Aula 06

Pilhas

Algoritmos e Estrutura de Dados II

2° Semestre - CDN

Prof. Dr. Dilermando Piva Jr.

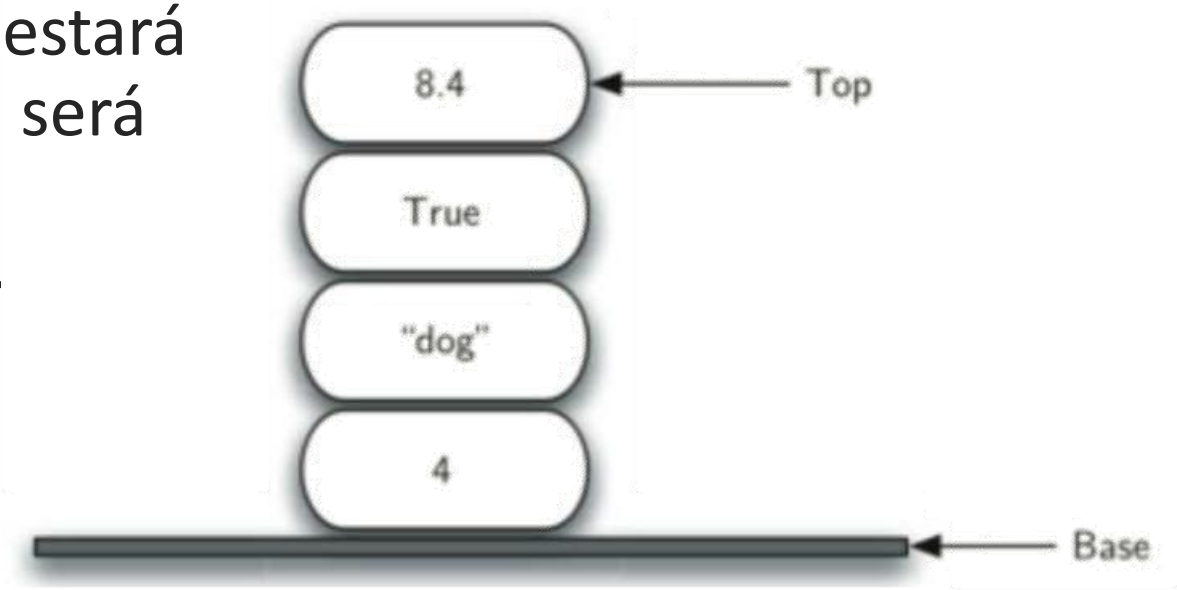
Conteúdo Programático - Planejamento

Semana	Data	Temas/Atividades
1	07/08	Acolhimento e Boas-vindas! Introdução a Disciplina. Formas de Avaliação e Percurso Pedagógico.
2	14/08	Tipo de dado abstrato. Introdução a Estrutura de Dados.
3	21/08	Complexidade de Algoritmos
4	28/08	Vetores não-Ordenados e busca sequencial
5	04/09	Vetores Ordenados e busca binária
6	11/09	Revisão de Programação Orientada a Objetos (POO)
7	18/09	Pilhas
8	25/09	Filas
9	02/10	Listas encadeadas
10	09/10	Recursão
11	*16/10*	***Primeira Avaliação Formal. (P1).*** *Correção da Avaliação após o intervalo.*
12	18/10	Algoritmos de Ordenação
13	23/10	Algoritmos de Ordenação
14	30/10	Árvores
15	06/11	Grafos
16	13/11	***Segunda Avaliação Formal (P2).*** Correção da Avaliação após o intervalo
17	27/11	***Apresentação PI do curso de CDN***
18	*04/12*	Tabela Hash (tabela de espalhamento) – Tópico extra.
19	*11/12*	***Exame / Avaliação Substitutiva.*** Correção da Avaliação após o intervalo. Finalização Disciplina
20	18/12	Finalização da disciplina.

Várias pilhas de livros...

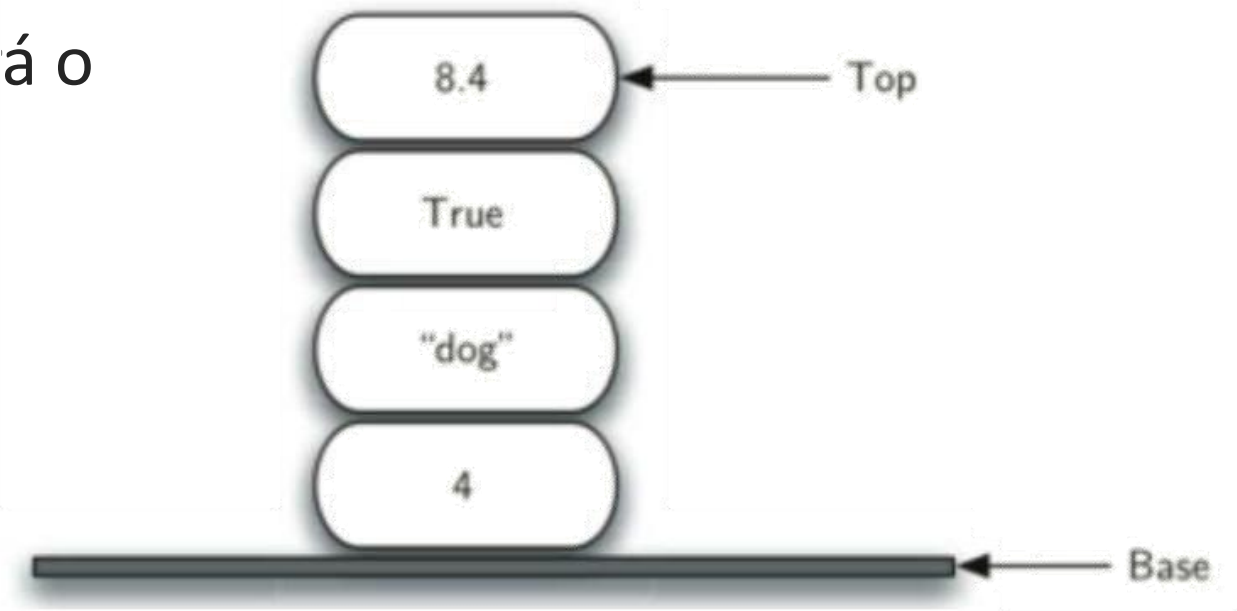
Pilhas

- Uma pilha (stack) e uma estrutura de dados linear em que a inserção e a remoção de elementos e realizada sempre na mesma extremidade, comumente de chamada de **topo**. O oposto do topo e a **base**.
- Quanto mais próximo da base está um elemento, há mais tempo ele está armazenado na estrutura.
- Por outro lado, um item inserido agora estará sempre no topo, o que significa que ele será o primeiro a ser removido
  (se não empilharmos novos elementos antes).



Pilhas

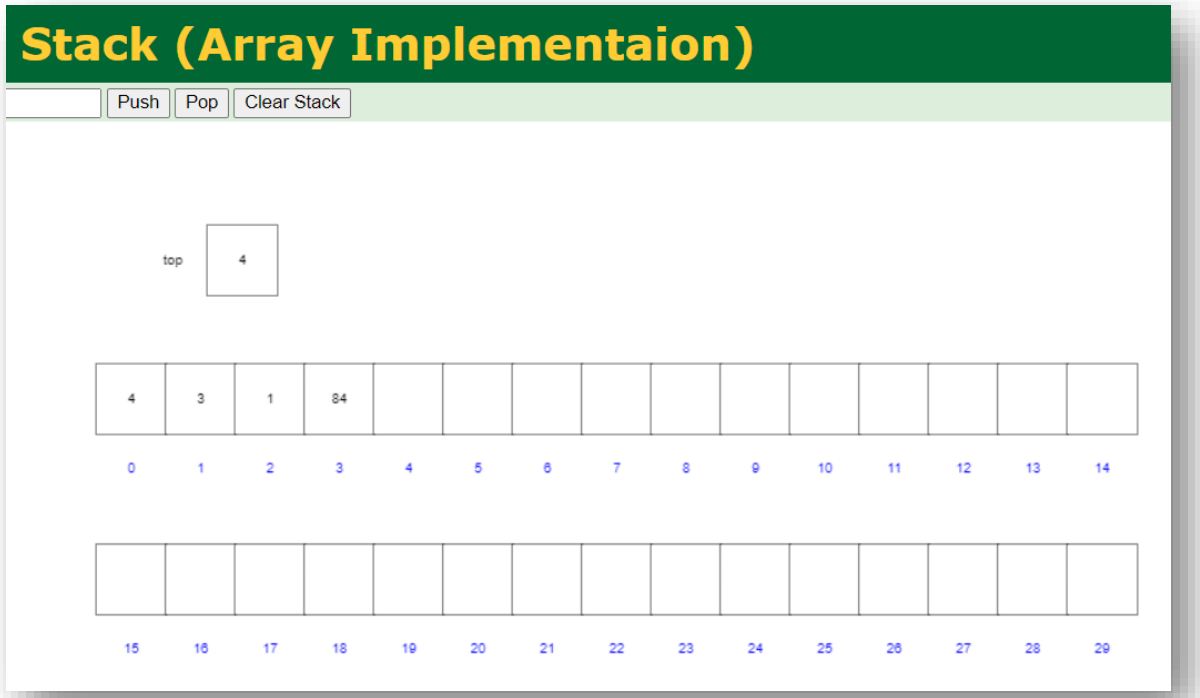
- Por esse princípio de ordenação inerente das pilhas, elas são conhecidas como a estrutura **LIFO**, do inglês, *Last In First Out*, ou seja, o último a entrar é o primeiro a sair... Também pode ser visto como: “primeiro a entrar e último a sair” (**FILO – First in Last out**).
- Essa intuição faz total sentido com uma pilha de livros (como visto no início da aula). O primeiro livro a ser empilhado fica na base da pilha e será o último a ser retirado.
- A ordem de remoção e o inverso da ordem de inserção.

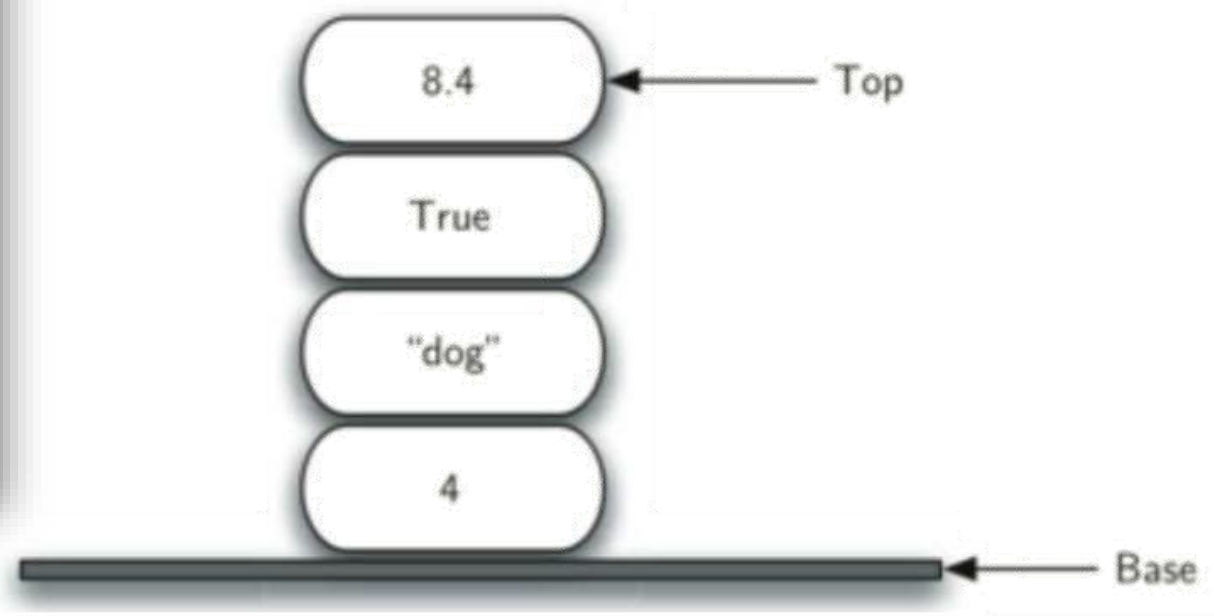


Pilhas

- Link para demonstração...

https://www.cs.usfca.edu/~galles/visualization/StackArray.html





Atividade com IA

- Para se aprofundar mais...:
  - No conceito de PILHAS...

- Faça individualmente, e depois compartilhe com o seu colega esses conceitos.

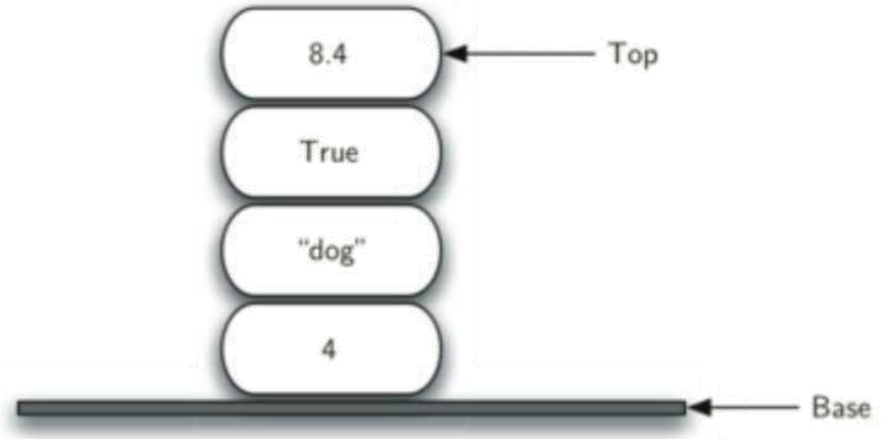
Peça para a IA:

Contexto: *Sou estudante de ciência de dados aprendendo estrutura de dados. Estou focado no conceito de* ***pilhas (stacks)*** *e quero uma explicação abrangente que inclua:* ***Definição formal:*** *O que é uma pilha? Quais são suas características principais (LIFO, operações básicas, etc.)?* ***Analogias do mundo real:*** *Exemplos concretos de como as pilhas funcionam em situações cotidianas (ex.: pilha de pratos, undo/ctrl+z, etc.).* ***Estrutura digital:*** *Como as pilhas são implementadas em programação? Explique com exemplos de código (em Python, se possível) as operações básicas (push, pop, peek, is_empty).* ***Aplicações em ciência de dados e computação:*** *Onde as pilhas são usadas em algoritmos, processamento de dados ou em bibliotecas/frameworks de data science?* ***Vantagens e desvantagens:*** *Quando usar uma pilha? Quais são as limitações?* ***Comparação com outras estruturas:*** *Como a pilha se diferencia de uma fila (queue) ou de uma lista encadeada?*

Por favor, explique de forma clara, com exemplos práticos e detalhes que ajudem a fixar o conceito.

Pilhas

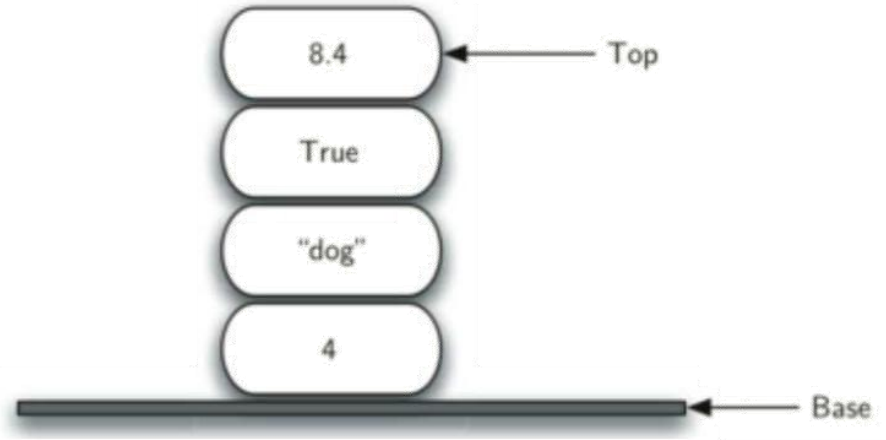
- Para que possamos implementar um classe Pilha (**Stack**), devemos ter em mente quais suas variáveis internas e quais as operações que podemos aplicar sobre os seus elementos.
- A listagem a seguir mostra como podemos construir a pilha da figura ao lado, a partir de uma pilha inicialmente vazia.
- Note que podemos utilizar uma lista **S** para armazenar os elementos da pilha.



Operação	Conteúdo	Retorno	Descrição
s.is_empty()	[]	True	Verifica se pilha está vazia
s.push(4)	[4]		Insere elemento no topo
s.push('dog')	[4, 'dog']		Insere elemento no topo
s.peek()	[4, 'dog']	'dog'	Consulta o elemento do topo (mantém)
s.push(True)	[4, 'dog', True]		Insere elemento no topo
s.size()	[4, 'dog', True]	3	Retorna número de elementos da pilha
s.is_empty()	[4, 'dog', True] False		Verifica se pilha está vazia
s.puch(8.4)	[4, 'dog', True, 8.4]		Insere elemento no topo
s.pop()	[4, 'dog', True]	8.4	Remove elemento do topo
s.pop()	[4, 'dog']	True	Remove elemento do topo
s.size()	[4, 'dog']	2	Retorna número de elementos da pilha

Pilhas

- Para que possamos implementar um classe Pilha (**Stack**), devemos ter em mente quais suas variáveis internas e quais as operações que podemos aplicar sobre os seus elementos.
- A listagem a seguir mostra como podemos construir a pilha da figura ao lado, a partir de uma pilha inicialmente vazia.
- Note que podemos utilizar uma lista **S** para armazenar os elementos da pilha.



Operação	Conteúdo	Retorno	Descrição
s.is_empty()	[]	True	Verifica se pilha está vazia
s.push(4)	[4]		Insere elemento no topo
s.push('dog')	[4, 'dog']		Insere elemento no topo
s.peek()	[4, 'dog']	'dog'	Consulta o elemento do topo (mantém)
s.push(True)	[4, 'dog', True]		Insere elemento no topo
s.size()	[4, 'dog', True]	3	Retorna número de elementos da pilha
s.is_empty()	[4, 'dog', True] False		Verifica se pilha está vazia
s.puch(8.4)	[4, 'dog', True, 8.4]		Insere elemento no topo
s.pop()	[4, 'dog', True]	8.4	Remove elemento do topo
s.pop()	[4, 'dog']	True	Remove elemento do topo
s.size()	[4, 'dog']	2	Retorna número de elementos da pilha

Pilhas

- Para implementar uma pilha em Python, iremos utilizar como atributo uma lista chamada `items`, que inicia vazia.
- Definiremos os métodos `push()` e `pop()` para inserção e remoção de elementos do topo, bem como `peek()`, `size()` e `is_empty()`, para consultar o elemento do topo, obter o número de elementos da pilha e verifica se a pilha está vazia.
- Note que na nossa implementação de pilha, tanto a inserção quanto a remoção tem complexidade O(1).

Pilhas

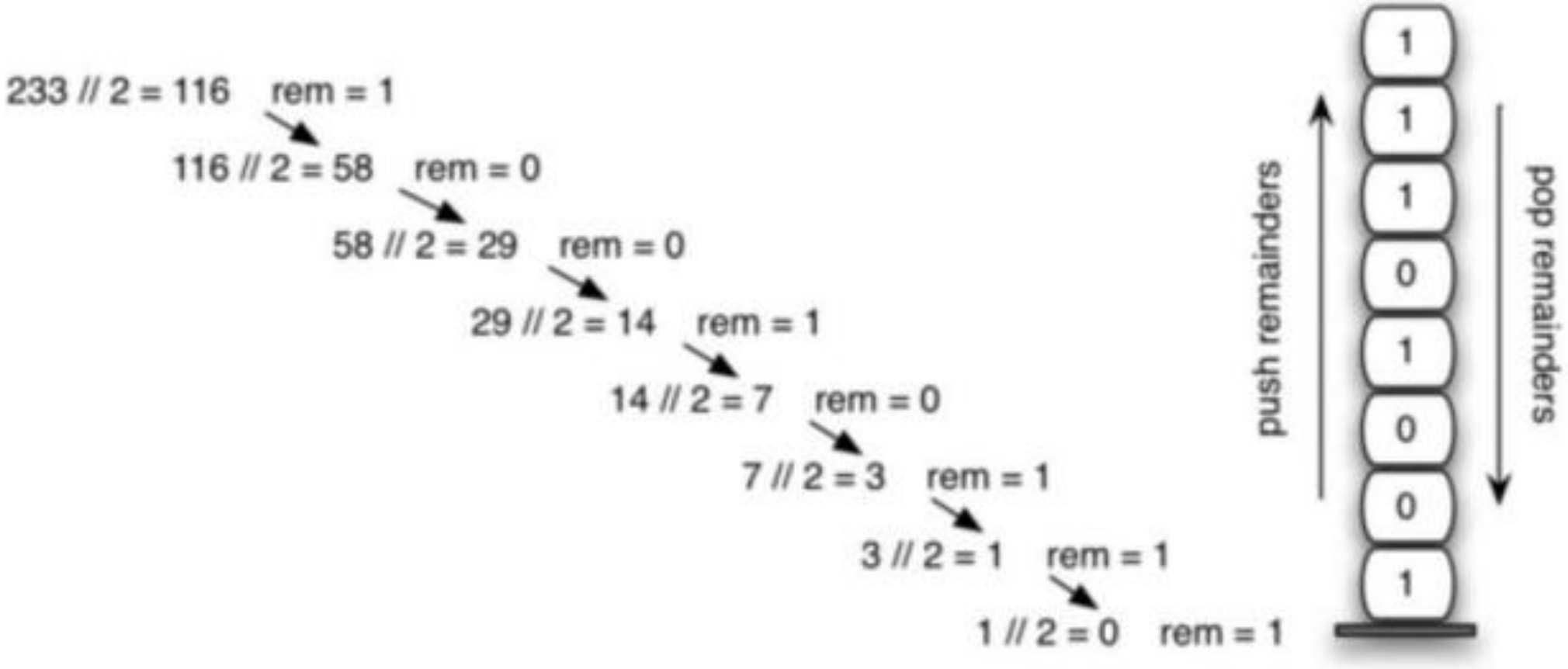
```
# Implementacao da classe Pilha
class Stack:
# Inicia com uma pilha vazia
def __init__(self):
    self.itens = []

# Verifica se pilha esta vazia
def is_empty(self):
    return self.itens == []

# Adiciona elemento no topo (topo e o final da lista)
def push(self, item):
    self.itens.append(item)
    print('PUSH %s' %item)

# Remove elemento do topo (final da lista)
def pop(self):
    print('POP')
    return self.itens.pop()

# Obtem o elemento do topo (mas nao remove)
def peek(self):
# Em Python, indice -1 retorna ultimo elemento (topo)
    return self.itens[-1]

# Retorna o numero de elementos da pilha
def size(self):
    return len(self.itens)

# Imprime pilha na tela
def print_stack(self):
    print(self.itens)
```

```
# TESTANDO
S = Stack()
S.print_stack()
S.push(1)
S.push(2)
S.push(3)
S.print_stack()
S.pop()
S.pop()
S.print_stack()
S.push(7)
S.push(8)
S.push(9)
S.print_stack()
print(S.is_empty())
```

VAMOS PARA A PRÁTICA ?!!!

Aplicações de Pilhas
Conversão de decimal para binário

Converter 233 em Binário = 11101001



Aplicações de Pilhas

Conversão de decimal para binário

```
from Pilha import Stack

# Função que converte um número decimal para binário
def decimal_binario(numero):
 s = Stack()
 while numero > 0:
    resto = numero % 2
    s.push(resto)
    numero = numero // 2
 binario = ''
 while not s.is_empty():
     binario = binario + str(s.pop())
 return binario
```

```
# TESTANDO
n = int(input('Entre com um número inteiro: '))
print(decimal_binario(n))
```

Exercícios...

1. (ENADE) Sobre uma pilha recém-criada (vazia), considere as seguintes operações em ordem: push(5), push(10), pop(), push(7), push(3), pop(). Qual será o resultado da última operação pop(), e qual a configuração final da pilha?

A. O último pop() retorna 3; pilha final = [5, 10, 7].
B. O último pop() retorna 7; pilha final = [5, 10].
C. O último pop() retorna 10; pilha final = [5, 7].
D. O último pop() retorna 7; pilha final = [5, 10, 7].
E. O último pop() retorna 3; pilha final = [5, 10].

Exercícios...

1. (ENADE) Sobre uma pilha recém-criada (vazia), considere as seguintes operações em ordem: push(5), push(10), pop(), push(7), push(3), pop(). Qual será o resultado da última operação pop(), e qual a configuração final da pilha?

A. O último pop() retorna 3; pilha final = [5, 10, 7].
B. O último pop() retorna 7; pilha final = [5, 10].
C. O último pop() retorna 10; pilha final = [5, 7].
D. O último pop() retorna 7; pilha final = [5, 10, 7].
E. O último pop() retorna 3; pilha final = [5, 10].

Exercícios...

2. (ENADE) Considere a implementação da classe **Stack** mostrada acima.
Após executar:

```
p = Stack()
p.push('a')
p.push('b')
x = p.peek()
p.pop()
y = p.peek()
```

Quais são os valores de x e y após essas operações?

A. x = 'a', y = 'b'
B. x = 'b', y = 'a'
C. x = 'b', y = 'b'
D. x = 'a', y = 'a'
E. x = None, y = None

Exercícios...

2. (ENADE) Considere a implementação da classe **Stack** mostrada acima. Após executar:

```
p = Stack()
p.push('a')
p.push('b')
x = p.peek()
p.pop()
y = p.peek()
```

Quais são os valores de x e y após essas operações?

A. x = 'a', y = 'b'
B. x = 'b', y = 'a'
C. x = 'b', y = 'b'
D. x = 'a', y = 'a'
E. x = None, y = None

Exercícios...

3. (ENADE) Suponha que temos uma pilha inicialmente vazia e executamos: push(1), push(2), push(3), pop(), push(4), pop(), pop(), pop(). Quantas vezes ocorre uma tentativa de pop() em uma pilha vazia?

A. 0 vezes
B. 1 vez
C. 2 vezes
D. 3 vezes
E. 4 vezes

Exercícios...

3. (ENADE) Suponha que temos uma pilha inicialmente vazia e executamos: push(1), push(2), push(3), pop(), push(4), pop(), pop(), pop(). Quantas vezes ocorre uma tentativa de pop() em uma pilha vazia?

A. 0 vezes
B. 1 vez
C. 2 vezes
D. 3 vezes
E. 4 vezes

Desafio 1

Validar Parênteses em uma Expressão

Objetivo: Verificar se os parênteses, colchetes e chaves em uma string estão balanceados e ordenados corretamente.

Exemplo:

- Entrada: "({[]})" → Saída: True
- Entrada: "([)]" → Saída: False

Desafio 2

Sistema Simples de Undo/Redo

Objetivo: Implementar um histórico de ações para desfazer e refazer operações em um editor de texto básico.

Exemplo:

- Adicionar "Hello" → Texto: "Hello"
- Adicionar " World" → Texto: "Hello World"
- Undo → Texto: "Hello"
- Redo → Texto: "Hello World"

Desafio 3

Simulação da Torre de Hanoi

Objetivo: Modelar as torres e discos usando pilhas e validar movimentos de acordo com as regras do jogo.

Exemplo:

- Torre A inicia com discos [3, 2, 1] (3 no fundo)
- Mover disco 1 de A para C → Válido
- Mover disco 2 de A para B → Válido
- Mover disco 2 de B para C → Inválido (disco 2 > disco 1)

Utilização de Pilhas *(na prática em Ciência de Dados)*

1.PROCESSAMENTO DE DADOS EM PIPELINES

Contexto: Em pipelines de dados, é comum aplicar transformações sequenciais (ex: limpeza, normalização, enriquecimento).

Uso de Pilhas:

- **Undo/Redo de Transformações:** Sistemas como ferramentas ETL (Extract, Transform, Load) podem usar pilhas para armazenar estados intermediários dos dados. Se uma transformação falhar, é possível "desfazer" (*undo*) revertendo para o estado anterior (pop da pilha).

1.PROCESSAMENTO DE DADOS EM PIPELINES

Exemplo Prático:

```
historico_transformacoes = []
dados = carregar_dados()

historico_transformacoes.append(dados.copy())  # Salva estado atual
dados = normalizar_coluna(dados, 'idade')

historico_transformacoes.append(dados.copy())  # Salva novo estado
dados = remover_outliers(dados, 'salario')

# Desfazer última transformação
if houve_erro:
    dados = historico_transformacoes.pop()  # Volta ao estado anterior
```

2.ANÁLISE DE DADOS COM BACKTRACKING

Contexto: Em algoritmos de otimização ou análise exploratória, pode ser necessário testar hipóteses e retroceder se não forem válidas.

Uso de Pilhas:

- **Exploração de Caminhos em Grafos ou Árvores:** Por exemplo, em algoritmos de busca em profundidade (DFS), uma pilha armazena os nós a serem visitados.

2. ANÁLISE DE DADOS COM BACKTRACKING

Exemplo Prático:

```
def dfs(grafo, no_inicial):
    visitados = set()
    pilha = [no_inicial]

    while pilha:
        no = pilha.pop()
        if no not in visitados:
            visitados.add(no)
            print(f"Visitando: {no}")  # Ex: análise de conexões em redes sociais
            for vizinho in grafo[no]:
                pilha.append(vizinho)
```

3. MACHINE LEARNING E HIPERPARÂMETROS

Contexto: Durante o ajuste de modelos, testamos combinações de hiperparâmetros e podemos retroceder se uma combinação não performar bem.

Uso de Pilhas:

- **Backtracking em Grid Search:** Armazenar combinações testadas para evitar repetições ou voltar a configurações anteriores.

3. MACHINE LEARNING E HIPERPARÂMETROS

Exemplo Prático:

```
historico_hiperparametros = []
melhor_acuracia = 0
melhores_params = None

for params in combinacoes_hiperparametros:
    historico_hiperparametros.append(params)  # Push
    modelo = treinar_modelo(params)
    acuracia = avaliar_modelo(modelo)

    if acuracia < 0.7:  # Critério de parada
        params_ruins = historico_hiperparametros.pop()
        print(f"Descartando: {params_ruins}")
    else:
        if acuracia > melhor_acuracia:
            melhor_acuracia = acuracia
            melhores_params = params
```

Mais exemplos:

- Versionamento de Conjunto de Dados (uma pilha pode armazenar versões anteriores dos dados para facilitar o rollback)
- Validação de Dados em Formatos Aninhados (verificar tags/chaves estão fechadas na ordem correta (ex:)
- Gerenciamento de Sessões em Análise de Comportamento (analisar o histórico de navegação para análise de caminhos mais comuns ou retenção)
- ...

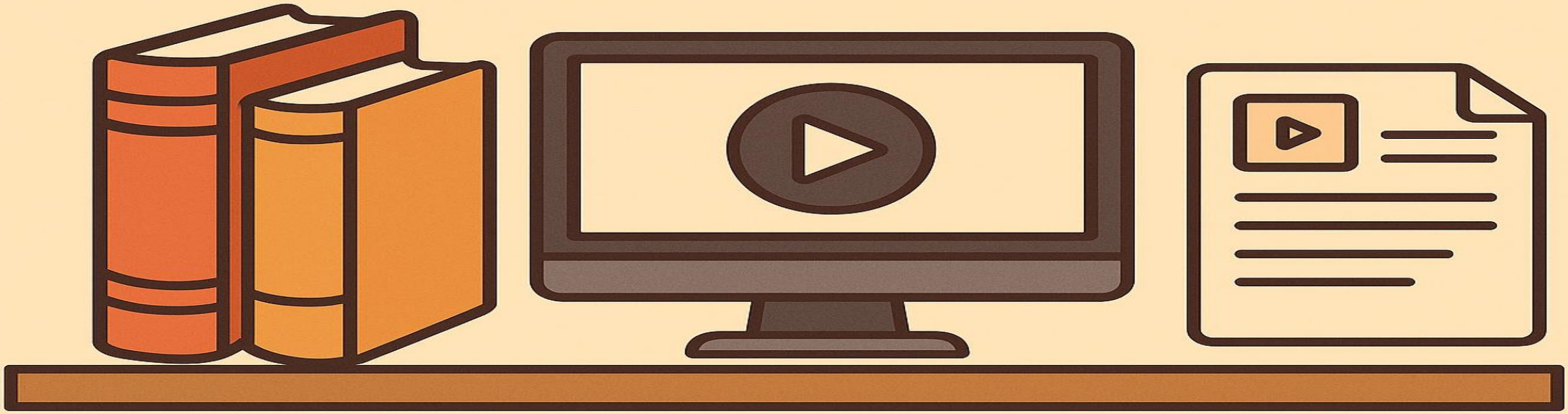
Por que Pilhas são úteis em Ciência de Dados?

1. **Eficiência:** Operações de inserção e remoção em pilhas são O(1), ideais para fluxos de dados em tempo real.

2. **Simplicidade:** Facilitam o gerenciamento de estados temporários (ex: versões de dados).

3. **Rastreabilidade:** Permitem reconstruir o histórico de operações para auditoria ou *debugging*.

Próxima Aula



- Ler o capítulo 7 do livro “Estrutura de Dados com Python”

Boa semana e bons estudos!!