Programação Orientada a Objetos – Aula 03

Prof. João Luiz
joao.laoliveira@gmail.com

Boas Práticas de Programação 

Escreva Código Claro

- Deve ser óbvio o que seu código faz (idealmente)
- Não sendo possível, utilize comentários objetivos

Princípios

- Bonito é melhor que feio
- Explícito é melhor que implícito
- Simples é melhor que complexo
- Complexo é melhor que complicado
- Linear é melhor que aninhado
- Esparso é melhor que denso
- Legibilidade conta

Zen de Python (PEP 20)

Nomes de Variáveis

- Devem dizer o que elas **contém**
- Utilizar nomes **claros** e **significativos**
- Utilizar uma **convenção** no seu código

Convenções de Escrita (PEP 8)

- camelCase: não recomendado (APIs antigas, código legado)
- snake_case: **variáveis** e **funções** (atributos e métodos)
- PascalCase: **classes**. Ex: MinhaClasse
- UPPER_CASE: **constantes** (screaming snake case)

- Nomes Descritivos: total_itens, user_age
- Prefixos e Sufixos: is_active, user_list
- Evitar apenas uma letra: **loops diretos**, vida muito curta
- Métodos e atributos privados: _my_private_method

Visibilidade no Python

Padrão	Nome técnico	Uso / Significado
atributo	público	Acesso livre.
_atributo	protegido (convenção)	“Interno, não use fora da classe.”
__atributo	name mangling	Evita sobrescrita em herança, “quase privado”.
__metodo__	dunder (especial)	Usado pelo Python, não invente novos.

Convenções são Importantes!

Principalmente em uma linguagem com tanta liberdade quanto o Python

Comentários

- Utilizar apenas quando trecho de código que não for autoexplicativo.

- Sempre buscar escrever códigos que não necessite de comentários

- Devem sempre ser claros e objetivos!

MAS..... 

Não faça comentários WET

- **Wrote Everything Twice** ou **Wasted Everyone's Time**

Evite Comentários em Linha

- Convenção de 80 caracteres no máximo

Cuidado com acentuação!

Indentação 🍷 🗿

4 espaços para cada indentação (tabulação pode dar problemas com editores e OS diferentes)

Linhas Vazias

- **Duas linhas** vazias: Antes e depois de definições de funções e de classes
- **Uma linha** vazia: Entre métodos dentro de classe e para organização de código (blocos lógicos).
- Mais que duas linhas: NUNCA!
- **Uma linha** vazia no **final** do código.

Sempre Teste o seu Código

- Manual

- Python possui ferramentas de testes automatizados (aulas futuras)

Princípio D. R. Y.

- Don’t Repeat Yourself!
- Tarefas ou lógicas repetidas devem ser centralizadas
- Reutilização de código: encapsulamento de código.

Vantagens do D. R. Y.

- Manutenção mais fácil: centralização de tarefas e lógicas
- Menos erros: Toyota Corolla
- Código mais limpo: Leitura e compreensão, eficiência

Modularidade

- Dividir o código em subtarefas **independentes** e **intercambiáveis**.
- Os módulos são responsáveis por uma parte específica do todo
- Módulos: Funções e métodos (Classes e Módulos)

Princípios da Modularidade

- **Separar Responsabilidades**

- **Independência de Implementação**: Interfaces bem definidas, permitindo que o código interno do módulo seja alterado sem afetar outros módulos.

Benefícios da Modularidade

- Reutilização de Código: D.R.Y. -> andam sempre juntos
- Manutenção Facilitada
- Colaboração Melhorada
- Testabilidade

Guias de Estilo

Conjuntos de regras e padrões para escrita e formatação de código

- PEP 8
  - https://peps.python.org/pep-0008/
  - https://wiki.python.org.br/GuidaDeEstilo

- Google Python Style Guide: Expande algumas questões em aberto da PEP 8 (docstrings).
  - https://google.github.io/styleguide/pyguide.html

Importações

- Topo do arquivo

- Devem ser agrupadas
  - Módulos nativos
  - Módulos de terceiros
  - Módulos próprios

- Evitar import * ou import do módulo completo (mod. extensos)

Excessões

- Para casos Excepcionais: Erros inesperados
- Não capturar exceções gerais: Especificar o tipo de exceção. Mascara erros inesperados
- Minimizar o **Bloco Try**: Evitar capturar exceção que não foi antecipada
- Utilizar exceções personalizadas: Tratamento de erro mais claro

Acesso a Atributos

- Utilizar Acesso Público a Atributos (geral)
- Preferir Funções Simples para Propriedades: se for necessário, manter simples e sem efeitos colaterais.
- Evitar Atributos de Classe Mutáveis

Outras Recomendações

- Compreensões de Lista e Geradores: Simplificação e legibilidade
- Use Funções em Vez de Classes: Classe com um método
- Utilize Argumentos Nomeados

Docstrings

- Strings usadas no início de um módulo, classe, método ou função
- Explica o **propósito** e o **comportamento** do bloco
- Usadas para **documentação** Prática importante para tornar o código mais legível e manutenível
- Os princípios visto não são utilizados em documentação

Componentes de uma Docstrings

- **Resumo**: Uma linha
- **Descrição**: Se necessário
- **Args**: Lista dos args. nome_arg (tipo) e descrição
- **Returns**: Descrição do valor de retorno(tipo)
- **Raises**: (Opcional) Descrição das exceções que a função pode lançar.
- **Examples**: (Opcional) Exemplos de como a função pode ser usada

Type Hinting

Sugestão de tipos

- Python possui tipagem dinâmica
- Podemos sugerir tipos esperados como parâmetros e retornos de funções
- Serve para documentação, o Pyhton ignora. No entanto, existem ferramentas para checkar se o *typing* está sendo respeitado.

Tipos Unificados

- Para registrar mais que um tipo no *type hinting*.

- Módulo **typing**

Obrigado!

Prof. João Luiz

joao.laoliveira@gmail.com