Roteiro Didático – Desenvolvimento da Atividade de Pilhas e Filas

Passo 1: Formação dos grupos

Reúnam-se em duplas para estimular a colaboração e o debate durante a execução.

Passo 2: Introdução conceitual rápida

Revejam, juntos, o que são pilhas (LIFO) e filas (FIFO), suas utilizações e principais comandos:

- Pilha: **`push`**, **`pop`**, **`peek`**, **`isEmpty`**
- Fila: **`enqueue`**, **`dequeue`**, **`front`**, **`isEmpty`**

Exemplos reais: pilha de pratos, fila de impressora, histórico de navegador.

Passo 3: Implementação da Pilha

Criem uma classe Stack em Python, utilizando lista.

Incluam as operações: **`push`**, **`pop`**, **`peek`**, **`isEmpty`**.

```python
class Stack:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return len(self.items) == 0
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop() if not self.isEmpty() else None
    def peek(self):
        return self.items[-1] if not self.isEmpty() else None
# Testes:
stack = Stack()
stack.push(10)
stack.push(20)
stack.push(30)
print("Após pushes:", stack.items)
print("Pop:", stack.pop())
print("Peek:", stack.peek())
print("Está vazia?", stack.isEmpty())
```

Comentem entre si o que cada linha faz e executem os testes, verificando se o comportamento está correto.

Passo 4: Implementação da Fila

Desenvolvam uma classe Queue em Python, também usando listas.

Incluam as operações: **`enqueue`**, **`dequeue`**, **`front`**, **`isEmpty`**.

```python
class Queue:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return len(self.items) == 0
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        return self.items.pop(0) if not self.isEmpty() else None
    def front(self):
        return self.items[0] if not self.isEmpty() else None
# Testes:
queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print("Após enqueues:", queue.items)
print("Front:", queue.front())
print("Dequeue:", queue.dequeue())
print("Fila após dequeue:", queue.items)
print("Está vazia?", queue.isEmpty())
```

Testem a fila criando situações reais (fila de atendimento, chamada para impressão, etc).

Passo 5: Experimente situações de erro

Modifiquem as classes para simular overflow (para pilhas e filas com capacidade fixa) e underflow (tentativa de remover de estrutura vazia).

Incluam mensagens explicativas quando o erro ocorrer.

Discutam como essas situações acontecem em sistemas reais.

Passo 6: Desafio de aplicação

Implementem juntos a simulação de um atendimento ao cliente (exemplo abaixo):

```python
class CustomerServiceQueue:
    def __init__(self):
        self.queue = []
    def arrive(self, customer):
        self.queue.append(customer)
    def serve(self):
        return self.queue.pop(0) if self.queue else None
    def isempty(self):
        return len(self.queue) == 0
    def size(self):
        return len(self.queue)
# Testando:
csq = CustomerServiceQueue()
csq.arrive("Cliente 1")
csq.arrive("Cliente 2")
csq.arrive("Cliente 3")
print("Fila após chegada:", csq.queue)
print("Atendendo:", csq.serve())
print("Fila após atendimento:", csq.queue)
while not csq.isempty():
    print("Atendendo:", csq.serve())
```

Discutam juntos o resultado e explorem o que acontece quando não há mais clientes na fila.

Passo 7: Registro da atividade

Compilação: Reúnam o código desenvolvido, comentários e resultados dos testes em um notebook Jupyter ou arquivo texto bem organizado.

Descrevam qualquer desafio, erro encontrado, resolução discutida em grupo e aprendizado adquirido.

Passo 8: Reflexão colaborativa

Respondam em grupo:

- Onde pilhas e filas aparecem fora do laboratório?
- Qual estrutura foi mais intuitiva de implementar e por quê?
- Como erros (overflow/underflow) podem ser prevenidos?
- Como trabalharam juntos para encontrar soluções?

Passo 9: Entrega

Enviem o arquivo pelo sistema AVA conforme instruções da disciplina.