaticamente e até faz pull requests com correções.
- **Prática:** Usar frameworks como LangChain ou AutoGPT para criar agentes que interagem com o GitHub, testam código, escrevem documentação e automatizam tarefas reais.
- **Diferencial:** Simula o futuro do desenvolvimento, onde IA não só assiste, mas age como colaborador autônomo.

4. Hackathon Integrado com IA Generativa

- **Proposta:** Realizar um hackathon dentro do curso, onde equipes (ou o aluno individualmente) recebem desafios-surpresa gerados por IA, com requisitos, datasets e até restrições criativas.
- **Prática:** A IA pode criar cenários fictícios, gerar dados sintéticos, propor problemas e até avaliar soluções, tornando a experiência dinâmica e imprevisível.
- **Diferencial:** Traz gamificação e imprevisibilidade, além de treinar o aluno para situações reais de mercado.

5. Módulo de Engenharia de Prompt e Segurança em IA

- **Proposta:** Ensinar técnicas avançadas de engenharia de prompt para extrair o máximo das ferramentas de IA, incluindo prompts encadeados, contextos longos, e mitigação de riscos (prompt injection, vazamento de dados).
- **Prática:** O aluno cria prompts que automatizam tarefas complexas, testa limites das ferramentas e aprende a proteger seu código e dados contra ataques de IA.
- **Diferencial:** Poucos cursos abordam segurança em IA aplicada ao desenvolvimento cotidiano.

6. Integração com Hardware e IoT via iPad/Mac

- **Proposta:** Mostrar como usar Python para controlar dispositivos físicos (Arduino, Raspberry Pi, sensores) a partir do Mac ou até remotamente via Codespaces.
- **Prática:** Projeto final pode incluir automação residencial, coleta de dados ambientais ou integração com APIs de dispositivos inteligentes.
- **Diferencial:** Une software, hardware e nuvem, expandindo o horizonte do aluno para além do código tradicional.

7. Portfólio Dinâmico com Deploy Automatizado

- **Proposta:** Ensinar a criar um portfólio online que se atualiza automaticamente a cada novo projeto ou commit, usando GitHub Actions, Docker e integração com plataformas como Vercel, Netlify ou Hugging Face Spaces.
- **Prática:** O aluno vê seu progresso refletido em tempo real, com dashboards, gráficos de evolução e até badges de conquistas gerados por IA.
- **Diferencial:** O portfólio vira uma vitrine viva, útil para carreira e networking.

8. Mentoria Virtual com IA Personalizada

- **Proposta:** Cada aluno configura um "mentor virtual" (chatbot customizado) treinado com seu histórico de dúvidas, projetos e preferências, que acompanha o progresso, sugere estudos e até simula entrevistas técnicas.
- **Prática:** Usar LLMs open-source ou APIs para criar esse mentor, que pode ser acessado via iPad, Mac ou Codespaces.
- **Diferencial:** Personalização extrema do aprendizado, com feedback contínuo e adaptativo.

9. Simulação de Incidentes e Resposta Rápida

- **Proposta:** O curso inclui "incidentes simulados" (bugs críticos, falhas de segurança, perda de dados) gerados por IA, e o aluno precisa diagnosticar, corrigir e documentar a resposta, como em equipes profissionais.
- **Prática:** O ambiente Codespaces pode ser "infectado" com um bug, e o aluno deve usar ferramentas aprendidas para restaurar o sistema.
- **Diferencial:** Treina habilidades de troubleshooting e resposta a crises, essenciais para desenvolvedores avançados.

10. Exploração de Computação Quântica e IA

- **Proposta:** Introduzir conceitos básicos de computação quântica, mostrar como rodar algoritmos simples em simuladores (ex: Qiskit) e discutir como IA pode acelerar pesquisas nessa área.
- **Prática:** O aluno executa experimentos quânticos via Python, integra resultados em dashboards e discute aplicações futuras.
- **Diferencial:** Pouquíssimos cursos conectam desenvolvimento tradicional, IA e computação quântica.

Como implementar?

Essas ideias podem ser módulos extras, desafios opcionais ou trilhas avançadas dentro do curso. Podem ser adaptadas conforme o interesse dos alunos, com tutoriais, projetos práticos e até colaboração entre participantes.

Rascunho de ROTEIRO

- **Objetivo**
- **Conteúdo teórico**
- **Projeto prático**
- **Ferramentas e recursos**
- **Diferenciais**
- **Como encaixar no fluxo do curso**

Se quiser, posso criar roteiros semanais, planos de aula, exemplos de projetos, ou até listas de recursos para cada etapa. Vamos começar pelo primeiro módulo inovador:

1. Desenvolvimento Multimodal com IA

Objetivo

Capacitar o aluno a criar aplicações que integram texto, imagem, áudio e vídeo usando APIs de IA multimodal (GPT-4o, Gemini, Stable Diffusion, Whisper).

Conteúdo Teórico

- O que é IA multimodal? Exemplos de aplicações reais.
- APIs e frameworks: OpenAI, Google Gemini, Hugging Face, Stability AI.
- Conceitos de processamento de imagem, áudio e vídeo.
- Fluxos de integração: como conectar diferentes modelos em uma pipeline.

Projeto Prático

- Criar um app Python que:
    - Recebe comandos por voz (Whisper).
    - Gera imagens a partir de texto (Stable Diffusion).
    - Responde perguntas sobre imagens (GPT-4o Vision).
    - Integra tudo em uma interface web (Streamlit ou Flask).
- Desafio extra: permitir upload de áudio ou vídeo e gerar respostas multimodais.

Ferramentas

- Python, Streamlit/Flask, APIs OpenAI/Gemini, Hugging Face Transformers, Docker, Codespaces.

Diferencial

- Pouquíssimos cursos ensinam integração multimodal real, especialmente usando iPad/Mac e Codespaces.
- O aluno sai capaz de criar apps que combinam voz, imagem e texto, algo de ponta no mercado.

2. Laboratório de Automação de Ambientes com Infraestrutura como Código

Objetivo

Ensinar a automatizar a criação de ambientes de desenvolvimento na nuvem usando Terraform, Ansible ou similares.

Conteúdo Teórico

- O que é Infraestrutura como Código (IaC)?
- Diferença entre scripts de automação e IaC declarativo.
- Provisionamento de ambientes: containers, VMs, Codespaces, bancos de dados.

Projeto Prático

- Criar scripts Terraform/Ansible que:
    - Montam todo o ambiente do projeto (Codespace, containers, banco de dados, permissões).
    - Permitem que qualquer pessoa replique o ambiente com um comando.
- Desafio extra: automatizar deploy de aplicações e pipelines CI/CD.

Ferramentas

- Terraform, Ansible, Docker, GitHub Actions, Codespaces.

Diferencial

- Aproxima o curso do DevOps moderno, preparando para times ágeis e cloud-native.

3. Desafio de Programação com Agentes Autônomos

Objetivo

Construir e treinar agentes de IA que navegam pelo repositório, sugerem melhorias, abrem issues e fazem pull requests automaticamente.

Conteúdo Teórico

- O que são agentes autônomos? Exemplos: AutoGPT, LangChain Agents.
- Fluxo de trabalho de agentes: observação, decisão, ação.
- Riscos e limitações dos agentes autônomos.

Projeto Prático

- Usar LangChain ou AutoGPT para criar um agente que:
    - Analisa o código do projeto.
    - Sugere melhorias e abre issues no GitHub.
    - Faz pull requests com correções automáticas.
- Desafio extra: agente que escreve documentação e testa código.

Ferramentas

- Python, LangChain, AutoGPT, GitHub API, Codespaces.

Diferencial

- Simula o futuro do desenvolvimento, onde IA age como colaborador autônomo.

4. Hackathon Integrado com IA Generativa

Objetivo

Realizar um hackathon onde desafios, dados e avaliações são gerados por IA.

Conteúdo Teórico

- O que é hackathon? Como IA pode criar e avaliar desafios.
- Gamificação e criatividade em ambientes de desenvolvimento.

Projeto Prático

- IA gera desafios-surpresa, datasets sintéticos e requisitos.
- Alunos (individualmente ou em equipes) resolvem problemas, submetem soluções e recebem feedback da IA.
- Desafio extra: IA avalia código, sugere melhorias e pontua soluções.

Ferramentas

- Python, OpenAI API, Hugging Face, Codespaces.

Diferencial

- Experiência dinâmica e imprevisível, treinando para situações reais de mercado.

5. Engenharia de Prompt e Segurança em IA

Objetivo

Ensinar técnicas avançadas de engenharia de prompt e mitigação de riscos em IA.

Conteúdo Teórico

- O que é engenharia de prompt? Exemplos de prompts encadeados e contextos longos.
- Riscos: prompt injection, vazamento de dados, segurança em IA.

Projeto Prático

- Criar prompts que automatizam tarefas complexas.
- Testar limites das ferramentas de IA.
- Implementar proteções contra ataques de IA.

Ferramentas

- Python, OpenAI API, Gemini, Codespaces.

Diferencial

- Poucos cursos abordam segurança em IA aplicada ao desenvolvimento cotidiano.

6. Integração com Hardware e IoT via iPad/Mac

Objetivo

Mostrar como usar Python para controlar dispositivos físicos (Arduino, Raspberry Pi, sensores) a partir do Mac ou remotamente via Codespaces.

Conteúdo Teórico

- O que é IoT? Exemplos de integração hardware-software.
- Protocolos de comunicação: serial, MQTT, HTTP.

Projeto Prático

- Projeto de automação residencial, coleta de dados ambientais ou integração com APIs de dispositivos inteligentes.
- Desafio extra: dashboard web para monitorar dispositivos em tempo real.

Ferramentas

- Python, Arduino/Raspberry Pi, Streamlit/Flask, Codespaces.

Diferencial

- Une software, hardware e nuvem, expandindo o horizonte do aluno para além do código tradicional.

7. Portfólio Dinâmico com Deploy Automatizado

Objetivo

Ensinar a criar um portfólio online que se atualiza automaticamente a cada novo projeto ou commit.

Conteúdo Teórico

- O que é portfólio dinâmico? Ferramentas de deploy automatizado.
- Integração com GitHub Actions, Docker, Vercel, Netlify, Hugging Face Spaces.

Projeto Prático

- Criar portfólio que mostra progresso em tempo real, dashboards, gráficos de evolução e badges de conquistas gerados por IA.
- Desafio extra: deploy automático de projetos e integração com redes sociais.

Ferramentas

- Python, GitHub Actions, Docker, Vercel/Netlify, Codespaces.

Diferencial

- O portfólio vira uma vitrine viva, útil para carreira e networking.

8. Mentoria Virtual com IA Personalizada

Objetivo

Configurar um "mentor virtual" (chatbot customizado) treinado com histórico de dúvidas, projetos e preferências do aluno.

Conteúdo Teórico

- O que é mentoria virtual? Como treinar chatbots personalizados.
- Feedback adaptativo e acompanhamento de progresso.

Projeto Prático

- Usar LLMs open-source ou APIs para criar mentor virtual acessível via iPad, Mac ou Codespaces.
- Desafio extra: mentor que simula entrevistas técnicas e sugere trilhas de estudo.

Ferramentas

- Python, Hugging Face, OpenAI API, Codespaces.

Diferencial

- Personalização extrema do aprendizado, com feedback contínuo e adaptativo.

9. Simulação de Incidentes e Resposta Rápida

Objetivo

Incluir "incidentes simulados" (bugs críticos, falhas de segurança, perda de dados) gerados por IA para treinar diagnóstico e correção.

Conteúdo Teórico

- O que é troubleshooting? Técnicas de resposta a crises.
- Documentação de incidentes e boas práticas de recuperação.

Projeto Prático

- Ambiente Codespaces “infectado” com bug, aluno deve restaurar sistema usando ferramentas aprendidas.
- Desafio extra: documentar resposta e criar plano de prevenção.

Ferramentas

- Python, Codespaces, GitHub Actions.

Diferencial

- Treina habilidades essenciais para desenvolvedores avançados.

10. Exploração de Computação Quântica e IA

Objetivo

Introduzir conceitos básicos de computação quântica e mostrar como IA pode acelerar pesquisas nessa área.

Conteúdo Teórico

- O que é computação quântica? Algoritmos e simuladores (Qiskit).
- Aplicações de IA em pesquisa quântica.

Projeto Prático

- Executar experimentos quânticos via Python, integrar resultados em dashboards e discutir aplicações futuras.
- Desafio extra: criar pipeline que une IA e computação quântica para resolver problemas reais.

Ferramentas

- Python, Qiskit, OpenAI API, Codespaces.

Diferencial

- Pouquíssimos cursos conectam desenvolvimento tradicional, IA e computação quântica.

Próximos Passos

Posso detalhar cada módulo com:

- Plano de aula semanal
- Exemplos de projetos práticos
- Recursos e links para APIs e ferramentas
- Sugestão de desafios e avaliações

ROTEIRO COMPLETO DOS MÓDULOS INOVADORES

1. Desenvolvimento Multimodal com IA

Objetivo:
Capacitar o aluno a criar aplicações que integram texto, imagem, áudio e vídeo usando APIs de IA multimodal.

Conteúdo Teórico:

- O que é IA multimodal?
- Exemplos de aplicações reais (assistentes, apps de acessibilidade, arte generativa).
- APIs e frameworks: OpenAI GPT-4o, Gemini, Stable Diffusion, Whisper, Hugging Face.
- Conceitos de processamento de imagem, áudio e vídeo.
- Pipeline multimodal: como conectar diferentes modelos.

Projeto Prático:

- App Python que:
    - Recebe comandos por voz (Whisper).
    - Gera imagens a partir de texto (Stable Diffusion).
    - Responde perguntas sobre imagens (GPT-4o Vision).
    - Integra tudo em uma interface web (Streamlit ou Flask).
- Desafio extra: upload de áudio/vídeo e geração de respostas multimodais.

Ferramentas:
Python, Streamlit/Flask, APIs OpenAI/Gemini, Hugging Face Transformers, Docker, Codespaces.

Diferenciais:

- Integração multimodal real, usando Mac/iPad e Codespaces.
- Aplicações de ponta, raras em cursos convencionais.

2. Laboratório de Automação de Ambientes com Infraestrutura como Código

Objetivo:
Ensinar automação da criação de ambientes de desenvolvimento na nuvem usando Terraform, Ansible ou similares.

Conteúdo Teórico:

- O que é Infraestrutura como Código (IaC)?
- Scripts de automação vs IaC declarativo.
- Provisionamento de containers, VMs, Codespaces, bancos de dados.

Projeto Prático:

- Scripts Terraform/Ansible que:
    - Montam todo o ambiente do projeto (Codespace, containers, banco de dados, permissões).
    - Permitem replicação do ambiente com um comando.
- Desafio extra: deploy automatizado de aplicações e pipelines CI/CD.

Ferramentas:
Terraform, Ansible, Docker, GitHub Actions, Codespaces.

Diferenciais:

- Aproxima do DevOps moderno, preparando para times ágeis e cloud-native.

3. Desafio de Programação com Agentes Autônomos

Objetivo:
Construir e treinar agentes de IA que navegam pelo repositório, sugerem melhorias, abrem issues e fazem pull requests automaticamente.

Conteúdo Teórico:

- O que são agentes autônomos? Exemplos: AutoGPT, LangChain Agents.
- Fluxo de trabalho: observação, decisão, ação.
- Riscos e limitações dos agentes autônomos.

Projeto Prático:

- Usar LangChain ou AutoGPT para criar agente que:
    - Analisa código do projeto.
    - Sugere melhorias e abre issues no GitHub.

    - Faz pull requests com correções automáticas.
- Desafio extra: agente que escreve documentação e testa código.

Ferramentas:
Python, LangChain, AutoGPT, GitHub API, Codespaces.

Diferenciais:

- Simula o futuro do desenvolvimento, IA como colaborador autônomo.

4. Hackathon Integrado com IA Generativa

Objetivo:
Realizar hackathon onde desafios, dados e avaliações são gerados por IA.

Conteúdo Teórico:

- O que é hackathon?
- Como IA pode criar e avaliar desafios.
- Gamificação e criatividade em ambientes de desenvolvimento.

Projeto Prático:

- IA gera desafios-surpresa, datasets sintéticos e requisitos.
- Alunos resolvem problemas, submetem soluções e recebem feedback da IA.
- Desafio extra: IA avalia código, sugere melhorias e pontua soluções.

Ferramentas:
Python, OpenAI API, Hugging Face, Codespaces.

Diferenciais:

- Experiência dinâmica e imprevisível, simulando situações reais de mercado.

5. Engenharia de Prompt e Segurança em IA

Objetivo:
Ensinar técnicas avançadas de engenharia de prompt e mitigação de riscos em IA.

Conteúdo Teórico:

- O que é engenharia de prompt?
- Exemplos de prompts encadeados e contextos longos.

- Riscos: prompt injection, vazamento de dados, segurança em IA.

Projeto Prático:

- Criar prompts que automatizam tarefas complexas.
- Testar limites das ferramentas de IA.
- Implementar proteções contra ataques de IA.

Ferramentas:
Python, OpenAI API, Gemini, Codespaces.

Diferenciais:

- Segurança em IA aplicada ao desenvolvimento cotidiano.

6. Integração com Hardware e IoT via iPad/Mac

Objetivo:
Mostrar como usar Python para controlar dispositivos físicos (Arduino, Raspberry Pi, sensores) a partir do Mac ou remotamente via Codespaces.

Conteúdo Teórico:

- O que é IoT?
- Exemplos de integração hardware-software.
- Protocolos de comunicação: serial, MQTT, HTTP.

Projeto Prático:

- Projeto de automação residencial, coleta de dados ambientais ou integração com APIs de dispositivos inteligentes.
- Desafio extra: dashboard web para monitorar dispositivos em tempo real.

Ferramentas:
Python, Arduino/Raspberry Pi, Streamlit/Flask, Codespaces.

Diferenciais:

- Une software, hardware e nuvem, expandindo o horizonte para além do código tradicional.

7. Portfólio Dinâmico com Deploy Automatizado

Objetivo:
Ensinar a criar um portfólio online que se atualiza automaticamente a cada novo projeto ou commit.

Conteúdo Teórico:

- O que é portfólio dinâmico?
- Ferramentas de deploy automatizado.
- Integração com GitHub Actions, Docker, Vercel, Netlify, Hugging Face Spaces.

Projeto Prático:

- Criar portfólio que mostra progresso em tempo real, dashboards, gráficos de evolução e badges de conquistas gerados por IA.
- Desafio extra: deploy automático de projetos e integração com redes sociais.

Ferramentas:
Python, GitHub Actions, Docker, Vercel/Netlify, Codespaces.

Diferenciais:

- Portfólio como vitrine viva, útil para carreira e networking.

8. Mentoria Virtual com IA Personalizada

Objetivo:
Configurar um "mentor virtual" (chatbot customizado) treinado com histórico de dúvidas, projetos e preferências do aluno.

Conteúdo Teórico:

- O que é mentoria virtual?
- Como treinar chatbots personalizados.
- Feedback adaptativo e acompanhamento de progresso.

Projeto Prático:

- Usar LLMs open-source ou APIs para criar mentor virtual acessível via iPad, Mac ou Codespaces.
- Desafio extra: mentor que simula entrevistas técnicas e sugere trilhas de estudo.

Ferramentas:
Python, Hugging Face, OpenAI API, Codespaces.

Diferenciais:

- Personalização extrema do aprendizado, com feedback contínuo e adaptativo.

9. Simulação de Incidentes e Resposta Rápida

Objetivo:
Incluir "incidentes simulados" (bugs críticos, falhas de segurança, perda de dados) gerados por IA para treinar diagnóstico e correção.

Conteúdo Teórico:

- O que é troubleshooting?
- Técnicas de resposta a crises.
- Documentação de incidentes e boas práticas de recuperação.

Projeto Prático:

- Ambiente Codespaces "infectado" com bug, aluno deve restaurar sistema usando ferramentas aprendidas.
- Desafio extra: documentar resposta e criar plano de prevenção.

Ferramentas:
Python, Codespaces, GitHub Actions.

Diferenciais:

- Treina habilidades essenciais para desenvolvedores avançados.

10. Exploração de Computação Quântica e IA

Objetivo:
Introduzir conceitos básicos de computação quântica e mostrar como IA pode acelerar pesquisas nessa área.

Conteúdo Teórico:

- O que é computação quântica?
- Algoritmos e simuladores (Qiskit).
- Aplicações de IA em pesquisa quântica.

Projeto Prático:

- Executar experimentos quânticos via Python, integrar resultados em dashboards e discutir aplicações futuras.
- Desafio extra: pipeline que une IA e computação quântica para resolver problemas reais.

Ferramentas:
Python, Qiskit, OpenAI API, Codespaces.

Diferenciais:

- Pouquíssimos cursos conectam desenvolvimento tradicional, IA e computação quântica.

Sugestão de Sequência Semanal

- **Semana 1-2:** Módulo 1 (Multimodal)
- **Semana 3:** Módulo 2 (IaC)
- **Semana 4:** Módulo 3 (Agentes Autônomos)
- **Semana 5:** Módulo 4 (Hackathon IA)
- **Semana 6:** Módulo 5 (Prompt/Security)
- **Semana 7:** Módulo 6 (IoT/Hardware)
- **Semana 8:** Módulo 7 (Portfólio Dinâmico)
- **Semana 9:** Módulo 8 (Mentoria Virtual)
- **Semana 10:** Módulo 9 (Incidentes Simulados)
- **Semana 11:** Módulo 10 (Quântica & IA)
- **Semana 12:** Projeto integrador, revisão, avaliação e próximos passos.

ROTEIRO DETALHADO v0.2

MÓDULO 1 – Desenvolvimento Multimodal com IA

Objetivo

Capacitar o aluno a criar aplicações que integram texto, imagem, áudio e vídeo usando APIs de IA multimodal.

Conteúdo Teórico

- O que é IA multimodal? Exemplos (assistentes, apps de acessibilidade, arte generativa, análise de vídeo).
- APIs e frameworks: OpenAI GPT-4o, Gemini, Stable Diffusion, Whisper, Hugging Face.
- Conceitos de pipeline multimodal: entrada, processamento, saída.
- Limitações e desafios (latência, custos, privacidade).

Plano de Aula

1. **Introdução multimodal:** Demonstração de apps que combinam voz, imagem e texto.
2. **Explorando APIs:** Como obter chaves, limites gratuitos, documentação.
3. **Processamento de áudio:** Transcrição com Whisper, reconhecimento de comandos.
4. **Geração de imagem:** Stable Diffusion, prompts criativos, controle de parâmetros.
5. **Perguntas sobre imagens:** Uso do GPT-4o Vision ou Gemini para análise visual.
6. **Integração em interface web:** Streamlit/Flask, upload de arquivos, exibição de resultados.
7. **Desafio extra:** Upload de vídeo, sumarização automática, geração de legendas.

Projeto Prático

- App Python que:
    - Recebe comandos por voz (Whisper).
    - Gera imagens a partir de texto (Stable Diffusion).
    - Responde perguntas sobre imagens (GPT-4o Vision).
    - Integra tudo em uma interface web (Streamlit ou Flask).
- **Desafio:** Permitir upload de áudio/vídeo e gerar respostas multimodais.

Ferramentas

- Python, Streamlit/Flask, APIs OpenAI/Gemini, Hugging Face Transformers, Docker, Codespaces.

Avaliação

- Entrega do app funcional.
- Demonstração de integração multimodal.
- Relatório explicando decisões técnicas e limitações encontradas.

MÓDULO 2 – Laboratório de Automação de Ambientes com IaC

Objetivo

Ensinar automação da criação de ambientes de desenvolvimento na nuvem usando Terraform, Ansible ou similares.

Conteúdo Teórico

- O que é Infraestrutura como Código (IaC)?
- Scripts de automação vs IaC declarativo.
- Provisionamento de containers, VMs, Codespaces, bancos de dados.
- Princípios de idempotência, versionamento e rollback.

Plano de Aula

1. **Introdução à IaC:** Exemplos práticos, benefícios para times.
2. **Terraform:** Conceitos de provider, resource, state.
3. **Ansible:** Playbooks, roles, inventário.
4. **Provisionando Codespaces e Docker:** Scripts para ambientes padronizados.
5. **Automatizando banco de dados:** Deploy de PostgreSQL/MySQL via IaC.
6. **CI/CD:** Integração com GitHub Actions para provisionamento automático.
7. **Desafio extra:** Deploy de app completo com um comando.

Projeto Prático

- Scripts Terraform/Ansible que:
    - Montam todo o ambiente do projeto (Codespace, containers, banco de dados, permissões).
    - Permitem replicação do ambiente com um comando.

- **Desafio:** Deploy automatizado de aplicações e pipelines CI/CD.

Ferramentas

- Terraform, Ansible, Docker, GitHub Actions, Codespaces.

Avaliação

- Provisionamento de ambiente funcional do zero.
- Documentação do fluxo de automação.
- Demonstração de rollback e atualização incremental.

MÓDULO 3 – Desafio de Programação com Agentes Autônomos

Objetivo

Construir e treinar agentes de IA que navegam pelo repositório, sugerem melhorias, abrem issues e fazem pull requests automaticamente.

Conteúdo Teórico

- O que são agentes autônomos? Exemplos: AutoGPT, LangChain Agents.
- Fluxo de trabalho: observação, decisão, ação.
- Riscos e limitações dos agentes autônomos (alucinação, permissões, ética).

Plano de Aula

1. **Introdução a agentes:** Diferença entre assistente e agente autônomo.
2. **LangChain/AutoGPT:** Instalação, configuração, primeiros fluxos.
3. **Acesso ao GitHub via API:** Permissões, tokens, limites.
4. **Análise de código automatizada:** Lint, sugestões de refatoração.
5. **Abertura de issues e PRs automáticos:** Fluxo de automação real.
6. **Desafio extra:** Agente que escreve documentação e executa testes.

Projeto Prático

- Usar LangChain ou AutoGPT para criar agente que:
    - Analisa código do projeto.
    - Sugere melhorias e abre issues no GitHub.

    - Faz pull requests com correções automáticas.
- **Desafio:** Agente que escreve documentação e testa código.

Ferramentas

- Python, LangChain, AutoGPT, GitHub API, Codespaces.

Avaliação

- Demonstração do agente em ação (vídeo ou logs).
- Relatório de limitações e sugestões de melhoria.
- Análise crítica dos riscos de automação.

MÓDULO 4 – Hackathon Integrado com IA Generativa

Objetivo

Realizar hackathon onde desafios, dados e avaliações são gerados por IA.

Conteúdo Teórico

- O que é hackathon?
- Como IA pode criar e avaliar desafios.
- Gamificação e criatividade em ambientes de desenvolvimento.

Plano de Aula

1. **Preparação:** Formação de equipes ou solo, regras do hackathon.
2. **Geração de desafios por IA:** Uso de GPT para criar problemas, datasets, restrições.
3. **Execução:** Desenvolvimento sob pressão, colaboração, uso de IA como "juiz".
4. **Avaliação automática:** IA pontua soluções, sugere melhorias.
5. **Feedback:** Discussão dos resultados, aprendizado coletivo.

Projeto Prático

- IA gera desafios-surpresa, datasets sintéticos e requisitos.
- Alunos resolvem problemas, submetem soluções e recebem feedback da IA.
- **Desafio extra:** IA avalia código, sugere melhorias e pontua soluções.

Ferramentas

- Python, OpenAI API, Hugging Face, Codespaces.

Avaliação

- Pontuação automática da IA.
- Apresentação dos projetos.
- Feedback dos colegas e da IA.

MÓDULO 5 – Engenharia de Prompt e Segurança em IA

Objetivo

Ensinar técnicas avançadas de engenharia de prompt e mitigação de riscos em IA.

Conteúdo Teórico

- O que é engenharia de prompt?
- Exemplos de prompts encadeados e contextos longos.
- Riscos: prompt injection, vazamento de dados, segurança em IA.

Plano de Aula

1. **Fundamentos de prompt:** Estrutura, contexto, exemplos.
2. **Prompts avançados:** Encadeamento, contexto longo, few-shot learning.
3. **Ataques e riscos:** Prompt injection, vazamento de informações.
4. **Mitigação:** Filtragem, validação, sandboxing.
5. **Desafio extra:** Criar prompts que automatizam tarefas complexas e são resistentes a ataques.

Projeto Prático

- Criar prompts que automatizam tarefas complexas.
- Testar limites das ferramentas de IA.
- Implementar proteções contra ataques de IA.

Ferramentas

- Python, OpenAI API, Gemini, Codespaces.

Avaliação

- Teste de prompts em cenários reais e adversariais.
- Relatório de segurança e mitigação.
- Demonstração de automação via prompt.

MÓDULO 6 – Integração com Hardware e IoT via iPad/Mac

Objetivo

Mostrar como usar Python para controlar dispositivos físicos (Arduino, Raspberry Pi, sensores) a partir do Mac ou remotamente via Codespaces.

Conteúdo Teórico

- O que é IoT?
- Exemplos de integração hardware-software.
- Protocolos de comunicação: serial, MQTT, HTTP.

Plano de Aula

1. **Introdução à IoT:** Exemplos práticos, aplicações reais.
2. **Python com hardware:** Bibliotecas (pyserial, gpiozero, paho-mqtt).
3. **Comunicação remota:** Codespaces acessando hardware via rede.
4. **Projetos práticos:** Automação residencial, coleta de dados ambientais.
5. **Dashboard web:** Monitoramento em tempo real com Streamlit/Flask.
6. **Desafio extra:** Integração com APIs de dispositivos inteligentes.

Projeto Prático

- Projeto de automação residencial, coleta de dados ambientais ou integração com APIs de dispositivos inteligentes.
- **Desafio:** Dashboard web para monitorar dispositivos em tempo real.

Ferramentas

- Python, Arduino/Raspberry Pi, Streamlit/Flask, Codespaces.

Avaliação

- Demonstração do hardware integrado.
- Dashboard funcional.
- Relatório de desafios e soluções técnicas.

MÓDULO 7 – Portfólio Dinâmico com Deploy Automatizado

Objetivo

Ensinar a criar um portfólio online que se atualiza automaticamente a cada novo projeto ou commit.

Conteúdo Teórico

- O que é portfólio dinâmico?
- Ferramentas de deploy automatizado.
- Integração com GitHub Actions, Docker, Vercel, Netlify, Hugging Face Spaces.

Plano de Aula

1. **Portfólio tradicional vs dinâmico:** Exemplos e benefícios.
2. **Deploy automatizado:** GitHub Actions, integração contínua.
3. **Dashboards e badges:** Visualização de progresso, conquistas.
4. **Integração com redes sociais:** Compartilhamento automático.
5. **Desafio extra:** Deploy automático de projetos e integração com APIs externas.

Projeto Prático

- Criar portfólio que mostra progresso em tempo real, dashboards, gráficos de evolução e badges de conquistas gerados por IA.
- **Desafio:** Deploy automático de projetos e integração com redes sociais.

Ferramentas

- Python, GitHub Actions, Docker, Vercel/Netlify, Codespaces.

Avaliação

- Portfólio online funcional.
- Deploy automatizado comprovado.
- Relatório de integração e automação.

MÓDULO 8 – Mentoria Virtual com IA Personalizada

Objetivo

Configurar um "mentor virtual" (chatbot customizado) treinado com histórico de dúvidas, projetos e preferências do aluno.

Conteúdo Teórico

- O que é mentoria virtual?
- Como treinar chatbots personalizados.
- Feedback adaptativo e acompanhamento de progresso.

Plano de Aula

1. **Chatbots vs mentores:** Diferenças e aplicações.
2. **Treinamento de LLMs:** Fine-tuning, embeddings, contexto personalizado.
3. **Integração com histórico do aluno:** Logs, preferências, projetos.
4. **Feedback adaptativo:** Sugestão de estudos, simulação de entrevistas técnicas.
5. **Desafio extra:** Mentor que acompanha progresso e sugere trilhas de estudo.

Projeto Prático

- Usar LLMs open-source ou APIs para criar mentor virtual acessível via iPad, Mac ou Codespaces.
- **Desafio:** Mentor que simula entrevistas técnicas e sugere trilhas de estudo.

Ferramentas

- Python, Hugging Face, OpenAI API, Codespaces.

Avaliação

- Demonstração do mentor em ação.
- Relatório de personalização e adaptação.

- Feedback do aluno sobre utilidade do mentor.

MÓDULO 9 – Simulação de Incidentes e Resposta Rápida

Objetivo

Incluir “incidentes simulados” (bugs críticos, falhas de segurança, perda de dados) gerados por IA para treinar diagnóstico e correção.

Conteúdo Teórico

- O que é troubleshooting?
- Técnicas de resposta a crises.
- Documentação de incidentes e boas práticas de recuperação.

Plano de Aula

1. **Incidentes reais:** Exemplos históricos, impactos.
2. **Simulação de bugs:** Geração automática de falhas por IA.
3. **Diagnóstico e correção:** Ferramentas, logs, rollback.
4. **Documentação:** Relato do incidente, plano de prevenção.
5. **Desafio extra:** Recuperação de ambiente em tempo limitado.

Projeto Prático

- Ambiente Codespaces “infectado” com bug, aluno deve restaurar sistema usando ferramentas aprendidas.
- **Desafio:** Documentar resposta e criar plano de prevenção.

Ferramentas

- Python, Codespaces, GitHub Actions.

Avaliação

- Tempo de resposta ao incidente.
- Qualidade da documentação.
- Efetividade das soluções aplicadas.

MÓDULO 10 – Exploração de Computação Quântica e IA

Objetivo

Introduzir conceitos básicos de computação quântica e mostrar como IA pode acelerar pesquisas nessa área.

Conteúdo Teórico

- O que é computação quântica?
- Algoritmos e simuladores (Qiskit).
- Aplicações de IA em pesquisa quântica.

Plano de Aula

1. **Fundamentos de quântica:** Qubits, portas lógicas, superposição.
2. **Simuladores:** Qiskit, execução local e em nuvem.
3. **Algoritmos clássicos vs quânticos:** Exemplos práticos.
4. **IA acelerando pesquisa:** Otimização de circuitos, análise de resultados.
5. **Desafio extra:** Pipeline que une IA e computação quântica para resolver problemas reais.

Projeto Prático

- Executar experimentos quânticos via Python, integrar resultados em dashboards e discutir aplicações futuras.
- **Desafio:** Pipeline que une IA e computação quântica para resolver problemas reais.

Ferramentas

- Python, Qiskit, OpenAI API, Codespaces.

Avaliação

- Execução de experimentos quânticos.
- Integração com IA para análise.
- Relatório de aplicações e limitações.

Como usar este roteiro

- Cada módulo pode ser trabalhado em 1-2 semanas, com encontros síncronos e atividades assíncronas.
- Os projetos práticos são incrementais e podem compor um portfólio final.
- Avaliações são sempre baseadas em entrega funcional, documentação e reflexão crítica.
- O curso pode ser adaptado para bootcamp intensivo, trilha de especialização ou formação continuada.

PASSO-A-PASSO v0.3

MÓDULO 1 – Desenvolvimento Multimodal com IA

Objetivo

Capacitar o aluno a criar aplicações que integram texto, imagem, áudio e vídeo usando APIs de IA multimodal.

Conteúdo Teórico

- O que é IA multimodal? Exemplos (assistentes, apps de acessibilidade, arte generativa, análise de vídeo).
- APIs e frameworks: OpenAI GPT-4o, Gemini, Stable Diffusion, Whisper, Hugging Face.
- Conceitos de pipeline multimodal: entrada, processamento, saída.
- Limitações e desafios (latência, custos, privacidade).

Plano de Aula

1. **Introdução multimodal:** Demonstração de apps que combinam voz, imagem e texto.
2. **Explorando APIs:** Como obter chaves, limites gratuitos, documentação.
3. **Processamento de áudio:** Transcrição com Whisper, reconhecimento de comandos.
4. **Geração de imagem:** Stable Diffusion, prompts criativos, controle de parâmetros.
5. **Perguntas sobre imagens:** Uso do GPT-4o Vision ou Gemini para análise visual.
6. **Integração em interface web:** Streamlit/Flask, upload de arquivos, exibição de resultados.
7. **Desafio extra:** Upload de vídeo, sumarização automática, geração de legendas.

Projeto Prático

- App Python que:
    - Recebe comandos por voz (Whisper).
    - Gera imagens a partir de texto (Stable Diffusion).
    - Responde perguntas sobre imagens (GPT-4o Vision).
    - Integra tudo em uma interface web (Streamlit ou Flask).
- **Desafio:** Permitir upload de áudio/vídeo e gerar respostas multimodais.

Ferramentas

- Python, Streamlit/Flask, APIs OpenAI/Gemini, Hugging Face Transformers, Docker, Codespaces.

Avaliação

- Entrega do app funcional.
- Demonstração de integração multimodal.
- Relatório explicando decisões técnicas e limitações encontradas.

MÓDULO 2 – Laboratório de Automação de Ambientes com IaC

Objetivo

Ensinar automação da criação de ambientes de desenvolvimento na nuvem usando Terraform, Ansible ou similares.

Conteúdo Teórico

- O que é Infraestrutura como Código (IaC)?
- Scripts de automação vs IaC declarativo.
- Provisionamento de containers, VMs, Codespaces, bancos de dados.
- Princípios de idempotência, versionamento e rollback.

Plano de Aula

1. **Introdução à IaC:** Exemplos práticos, benefícios para times.
2. **Terraform:** Conceitos de provider, resource, state.
3. **Ansible:** Playbooks, roles, inventário.
4. **Provisionando Codespaces e Docker:** Scripts para ambientes padronizados.
5. **Automatizando banco de dados:** Deploy de PostgreSQL/MySQL via IaC.
6. **CI/CD:** Integração com GitHub Actions para provisionamento automático.
7. **Desafio extra:** Deploy de app completo com um comando.

Projeto Prático

- Scripts Terraform/Ansible que:
    - Montam todo o ambiente do projeto (Codespace, containers, banco de dados, permissões).
    - Permitem replicação do ambiente com um comando.

- **Desafio:** Deploy automatizado de aplicações e pipelines CI/CD.

Ferramentas

- Terraform, Ansible, Docker, GitHub Actions, Codespaces.

Avaliação

- Provisionamento de ambiente funcional do zero.
- Documentação do fluxo de automação.
- Demonstração de rollback e atualização incremental.

MÓDULO 3 – Desafio de Programação com Agentes Autônomos

Objetivo

Construir e treinar agentes de IA que navegam pelo repositório, sugerem melhorias, abrem issues e fazem pull requests automaticamente.

Conteúdo Teórico

- O que são agentes autônomos? Exemplos: AutoGPT, LangChain Agents.
- Fluxo de trabalho: observação, decisão, ação.
- Riscos e limitações dos agentes autônomos (alucinação, permissões, ética).

Plano de Aula

1. **Introdução a agentes:** Diferença entre assistente e agente autônomo.
2. **LangChain/AutoGPT:** Instalação, configuração, primeiros fluxos.
3. **Acesso ao GitHub via API:** Permissões, tokens, limites.
4. **Análise de código automatizada:** Lint, sugestões de refatoração.
5. **Abertura de issues e PRs automáticos:** Fluxo de automação real.
6. **Desafio extra:** Agente que escreve documentação e executa testes.

Projeto Prático

- Usar LangChain ou AutoGPT para criar agente que:
    - Analisa código do projeto.
    - Sugere melhorias e abre issues no GitHub.

    - Faz pull requests com correções automáticas.
- **Desafio:** Agente que escreve documentação e testa código.

Ferramentas

- Python, LangChain, AutoGPT, GitHub API, Codespaces.

Avaliação

- Demonstração do agente em ação (vídeo ou logs).
- Relatório de limitações e sugestões de melhoria.
- Análise crítica dos riscos de automação.

MÓDULO 4 – Hackathon Integrado com IA Generativa

Objetivo

Realizar hackathon onde desafios, dados e avaliações são gerados por IA.

Conteúdo Teórico

- O que é hackathon?
- Como IA pode criar e avaliar desafios.
- Gamificação e criatividade em ambientes de desenvolvimento.

Plano de Aula

1. **Preparação:** Formação de equipes ou solo, regras do hackathon.
2. **Geração de desafios por IA:** Uso de GPT para criar problemas, datasets, restrições.
3. **Execução:** Desenvolvimento sob pressão, colaboração, uso de IA como "juiz".
4. **Avaliação automática:** IA pontua soluções, sugere melhorias.
5. **Feedback:** Discussão dos resultados, aprendizado coletivo.

Projeto Prático

- IA gera desafios-surpresa, datasets sintéticos e requisitos.
- Alunos resolvem problemas, submetem soluções e recebem feedback da IA.
- **Desafio extra:** IA avalia código, sugere melhorias e pontua soluções.

Ferramentas

- Python, OpenAI API, Hugging Face, Codespaces.

Avaliação

- Pontuação automática da IA.
- Apresentação dos projetos.
- Feedback dos colegas e da IA.

MÓDULO 5 – Engenharia de Prompt e Segurança em IA

Objetivo

Ensinar técnicas avançadas de engenharia de prompt e mitigação de riscos em IA.

Conteúdo Teórico

- O que é engenharia de prompt?
- Exemplos de prompts encadeados e contextos longos.
- Riscos: prompt injection, vazamento de dados, segurança em IA.

Plano de Aula

1. **Fundamentos de prompt:** Estrutura, contexto, exemplos.
2. **Prompts avançados:** Encadeamento, contexto longo, few-shot learning.
3. **Ataques e riscos:** Prompt injection, vazamento de informações.
4. **Mitigação:** Filtragem, validação, sandboxing.
5. **Desafio extra:** Criar prompts que automatizam tarefas complexas e são resistentes a ataques.

Projeto Prático

- Criar prompts que automatizam tarefas complexas.
- Testar limites das ferramentas de IA.
- Implementar proteções contra ataques de IA.

Ferramentas

- Python, OpenAI API, Gemini, Codespaces.

Avaliação

- Teste de prompts em cenários reais e adversariais.
- Relatório de segurança e mitigação.
- Demonstração de automação via prompt.

MÓDULO 6 – Integração com Hardware e IoT via iPad/Mac

Objetivo

Mostrar como usar Python para controlar dispositivos físicos (Arduino, Raspberry Pi, sensores) a partir do Mac ou remotamente via Codespaces.

Conteúdo Teórico

- O que é IoT?
- Exemplos de integração hardware-software.
- Protocolos de comunicação: serial, MQTT, HTTP.

Plano de Aula

1. **Introdução à IoT:** Exemplos práticos, aplicações reais.
2. **Python com hardware:** Bibliotecas (pyserial, gpiozero, paho-mqtt).
3. **Comunicação remota:** Codespaces acessando hardware via rede.
4. **Projetos práticos:** Automação residencial, coleta de dados ambientais.
5. **Dashboard web:** Monitoramento em tempo real com Streamlit/Flask.
6. **Desafio extra:** Integração com APIs de dispositivos inteligentes.

Projeto Prático

- Projeto de automação residencial, coleta de dados ambientais ou integração com APIs de dispositivos inteligentes.
- **Desafio:** Dashboard web para monitorar dispositivos em tempo real.

Ferramentas

- Python, Arduino/Raspberry Pi, Streamlit/Flask, Codespaces.

Avaliação

- Demonstração do hardware integrado.
- Dashboard funcional.
- Relatório de desafios e soluções técnicas.

MÓDULO 7 – Portfólio Dinâmico com Deploy Automatizado

Objetivo

Ensinar a criar um portfólio online que se atualiza automaticamente a cada novo projeto ou commit.

Conteúdo Teórico

- O que é portfólio dinâmico?
- Ferramentas de deploy automatizado.
- Integração com GitHub Actions, Docker, Vercel, Netlify, Hugging Face Spaces.

Plano de Aula

1. **Portfólio tradicional vs dinâmico:** Exemplos e benefícios.
2. **Deploy automatizado:** GitHub Actions, integração contínua.
3. **Dashboards e badges:** Visualização de progresso, conquistas.
4. **Integração com redes sociais:** Compartilhamento automático.
5. **Desafio extra:** Deploy automático de projetos e integração com APIs externas.

Projeto Prático

- Criar portfólio que mostra progresso em tempo real, dashboards, gráficos de evolução e badges de conquistas gerados por IA.
- **Desafio:** Deploy automático de projetos e integração com redes sociais.

Ferramentas

- Python, GitHub Actions, Docker, Vercel/Netlify, Codespaces.

Avaliação

- Portfólio online funcional.
- Deploy automatizado comprovado.
- Relatório de integração e automação.

MÓDULO 8 – Mentoria Virtual com IA Personalizada

Objetivo

Configurar um "mentor virtual" (chatbot customizado) treinado com histórico de dúvidas, projetos e preferências do aluno.

Conteúdo Teórico

- O que é mentoria virtual?
- Como treinar chatbots personalizados.
- Feedback adaptativo e acompanhamento de progresso.

Plano de Aula

1. **Chatbots vs mentores:** Diferenças e aplicações.
2. **Treinamento de LLMs:** Fine-tuning, embeddings, contexto personalizado.
3. **Integração com histórico do aluno:** Logs, preferências, projetos.
4. **Feedback adaptativo:** Sugestão de estudos, simulação de entrevistas técnicas.
5. **Desafio extra:** Mentor que acompanha progresso e sugere trilhas de estudo.

Projeto Prático

- Usar LLMs open-source ou APIs para criar mentor virtual acessível via iPad, Mac ou Codespaces.
- **Desafio:** Mentor que simula entrevistas técnicas e sugere trilhas de estudo.

Ferramentas

- Python, Hugging Face, OpenAI API, Codespaces.

Avaliação

- Demonstração do mentor em ação.
- Relatório de personalização e adaptação.

- Feedback do aluno sobre utilidade do mentor.

MÓDULO 9 – Simulação de Incidentes e Resposta Rápida

Objetivo

Incluir “incidentes simulados” (bugs críticos, falhas de segurança, perda de dados) gerados por IA para treinar diagnóstico e correção.

Conteúdo Teórico

- O que é troubleshooting?
- Técnicas de resposta a crises.
- Documentação de incidentes e boas práticas de recuperação.

Plano de Aula

1. **Incidentes reais:** Exemplos históricos, impactos.
2. **Simulação de bugs:** Geração automática de falhas por IA.
3. **Diagnóstico e correção:** Ferramentas, logs, rollback.
4. **Documentação:** Relato do incidente, plano de prevenção.
5. **Desafio extra:** Recuperação de ambiente em tempo limitado.

Projeto Prático

- Ambiente Codespaces “infectado” com bug, aluno deve restaurar sistema usando ferramentas aprendidas.
- **Desafio:** Documentar resposta e criar plano de prevenção.

Ferramentas

- Python, Codespaces, GitHub Actions.

Avaliação

- Tempo de resposta ao incidente.
- Qualidade da documentação.
- Efetividade das soluções aplicadas.

MÓDULO 10 – Exploração de Computação Quântica e IA

Objetivo

Introduzir conceitos básicos de computação quântica e mostrar como IA pode acelerar pesquisas nessa área.

Conteúdo Teórico

- O que é computação quântica?
- Algoritmos e simuladores (Qiskit).
- Aplicações de IA em pesquisa quântica.

Plano de Aula

1. **Fundamentos de quântica:** Qubits, portas lógicas, superposição.
2. **Simuladores:** Qiskit, execução local e em nuvem.
3. **Algoritmos clássicos vs quânticos:** Exemplos práticos.
4. **IA acelerando pesquisa:** Otimização de circuitos, análise de resultados.
5. **Desafio extra:** Pipeline que une IA e computação quântica para resolver problemas reais.

Projeto Prático

- Executar experimentos quânticos via Python, integrar resultados em dashboards e discutir aplicações futuras.
- **Desafio:** Pipeline que une IA e computação quântica para resolver problemas reais.

Ferramentas

- Python, Qiskit, OpenAI API, Codespaces.

Avaliação

- Execução de experimentos quânticos.
- Integração com IA para análise.
- Relatório de aplicações e limitações.

Como usar este roteiro

- Cada módulo pode ser trabalhado em 1-2 semanas, com encontros síncronos e atividades assíncronas.
- Os projetos práticos são incrementais e podem compor um portfólio final.
- Avaliações são sempre baseadas em entrega funcional, documentação e reflexão crítica.
- O curso pode ser adaptado para bootcamp intensivo, trilha de especialização ou formação continuada.

Detalhamento v0.4

- Estrutura de aulas semanais
- Teoria aprofundada
- Práticas guiadas e desafios
- Ferramentas e links recomendados
- Sugestões de avaliação e documentação
- Dicas para adaptação em bootcamp, especialização ou formação continuada

MÓDULO 1 – Desenvolvimento Multimodal com IA

Semana 1: Introdução e Fundamentos

- **Teoria:** O que é IA multimodal? Exemplos reais (assistentes, apps de acessibilidade, arte generativa, análise de vídeo).
- **Prática:** Demonstração de apps multimodais. Discussão sobre aplicações e limitações (latência, custos, privacidade).
- **Ferramentas:** Python, Streamlit/Flask, APIs OpenAI GPT-4o, Gemini, Stable Diffusion, Whisper, Hugging Face.

Semana 2: APIs e Pipeline Multimodal

- **Teoria:** Como funcionam APIs de IA multimodal. Pipeline: entrada, processamento, saída.
- **Prática:** Obtenção de chaves de API, configuração de ambiente Docker/Codespaces.
- **Exercício:** Testar transcrição de áudio com Whisper, geração de imagem com Stable Diffusion, perguntas sobre imagens com GPT-4o Vision.
- **Desafio:** Integrar tudo em uma interface web (Streamlit ou Flask).

Semana 3: Projeto Prático e Avaliação

- **Projeto:** App Python que recebe comandos por voz, gera imagens, responde perguntas sobre imagens e integra tudo em uma interface web.
- **Desafio extra:** Upload de áudio/vídeo e geração de respostas multimodais.
- **Avaliação:** Entrega do app funcional, relatório explicando decisões técnicas e limitações encontradas.

MÓDULO 2 – Laboratório de Automação de Ambientes com IaC

Semana 4: Fundamentos de IaC

- **Teoria:** O que é Infraestrutura como Código (IaC)? Scripts de automação vs IaC declarativo. Princípios de idempotência, versionamento e rollback.
- **Prática:** Exemplos práticos com Terraform e Ansible.

Semana 5: Provisionamento Automatizado

- **Teoria:** Provisionamento de containers, VMs, Codespaces, bancos de dados.
- **Prática:** Scripts Terraform/Ansible para montar ambientes completos (Codespace, containers, banco de dados, permissões).
- **Exercício:** Replicar ambiente com um comando.

Semana 6: CI/CD e Deploy Automatizado

- **Teoria:** Integração com GitHub Actions para provisionamento automático.
- **Prática:** Deploy automatizado de aplicações e pipelines CI/CD.
- **Avaliação:** Provisionamento de ambiente funcional do zero, documentação do fluxo de automação, demonstração de rollback e atualização incremental.

MÓDULO 3 – Desafio de Programação com Agentes Autônomos

Semana 7: Agentes Autônomos e Fluxos

- **Teoria:** O que são agentes autônomos? Exemplos: AutoGPT, LangChain Agents. Fluxo de trabalho: observação, decisão, ação. Riscos e limitações (alucinação, permissões, ética).
- **Prática:** Instalação e configuração de LangChain/AutoGPT.

Semana 8: Automação no GitHub

- **Teoria:** Acesso ao GitHub via API: permissões, tokens, limites.
- **Prática:** Agente que analisa código, sugere melhorias, abre issues e faz pull requests automáticos.

- **Desafio:** Agente que escreve documentação e executa testes.

Semana 9: Avaliação e Reflexão

- **Avaliação:** Demonstração do agente em ação (vídeo ou logs), relatório de limitações e sugestões de melhoria, análise crítica dos riscos de automação.

MÓDULO 4 – Hackathon Integrado com IA Generativa

Semana 10: Preparação e Geração de Desafios

- **Teoria:** O que é hackathon? Como IA pode criar e avaliar desafios. Gamificação e criatividade.
- **Prática:** Formação de equipes ou solo, regras do hackathon. Uso de GPT para criar problemas, datasets, restrições.

Semana 11: Execução e Avaliação

- **Prática:** Desenvolvimento sob pressão, colaboração, uso de IA como "juiz". IA pontua soluções, sugere melhorias.
- **Desafio:** IA avalia código, sugere melhorias e pontua soluções.

Semana 12: Feedback e Apresentação

- **Avaliação:** Pontuação automática da IA, apresentação dos projetos, feedback dos colegas e da IA.

MÓDULO 5 – Engenharia de Prompt e Segurança em IA

Semana 13: Fundamentos e Riscos

- **Teoria:** O que é engenharia de prompt? Exemplos de prompts encadeados e contextos longos. Riscos: prompt injection, vazamento de dados, segurança em IA.
- **Prática:** Estrutura, contexto, exemplos de prompts.

Semana 14: Prompts Avançados e Mitigação

- **Teoria:** Encadeamento, contexto longo, few-shot learning. Ataques e riscos: prompt injection, vazamento de informações.
- **Prática:** Filtragem, validação, sandboxing.

Semana 15: Projeto Prático e Avaliação

- **Projeto:** Criar prompts que automatizam tarefas complexas e são resistentes a ataques.
- **Avaliação:** Teste de prompts em cenários reais e adversariais, relatório de segurança e mitigação, demonstração de automação via prompt.

MÓDULO 6 – Integração com Hardware e IoT via iPad/Mac

Semana 16: Fundamentos de IoT

- **Teoria:** O que é IoT? Exemplos de integração hardware-software. Protocolos de comunicação: serial, MQTT, HTTP.
- **Prática:** Bibliotecas Python (pyserial, gpiozero, paho-mqtt).

Semana 17: Projetos Práticos

- **Prática:** Automação residencial, coleta de dados ambientais, integração com APIs de dispositivos inteligentes.
- **Desafio:** Dashboard web para monitorar dispositivos em tempo real.

Semana 18: Avaliação

- **Avaliação:** Demonstração do hardware integrado, dashboard funcional, relatório de desafios e soluções técnicas.

MÓDULO 7 – Portfólio Dinâmico com Deploy Automatizado

Semana 19: Portfólio Tradicional vs Dinâmico

- **Teoria:** O que é portfólio dinâmico? Ferramentas de deploy automatizado. Integração com GitHub Actions, Docker, Vercel, Netlify, Hugging Face Spaces.
- **Prática:** Exemplos e benefícios.

Semana 20: Deploy Automatizado e Dashboards

- **Prática:** GitHub Actions, integração contínua. Dashboards e badges: visualização de progresso, conquistas. Compartilhamento automático em redes sociais.

Semana 21: Projeto Prático e Avaliação

- **Projeto:** Criar portfólio que mostra progresso em tempo real, dashboards, gráficos de evolução e badges de conquistas gerados por IA.
- **Avaliação:** Portfólio online funcional, deploy automatizado comprovado, relatório de integração e automação.

MÓDULO 8 – Mentoria Virtual com IA Personalizada

Semana 22: Fundamentos de Mentoria Virtual

- **Teoria:** O que é mentoria virtual? Como treinar chatbots personalizados. Feedback adaptativo e acompanhamento de progresso.
- **Prática:** Diferenças entre chatbots e mentores.

Semana 23: Treinamento e Integração

- **Prática:** Fine-tuning, embeddings, contexto personalizado. Integração com histórico do aluno: logs, preferências, projetos.

Semana 24: Projeto Prático e Avaliação

- **Projeto:** Mentor virtual acessível via iPad, Mac ou Codespaces. Simulação de entrevistas técnicas e sugestão de trilhas de estudo.
- **Avaliação:** Demonstração do mentor em ação, relatório de personalização e adaptação, feedback do aluno sobre utilidade do mentor.

MÓDULO 9 – Simulação de Incidentes e Resposta Rápida

Semana 25: Incidentes Reais e Simulação

- **Teoria:** O que é troubleshooting? Técnicas de resposta a crises. Documentação de incidentes e boas práticas de recuperação.
- **Prática:** Exemplos históricos, impactos.

Semana 26: Diagnóstico e Correção

- **Prática:** Geração automática de falhas por IA. Diagnóstico e correção: ferramentas, logs, rollback.

Semana 27: Projeto Prático e Avaliação

- **Projeto:** Ambiente Codespaces "infectado" com bug, aluno deve restaurar sistema usando ferramentas aprendidas. Documentar resposta e criar plano de prevenção.
- **Avaliação:** Tempo de resposta ao incidente, qualidade da documentação, efetividade das soluções aplicadas.

MÓDULO 10 – Exploração de Computação Quântica e IA

Semana 28: Fundamentos de Quântica

- **Teoria:** O que é computação quântica? Algoritmos e simuladores (Qiskit). Aplicações de IA em pesquisa quântica.
- **Prática:** Qubits, portas lógicas, superposição.

Semana 29: Simuladores e Algoritmos

- **Prática:** Qiskit, execução local e em nuvem. Algoritmos clássicos vs quânticos. IA acelerando pesquisa: otimização de circuitos, análise de resultados.

Semana 30: Projeto Prático e Avaliação

- **Projeto:** Executar experimentos quânticos via Python, integrar resultados em dashboards e discutir aplicações futuras. Pipeline que une IA e computação quântica para resolver problemas reais.
- **Avaliação:** Execução de experimentos quânticos, integração com IA para análise, relatório de aplicações e limitações.

Semana 31-32: Projeto Integrador, Revisão e Próximos Passos

- **Projeto Final:** Escolha de tema integrador (ex: dashboard multimodal com IA, automação IoT, agente autônomo para portfólio, etc.).
- **Revisão:** Recapitulação dos principais aprendizados, esclarecimento de dúvidas, avaliação prática e teórica.
- **Feedback:** Discussão sobre evolução, próximos passos, sugestões de especialização.
- **Certificação:** Entrega de certificado ou reconhecimento formal.

Recursos e Links Úteis

- OpenAI API
- Hugging Face Transformers
- Stable Diffusion
- Qiskit
- Streamlit
- Terraform
- Ansible
- LangChain
- AutoGPT
- GitHub Actions
- Codespaces