# Universidade Federal Rural de Pernambuco

## Linguagens de Programação

Atividade desenvolvida como parte da disciplina de Linguagens de Programação, ministrada pelo professor Gabriel Alves.

**1. “Pesquise e escreva com suas palavras semelhanças e diferenças das linguagens de programação Haskell, Python e Java”**

Se você já se deparou com mais de uma linguagem de programação, deve ter notado que elas costumam possuir algumas diferenças no modo em que lidamos com elas, que vão desde comandos básicos ao modo de execução do código. Mas nem todas são tão diferentes assim, elas costumam possuir certas semelhanças que se tornam mais evidentes com o tempo de experiência.

Sendo assim, vamos abordar algumas delas e citar onde há vantagens e desvantagens em cada uma delas.

Para começar, devo apresentar quais são e um pouco da história por trás de cada uma delas.

**Java**: O Java surgiu em 1991. Na época, ela buscava ser uma alternativa de concorrência às linguagens da época.

Com a idéia de ser uma linguagem que pudesse ser executada em diversos dispositivos, atualmente é uma linguagem bastante popular, porém em seu início teve complicações devido às limitações de hardware da época, por ser uma linguagem "pesada" e que terminava causando transtornos no tempo de lentidão e alto tempo de espera quando utilizada.

**Python**: O python surgiu em teve sua primeira release em 1991 e buscava ser uma linguagem de alto nível - o que significa ser mais próxima da compreensão humana com inspiração no C e no Shell Script - além de ser uma alternativa às linguagens que eram didaticamente difíceis de compreender.

**Haskell**: O Haskell surgiu em 1987, com um estilo de programar baseado no que deve ser feito (what) em vez de como deve ser feito (how), possui foco em cálculos matemáticos de fácil manutenção e que costuma ser utilizada a fins didáticos e de pesquisa por ser simples de se compreender.

Além da história, podemos citar diferenças técnicas entre as linguagens.
Abaixo temos uma comparação lado a lado entre cada uma delas.

| | **Java** | **Python** | **Haskell** |
|---|---|---|---|
| Tipo de dados na lista | Homogêneo | Heterogêneo | Homogêneo |
| Paradigma | OO | OO, Funcional, Imperativo | Funcional |
| Tipo de Execução | Compilada | Interpretada | Compilada |

| Tipagem | Fortemente Tipada | Tipagem Dinâmica | Fortemente Tipada |
|---|---|---|---|
| Chaveamento | Chaves | Identação | Chaves, Identação |
| Modificador de visibilidade | Public, Private, Protected | Public, Private | **not found** |

**2. “Reflita sobre os códigos criados: Você geralmente cria programas orientados a objetos ou estruturados?”**

Há diferenças marcantes entre lidar com o paradigma orientado a objetos e o paradigma imperativo. Ao desenvolver o código, a primeira observação é que no desenvolvimento orientado a objetos, centralizamos as informações, então não temos necessidade de criar muitas estruturas de dados para reunir os dados em que estamos trabalhando.

Ao criar um carro representado por objetos, optei por utilizar uma classe Veiculo que regulava regras para o caso de um veículo ser um Carro ou uma Moto. Foi possibilitada a dissociação das estruturas de dados em objetos singulares, mas que fazem sentido juntos: Roda, Veiculo, Carro e Moto. Deixando a lógica de união dos dados apenas na parte em que há interação com o usuário.

No paradigma imperativo, foi necessário a criação de uma lista para cada informação relevante: Número de Rodas (Sem flexibilidade para mais informações), Placa, Tipo de veículo (Moto ou Carro). Para haver adição de informações extras, teriam que ser criadas mais listas, isso indica uma tendência de ter que criar tantas listas para guardar as informações que dificultaria o acesso aos dados, inclusive por conta de ter que varrer a lista todas as vezes que for realizar alguma operação em algum dos veículos.

Costumo programar orientado a objetos sempre que no sistema haja uma tendência de armazenar muitas informações de diversos tipos - e isso acontece em quase todos os casos. A programação imperativa me tem sido útil apenas para testar códigos e soluções de pequenos problemas.

**3. “Não existe agora, e nem existirá, uma linguagem de programação na qual seja difícil escrever programas ruins.” - Larry Flon 1975.**

**Discorra sobre a afirmação de Larry Flon.**

É de domínio comum a qualquer programador com um pouco de experiência o fato que à medida que sistemas bons não costumam ser desenvolvidos por iniciantes. Quando falamos sobre a qualidade de um sistema, nos vêm um questionamento sobre o que classifica um sistema como bom ou ruim.
Seria a complexidade? Seria a estrutura? Seria algum aspecto inovador? Seria a dificuldade de ser reproduzido por outra pessoa?

A resposta pode não ser tão simples quanto parece, pois talvez um bom sistema possua um pouco de cada uma dessas características, mas não puramente apenas uma delas.

Um sistema ser complexo não basta para que seja bom, pois à medida que a complexidade aumenta, há uma tendência de aumentar também a dificuldade de manutenção.

Um sistema que possua uma estrutura bem definida e partes bem interconectadas possui qualidade no aspecto de compreensão, porém uma estrutura bem definida por si só, não garante que esse sistema resolverá problemas relevantes.

Um sistema que possua um aspecto inovador é interessante e pode possuir vantagem competitiva, porém ser inovador não garante a longevidade do programa.

Bons sistemas costumam possuir dificuldade de ser reproduzidos por outras pessoas, mas não necessariamente por serem difíceis, pois bons sistemas necessitam de alguém com uma boa compreensão dos problemas em que está resolvendo, da necessidade sem exagero de algoritmos complexos e que principalmente possam ser facilmente compreendidos e evoluídos por outras pessoas.

As linguagens de programação nos aproximam da solução de problemas específicos, mas além do código, um bom programa é escrito por um bom programador que tem consciência dos poderes e limitações. É interessante saber como resolver um problema, mas tão importante quanto, é saber até onde se pode ir utilizando as tecnologias conhecidas; Além de que bons programas, no contexto atual, já utilizam mais de uma linguagem em colaboração à solução de conjuntos de problemas.