## Laboratório Calculadora Simples

**Projeto:** Uma calculadora modular e orientada a objetos com funcionalidades crescentes.

**Objetivo didático:** Mostrar como o uso do Prefect traz:

* Clareza no versionamento e evolução de código.
* Automação do processo de CI/CD (push → deploy).
* Observabilidade (logs, histórico, status, falhas).
* Controle granular com retries, agendamento, dependências.
* Profissionalismo no ciclo de entrega contínua.

**🚀 Estrutura do novo laboratório**

| **Fase** | **Funcionalidade** | **Justificativa técnica** |
| --- | --- | --- |
| V0 | Adição e subtração | MVP básico, execução local |
| V1 | + Multiplicação | Atualização simples + novo deploy |
| V2 | + Divisão | Introduz tratamento de exceções |
| V3 | + Porcentagem | Função que simula cálculo financeiro simples |
| V4 | CLI simples ou Web API (opcional) | Interface de uso real do sistema |
| VFinal | Observabilidade via Prefect | Histórico, versionamento, controle de execução |

**📁 Estrutura sugerida do projeto**

bash

CopiarEditar

calculator\_prefect/

├── calculator/

│ ├── \_\_init\_\_.py

│ ├── base.py # classe abstrata / interface

│ ├── operations.py # implementação OO das operações

├── flows/

│ └── run\_calculator.py # fluxo Prefect que orquestra chamadas

├── tests/

│ └── test\_operations.py

├── .github/workflows/

│ └── deploy.yml

├── prefect.yaml

├── requirements.txt

└── README.md

**🧪 Diferenciais didáticos**

* Mostra evolução real de um projeto de software.
* Permite aplicar CI/CD a um sistema **que cresce com as entregas**.
* Pode ser integrado com testes, cobertura de código e até Sentry.
* Dá espaço para introduzir conceitos como: modularização, boas práticas, automação e versionamento.

**📝 E no relatório?**

Posso incluir:

* ✅ Nova seção com objetivo e plano do novo laboratório.
* ✅ Descrição detalhada de cada etapa (do V0 ao VFinal).
* ✅ Explicação técnica sobre o valor do Prefect em cada evolução.
* ✅ Código comentado das versões incrementais.

Se estiver de acordo, posso:

1. Criar o esqueleto da estrutura do projeto.
2. Escrever o prefect.yaml inicial.
3. Gerar o deploy.yml do GitHub.
4. Adicionar a nova seção no relatório .docx como parte do conteúdo didático.

**✅ Confirmação da Estrutura Didática do Novo Laboratório**

**🎯 Dois eixos centrais:**

**1. Orquestração com Prefect — O "Por quê"**

Mostrar aos alunos que **orquestrar não é apenas automatizar**, mas sim:

* **Controlar, versionar, auditar e escalar** workflows.
* Garantir **resiliência com retries**, **condições de execução**, agendamentos e visualização clara do pipeline.
* Evitar que falhas silenciosas prejudiquem a produção (monitoramento e logs).
* Preparar o projeto para integração com ferramentas como S3, bancos de dados, serviços externos, ML pipelines, etc.

**2. Engenharia de Software Moderna — O "Como"**

Construir um projeto:

* Modular.
* Escalável.
* Orientado a objetos.
* Com documentação, testes e CI/CD.

A proposta da **calculadora evolutiva** (com adição → subtração → multiplicação → divisão → porcentagem) **é perfeita** para ilustrar isso em pequenas doses funcionais, com cada versão gerando um novo deployment automatizado.

**✍️ Ação Imediata**

Proponho criar um novo documento com o seguinte conteúdo:

**🧩 Nome: Calculadora Orquestrada - Projeto Didático com Prefect**

E nele vamos estruturar:

1. **Objetivo pedagógico**: detalhar esses dois eixos.
2. **Etapas do projeto**:
   * V0 (adição e subtração)
   * V1 (+ multiplicação)
   * V2 (+ divisão com tratamento de erro)
   * V3 (+ porcentagem, simulação financeira)
   * V4 (CLI simples)
3. **Como o Prefect entra em cada etapa**.
4. **Pipeline de automação GitHub Actions** e gatilho por push.
5. **Estrutura modular do código (orientada a objetos)**.
6. **Visão final do ganho de orquestrar e versionar projetos reais.**

# Laboratório Didático: Orquestração com Prefect aplicada ao Desenvolvimento de uma Calculadora Modular

**🎯 Objetivo**

Desenvolver um laboratório didático que evidencie a importância da orquestração de tarefas com Prefect, aliado às boas práticas de Engenharia de Software. O laboratório deverá mostrar o ciclo de vida completo de um projeto de software: desde a fase de projeto (fundamental e anterior à codificação), passando pelo desenvolvimento modular orientado a objetos, até a automação de deploys e execuções usando GitHub Actions e Prefect Cloud. A proposta também busca destacar a importância da reprodutibilidade, escalabilidade, organização de sistemas reais e a necessidade de um bom projeto da solução antes de iniciar a codificação.

**🧩 Justificativa**

Alunos e profissionais iniciantes geralmente subestimam a importância de ferramentas de orquestração, acreditando que apenas "rodar um script" seja suficiente. Este laboratório irá demonstrar, de maneira prática e visual, os reais ganhos de controle, rastreabilidade, automação e confiabilidade trazidos por uma ferramenta como o Prefect. Simultaneamente, a proposta reforça a importância do projeto da solução e da aplicação das melhores práticas de Engenharia de Software desde o início do desenvolvimento, evidenciando o papel das etapas de análise, modelagem e planejamento no sucesso da implementação.

**🏗️ Estrutura Didática (Etapas)**

**✅ Etapa 0 — Preparação do Ambiente**

* Instalação e configuração do ambiente virtual, dependências e Prefect.
* Criação do repositório no GitHub com estrutura inicial.
* Testes de conectividade com Prefect Cloud.

**🔸 Etapa 1 — Projeto da Solução e Estrutura Base**

* Definição do escopo inicial: Calculadora com operações básicas (adição e subtração).
* Uso de ferramentas de modelagem (Canvas, Lucidchart, draw.io).
* Estrutura modular e orientada a objetos.
* Criação da estrutura de diretórios e arquivos principais.
* Documentação do projeto com UML simples (classes e fluxo).

**🔸 Etapa 2 — Primeiro Deploy (Adição e Subtração)**

* Implementação inicial das operações básicas.
* Testes locais com Prefect Flow.
* Deploy da primeira versão com GitHub Actions.

**🔸 Etapa 3 — Ampliação Funcional (Multiplicação e Divisão)**

* Expansão incremental do projeto.
* Refatorações, modularização e tratamento de erros.
* Novo commit + push → novo deploy automatizado.

**🔸 Etapa 4 — Finalização de Funcionalidades (Porcentagem)**

* Adição da última funcionalidade planejada.
* Validação final do comportamento.
* Push final e execução orquestrada via Prefect Cloud.

**🔸 Etapa 5 — Integração com Prefect e Visualização**

* Conversão dos scripts em flows.
* Observação da execução e logs no painel Prefect.
* Avaliação dos ganhos e comparação com execução local/manual.

**🔍 Metodologia**

* **Projeto da solução antes do código**, com apoio de ferramentas de modelagem (Canvas, Lucidchart, draw.io).
* Aplicação de princípios de modularidade, encapsulamento e orientação a objetos.
* Uso de diagramas UML e documentação progressiva.
* Git e GitHub como base de versionamento e integração.
* Prefect para controle de execução e rastreabilidade.
* GitHub Actions para automação do deploy e execução.
* Avaliação visual e técnica dos resultados com e sem orquestração.
* Reflexão crítica ao fim de cada etapa sobre os ganhos obtidos.

**🧭 Resultado Esperado**

* Um projeto de software simples, mas tecnicamente bem estruturado, escalável e modular.
* Orquestração eficaz com Prefect, evidenciando benefícios de automação e rastreabilidade.
* CI/CD funcional com deploy automático a partir do GitHub.
* Relatório técnico final detalhando todo o processo, decisões e aprendizados.

**✅ Pontos-Chave**

* Prefect é mais do que um agendador: é uma ferramenta de orquestração e confiabilidade.
* Engenharia de Software começa com **projeto da solução**, não com código.
* Boas práticas geram código reusável, manutenível e escalável.
* CI/CD não é um luxo — é um diferencial competitivo real.

**🧠 Observação Didática**

Este laboratório poderá ser adaptado para diferentes níveis (graduação, pós) ajustando o grau de complexidade da calculadora e os requisitos de automação. Também poderá ser expandido com integração a banco de dados, autenticação ou visualização via Dash/Streamlit em etapas futuras.

## Estrutura Didática do Repositório (com explicações)

### ✅ Etapa 0 — Preparação do Ambiente e Documentação Inicial

🧭 Contexto

Estamos iniciando o projeto calculadora\_simples, cujo objetivo final é demonstrar a importância da orquestração com Prefect, aliada às boas práticas de Engenharia de Software.

O projeto será construído incrementalmente, com deploys progressivos após cada nova funcionalidade, simulando um ciclo real de evolução de sistemas.

### 🔹 Ambiente de Desenvolvimento

* 📁 Repositório: calculadora\_simples
* 💻 Ambiente virtual: DEV
* 🐍 Python: 3.12.5
* 📂 Diretório atual: /Users/accol/Library/Mobile Documents/com~apple~CloudDocs/UNIVERSIDADES/UFF/CARREIRA\_POS/PPGEET/ORQUESTRACAO/LABORATORIO/calculadora\_simples

### 📁 Estrutura Inicial do Projeto

Texto

O conteúdo gerado por IA pode estar incorreto.

**✅** Ações obrigatórias nesta etapa

#### 1. Criar o arquivo docs/readme.txt

Este arquivo conterá a **especificação da versão inicial da calculadora**, com foco apenas em **adição e subtração**. Deve conter:

* Nome do sistema
* Objetivo
* Funcionalidades mínimas
* Público-alvo
* Forma de uso esperada

#### 2. Gerar requirements.txt

Com os pacotes usados no projeto e versões fixas. Incluir:

prefect==2.16.4

griffe==0.35.2

Outros pacotes serão adicionados conforme a evolução.

#### 3. Configurar infra/prefect.yaml

Este arquivo será responsável por indicar como o Prefect deve executar os flows. Terá a configuração git\_clone para repositório GitHub.

#### 4. Criar .gitignore e .prefectignore

Para evitar que arquivos locais/ocultos sejam versionados:

Texto

O conteúdo gerado por IA pode estar incorreto.

#### 5. Visibilidade no VSCode (macOS)

Para visualizar arquivos ocultos como .gitignore e .prefectignore:

1. Pressione ⌘ + , para abrir Preferências;
2. Busque por files.exclude e certifique-se que arquivos como .git\* não estão ocultos;
3. Alternativamente, abra a paleta de comandos (⌘ + Shift + P) e busque por Preferences: Open Settings (JSON) e ajuste:

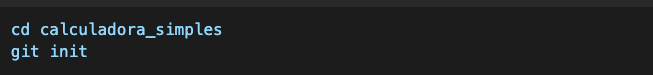
Uma imagem contendo Forma

O conteúdo gerado por IA pode estar incorreto.

#### 6. Criar repositório remoto no GitHub

* Nome: CALCULADORA\_SIMPLES
* Inicializar sem README (para evitar conflito)

#### 7. Inicializar versionamento com Git localmente



#### 8. Conectar repositório remoto



#### 9. Primeiro commit e push

Texto

O conteúdo gerado por IA pode estar incorreto.

## 📘 ETAPA 1 — Modelagem da Solução e Projeto da Calculadora OO

### 🎯 Objetivo

Desenhar e documentar a arquitetura inicial da calculadora com base nos princípios da Engenharia de Software, aplicando boas práticas de modularização, orientação a objetos e escalabilidade. Antes de codificar, será realizado o **projeto da solução**, que norteará todas as etapas seguintes.

### 🧠 Justificativa

É fundamental que o projeto de um sistema, por mais simples que pareça, seja realizado com base em boas práticas. Isso inclui:

* Clareza nos requisitos e escopo
* Organização modular e reutilizável
* Documentação desde o início
* Adoção de estratégias que favoreçam o crescimento do projeto

Além disso, será evidenciada a **importância da orquestração** em ciclos de evolução, por meio de deploys programados com o Prefect a cada nova funcionalidade adicionada.

### 🧱 Ferramentas e Estratégias Utilizadas

* **Modelagem visual:** ferramenta https://app.diagrams.net para os diagramas
* **Canvas de Projeto:** descrevendo contexto, escopo, usuários e funcionalidades
* **UML simplificada:** para representação das classes e modularização
* **Documentação contínua:** no diretório docs/

### 🛠️ Tarefas desta Etapa

**1. Criar o Canvas do Projeto (arquivo .md ou .pdf em docs/)**

* Nome do sistema
* Problema que resolve
* Usuários
* Funcionalidades esperadas
* Restrições e premissas

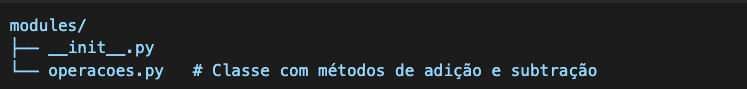
**2. Criar o Diagrama de Classes Inicial**

* Representar as classes Calculadora (base), e Operacoes
* Métodos públicos: somar(), subtrair()
* Salvar como imagem em docs/diagrama\_classes\_v1.png

**3. Documentar Arquitetura da Solução**

* Arquivo docs/arquitetura.md
* Descrever:
  + Modularização pretendida (modules/, flows/, tests/)
  + Justificativa das separações
  + Como será feita a expansão futura (multiplicação, divisão, porcentagem)

### 4. Criar estrutura de classes base em modules/

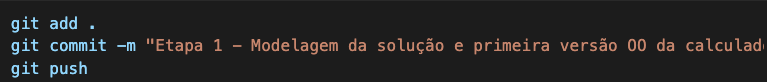


## 5. Planejar primeiro Flow em flows/

* Nome: flow\_versao1.py
* Responsável por orquestrar a execução da versão inicial da calculadora

## 6. Atualizar requirements.txt se necessário

## Commit e Push da Etapa 1



## Ações da ETAPA 1

## 📌 Canvas do Projeto – Calculadora Simples com Orquestração Prefect

|  |  |
| --- | --- |
| Bloco | Descrição |
| Nome do Sistema | Calculadora Simples Orquestrada |
| Problema que Resolve | Automatiza operações básicas de forma estruturada, modular e escalável. |
| Público-Alvo / Usuários | Estudantes e profissionais aprendendo boas práticas de Engenharia de Software e Orquestração de Projetos |
| Funcionalidades Esperadas | - Soma - Subtração - Multiplicação - Divisão - Porcentagem |
| Módulos do Sistema | - modules/operacoes.py (lógica OO) - flows/ (orquestração) - infra/ (deploy futuro) |
| Expansões Previstas | Novas operações, interface web (futura), relatórios de uso |
| Premissas | - Código em Python 3.12.5 - Projeto modular - Deploy via Prefect Cloud com GitHub Actions |
| Restrições | - Código deve ser funcional a cada commit - Testes obrigatórios - Compatível com GitHub Actions |

## 📌 **UML – Diagrama de Classes (Versão 1)**

### Objetivo:

Representar graficamente as classes e seus relacionamentos iniciais no projeto da calculadora orientada a objetos. Nesta versão inicial, focaremos nas operações de **soma** e **subtração**, com possibilidade de extensão modular.

### 🧩 Estrutura proposta:

Diagrama

O conteúdo gerado por IA pode estar incorreto.

### 📁 Local para salvar:

O arquivo será salvo como:

📄 docs/diagrama\_classes\_v1.png

### Criando chaves – importante:

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Agora que você está com a **API Key recém-criada** do Prefect Cloud visível (conforme imagem), o próximo passo é usá-la **IMEDIATAMENTE**, antes de fechar esse modal — pois ela **não poderá mais ser visualizada depois**.

### ✅ Passos Imediatos – Agora no **Terminal (VSCode)**

Execute o seguinte comando abaixo diretamente no terminal (com o ambiente virtual DEV ativado):

pnu\_IbAMNqegMi3HB3yNGxPvAqiklD1uWQ2kjdQA

prefect cloud login -k pnu\_lbAMNqegMi3HB3yNGxPvAqikID1uWQ2kjdQA

**Nota:** Isso salvará a chave API no seu perfil local do Prefect, permitindo autenticar seu agente local e o CLI com o **Prefect Cloud**.

## Próximo Passo:

**👉 Definir os Secrets no GitHub**

1. Acesse o repositório no GitHub
2. Vá até: Settings > Secrets and variables > Actions
3. Clique em **New repository secret**
4. Adicione:
   * Name: PREFECT\_API\_KEY
   * Value: pnu\_lbAMNqegMi3HB3yNGxPvAqikID1uWQ2kjdQA
5. Clique em **New repository secret** novamente:
   * Name: PREFECT\_API\_URL
   * Value:  
     https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982

Depois disso, validamos juntos o prefect-deploy.yml para garantir que o workflow do GitHub esteja correto e preparado para **executar deploys automáticos** com segurança e eficiência.

Quando estiver pronto, só me dar o sinal que seguimos com os secrets e depois revisamos o arquivo prefect-deploy.yml. Vamos fazer desse laboratório o padrão de excelência!

### ✅ Notas Importantes

* **Gatilho:** somente em push no main e com alterações relevantes.
* **Secrets esperados:**
  + PREFECT\_API\_KEY: sua chave do Prefect Cloud
  + PREFECT\_API\_URL: URL do seu workspace Prefect
* **Deploy com retry:** evita falhas momentâneas por conectividade ou instabilidade de rede.

### 📌 Explicações Didáticas (para seus alunos)

|  |  |
| --- | --- |
| Seção | Explicação |
| on.push.paths | Garante que o deploy só será feito se algo relevante mudar |
| prefect deploy com retry | Evita falhas momentâneas de rede ou API |
| secrets | O segredo nunca fica visível e é seguro via GitHub |
| curl -I | Confirma conectividade com a Prefect Cloud antes de tentar o login |

### ✅ Objetivo agora

Verificar e, se necessário, redefinir a variável de ambiente PREFECT\_API\_KEY no seu terminal, para evitar conflitos com chaves antigas.

### 🧪 **1. Verificar valor atual**

Execute o comando abaixo para verificar a variável que está carregada atualmente:

prefect config view

### 🧹 **2. Limpar a variável antiga do ambiente**

Se a PREFECT\_API\_KEY exibida ainda estiver apontando para uma chave antiga, rode o seguinte comando para **desvinculá-la temporariamente da sessão atual**:

unset PREFECT\_API\_KEY

Caso também tenha configurado no seu shell de forma permanente (.zshrc, .bash\_profile, etc), comente ou remova a linha correspondente.

### 🔐 **3. Reconfigurar com a nova chave (caso necessário)**

Depois disso, configure manualmente com a nova chave (copiada da Prefect Cloud, já validada):

export PREFECT\_API\_KEY="sua-nova-chave-aqui"

Você pode também salvar isso em seu shell profile se desejar manter permanente, mas isso é opcional neste momento.

### 🌐 **4. Verificar novamente**

Agora rode:

prefect cloud login --key $PREFECT\_API\_KEY --workspace-url $PREFECT\_API\_URL

E depois:

prefect config view

Confirme se está tudo certo com os dois valores:

* PREFECT\_API\_KEY: sua nova chave
* PREFECT\_API\_URL: o link do seu workspace, como:

<https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982>

### 🔄 Próximo passo:

Se quiser garantir que não há nenhuma outra variável de ambiente sobrescrevendo isso, você pode rodar:

env | grep PREFECT

Isso vai listar todas as variáveis do sistema que começam com PREFECT\_. Deve aparecer apenas o que foi configurado no profile — sem variáveis de ambiente conflitantes.

...**não retornou nada**, o que significa que:

### ✅ **Diagnóstico Perfeito para Seguirmos**

* **Não há variáveis de ambiente PREFECT\_API\_KEY ou PREFECT\_API\_URL ativas no seu terminal.**
* O Prefect está **usando exclusivamente os valores salvos no perfil padrão**, conforme retornado por prefect config view.

Isso é **exatamente o comportamento desejado** neste momento:

|  |  |  |
| --- | --- | --- |
| Variável | Fonte | Status |
| PREFECT\_API\_KEY | profile | ✅ OK |
| PREFECT\_API\_URL | profile | ✅ OK |
| `env | grep PREFECT` | terminal env |

### 📌 Agora podemos seguir com segurança para:

1. **Configuração dos secrets no GitHub**
   * PREFECT\_API\_KEY → cole a chave nova (você gerou e viu hoje).
   * PREFECT\_API\_URL → https://api.prefect.cloud/api/accounts/284c9335-75c6-4022-ac98-4aa4d49d7f95/workspaces/fbc659c6-2423-42f4-b053-8224ac12d982
2. **Rodar um prefect deploy localmente para validar.**
3. **Depois disso**, rodar o GitHub Actions com o .yml atualizado.

### ✅ Verificação antes do deploy automatizado:

|  |  |
| --- | --- |
| Item | Status |
| Repositório GitHub CALCULADORA\_SIMPLES | ✅ Criado |
| prefect.yaml | ✅ Refatorado (gatilho por push, sem agendamento) |
| prefect-deploy.yml | ✅ Preparado |
| Variáveis secretas no GitHub | ✅ Presentes (PREFECT\_API\_KEY e PREFECT\_API\_URL) |
| Flow flow\_versao1.py | ✅ Pronto |
| Script de operações base (operacoes.py) | ✅ Implementado |
| Requisitos (requirements.txt) | ✅ Atualizado |
| Documentação (Canvas, UML, arquitetura) | ✅ Concluída na docs/ |

### 📦 **Próximo passo agora**:

#### 👉 Rodar o primeiro deploy manualmente para validar se tudo está funcionando:

No terminal com o ambiente virtual DEV ativado, execute:

prefect deploy -n hello-deploy

Nota: se **o deploy funcionar com sucesso**, **fazemos um push no GitHub** e **validamos a execução automática** via GitHub Actions (deploy automatizado).

⚠️ Caso algo falhe no deploy, o erro nos mostrará **se é de conectividade, autenticação, work pool ou estrutura do flow** — e resolveremos juntos.

## Comando para executar um teste local com Python:

python flows/flow\_versao1.py