Avaliação

Nome: Victor Martins Vieira

Quantos tópicos você leu?

> 20
> Na data de entrega dessa avaliação, os tópicos 19 e 20 já haviam sido disponibilizadas.

Você esclareceu pelo menos 50% das suas dúvidas pertinentes aos tópicos?

> Sim.

Seu repositório poo-2023-01 contém o arquivo README.md que informa a finalidade deste repositório para os visitantes?

> Sim, entretanto não contém a descrição de de todos as funcionalidades e meios de interação com o projeto, sendo um artefato que vem evoluindo conforme aparecem novas demandas, a exemplo das estatísticas de número de classes e quantidade de código escrito solicitados nessa avaliação.
> Ademais, alguns módulos incluem seus próprios readme's, a fim de disponibilizar, mais facilmente, outros artefatos solicitados nos tópicos, a exemplo dos diagramas e esclarecer quaisquer divergências.

Suas respostas no seu repositório poo-2023-01 estão em diretórios em conformidade com o que foi especificado, ou seja, seus diretórios tem como nomes t07, t08 e assim por diante?

> Sim.

Você respondeu quantos tópicos?

> 11
> Sendo estes os tópicos, 1, 7, 8, 9, 10, 11, 12, 13, 17, 18 e 19. Na data de entrega dessa avaliação, ainda não concluí o tópico 20, e os demais tópicos não solicitavam entrega de nenhum tipo de artefato.

Quantas classes você criou?

> 264 classes, distribuídas em 101 arquivos em 10 diretórios; compreendendo os tópicos listados na questão anterior.
> Python não estipula a convenção/necessidade de se haver uma única classe por arquivo, sendo recomendado o agrupamento de classes e funções correlatas em um único módulo (arquivo.py).
> Com o intuito de facilitar eventuais correções, no repositório, implementei um módulo para cada

> proposta de exercício dentro dos tópicos - o que se mostrou conveniente, haja vista que alguns dos exercícios incluem a implementações diferentes de classes com o mesmo nome.

Conte quantas linhas de código você criou.

> 2013 linhas.

Seu código está organizado em diretórios conforme a convenção introduzida pela ferramenta Maven?

> Não, uma vez que não utilizei a linguagem Java para a implementação das classes propostas nos tópicos.
> Não há, em python, uma convenção definida, a nível de linguagem, para estruturar projetos.
> Também não encontrei, em minhas pesquisas, uma ferramenta similar ao Maven para a linguagem, e nenhuma das ferramentas utilizadas demandam uma estrutura específica como ocorre com o Maven.
>
> Sendo assim, os seguintes artigos foram utilizados como guia para estruturar o projeto/repositório:
> Python Application Layouts: A Reference – Real Python (https://realpython.com/python-application-layouts/)
> Structuring Your Project — The Hitchhiker's Guide to Python (https://docs.python-guide.org/writing/structure/)

Contém o arquivo pom.xml que informa como compilar, por exemplo, o código produzido?

> O projeto contém o arquivo pyproject.toml, que é o formato introduzido pela PEP 518 (Specifying Minimum Build System Requirements for Python Projects — https://peps.python.org/pep-0518/) para armazenar dependências e configurações relativas ao projeto.
> Até o momento, nenhum exercício especificou um ponto de entrada para que o projeto seja compilado em um executável. Caso especifique, há ferramentas para tal, e as configurações seriam armazenadas em tal arquivo.
> Novamente, não foi encontrado um framework que agrupe todas as funcionalidades do Maven demonstradas em sala, sendo usado, portanto, um conjunto de ferramentas atualmente agrupadas e gerenciadas pela ferramenta Poetry (https://python-poetry.org/).

Durante as aulas foi sugerido um repositório exemplo para sua orientação. Caso não tenha empregado ele, sabe diferençar o que você ganha e o que você perde quando não o emprega?

> Sim, conforme já discutido nas respostas anteriores.

Compreende a finalidade do arquivo .gitignore?

> Sim.

Suas respostas que envolvem código contém um arquivo .gitignore correspondente?

> O arquivo .gitignore foi definido a nível de projeto, visto que algumas das pastas a serem ignoradas pelo sistema de controle de versões se repetiam dentro de cada pacote, não havendo a necessidade de incluir múltiplos arquivos .gitignore.

Código em Java usa linhas em branco para demarcar a separação de um tópico de outro, seja dentro de um mesmo método ou entre métodos e também entre elementos de uma classe, por exemplo, package é separado dos imports por uma linha em branco e os imports, por sua vez, separados da declaração da classe, por uma linha em branco. Seu código usa linhas em branco para essa finalidade?

> Sim. Linhas em branco constituem 32.7% dos arquivos de código, sendo utilizadas para fins de separação e melhoria da visualização do código, estando conforme à PEP 8 (Style Guide for Python Code — https://peps.python.org/pep-0008/#blank-lines).

Há linhas em branco, por exemplo, duas ou mais em seu código, simplesmente deixadas lá simplesmente porque não foram removidas? Naturalmente, isto não é desejável.

> Não.
> Parte da validação realizada é a análise de conformidade do código à PEP 8, por meio da ferramenta flake8 (https://flake8.pycqa.org/). Linhas em branco a mais que o determinado no guia de estilo seriam reportadas como erro.

Você entendeu para que serve o arquivo pom.xml, a ferramenta Maven e porque o código em Java é organizado desta forma?

> Sim, apesar de não estar utilizando a linguagem Java para implementar as classes presentes nos tópicos de exercícios.

Ao baixar seu repositório poo-2023-01 e em cada diretório criado para responder o tópico em questão, quantos deles o comando mvn compile compila de forma satisfatória o código produzido?

> O projeto foi implementado em Python, portanto não faz uso da ferramenta Maven.
> Entretanto, ao executar a análise estática do código, com a ferramenta mypy (https://mypy-lang.org/) que apresenta todas as dicas de tipo, conforme a PEP 484 (Type Hints — https://peps.python.org/pep-0484/), não é reportado nenhum erro ou inconsistência.

A convenção para nomear packages em Java é usar apenas letras minúsculas. Ou seja, diferente da nomeação de classes, por exemplo, que deve se iniciar por maiúsculas. Responda SIM caso tenha atendido esta convenção ou NÃO, caso contrário.

> Nomes de módulos e pacotes no projeto encontram-se em conformidade com a PEP 8 (Style Guide for Python Code — https://peps.python.org/pep-0008/#package-and-module-names), sendo também verificados pela ferramenta flake8.

As classes criadas começam com letra maiúscula e não possuem acento? Java segue a convenção conhecida por CamelCase, por exemplo, uma classe para classificar os guarda-chuvas seria normalmente nomeada por GuardaChuva. Os nomes das suas classes seguem esta convenção?

> Os nomes de classes implementadas no projeto encontram-se na convenção CapWords/UpperCamelCase, conforme PEP 8 (Style Guide for Python Code — https://peps.python.org/pep-0008/#class-names), sendo também verificados pela ferramenta flake8.

Nomes de packages devem seguir a convenção de usar “domínios reversos”. Responda SIM se usou esta convenção na nomeação dos packages ou NÃO, caso contrário.

> Não. Não há a determinação dessa convenção para pacotes e módulos escritos em Python na PEP 8 (Style Guide for Python Code — https://peps.python.org/pep-0008/#package-and-module-names), e o projeto não faz uso de nenhuma ferramenta que dependa da utilização de domínios reversos para nomeação dos pacotes.

Após o nome da classe deve seguir um espaço, ou seja, a classe Saude, porque classes em Java não empregam acentos, deve ser declarada como class Saude { com espaço entre o abre chaves e a palavra Saude.

> Tal convenção não se aplica a Python. O código implementado encontra-se conforme a PEP 8 (Style Guide for Python Code — https://peps.python.org/pep-0008/#whitespace-in-expressions-and-statements) sendo verificado, pela ferramenta flake8.

Nomes de propriedades e variáveis em Java, por convenção, são iniciadas por letra minúscula. Responda SIM se seguiu esta convenção ou NÃO, caso contrário.

> Sim. O código implementado encontra-se conforme a PEP 8 (Style Guide for Python Code — https://peps.python.org/pep-0008/#function-and-variable-names), sendo verificado pela ferramenta flake8.

Quando se faz uso da orientação a objetos existe um princípio ou orientação conhecida por “programação para interface”. Nestes tópicos houve oportunidade de fazer uso desta orientação, por exemplo, ao usar a estrutura de dados “lista”. Em Java existe a interface List e várias classes que implementam esta interface, dentre elas, ArrayList, Stack, Vector e outras. Neste caso, a sugestão é fazer uso de uma referência para a interface, em vez da referência para classe. Desta forma, pode-se, se preciso, trocar a implementação, por exemplo, ArrayList por outra implementação, sem afetar o restante do código. Você pode consultar

https://www.baeldung.com/cs/program-to-interface para mais informações. Você compreendeu esta orientação?

> Sim.

Você fez uso da orientação ou princípio sugerido acima?

> Não exatamente.
> Python, enquanto linguagem, não oferece suporte nativo a interfaces, optando pela opção de herança múltipla, conforme dado no referido artigo como uma alternativa. Ainda assim, é possível declarar interfaces formais por meio de classes e métodos abstratos, como uma alternativa. O que foi implementado nos exercícios que solicitavam a declaração e implementação de interfaces.

Quando se usa o git, ao realizar um “commit”, você pede para este software que considere todas as mudanças realizadas como uma unidade. Houve cuidado na definição das mensagens associadas aos commits ao longo da realização das tarefas deste tópico?

> Sim, buscou-se a utilização de mensagens descritivas para cada um dos commits que constituem o projeto.

O tópico 18 requisita a criação de um programa, você entendeu o que foi requisitado?

> Sim.

O tópico 18 requisita a criação de um programa, você implementou o programa?

> Sim.