UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE CRATEÚS

Trabalho de Fundamentos de Linguagens de Programação

1. Introdução

O presente trabalho tem como objetivo proporcionar aos alunos uma vivência prática e crítica dos principais conceitos de linguagens de programação, conforme estruturados no livro *Conceitos de Linguagens de Programação* de Robert W. Sebesta.

A atividade propõe desafios que abrangem os 14 tópicos fundamentais estudados ao longo da disciplina, incentivando a pesquisa, a análise comparativa e o desenvolvimento de pequenos experimentos ou exemplos práticos.

Cada aluno deverá produzir seu próprio trabalho de forma independente, sendo expressamente recomendado que as soluções sejam personalizadas para evitar cópias e plágio.

2. Desenvolvimento

O trabalho é composto por **14 desafios**, um para cada tópico estudado em aula.
Cada desafio será **aberto**, para que cada aluno possa abordá-lo de forma única, com liberdade para escolher exemplos, linguagens e formas de apresentação (texto, código, figuras, diagramas).

Desafios

1. **Introdução às Linguagens de Programação**
   ➔ Pesquise e monte uma linha do tempo da evolução das linguagens, destacando **marcos que você considerar mais relevantes**.

2. **Ambientes de Programação**
   ➔ Elabore um **diagrama próprio** que explique compiladores, interpretadores e máquinas virtuais com exemplos **de sua escolha**.

3. **Descrições Sintáticas e Semânticas**
   ➔ Crie uma **mini-gramática fictícia** para uma linguagem que você mesmo inventar,

com exemplos de análise léxica.

4. **Tipos de Dados**
   ➔ Compare a tipagem entre **três linguagens de sua escolha**, mostrando exemplos breves e comentados.

5. **Estruturas de Controle**
   ➔ Desenvolva um programa simples que use estruturas de seleção, repetição e controle de fluxo, **criando um contexto original**.

6. **Subprogramas**
   ➔ Implemente funções que demonstrem passagem de parâmetros por valor e por referência em **linguagens diferentes** que você dominar.

7. **Implementação de Subprogramas**
   ➔ Desenhe e explique a pilha de chamadas de um exemplo recursivo **definido por você**.

8. **Programação Orientada a Objetos**
   ➔ Modele uma hierarquia simples de classes **em um domínio de sua preferência** (ex.: personagens, transportes, serviços).

9. **Concorrência**
   ➔ Explique a diferença entre threads e processos e implemente um exemplo de **concorrência personalizado**.

10. **Gerenciamento de Memória**
    ➔ Pesquise como ocorre a gestão de memória em duas linguagens distintas e monte um **quadro comparativo de sua autoria**.

11. **Programação Funcional**
    ➔ Implemente uma solução funcional com recursão e funções de alta ordem, **escolhendo um problema real ou fictício**.

12. **Programação Lógica**
    ➔ Modele um pequeno problema lógico (ex.: resolver um quebra-cabeça, genealogia) usando **sintaxe inspirada em Prolog**.

13. **Linguagens para Scripts e Web**
    ➔ Crie um pequeno **roteiro ou script** usando uma linguagem de script para automação ou manipulação de dados, **com contexto próprio**.

14. **Tendências em Linguagens de Programação**
    ➔ Escolha uma linguagem emergente, investigue e elabore uma **apresentação textual**

crítica sobre ela.

3. Entrega

A entrega será feita por meio da criação de um **repositório público no GitHub** seguindo estas orientações:

```
Fundamentos-Linguagens-UFC/
│
├── README.md                --> Explicação geral do trabalho e estrutura
do repositório
│
├── 01-introducao/            --> README.md com explicação e materiais da
Introdução às Linguagens
│
├── 02-ambientes/             --> README.md + diagramas de ambientes de
programação
│
├── 03-sintaxe-semantica/     --> README.md + gramática criada
│
├── 04-tipos-de-dados/        --> README.md + comparativo de tipos
│
├── 05-estruturas-de-controle/--> README.md + código de exemplo
│
├── 06-subprogramas/          --> README.md + exemplos de funções
│
├── 07-implementacao-subprogramas/ --> README.md + desenho da pilha
│
├── 08-orientacao-objetos/    --> README.md + modelagem de classes
│
├── 09-concorrencia/          --> README.md + exemplo de concorrência
│
├── 10-gerenciamento-memoria/ --> README.md + quadro comparativo
│
├── 11-programacao-funcional/ --> README.md + exemplo funcional
│
├── 12-programacao-logica/    --> README.md + problema lógico
│
├── 13-scripts-web/           --> README.md + script criado
```

```
│
└── 14-tendencias/            --> README.md + análise crítica
```

O link do repositório deve ser entregue para bruno@crateus.ufc.br com o título de email sendo:

`[Entrega] Fundamentos-Linguagens-UFC`

4. Observações Finais

- A avaliação considerará **originalidade, completude, clareza na documentação** e **qualidade das implementações**.
- Trabalhos com sinais evidentes de cópia ou plágio serão desconsiderados.
- O repositório deve ser enviado até **23h59 do dia 21/07/2025**, com o link postado na plataforma oficial da disciplina (ou enviado por e-mail, conforme instruções posteriores).

Embora os alunos tenham liberdade para escolher linguagens e ferramentas, sugere-se o uso da seguinte stack para cada desafio:

Desafio	**Stack Sugerida**
01 - Introdução às Linguagens de Programação	Documento em Markdown (`README.md`) com imagem/linha do tempo (pode usar Canva, Lucidchart, ou outro editor online)
02 - Ambientes de Programação	Diagrama feito com Draw.io, Lucidchart ou Canva + explicação em Markdown
03 - Descrições Sintáticas e Semânticas	Markdown + gramática escrita em pseudocódigo ou exemplos pequenos em Python
04 - Tipos de Dados	Código curto em **Python**, **C** e **JavaScript**; comparação documentada em Markdown

05 - Estruturas de Controle	Código em **Python**, **Java**, **C++**, ou **JavaScript** com uso de estruturas de decisão e repetição
06 - Subprogramas	Exemplos em **C** e **Python** (funções simples mostrando valor/referência)
07 - Implementação de Subprogramas	Diagrama feito em Draw.io + código exemplo (pode ser recursão simples em **Python** ou **C**)
08 - Programação Orientada a Objetos	Pequeno projeto em **Java**, **Python**, ou **C#**, com hierarquia de classes e herança
09 - Concorrência	Implementação em **Python (threading ou multiprocessing)** ou **Java (threads)**
10 - Gerenciamento de Memória	Tabela comparativa entre **Java** (GC automático) e **C** (malloc/free manual) em Markdown
11 - Programação Funcional	Exemplo de funções de alta ordem e recursão usando **Python** (functools) ou **JavaScript** (ES6)
12 - Programação Lógica	Código em **Prolog** (GNU Prolog, SWI-Prolog) ou simulação em pseudocódigo
13 - Linguagens para Scripts e Web	Exemplo em **Python** (scripts de automação usando os módulos `os`, `shutil` ou `requests`) ou **JavaScript** (Node.js básico)

14 - Tendências em Linguagens de Programação	Pesquisa e documentação sobre **Rust**, **Go**, **Kotlin** ou **TypeScript**, escrita em Markdown

5. Avaliação

A nota final da disciplina será composta pela **média simples** entre duas componentes:

1. Trabalho Prático

- Avaliação do repositório entregue no GitHub conforme as instruções estabelecidas.
- Será atribuída uma nota de **0 a 10** considerando critérios como:
    - Cumprimento das instruções;
    - Completude das soluções;
    - Qualidade das implementações;
    - Clareza e organização da documentação;
    - Originalidade e criatividade.

2. Prova Escrita

- A prova escrita conterá **uma questão dissertativa** para cada um dos 14 tópicos estudados.
- Cada aluno deverá **escolher 10 tópicos** entre os 14 disponíveis e responder às respectivas questões.
- Cada resposta terá peso equivalente e será avaliada de **0 a 10 pontos** individualmente.
- Será avaliada a capacidade de:
    - Explicar os conceitos com clareza e profundidade;

- Relacionar teoria e prática;
- Demonstrar domínio crítico do conteúdo.