Módulo 3 - Estrutura de dados e algoritmos

Fala, **dev**!

Neste módulo, vamos aprofundar nossos conhecimentos em estruturas de dados e algoritmos, dois pilares fundamentais para qualquer programador. Vamos explorar esses conceitos de forma didática e prática. Preparado? Vamos lá!

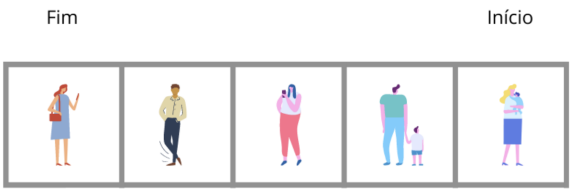
O que são estruturas de dados?

As estruturas de dados são como a chave para organizar o caos no mundo da programação. Imagine que você é o administrador de uma biblioteca gigante com milhares de livros. Como você organizaria todos esses livros para que qualquer pessoa pudesse encontrar o que procura facilmente? É para resolver esses problemas que nós podemos utilizar as **Estruturas de Dados**.

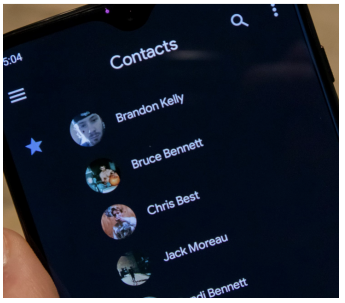
No computador, podemos fazer algo parecido. Quando temos muitos pedaços de informação, como números ou nomes, colocamos eles em “caixinhas especiais” chamadas de **Estruturas de Dados**. Ou seja, são como ferramentas que ajudam a organizar informações para serem acessadas e manipuladas de maneira eficiente.

Agora vou te dar alguns exemplos:

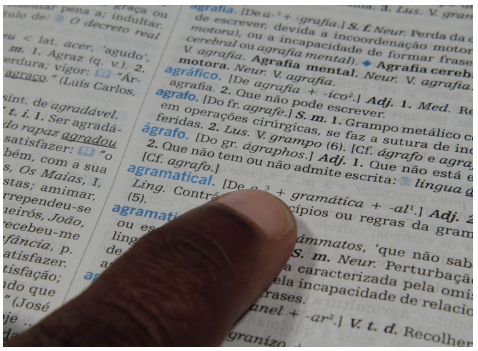
1. **Fila da Padaria (Fila - Queue):** Você já esteve em uma padaria onde as pessoas formam uma fila para comprar pães e bolos? Essa é uma fila na vida real, que funciona exatamente como uma Estrutura de Dados chamada "Queue" (Fila). A primeira pessoa que chega é a primeira a ser atendida.



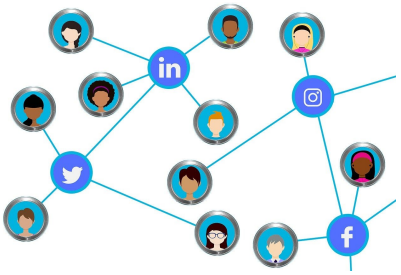
2. **Agenda de Contatos (Lista - Array):** A agenda de contatos no seu celular é um exemplo de uma lista (array). Cada contato é um item na lista, e você pode acessá-los por posição, assim como na estante de livros que mencionamos anteriormente.



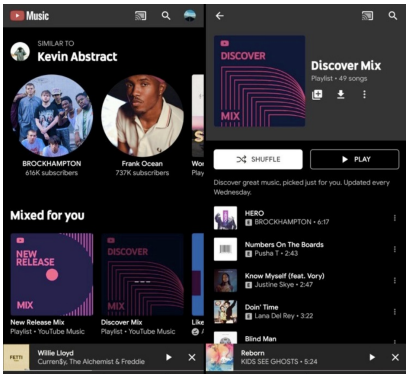
3. **Dicionário (Dicionário - Hashmap):** Pense em um dicionário físico onde você procura palavras e suas definições. No mundo digital, um exemplo é o mecanismo de busca na internet, como o Google. Você digita uma palavra-chave (como "receitas de bolo") e obtém uma lista de resultados relacionados, como um dicionário onde você encontra informações relevantes.



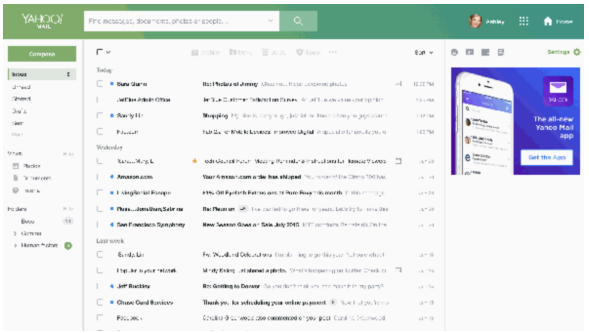
4. **Redes Sociais (Grafo - Graph):** Redes sociais como o Facebook ou o Instagram são como um grande grafo, outra Estrutura de Dados. Os perfis são os nós (ou vértices) e as amizades ou seguidores são as arestas (ou conexões) que ligam esses nós. Isso permite que as redes sociais identifiquem quem é amigo de quem e mostrem seu conteúdo para as pessoas certas.



5. **Playlist de Música (Lista Encadeada - Linked List):** Algumas playlists de música são criadas com uma Estrutura de Dados chamada "Linked List" (Lista Encadeada). Cada música está ligada à próxima, como uma corrente, permitindo que você avance ou retroceda facilmente na lista de reprodução.



6. **Caixa de Emails (Pilha - Stack):** Imagine sua caixa de entrada de e-mail como uma pilha de cartas. A última carta que você recebeu está sempre no topo da pilha. Quando você lê ou remove uma carta, é como desempilhar a última. É por isso que você lê e-mails na ordem em que são recebidos - é uma pilha.



Com certeza você já se deparou com os exemplos acima no seu cotidiano, e agora tem o conhecimento de como essas informações são estruturadas.

Array (Matriz)

Um array em linguagem de programação é uma estrutura de dados que nos permite armazenar uma coleção ordenada de elementos sob um único nome. Esses elementos podem ser números, palavras, objetos, ou qualquer outro tipo de dado que a linguagem de programação suporte.

A principal característica de um **array** é a sua capacidade de organizar os elementos em posições numeradas, **começando geralmente do zero**. Isso facilita o acesso e a manipulação desses elementos, uma vez que você pode se referir a eles pelo índice da posição em que estão armazenados.

Para inicializar uma **variável do tipo Array**, podemos iniciar sem itens: `array = []`

Ou com alguns itens de início:
`array = ["texto", 123, 4.56, True]`

E esses itens podem ser de diversos tipos (
string, número inteiro, número flutuante, booleano).

Para acessar uma posição do array, nós utilizamos uma contagem a partir do 0 (zero). Isso é comum ao lidar com linguagens de programação, pois o computador "conta de 0 a 10 e não de 1 a 10", como nós humanos.

Por isso que no
Python, ao utilizarmos a estrutura de repetição `for valor in range(10)`, sem dizer para função `range()` que deve começar do 1 (um), ele nos traz um resultado a partir do 0 (zero).

Por exemplo, se você tiver um array de números inteiros chamado `idades`, poderá acessar o primeiro elemento usando `idades[0]`, o segundo usando `idades[1]`, e assim por diante.

Em **Python**:

```
idades = [10, 15, 16, 20]

print(idades[0]) # 10
print(idades[1]) # 15
print(idades[2]) # 16
print(idades[3]) # 20
print(idades[4]) # Ao acessar uma posição inexistente, isso resultará em ERROR
```

Não apenas podemos acessar um elemento específico em uma lista, mas também podemos recuperar um período de itens consecutivos dentro da lista. Para fazer isso, utilizamos o operador `:` (dois pontos) para definir o início e o fim do intervalo que desejamos acessar. Isso é particularmente útil quando queremos trabalhar com uma parte específica de uma lista sem a necessidade de percorrê-la inteira.

Vamos explorar isso em um exemplo prático de código para entender melhor. Suponha que temos uma lista chamada `numeros` e queremos acessar os elementos do índice 2 ao índice 5.

Em **Python**:

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
intervalo = numeros[2:6]

print(intervalo) # Resultado: [2, 3, 4, 5]
```

Neste exemplo, `numeros[2:6]` nos dará uma nova lista contendo os elementos dos índices 2, 3, 4 e 5 da lista `numeros`. O intervalo começa no índice 2 e vai até, mas **não inclui**, o índice 6.

Essa capacidade de acessar períodos de itens em listas é valiosa para tarefas como filtrar dados específicos, criar subconjuntos de informações e muito mais. Portanto, lembre-se do operador `:` ao trabalhar com listas em Python, pois ele pode simplificar bastante seu código e tornar suas operações mais eficientes."

👉 Tamanho / Length (comprimento)

len()

- O método `len()` em Python é uma função embutida (built-in) que nos permite determinar o comprimento ou o número de elementos em uma estrutura de dados, como uma lista, uma string, uma tupla, entre outras.

- **Listas**: Para listas, o `len ()` retorna o número de elementos contidos na lista.

Em **Python**:

```
frutas = ["maçã", "banana", "laranja"]

tamanho = len(frutas) # Resultado: 3

# Retorna 3, pois há 3 elementos na lista.
```

- **Strings**: Para strings, o `len ()` retorna o número de caracteres na string.

Em **Python**:

```
texto = "Olá, mundo!"

tamanho = len(texto) # Resultado: 12

# Retorna 12, pois há 12 caracteres na string.
```

> O método `len ()` é útil quando você precisa saber o tamanho de uma estrutura de dados em seu programa. Pode ser usado em loops, condicionais e em várias outras situações em que você precisa tomar decisões com base na quantidade de elementos ou caracteres presentes em uma variável.

👉 Métodos / funções de manipulação de lista (array) em Python

As listas (*arrays*) são estruturas de dados que permitem armazenar e manipular coleções de elementos. Vamos explorar alguns métodos importantes que nos ajudam a trabalhar com listas de maneira eficiente:

---

append()

- O método `append ()` é utilizado para adicionar um elemento ao final de uma lista.

Por exemplo, se temos uma lista (*array*) chamada `frutas` e queremos adicionar a fruta "maçã" ao final da lista, podemos fazer o seguinte.

Em **Python**:

```
frutas = ["banana", "laranja", "uva"]
frutas.append("maçã")

print(frutas) # Resultado: ["banana", "laranja", "uva", "maçã"]
```

➖ **pop(*index*)**

- O método `pop()` é usado para remover um elemento de uma lista com base no índice fornecido e, opcionalmente, retornar o valor removido.
- `index` é o valor referente ao índice que queremos remover. Caso deixemos vazio, ele irá passar o índice `-1` como padrão e remover o último item da lista (*array*).

Por exemplo, se quisermos remover o elemento "laranja" da lista `frutas`, podemos fazer o seguinte.

Em **Python**:

```
frutas = ["banana", "laranja", "uva"]
fruta_removida = frutas.pop(1)

print(frutas) # Resultado: ["banana", "uva"]
print(fruta_removida) # Resultado: "laranja"
```

⏪ **reverse()**

- O método `reverse()` inverte a ordem dos elementos em uma lista (*array*).

Em **Python**:

```
frutas = ["banana", "laranja", "uva"]
frutas.reverse()

print(frutas) # Resultado: ["uva", "laranja", "banana"]
```

🧹 **clear()**

- O método `clear()` é usado para remover todos os elementos de uma lista, deixando-a vazia.

Por exemplo, se temos uma lista (*array*) chamada `frutas` já com alguns itens e queremos limpar/excluir todos os elementos, podemos fazer o seguinte.

Em **Python**:

```
frutas = ["banana", "laranja", "uva"]
frutas.clear()

print(frutas) # Resultado: []
```

Arrays são amplamente utilizados na programação para armazenar e processar dados de maneira eficiente. Eles são úteis quando você precisa lidar com conjuntos de informações semelhantes, como uma lista de nomes, notas de alunos, ou registros de vendas. É uma maneira poderosa de organizar e trabalhar com dados em programas de computador.

Para saber mais, eu vou deixar dois links de um material externo para consulta:

5. Estruturas de dados

Esse capítulo descreve algumas coisas que você já aprendeu em detalhes e adiciona algumas coisas novas também. Mais sobre listas: O tipo de dado lista tem ainda mais métodos. Aqui estão apresentado...

https://docs.python.org/pt-br/3/tutorial/datastructures.html

Listas — Como pensar como um Cientista da Computação: Edição Interativa em Python

Uma lista (list) em Python é uma sequência ou coleção ordenada de valores. Cada valor na lista é identificado por um índice. O valores que formam uma lista são chamados elementos ou itens.

https://panda.ime.usp.br/pensepy/static/pensepy/09-Listas/listas.html

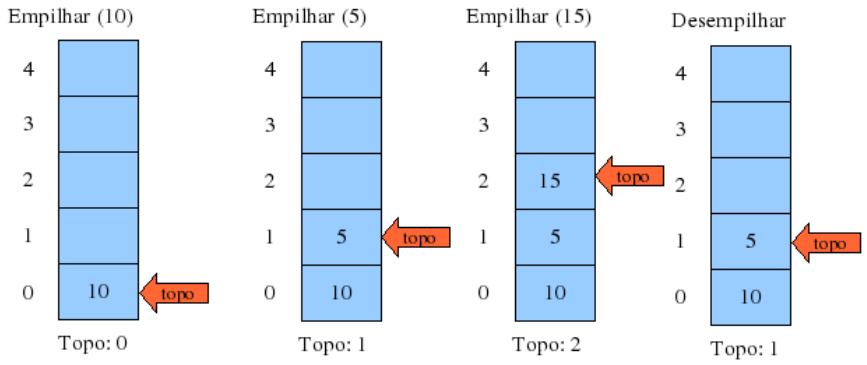
Agora, vamos entender como podemos **manipular o Array** para resolver diversos problemas. 👇

Pilhas (Stacks) - LIFO

A Pilha é uma estrutura de dados bastante simples, basta você pensar em uma pilha de pratos, onde o último prato colocado, é o primeiro a ser retirado da pilha.

> Esse fluxo de **"último item a entrar na pilha, é o primeiro item a sair da fila"** é chamado de **LIFO**, que é a sigla para *Last In, First Out*.
>
> Traduzindo do inglês para o português, entendemos:
> **Primeiro** a entrar, **Último** a sair.

Exemplo visual:



A pilha pode ser representada como na imagem acima, onde temos uma lista organizada de baixo para cima. Isso é, todos os itens adicionados na lista, vão para o topo, sendo empilhados. Agora, ao desempilhar, vamos buscar o último item adicionado, aquele que está no topo da lista.

Para que possamos manipular uma lista (*array*) de forma que se comporte como uma **pilha**, podemos definir o código abaixo.

Primeiro criamos uma função chamada `adicionar_lifo` que recebe dois parâmetros, `pilha` e `item`, onde a `pilha` é a lista que queremos adicionar o `item`. E dentro dessa função, usamos o método `append(item)` (que também é uma função) para adicionar o item no final da lista, que é o topo da pilha.

```
def adicionar_lifo(pilha, item):
  return pilha.append(item)
```

Depois criamos outra função chamada `remover_lifo` que recebe apenas um parâmetro, que é a `pilha` que queremos remover um item do topo (desempilhar). Assim, fazemos o uso do método `pop()` e não passamos

nenhum parâmetro, pois por padrão, o método `pop()` já repassa o índice `-1` e remove o último item da lista, que é justamente o comportamento que esperamos dentro de uma pilha: remover o último item adicionado.

```
def remover_lifo(pilha):
  return pilha.pop()
```

Podemos representar o uso desse código em **Python**:

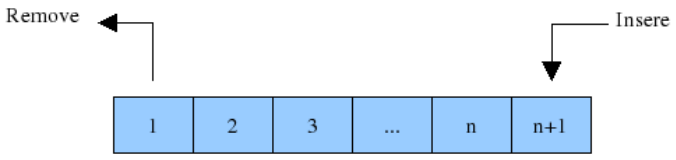
```
# Define uma função para adicionar a pilha
# usando o método append()
def adicionar_lifo(pilha, item):
  return pilha.append(item)

# Define uma função para remover da pilha
# usando o método pop()
def remover_lifo(pilha):
  return pilha.pop()

# Inicializa um array (lista) com alguns itens
pilha_pratos = ['P1', 'P2', 'P3', 'P4']

# Invoca a função para adicionar um item diretamente na pilha
adicionar_lifo(pilha_pratos, 'P5')
print(pilha_pratos) # Resultado: ['P1', 'P2', 'P3', 'P4', 'P5']

# Invoca a função para remover um item diretamente da pilha
remover_lifo(pilha_pratos)
print(pilha_pratos) # Resultado: ['P1', 'P2', 'P3', 'P4']
```

Filas (Queues) - FIFO

A Fila é uma estrutura de dados bem conhecida, basta você pensar em uma fila de banco, onde o primeiro a chegar é o primeiro a ser atendido.

> Esse fluxo de **"primeiro item a entrar na pilha, é o primeiro item a sair da fila"** é chamado de **FIFO**, que é a sigla para *First In, First Out*.
>
> Traduzindo do inglês para o português, entendemos:
> **Primeiro** a entrar, **Primeiro** a sair.

Exemplo visual:



Para que possamos manipular uma lista (*array*) de forma que se comporte como uma **fila**, podemos definir o código abaixo.

Primeiro criamos uma função chamada `adicionar_fifo` que recebe dois parâmetros, `pilha` e `item`, onde a `pilha` é a lista que queremos adicionar o `item`. E dentro dessa função, usamos o método `append(item)` (que também é uma função) para adicionar o item no final da lista, que é o topo da pilha.

```
def adicionar_fifo(fila, item):
  return fila.append(item)
```

Depois criamos outra função chamada `remover_lifo_pop` que recebe apenas um parâmetro, que é a `fila` que queremos remover um item do início. Assim, fazemos o uso do método `pop()` e passamos o parâmetro com índice 0 (zero), sendo `pop(0)`, para que remova o primeiro item da lista que está na posição inicial (0 - zero). Assim, temos o comportamento que esperamos dentro de uma fila: remover o primeiro item adicionado.

```
def remover_fifo_pop(fila):
  return fila.pop(0)
```

Agora, a título de aprendizado, podemos testar uma outra forma de remover um item da fila. Para isso vamos definir uma função chamada `remover_fifo` que também irá receber apenas um parâmetro, que é a `fila` que queremos remover um item do início. Entretanto, não utilizaremos o método `pop()` e sim o conceito de acessar um período de índices dentro de uma lista (array), utilizando o operador `:` (dois pontos), como aprendemos anteriormente.

Só que desse modo, não podemos simplesmente invocar a função, precisamos também redefinir a nossa variável `fila` para o resultado retornado (por meio do `return` da função), pois assim **definiremos o novo valor da fila de forma explícita**.

Diferentemente do `pop()`, que manipula uma lista de **forma implícita**, já que atua diretamente na variável onde está a fila sem precisar reatribui-la.


```
def remover_fifo(fila):
  return fila[1:]

fila_banco = remover_fifo(fila_banco)
```

Podemos representar o uso desse código em **Python**:

```
# Define uma função para adicionar a fila
# usando o método append()
def adicionar_fifo(fila, item):
  return fila.append(item)

# Define uma função para remover da fila
# usando o retorno de um período
def remover_fifo(fila):
  return fila[1:]

# Define uma função para remover da fila
# usando o método pop()
def remover_fifo_pop(fila):
  return fila.pop(0)

# Inicializa um array (lista) com alguns itens
fila_banco = ["João", "Karol", "Davi", "Raquel"]

# Invoca a função para adicionar um item diretamente na fila
```

```
adicionar_fifo(fila_banco, "Sarah")
print(fila_banco) # Resultado: ["João", "Karol", "Davi", "Raquel", "Sarah"]

# Invoca a função para remover um item diretamente da fila
remover_fifo_pop(fila_banco)
print(fila_banco) # Resultado: ["Karol", "Davi", "Raquel", "Sarah"]

# Invoca a função para remover um item da fila
# retornando uma nova fila
fila_banco = remover_fifo(fila_banco)

print(fila_banco) # Resultado: ["Davi", "Raquel", "Sarah"]
```

Estruturas de repetição e estruturas de dados

Vamos falar sobre estruturas de repetição em Python, como o `for` e o `while`, e como usá-los para iterar sobre listas e dicionários.

1. Estrutura de Repetição `for` com Listas:

O `for` é ótimo para percorrer cada item em uma lista. Imagine que temos uma lista de frutas e queremos imprimir cada uma delas:

```
frutas = ["maçã", "banana", "laranja", "uva"]

for fruta in frutas:
    print(fruta)
```

Neste exemplo, o `for` percorre a lista `frutas` e a variável `fruta` recebe cada elemento da lista em cada iteração. O resultado será a impressão de cada fruta.

2. Estrutura de Repetição `for` com Dicionários:

Agora, vamos usar o `for` para iterar sobre um dicionário. Suponha que temos um dicionário de contatos e queremos imprimir cada nome e número de telefone:

```
contatos = {
    "Alice": "123-456-7890",
    "Bob": "987-654-3210",
    "Carol": "555-123-4567"
}

for nome, telefone in contatos.items():
    print(f"{nome}: {telefone}")
```

Neste caso, o `for` percorre o dicionário, e a variável `nome` recebe a chave (nome do contato) e a variável `telefone` recebe o valor (número de telefone) em cada iteração.

3. Estrutura de Repetição `while` com Listas:

O `while` é útil quando você precisa repetir algo enquanto uma condição for verdadeira. Vamos usar um exemplo de lista para encontrar um item específico:

```
nomes = ["Alice", "Bob", "Carol", "David"]
nome_procurado = "Carol"
encontrado = False
indice = 0

while not encontrado and indice < len(nomes):
```

```
    if nomes[indice] == nome_procurado:
        encontrado = True
    else:
        indice += 1

if encontrado:
    print(f"{nome_procurado} foi encontrado na posição {indice}")
else:
    print(f"{nome_procurado} não foi encontrado na lista")
```

Neste caso, usamos um loop `while` para verificar se o nome procurado está na lista. O loop continua até encontrar o nome ou percorrer toda a lista.

4. Estrutura de Repetição `while` com Dicionários (Parte 2):

Digamos que temos um dicionário de produtos e seus preços, e queremos encontrar o produto mais caro. Usaremos um loop `while` para fazer isso.

```
produtos = {
    "Maçã": 2.50,
    "Banana": 1.50,
    "Laranja": 2.00,
    "Abacaxi": 3.50,
}

produto_mais_caro = None
preco_mais_caro = 0

while produtos:
    produto, preco = produtos.popitem()
    if preco > preco_mais_caro:
        produto_mais_caro = produto
        preco_mais_caro = preco

print(f"O produto mais caro é '{produto_mais_caro}' com preço de R${preco_mais_caro:.2
f}")
```

Neste exemplo, estamos usando um loop `while` para percorrer o dicionário `produtos`. A cada iteração, verificamos se o preço do produto atual é maior do que o preço do produto mais caro encontrado até agora. Se for, atualizamos a variável `produto_mais_caro` com o nome do produto e a variável `preco_mais_caro` com o preço correspondente.

No final do loop, imprimimos o produto mais caro encontrado.

Espero que esses exemplos tenham sido úteis e fáceis de entender! As estruturas de repetição `for` e `while` são fundamentais para percorrer e manipular dados em Python.

Complexidade de um algoritmo

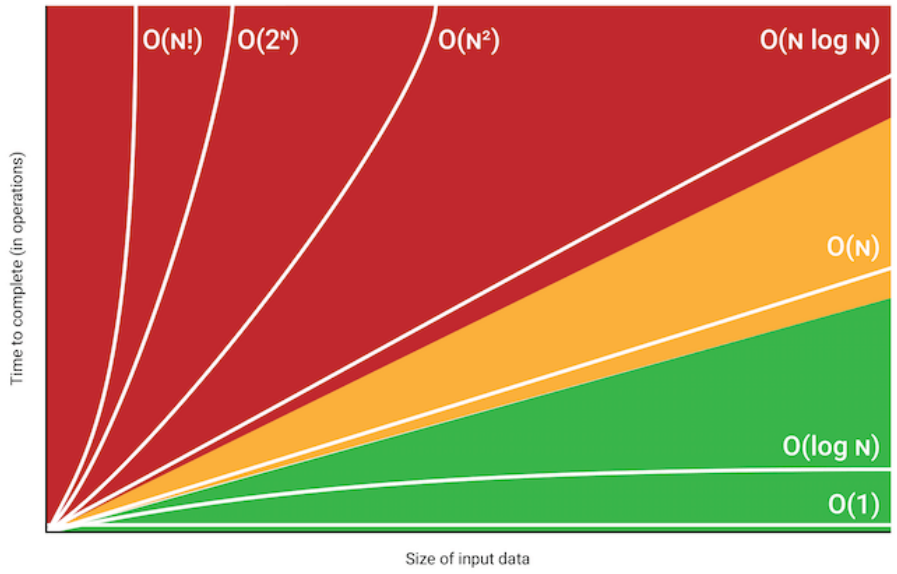
Notação Big O

A notação "Big O" (ou "O Grande O") é uma forma de descrever o desempenho ou eficiência de um algoritmo. Ela é usada para entender o quão rápido um algoritmo cresce à medida que a quantidade de dados de entrada aumenta. Vou explicar de forma simples:

> **Big O é como medimos o "quão rápido" ou "quão lento" um algoritmo é.**

A ideia principal é responder a pergunta: "Se eu dobrar a quantidade de dados que estou processando, o tempo que meu algoritmo leva para rodar também dobrará? Ou ele levará mais tempo?"

Aqui estão alguns exemplos simples para ajudar a entender:



1. **O(1) - Constante:** Isso significa que o algoritmo leva o mesmo tempo, não importa o tamanho dos dados. É como se o tempo fosse constante, não importa quão grande seja o problema. Um exemplo disso é acessar um elemento em uma lista pelo índice - isso leva um tempo constante, não importa o tamanho da lista.
2. **O(n) - Linear:** Isso significa que o tempo de execução do algoritmo cresce na mesma proporção que o tamanho dos dados. Se você tem uma lista com 100 elementos e leva 10 segundos para processá-la, se você tiver uma lista com 200 elementos, levará cerca de 20 segundos.
3. **O(n^2) - Quadrático:** Isso significa que o tempo de execução do algoritmo cresce quadrado em relação ao tamanho dos dados. Se você tem uma lista com 100 elementos e leva 10 segundos para processá-la, se você tiver uma lista com 200 elementos, levará cerca de 40 segundos (10 × 4).
4. **O(log n) - Logarítmico:** Este é um dos mais eficientes em termos de tempo. Significa que, à medida que a quantidade de dados (n) aumenta, o tempo de execução do algoritmo aumenta, mas não de forma linear. Em vez disso, ele cresce de acordo com o logaritmo do tamanho dos dados. Isso faz com que algoritmos com essa notação sejam muito rápidos para grandes conjuntos de dados.

   Um exemplo clássico é a busca binária em uma lista ordenada. Imagine ter uma lista de 1.000 números ordenados e você quer encontrar um número específico. Usando a busca binária, você divide a lista pela metade repetidamente, o que significa que você elimina metade dos elementos a cada passo. Isso é muito mais rápido do que verificar um por um.
5. **O(n log n) - Linearítmico:** Este é um pouco menos eficiente que o O(log n), mas ainda é muito rápido. Significa que o tempo de execução do algoritmo cresce linearmente com os dados, mas também é multiplicado pelo logaritmo do tamanho dos dados. Isso é comum em algoritmos de classificação eficientes, como o algoritmo QuickSort e o algoritmo MergeSort. Quando você classifica uma grande lista de dados, esses algoritmos dividem a lista em partes menores, classificam cada parte e, em seguida, mesclam essas partes classificadas. O logaritmo entra na divisão e mesclagem das partes, tornando o algoritmo geral muito rápido, mesmo para grandes conjuntos de dados.

A ideia é escolher algoritmos com notações "Big O" menores sempre que possível, porque eles são mais eficientes. Por exemplo, um algoritmo O(n) é mais eficiente do que um O(n^2).

Pense no "Big O" como uma maneira de comparar algoritmos e escolher o mais eficiente para o seu problema. Quanto menor a notação "Big O", mais rápido o algoritmo será em relação ao tamanho dos dados.

Eu aproveito para deixar algumas video-aulas públicas no YouTube para auxiliar na compreensão:

NOTAÇÃO BIG-O COMO CLASSIFICAR A COMPLEXIDADE DE UM CÓDIGO? VOCÊ PRECISA SABER!

Sua aplicação escala ou desaba? Saiba que um simples trecho de código é capaz de destruir a performance da sua aplicação.

https://youtu.be/KjJwx-AB4KI?si=FdFqG76tHxLTINMS

O que é a Notação Big O em Algoritmos

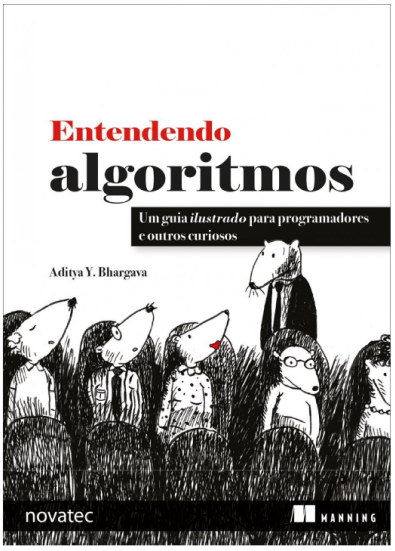
O que é a Notação Big O em Algoritmos

Neste vídeo explico o que é a Notação Big O, empregada para realizar comparação de

https://www.youtube.com/watch?v=JTIpS1WIgXg&ab_channel=BósonTreinamentos

Aproveite para seguir os canais 😉

Literatura: Além do analógico



Caso você tenha curiosidade, separei alguns livros para que você possa já deixar na sua rotina de estudos e aprender um pouco mais sobre os assuntos que abordamos até aqui.

O livro “Entendendo Algoritmos” é bem amigável, cada página possui uma ilustração dos temas e explicações abordadas, tornando a experiência bem interativa. E o livro “Código Limpo” é padrão, um pouco mais denso por não ter ilustrações, mas é uma leitura essencial para quem quer ser um programador.

Dica: se você não tem costume de ler, tente tornar a leitura um hábito; separe 10 minutinhos todos os dias, leia de 2 a 3 páginas e faça as suas anotações sempre que possível. O importante é exercitar a leitura, aprendendo um pouco a cada dia.

Links para compra pela Amazon

Entendendo Algoritmos: Um Guia Ilustrado Para Programadores e Outros Curiosos

Compre online Entendendo Algoritmos: Um Guia Ilustrado Para Programadores e Outros Curiosos, de Bhargava, Aditya Y. na Amazon. Frete GRÁTIS em milhares de produtos com o Amazon Prime. Encontre diversos livros escritos por Bhargava, Aditya Y. com ótimos preços.

https://a.co/d/24U1Ncj

Código limpo: Habilidades práticas do Agile Software

Compre online Código limpo: Habilidades práticas do Agile Software, de Martin, Robert C. na Amazon. Frete GRÁTIS em milhares de produtos com o Amazon Prime. Encontre diversos livros escritos por Martin, Robert C. com ótimos preços.

https://a.co/d/dA7vLGR

Projeto - Hands on 👋💻

Sistema Bancário

Neste exemplo, vamos criar um algoritmo que simule as principais operações de um sistema bancário. Para isso, utilizaremos todos os conhecimentos aprendidos até aqui, aplicando estruturas de repetição e estruturas de dados na prática para resolver o nosso problema. As interações com o programa irão acontecer por meio de *inputs* que serão informados a cada etapa.

Introdução

Crie uma lógica para simular um fluxo de um sistema bancário.

Esse programa deverá ter:

1. Um menu com seleção de operação
2. Criação de contas
3. Remoção de contas
4. Listagem de todas as contas criadas
5. Adicionar saldo / Remover saldo
6. Transferir valor entre contas
7. Consultar saldo de uma conta específica

As contas devem ser armazenadas em uma lista e cada conta deve ser um dicionário (chave-valor) contendo as seguintes chaves:

1. numero_conta
2. saldo_conta

🚩 Aprofundamento os conhecimentos

Quebra de linha com \n

🏁 Para “quebrar uma linha” enquanto estamos programando, por exemplo, ao utilizar um `print()` ou um `input()`, onde passamos uma mensagem como parâmetro, nós pode utilizar do `\n` (contra-barra n) para sinalizar uma “quebra de linha”.

Formatação especial de números utilizando template strings

Em Python, o `:.2f` é usado em f-strings para formatar números como ponto flutuante (números com casas decimais). Vou explicar cada parte:

- O `:` indica que estamos fazendo uma formatação especial.
- O `.2` indica que queremos duas casas decimais após o ponto decimal.
- O `f` significa que estamos formatando um número de ponto flutuante (ou seja, um número real).

Agora, vamos ver um exemplo para entender melhor. Suponha que temos uma variável `preco` com o valor `10.56789` e queremos formatá-la com duas casas decimais:

```
preco = 10.56789
mensagem = f"O preço é: {preco:.2f} reais"
print(mensagem)
```

Aqui, a f-string `{preco:.2f}` vai formatar o valor da variável `preco` como `10.57`, arredondando para duas casas decimais.

Isso é útil quando você precisa exibir números de forma mais legível, como em preços, porcentagens ou medidas com casas decimais.

Para saber mais sobre o uso de *f-strings* (*template strings*) e formação especial:

Manipulação de Strings com f-strings no Python

Aprenda a manipular strings em Python de um jeito fácil e descomplicado com f-strings!

https://pythonacademy.com.br/blog/f-strings-no-python

string — Common string operations

Source code: Lib/string.py String constants: The constants defined in this module are: Custom String Formatting: The built-in string class provides the ability to do complex variable substitutions ...

https://docs.python.org/3/library/string.html#format-specification-mini-language

Python 3's F-Strings: An Improved String Formatting Syntax (Guide) – Real Python

As of Python 3.6, f-strings are a great new way to format strings. Not only are they more readable, more concise, and less prone to error than other ways of formatting, but they are also faster! By the end of this article, you will learn how and why to start using f-strings today.

https://realpython.com/python-f-strings/

💻 Escrevendo código

▼ Código - Parte 1

Vamos criar a base do programa com menu.

```
print(">>> Bem vindo ao sistema Bancário <<<")
programa_ativo = True
while programa_ativo == True:
  print("\n### Menu ###")
  print("0 - Sair")
  numero_operacao = input("Selecione a operação que deseja realizar:\n>>> ")

  if numero_operacao == "0":
    print("\nSistema encerrado.")
    programa_ativo = False

  else:
    print("\nOperação invalida.")
```

▼ Código - Parte 2

Vamos reestruturar o nosso código. Adicionar mais duas operações e trocar a nossa condição do WHILE para algo mais simples, objetivo e funcional.

```
contas = []

print(">>> Bem vindo ao sistema Bancário <<<")

while True:
  print("\n### Menu ###")
  print("0 - Sair")
  print("1 - Criar uma nova conta")
  print("2 - Remover uma conta")
  numero_operacao = input("Selecione a operação que deseja realizar:\n>>> ")

  if numero_operacao == "0":
    print("\nSistema encerrado.")
    break

  # Operação 1 - Criar uma nova conta
  elif numero_operacao == "1":
    verificador_conta_existe = False
    dados_nova_conta = {}
    dados_nova_conta['numero_conta'] = input("Digite o número da nova conta:\n>>> ")

    for conta_em_consulta in contas:
      if conta_em_consulta['numero_conta'] == dados_nova_conta['numero_conta']:
        verificador_conta_existe = True

    if verificador_conta_existe == True:
      print("\nO número da conta já existe. Tente novamente.")
    else:
      dados_nova_conta['saldo_conta'] = float(input("Digite o saldo da nova conta:\n>>
      contas.append(dados_nova_conta)
      print("\nOperacao efetuada com sucesso.")

  # Operação 2 - Remover uma conta
  elif numero_operacao == "2":
    numero_conta = input("Digite o número da conta para remover:\n>>> ")
    conta_encontrada = False
    for conta_em_consulta in contas:
      if conta_em_consulta['numero_conta'] == numero_conta:
        conta_encontrada = True
        contas.remove(conta_em_consulta)
        print("\nOperação efetuada com sucesso.")
    if not conta_encontrada:
      print("\nO número da conta não existe. Tente novamente.")

  # Operação inválida
  else:
    print("\nOperação invalida.")
```

▼ Código - Parte 3

Vamos reestruturar (refatorar) o nosso código e passar a usar funções.

```
contas = []

def criar_conta():
  verificador_conta_existe = False
  dados_nova_conta = {}
  dados_nova_conta['numero_conta'] = input('Digite o número da nova conta:\n>>> ')

  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == dados_nova_conta['numero_conta']:
      verificador_conta_existe = True
      break

  if verificador_conta_existe == True:
    print('\nO número da conta já existe. Tente novamente.')
  else:
    dados_nova_conta['saldo_conta'] = float(input('Digite o saldo da nova conta:\n>>>
    contas.append(dados_nova_conta)
    print('\nOperação efetuada com sucesso.')

def remover_conta():
  numero_conta = input('Digite o número da conta para remover:\n>>> ')
  conta_encontrada = False
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      conta_encontrada = True
      contas.remove(conta_em_consulta)
      print('\nOperação efetuada com sucesso.')
      break
  if not conta_encontrada:
    print('\nO número da conta não existe. Tente novamente.')

print(">>> Bem vindo ao sistema Bancário <<<")

while True:
  print("\n### Menu ###")
  print("0 - Sair")
  print("1 - Criar uma nova conta")
  print("2 - Remover uma conta")
  numero_operacao = input("Selecione a operação que deseja realizar:\n>>> ")

  if numero_operacao == "0":
    print("\nSistema encerrado.")
    break

  # Operação 1 - Criar uma nova conta
  elif numero_operacao == "1":
    criar_conta()

  # Operação 2 - Remover uma conta
  elif numero_operacao == "2":
    remover_conta()

  # Operação inválida
```

```
  else:
    print("\nOperação invalida.")
```

▼ Código - Parte 4

Vamos criar mais uma operação: Listar Contas.

```
contas = []

def criar_conta():
  verificador_conta_existe = False
  dados_nova_conta = {}
  dados_nova_conta['numero_conta'] = input('Digite o número da nova conta:\n>>> ')

  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == dados_nova_conta['numero_conta']:
      verificador_conta_existe = True
      break

  if verificador_conta_existe == True:
    print('\nO número da conta já existe. Tente novamente.')
  else:
    dados_nova_conta['saldo_conta'] = float(input('Digite o saldo da nova conta:\n>>>
    contas.append(dados_nova_conta)
    print('\nOperação efetuada com sucesso.')

def remover_conta():
  numero_conta = input('Digite o número da conta para remover:\n>>> ')
  conta_encontrada = False
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      conta_encontrada = True
      contas.remove(conta_em_consulta)
      print('\nOperação efetuada com sucesso.')
      break
  if not conta_encontrada:
    print('\nO número da conta não existe. Tente novamente.')

def listar_contas():
  if len(contas) == 0:
    print('\nNão há contas cadastradas.')
  else:
    index = 1
    for conta_em_consulta in contas:
      if conta_em_consulta['saldo_conta'] >= 0:
        status = 'Positivo'
      else:
        status = 'Negativo'
      # format string - f string
      print(f"{index} - Número da conta: {conta_em_consulta['numero_conta']} | Saldo d
      index += 1

print(">>> Bem vindo ao sistema Bancário <<<")

while True:
```

```
  print("\n### Menu ###")
  print("0 - Sair")
  print("1 - Criar uma nova conta")
  print("2 - Remover uma conta")
  print("3 - Listar todas as contas")
  numero_operacao = input("Selecione a operação que deseja realizar:\n>>> ")

  if numero_operacao == "0":
    print("\nSistema encerrado.")
    break

  # Operação 1 - Criar uma nova conta
  elif numero_operacao == "1":
    criar_conta()

  # Operação 2 - Remover uma conta
  elif numero_operacao == "2":
    remover_conta()

  # Operação 3 - Listar todas as contas
  elif numero_operacao == "3":
    listar_contas()

  # Operação inválida
  else:
    print("\nOperação invalida.")
```

▼ Código - Parte 5

Agora, vamos adicionar mais duas operações:
Adicionar Saldo (crédito) e Remover Saldo (débito)

```
contas = []

def criar_conta():
  verificador_conta_existe = False
  dados_nova_conta = {}
  dados_nova_conta['numero_conta'] = input('Digite o número da nova conta:\n>>> ')

  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == dados_nova_conta['numero_conta']:
      verificador_conta_existe = True
      break

  if verificador_conta_existe == True:
    print('\nO número da conta já existe. Tente novamente.')
  else:
    dados_nova_conta['saldo_conta'] = float(input('Digite o saldo da nova conta:\n>>>
    contas.append(dados_nova_conta)
    print('\nOperação efetuada com sucesso.')

def remover_conta():
  numero_conta = input('Digite o número da conta para remover:\n>>> ')
  conta_encontrada = False
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
```

```
      conta_encontrada = True
      contas.remove(conta_em_consulta)
      print('\nOperação efetuada com sucesso.')
      break
  if not conta_encontrada:
    print('\nO número da conta não existe. Tente novamente.')

def listar_contas():
  if len(contas) == 0:
    print('\nNão há contas cadastradas.')
  else:
    index = 1
    for conta_em_consulta in contas:
      if conta_em_consulta['saldo_conta'] >= 0:
        status = 'Positivo'
      else:
        status = 'Negativo'
      # format string - f string
      print(f"{index} - Número da conta: {conta_em_consulta['numero_conta']} | Saldo d
      index += 1

def adicionar_saldo_em_conta():
  numero_conta = input("Digite o número da conta:\n>>> ")
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      valor_credito = float(input("Digite o valor do crédito:\n>>> R$ "))
      if valor_credito < 0:
        print("\nValores negativos não são aceitos. Tente novamente.")
        return
      else:
        conta_em_consulta['saldo_conta'] += valor_credito
        print("\nOperação efetuada com sucesso.")
        return
  print("\nO número da conta não foi encontrado. Tente novamente.")


def remover_saldo_em_conta():
  numero_conta = input("Digite o número da conta:\n>>> ")
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      valor_debito = float(
      input("Digite o valor do débito:\n>>> R$ "))
      if valor_debito < 0:
        print("\nValores negativos não são aceitos. Tente novamente.")
        return
      else:
        conta_em_consulta['saldo_conta'] -= valor_debito
        print("\nOperação efetuada com sucesso.")
        return
  print("\nO número da conta não foi encontrado. Tente novamente.")

print(">>> Bem vindo ao sistema Bancário <<<")

while True:
  print("\n### Menu ###")
```

```
  print("0 - Sair")
  print("1 - Criar uma nova conta")
  print("2 - Remover uma conta")
  print("3 - Listar todas as contas")
  print("4 - Adicionar saldo (creditar)")
  print("5 - Remover saldo (debitar)")
  numero_operacao = input("Selecione a operação que deseja realizar:\n>>> ")

  if numero_operacao == "0":
    print("\nSistema encerrado.")
    break

  # Operação 1 - Criar uma nova conta
  elif numero_operacao == "1":
    criar_conta()

  # Operação 2 - Remover uma conta
  elif numero_operacao == "2":
    remover_conta()

  # Operação 3 - Listar todas as contas
  elif numero_operacao == "3":
    listar_contas()

  # Operação 4 - Crédito
  elif numero_operacao == "4":
    adicionar_saldo_em_conta()

  # Operação 5 - Débito
  elif numero_operacao == "5":
    remover_saldo_em_conta()

  # Operação inválida
  else:
    print("\nOperação invalida.")
```

▼ Código - Parte 6

Vamos criar a operação de Transferir Saldo entre Contas.

```
contas = []

def criar_conta():
  verificador_conta_existe = False
  dados_nova_conta = {}
  dados_nova_conta['numero_conta'] = input('Digite o número da nova conta:\n>>> ')

  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == dados_nova_conta['numero_conta']:
      verificador_conta_existe = True
      break

  if verificador_conta_existe == True:
    print('\nO número da conta já existe. Tente novamente.')
  else:
    dados_nova_conta['saldo_conta'] = float(input('Digite o saldo da nova conta:\n>>>
```

```
    contas.append(dados_nova_conta)
    print('\nOperação efetuada com sucesso.')

def remover_conta():
  numero_conta = input('Digite o número da conta para remover:\n>>> ')
  conta_encontrada = False
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      conta_encontrada = True
      contas.remove(conta_em_consulta)
      print('\nOperação efetuada com sucesso.')
      break
  if not conta_encontrada:
    print('\nO número da conta não existe. Tente novamente.')

def listar_contas():
  if len(contas) == 0:
    print('\nNão há contas cadastradas.')
  else:
    index = 1
    for conta_em_consulta in contas:
      if conta_em_consulta['saldo_conta'] >= 0:
        status = 'Positivo'
      else:
        status = 'Negativo'
      # format string - f string
      print(f"{index} - Número da conta: {conta_em_consulta['numero_conta']} | Saldo d
      index += 1

def adicionar_saldo_em_conta():
  numero_conta = input('Digite o número da conta:\n>>> ')
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      valor_credito = float(input('Digite o valor do crédito:\n>>> R$ '))
      if valor_credito < 0:
        print('\nValores negativos não são aceitos. Tente novamente.')
        return
      else:
        conta_em_consulta['saldo_conta'] += valor_credito
        print('\nOperação efetuada com sucesso.')
        return
  print('\nO número da conta não foi encontrado. Tente novamente.')

def remover_saldo_em_conta():
  numero_conta = input('Digite o número da conta:\n>>> ')
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      valor_debito = float(input('Digite o valor do débito:\n>>> R$ '))
      if valor_debito < 0:
        print('\nValores negativos não são aceitos. Tente novamente.')
        return
      else:
        conta_em_consulta['saldo_conta'] -= valor_debito
        print('\nOperação efetuada com sucesso.')
        return
```

```
  print('\nO número da conta não foi encontrado. Tente novamente.')

def transferir_saldo_entre_contas():
  conta_partida = input('Digite o número da sua conta:\n>>> ')
  verificador_etapa = 0
  # Procura a primeira conta
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == conta_partida:
      verificador_etapa = 1
      conta_destino = input('Digite o número da conta destino:\n>>> ')
      # Verifica se é uma transferencia para a mesma conta
      if conta_destino != conta_partida:
        # Procura a segunda conta
        for conta_destino_em_consulta in contas:
          if conta_destino_em_consulta['numero_conta'] == conta_destino:
            verificador_etapa = 3
            valor_transferencia = float(input('Digite o valor da transferência:\n>>> R
            # Verifica se o valor é negativo
            if valor_transferencia < 0:
              print('\nNão é possível transferir um valor negativo.')
              break
            # Verifica se o saldo é suficiente
            elif valor_transferencia > conta_em_consulta['saldo_conta']:
              print('\nNão é possível transferir um valor maior que o seu saldo.')
              break
            else:
              conta_em_consulta['saldo_conta'] -= valor_transferencia
              conta_destino_em_consulta['saldo_conta'] += valor_transferencia
              print('\nOperação efetuada com sucesso.')
              break

        print()
      else:
        verificador_etapa = 2
        break

  if verificador_etapa == 0:
    print('\nA sua conta não foi encontrada. Tente novamente.')
  elif verificador_etapa == 1:
    print('\nA conta de destino não foi encontrada. Tente novamente.')
  elif verificador_etapa == 2:
    print('\nNão é possível transferir dinheiro para a mesma conta.')

print(">>> Bem vindo ao sistema Bancário <<<")

while True:
  print("\n### Menu ###")
  print("0 - Sair")
  print("1 - Criar uma nova conta")
  print("2 - Remover uma conta")
  print("3 - Listar todas as contas")
  print("4 - Adicionar saldo (creditar)")
  print("5 - Remover saldo (debitar)")
  print("6 - Transferir saldo entre contas")
  numero_operacao = input("Selecione a operação que deseja realizar:\n>>> ")
```

```
  if numero_operacao == "0":
    print("\nSistema encerrado.")
    break

  # Operação 1 - Criar uma nova conta
  elif numero_operacao == "1":
    criar_conta()

  # Operação 2 - Remover uma conta
  elif numero_operacao == "2":
    remover_conta()

  # Operação 3 - Listar todas as contas
  elif numero_operacao == "3":
    listar_contas()

  # Operação 4 - Crédito
  elif numero_operacao == "4":
    adicionar_saldo_em_conta()

  # Operação 5 - Débito
  elif numero_operacao == "5":
    remover_saldo_em_conta()

  # Operação 6 - Transferir entre contas
  elif numero_operacao == "6":
    transferir_saldo_entre_contas()

  # Operação inválida
  else:
    print("\nOperação invalida.")
```

▼ Código - Parte 7

Por fim, vamos criar a operação de Consultar Saldo.

```
contas = []

def criar_conta():
  verificador_conta_existe = False
  dados_nova_conta = {}
  dados_nova_conta['numero_conta'] = input('Digite o número da nova conta:\n>>> ')

  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == dados_nova_conta['numero_conta']:
      verificador_conta_existe = True
      break

  if verificador_conta_existe == True:
    print('\nO número da conta já existe. Tente novamente.')
  else:
    dados_nova_conta['saldo_conta'] = float(input('Digite o saldo da nova conta:\n>>>
    contas.append(dados_nova_conta)
    print('\nOperação efetuada com sucesso.')
```

```
def remover_conta():
  numero_conta = input('Digite o número da conta para remover:\n>>> ')
  conta_encontrada = False
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      conta_encontrada = True
      contas.remove(conta_em_consulta)
      print('\nOperação efetuada com sucesso.')
      break
  if not conta_encontrada:
    print('\nO número da conta não existe. Tente novamente.')

def listar_contas():
  if len(contas) == 0:
    print('\nNão há contas cadastradas.')
  else:
    index = 1
    for conta_em_consulta in contas:
      if conta_em_consulta['saldo_conta'] >= 0:
        status = 'Positivo'
      else:
        status = 'Negativo'
      # format string - f string
      print(f"{index} - Número da conta: {conta_em_consulta['numero_conta']} | Saldo d
      index += 1

def adicionar_saldo_em_conta():
  numero_conta = input('Digite o número da conta:\n>>> ')
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      valor_credito = float(input('Digite o valor do crédito:\n>>> R$ '))
      if valor_credito < 0:
        print('\nValores negativos não são aceitos. Tente novamente.')
        return
      else:
        conta_em_consulta['saldo_conta'] += valor_credito
        print('\nOperação efetuada com sucesso.')
        return
  print('\nO número da conta não foi encontrado. Tente novamente.')

def remover_saldo_em_conta():
  numero_conta = input('Digite o número da conta:\n>>> ')
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      valor_debito = float(input('Digite o valor do débito:\n>>> R$ '))
      if valor_debito < 0:
        print('\nValores negativos não são aceitos. Tente novamente.')
        return
      else:
        conta_em_consulta['saldo_conta'] -= valor_debito
        print('\nOperação efetuada com sucesso.')
        return
  print('\nO número da conta não foi encontrado. Tente novamente.')

def transferir_saldo_entre_contas():
```

```
  conta_partida = input('Digite o número da sua conta:\n>>> ')
  verificador_etapa = 0
  # Procura a primeira conta
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == conta_partida:
      verificador_etapa = 1
      conta_destino = input('Digite o número da conta destino:\n>>> ')
      # Verifica se é uma transferencia para a mesma conta
      if conta_destino != conta_partida:
        # Procura a segunda conta
        for conta_destino_em_consulta in contas:
          if conta_destino_em_consulta['numero_conta'] == conta_destino:
            verificador_etapa = 3
            valor_transferencia = float(input('Digite o valor da transferência:\n>>> R
            # Verifica se o valor é negativo
            if valor_transferencia < 0:
              print('\nNão é possível transferir um valor negativo.')
              break
            # Verifica se o saldo é suficiente
            elif valor_transferencia > conta_em_consulta['saldo_conta']:
              print('\nNão é possível transferir um valor maior que o seu saldo.')
              break
            else:
              conta_em_consulta['saldo_conta'] -= valor_transferencia
              conta_destino_em_consulta['saldo_conta'] += valor_transferencia
              print('\nOperação efetuada com sucesso.')
              break

        print()
      else:
        verificador_etapa = 2
        break

  if verificador_etapa == 0:
    print('\nA sua conta não foi encontrada. Tente novamente.')
  elif verificador_etapa == 1:
    print('\nA conta de destino não foi encontrada. Tente novamente.')
  elif verificador_etapa == 2:
    print('\nNão é possível transferir dinheiro para a mesma conta.')

def consultar_saldo_de_conta():
  numero_conta = input('Digite o número da conta para consultar o saldo:\n>>> ')
  for conta_em_consulta in contas:
    if conta_em_consulta['numero_conta'] == numero_conta:
      if conta_em_consulta['saldo_conta'] >= 0:
        status = 'Positivo'
      else:
        status = 'Negativo'
      print(f"\nO saldo é: R$ {conta_em_consulta['saldo_conta']:.2f} ({status})")
      return
  print('\nO número da conta não foi encontrado. Tente novamente.')

print(">>> Bem vindo ao sistema Bancário <<<")

while True:
```

```
  print("\n### Menu ###")
  print("0 - Sair")
  print("1 - Criar uma nova conta")
  print("2 - Remover uma conta")
  print("3 - Listar todas as contas")
  print("4 - Adicionar saldo (creditar)")
  print("5 - Remover saldo (debitar)")
  print("6 - Transferir valor entre contas")
  print("7 - Consultar saldo de uma conta")
  numero_operacao = input('Selecione a operação que deseja realizar:\n>>> ')

  if numero_operacao == "0":
    print('\nSistema encerrado.')
    break

  # Operação 1 - Criar uma nova conta
  elif numero_operacao == "1":
    criar_conta()

  # Operação 2 - Remover uma conta
  elif numero_operacao == "2":
    remover_conta()

  # Operação 3 - Listar todas as contas
  elif numero_operacao == "3":
    listar_contas()

  # Operação 4 - Adicionar saldo
  elif numero_operacao == "4":
    adicionar_saldo_em_conta()

  # Operação 5 - Remover saldo
  elif numero_operacao == "5":
    remover_saldo_em_conta()

  # Operação 6 - Transferir valor entre contas
  elif numero_operacao == "6":
    transferir_saldo_entre_contas()

  # Operação 7 - Consultar saldo de uma conta
  elif numero_operacao == "7":
    consultar_saldo_de_conta()

  # Operação inválida
  else:
    print('\nOperação inválida.')
```

Próximo módulo 

Muito conteúdo, né?

Caso tenha ficado alguma dúvida, aproveite para revisar a aula e não esqueça de responder aos exercícios deste módulo.

Agora, no próximo módulo vamos falar sobre como **"Gerenciar o seu código com maestria"**, isso significa que aprenderemos sobre como podemos utilizar o **Git** para organizar o nosso código e armazená-lo dentro de

plataformas com o **GitHub**.

Nessa etapa já iremos expor o que criamos até aqui para o mundo, colocando na Internet os seus primeiros códigos/projetos que servirão de portfólio desde já! Espero que esteja ansioso para aprender mais. Até a próxima!

Bibliografia

Estrutura de Dados e Algoritmos

COS-121
Estrutura de Dados e Algoritmos
2º Semestre de 2009
https://www.cos.ufrj.br/~rfarias/cos121/pilhas.html

Estrutura de Dados e Algoritmos

COS-121
Estrutura de Dados e Algoritmos
2º Semestre de 2009
https://www.cos.ufrj.br/~rfarias/cos121/filas.html