Desenvolvimento Python

Este é um material produzido para estudo da linguagem de programação Python.

Referências:

- https://penseallen.github.io/PensePython2e/
- https://docs.python.org/3/tutorial/index.html
- https://github.com/calexdev-hub/design-patterns/tree/main
- https://www.devmedia.com.br/conheca-os-padroes-de-projeto/957
- https://refactoring.guru/pt-br/design-patterns/strategy
- https://www.alura.com.br/conteudo/python-testes-automatizados-qualidade-codigo

Repositório do Estudo

- https://github.com/codes-by-rafaeladias/Clientes-DesignPatterns

Vídeo Tutorial

- https://youtu.be/5-VwDU_LJ9I

O que é Python?

Python é uma linguagem de programação de alto nível criada em 1991 por Guido van Rossum. O nome não vem da cobra, mas sim do grupo de comédia britânico *Monty Python's Flying Circus*, de quem o criador era fã. É utilizada para diversas possibilidades de aplicação, como por exemplo:

- Desenvolvimento de websites, jogos, aplicativos para dispositivos móveis e desktop;
- Inteligência Artificial;
- Ciência de Dados;
- Matemática e Estatística.

Python é uma das linguagens de programação mais famosas! A sua popularidade deve-se ao fato de ela apresentar uma sintaxe simples, contar com uma grande comunidade de desenvolvedores e disponibilizar muitas bibliotecas.

Atualmente, a versão mais atual da linguagem Python é a versão 3.13.

```
print("Olá, mundo!")
```

Como Instalar Python?

Para instalar Python em seu dispositivo, você pode acessar *https://www.python.org/downloads/* e selecionar o instalador compatível com o sistema operacional de seu dispositivo.

Após a instalação, certifique-se de que o comando *python* funciona no terminal:

```
python --version
```

Variáveis

- Variáveis são áreas de memória que podem armazenar valores que podem ser modificados.
- Python é uma linguagem Case Sensitive, isto é, sensível a letras maiúsculas e minúsculas. Portanto, a variável *nome* é diferente de *Nome*.
- *Snake_case* é o padrão utilizado para nomear variáveis: *primeiro_nome*.

Tipos de Dados

- Numérico: Pode ser inteiro, ponto flutuante ou complexo. Exemplos: Inteiros - *int*, *integer*; Ponto flutuante - *float*; Números complexos - *complex*.
- Texto: Exemplo: *String*.
- Lógicos ou Booleanos: Dados que só podem ser *True* ou *False*.
- Listas
- Tuplas
- Dicionários ou Mapas
- Conjuntos ou Sets
- None: Representa dados que não possuem valor atribuído.

Estruturas de Dados

Estruturas de dados são coleções que armazenam e manipulam dados.

Lista

É uma estrutura ordenada (os dados são armazenados de acordo com índices), mutável e heterogênea (permite armazenar dados de diferentes tipos).

```
minha_lista = ["Python", 1234]
tamanho_minha_lista = len(minha_lista)
print("A lista minha_lista tem", tamanho_minha_lista, "elementos")
```

Funções de uma lista:

```
minha_lista = ["Python", 1234]
tamanho_minha_lista = len(minha_lista) #retorna a quantidade de itens na lista
minha_lista.append("Ciência de Dados") #adiciona um dado ao fim da lista
```

```
minha_lista.insert(3, 5678) #adiciona um dado na posição 3
ultimo_dado_lista = minha_lista.pop() #retorna e remove o último elemento da lista
ultimo_dado_lista = minha_lista.pop(1) #retorna e remove o item na posição 1
del minha_lista[2] #remove o item na posição 2
minha_lista.remove("Python") #remove elemento da lista
minha_lista.clear() #limpa a lista

lista_frutas = ["Banana", "Maçã", "Abacaxi"]
quantidade_banana = lista_frutas.count("Banana") #conta quantos elementos Banana
indice_banana = lista_frutas.index("Banana") #primeiro índice que tem Banana
copia_lista_frutas = lista_frutas.copy() #copia a lista

lista_numeros = [7, 4, 5, 6, 1, 3, 2, 9, 8]
lista_numeros_inverso = lista_numeros.reverse() # inverte a lista
lista_numeros.sort() #ordena a lista, alterando
lista_numeros_ordenada = lista_numeros.sorted() #ordena a lista mas não afeta lista_nume
```

Tupla

É uma sequência de elementos separados por vírgulas,representados ou não entre parênteses. É imutável e heterogênea.

```
tupla_vazia = tuple()
print(tupla_vazia)
minha_lista = [0, 1, 2]
minha_tupla = tuple([minha_lista])
print(minha_tupla)
tupla_nomes = "Python, Java, Javascript"
print(tupla_nomes)
tupla_animais = ("Cachorro", "Gato")
print(tupla_animais)
```

Funções de uma tupla:

```
minha_lista = [0, 1, 2]
print("A lista minha_lista tem", minha_lista.count(0), "elementos de valor 0") #retorna
print("O elemento 2 está na posição", minha_lista.index(2), "na lista") #retorna índice
```

Dicionário

Também chamado de mapa, é uma estrutura que associa uma chave a seu valor, como por exemplo, associa uma palavra ao seu significado. É mutável, heterogênea e não possui ordem. Nela, a chave não pode ser repetida.

```
dicionario = dict()
print(dicionario)
dicionario["nome"] = "Maria"
print(dicionario)
numerais = {"1":"Um", "1º":"Primeiro"}
print(numerais)
```

Funções de um Dicionário

```
numerais = {"1":"Um", "1º":"Primeiro"}
print(numerais.items()) #imprime todos os pares
print(numerais.keys()) #imprime todas as chaves
print(numerais.values()) #imprime todos os valores
print(numerais.get("1")) #imprime o valor associada à chave 1
```

Padrões de Projeto

Um padrão de projeto pode ser entendido como uma solução recorrente para um problema em um contexto, mesmo que em projetos e áreas distintas. Os padrões de projetos ganharam destaque na Engenharia de Software, em 1995, com a publicação do livro *Padrões de Projeto – Soluções reutilizáveis de software orientado a objetos*, por um grupo de quatro atores que ficou conhecido como *Gang of Four - GoF*.

Usar padrões de projetos propicia o reaproveitamento de soluções e reduz a complexidade do processo de desenvolvimento de software. Eles se classificam em:

- Criacional: abstrai e ou adia o processo criação dos objetos.
- Comportamental: se concentra nos algoritmos e atribuições de responsabilidades entre os objetos.
- Estrutural: se preocupa com a forma como classes e objetos são compostos para formar estruturas maiores.

DAO - Data Access Object

É um padrão estrutural que busca organizar o código, isolando a camada de acesso ao banco de dados do restante do código.

Factory

É um padrão criacional que permite definir uma interface para criar objetos, mas as subclasses escolhem qual classe instanciar.

Adapter

É um padrão estrutural que converte a interface de uma classe por outra esperada pelos clientes. O que possibilita que classes com interfaces incompatíveis trabalhem em conjunto.

Repository

É um padrão usado para separar o aplicativo (elementos de programação, como classes, interfaces, métodos) do armazenamento de dados (banco de dados) e funciona como um mediador entre os dois. Ele fornece um conjunto de métodos para executar operações de leitura, escrita, edição e exclusão nos dados.

Strategy

É um padrão comportamental que permite que você defina uma família de algoritmos, coloque-os em classes separadas, e faça os objetos deles intercambiáveis.

Testes de Software

Em Engenharia de Software, os testes são realizados para verificar o comportamento de um sistema e podem indicar a presença de defeitos, sendo uma ferramenta essencial para a qualidade do software.

Há muitos tipos de testes, mas abaixo serão discutidos os testes unitários e os testes de integração.

Os testes unitários são executados para verificar uma pequena porção do código, rastreando possíveis erros ou problemas. Já os testes de integração testam o conjunto dos componentes e módulos que compõem um sistema para garantir que o sistema funcione de forma integrada.

Testes Automatizados

Os testes automatizados são popularmente usados, pois economizam tempo e dinheiro envolvidos no processo de testes de um software. Assim, como em outras linguagens de programação, Python possui frameworks e bibliotecas que possibilitam a escrita e execução de casos de testes de forma automática:

- **PyTest**
- **Unittest**
- **Robot**
- **Selenium**
- **Testify**
- **DocTest**