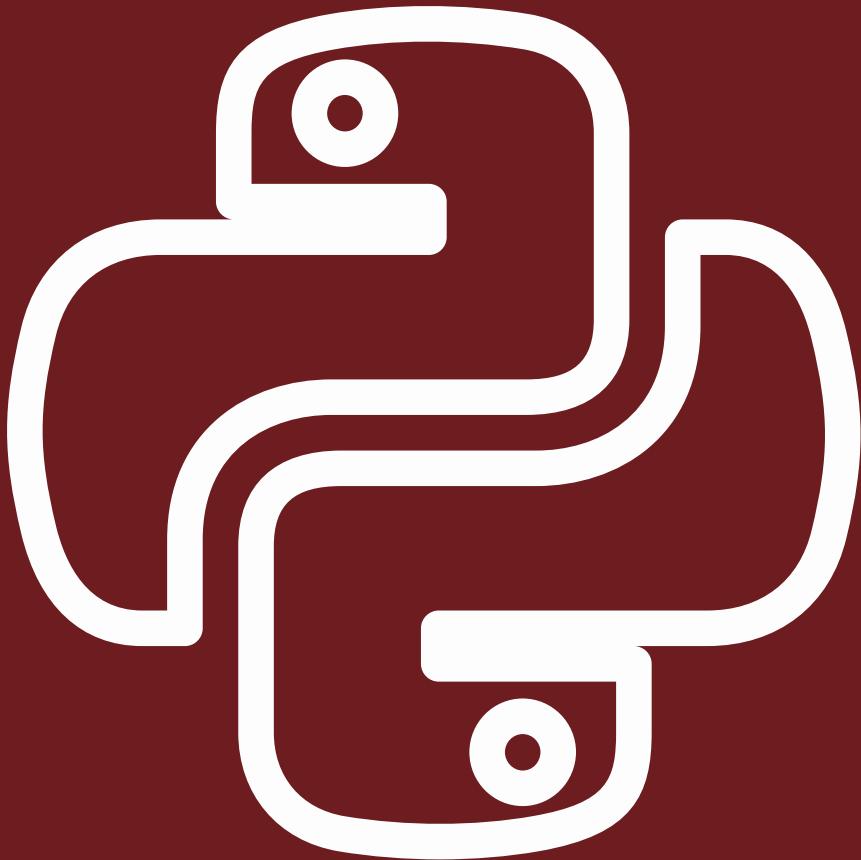


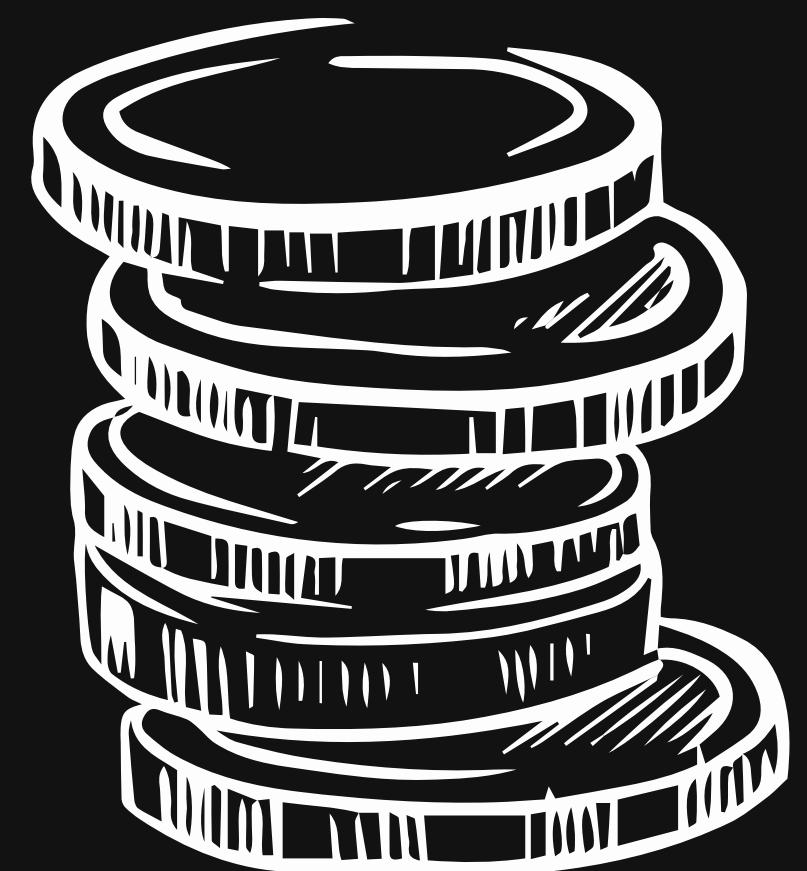
ESTRUTURA DE DADOS



UNIVERSIDADE DE
vassouras



PILHAS



SUPONDO QUE VOCÊ TEM A PILHA DE MOEDAS DO SLIDE ANTERIOR E VOCÊ IRÁ **RETIRAR** POR EXEMPLO A MOEDA DO **MEIO** DA PILHA, VOCÊ **DEVE CONCORDAR** QUE PROVAVELMENTE VOCÊ **VAI TER DIFICULDADE** EM **TIRAR** ESSA MOEDA SEM MEXER NAS MOEDAS QUE ESTÃO NA **PARTE SUPERIOR**.

O CONCEITO BÁSICO É ESSE.

OUTRO EXEMPLO

A - VOCÊ ESTÁ OCUPADO COM UM PROJETO DE LONGO PRAZO

B - É INTERROMPIDO POR UM COLEGA SOLICITANDO AJUDA EM UM OUTRO PROJETO

C - ENQUANTO ESTIVER TRABALHANDO EM B, ALGUÉM DA CONTABILIDADE APARECE PARA UMA REUNIÃO SOBRE DESPESAS DE VIAGEM

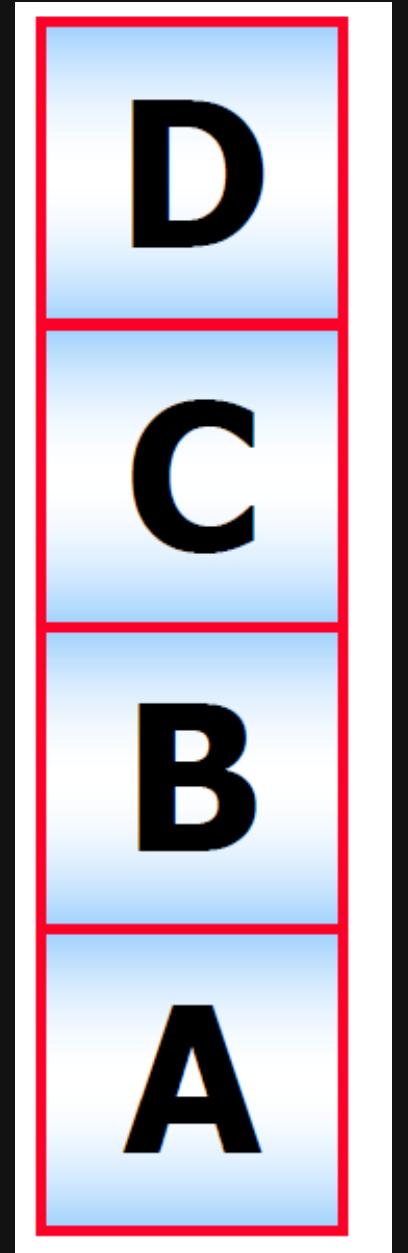
D - DURANTE A REUNIÃO, RECEBE UM TELEFONEMA DE EMERGÊNCIA DE ALGUÉM DE VENDAS E PASSA ALGUNS MINUTOS RESOLVENDO UM PROBLEMA RELACIONADO A UM NOVO PRODUTO

OUTRO EXEMPLO

QUANDO TIVER TERMINADO O TELEFONEMA **D**,
VOLTARÁ PARA A REUNIÃO **C**;

QUANDO TIVER ACABADO COM **C**, VOLTARÁ
PARA O PROJETO **B**.

E QUANDO TIVER TERMINADO COM **B**, PODERÁ
FINALMENTE VOLTAR PARA O PROJETO **A**.



PILHAS PERMITEM ACESSO A UM ITEM DE DADOS: O
ÚLTIMO ITEM INSERIDO.

SE O **ÚLTIMO** ITEM FOR REMOVIDO, O ITEM ANTERIOR AO
ÚLTIMO INSERIDO PODERÁ SER ACESSADO

APLICAÇÕES

- CORREÇÃO DE EXPRESSÕES ARITMÉTICAS, TAIS COMO $3^*(4 + 5)$;
- PERCORRIMENTO DE UMA ÁRVORE BINÁRIA;
- PESQUISA DO VÉRTICE DE UM GRAFO;
- MICROPROCESSADORES COM ARQUITETURA BASEADA EM PILHAS. QUANDO UM MÉTODO É CHAMADO, SEU ENDEREÇO DE RETORNO E SEUS PARÂMETROS SÃO EMPILHADOS EM UMA PILHA E QUANDO ELE RETORNA, SÃO DESEMPILHADOS.

OPERAÇÕES

- EMPILHAR : COLOCAR UM ITEM DE DADOS NO TOPO DA PILHA
- DESEMPILHAR: • REMOVER UM ITEM DO TOPO DA PILHA
- VER O TOPO: MOSTRA O ELEMENTO QUE ESTÁ NO TOPO DA PILHA
- ÚLTIMO-A-ENTRAR-PRIMEIRO-A-SAIR (**LIFO** - LAST-IN-FIRST-OUT)

IMPLEMENTAÇÃO

▼ Pilha

```
1 import numpy as np
```

[1]

✓ 0.2s

Python



```
1 class Pilha:
2
3     def __init__(self, capacidade): # construtor
4         self.__capacidade = capacidade # capacidade da pilha
5         self.__topo = -1 # topo da pilha
6         self.__valores = np.empty(
7             self.__capacidade, dtype=int
8         ) # array numpy vazio com capacidade igual a capacidade da pilha
9         # __ torna o atributo privado
10
11    def __pilha_cheia(self): # verifica se a pilha esta cheia
12        if (
13            self.__topo == self.__capacidade - 1
14        ): # se o topo for igual a capacidade - 1 (ultima posicao do array)
15            return True # pilha cheia
16        else:
17            return False # pilha vazia
18
19    def __pilha_vazia(self): # verifica se a pilha esta vazia
20        if self.__topo == -1: # se o topo for igual a -1
21            return True # pilha vazia
22        else:
23            return False # pilha cheia
24
```

```
24
25     def empilhar(self, valor): # empilha um valor
26         if self.__pilha_cheia(): # se a pilha estiver cheia
27             print("A pilha está cheia") # pilha cheia
28         else:
29             self.__topo += (
30                 1 # incrementa o topo da pilha por 1 (ultima posicao do array)
31             )
32             self.__valores[self.__topo] = (
33                 valor # adiciona o valor na ultima posicao do array
34             )
35
36     def desempilhar(self): # desempilha um valor
37         if self.__pilha_vazia(): # se a pilha estiver vazia
38             print("A pilha está vazia") # pilha vazia
39         else:
40             self.__topo -= (
41                 1 # decrementa o topo da pilha por 1 (ultima posicao do array)
42             )
43
44     def ver_topo(self): # verifica o valor do topo da pilha
45         if self.__topo != -1: # se o topo for diferente de -1 (ultima posicao do
array)
46             return self.__valores[self.__topo] # retorna o valor do topo da pilha
47         else:
48             return -1 # se o topo for igual a -1 (ultima posicao do array)
```

```
1 pilha = Pilha(5)
```

[3] ✓ 0.0s

```
1 pilha.ver_topo()
```

[4] ✓ 0.0s

... -1

```
1 pilha.empilhar(1)
```

```
2 pilha.ver_topo()
```

[5] ✓ 0.0s

... 1

> <

```
1 pilha.empilhar(1)
2 pilha.empilhar(2)
3 pilha.empilhar(3)
4 pilha.empilhar(4)
```

[14]

✓ 0.0s

```
1 pilha.empilhar(6)
```

[7]

✓ 0.0s

... A pilha está cheia

> <

```
1 pilha.ver_topo()
```

[8]

✓ 0.0s

...

4

```
1 pilha.desempilhar()
```

[9] ✓ 0.0s

```
1 pilha.ver_topo()
```

[10] ✓ 0.0s

... 3

▶ ▾

```
1 pilha.desempilhar()
```

```
2 pilha.desempilhar()
```

```
3 pilha.desempilhar()
```

[11] ✓ 0.0s

```
1 pilha.ver_topo()
```

[12] ✓ 0.0s

...
1

```
1 pilha.desempilhar()
```

[13] ✓ 0.0s

VALIDADOR DE EXPRESSÕES

- OS DELIMITADORES SÃO AS CHAVES { E }, OS COLCHETES [E] E OS PARÉNTESES (E)
- CADA DELIMITADOR DE ABERTURA OU À ESQUERDA DEVE SER CASADO COM UM DELIMITADOR DE FECHAMENTO OU À DIREITA
- TODA { DEVE SER SEGUIDA POR UMA } QUE CASE COM ELA

EXEMPLOS

- C[D] (CORRETO)
- A{B[C]D}E (CORRETO)
- A{B(C)D}E (INCORRETO -] NÃO CASA COM (
- A[B{C}D]E} (INCORRETO - NADA CASA COM } NO FINAL
- A{B(C} (INCORRETO - NADA CASA COM { DE ABERTURA

IMPLEMENTAÇÃO DO NOSSO VALIDADOR

Pilha

```
[1] 1 import numpy as np
```

✓ 0.1s

Python

```
1 class Pilha:
2     def __init__(self, capacidade):
3         self.capacidade = capacidade
4         self.topo = -1
5         # Array de chars
6         self.valores = np.chararray(self.capacidade, unicode = True)
7
8     def __pilha_cheia(self):
9         if self.topo == self.capacidade - 1:
10             return True
11         else:
12             return False
13
14     # Mudança para método público
15     def pilha_vazia(self):
16         if self.topo == -1:
17             return True
18         else:
19             return False
20
```

```
20
21     def empilhar(self, valor):
22         if self._pilha_cheia():
23             print('A pilha está cheia')
24         else:
25             self.topo += 1
26             self.valores[self.topo] = valor
27
28     # Retorno do valor desempilhado
29     def desempilhar(self):
30         if self.pilha_vazia():
31             print('A pilha está vazia')
32             return -1
33         else:
34             valor = self.valores[self.topo]
35             self.topo -= 1
36             return valor
37
38     def ver_topo(self):
39         if self.topo != -1:
40             return self.valores[self.topo]
41         else:
42             return -1
```

```
1 # Dados a serem validados, insira-os como no input a seguir. Nossa função mais  
    abaixo irá somente dar um return para as expressões que não atenderem a validação.  
2 # c[d]  
3 # a{b[c]d}e  
4 # a{b(c)d}e  
5 # a[b{c}d]e}  
6 # a{b(c)  
7  
8 # funcionarão os 2 primeiros casos, e os 3 últimos não.
```

Python

```
1 expressao = str(input('Digite uma expressão: '))  
2 pilha = Pilha(len(expressao))
```

✓ 1.5s

Python

∨≡ ∨↑ ∨↓ □ ⋯

```
1 for i in range(len(expressao)):
2     ch = expressao[i]
3     if ch == '{' or ch == '[' or ch == '(':
4         pilha.empilhar(ch)
5     elif ch == '}' or ch == ']' or ch == ')':
6         if not pilha.pilha_vazia():
7             chx = str(pilha.desempilhar())
8             if (ch == '}' and chx != '{') or (ch == ']' and chx != '[') or (ch == ')' and
9                 chx != '('):
10                print('Erro: ', ch, ' na posição ', i)
11                break
12            else:
13                print('Erro: ', ch, ' na posição ', i)
14 if not pilha.pilha_vazia():
15     print('Erro!')
```

✓ 0.0s

Python

A{B(C]D}E

```
▶ 
1 for i in range(len(expressao)):
2     ch = expressao[i]
3     if ch == '{' or ch == '[' or ch == '(':
4         pilha.empilhar(ch)
5     elif ch == '}' or ch == ']' or ch == ')':
6         if not pilha.pilha_vazia():
7             chx = str(pilha.desempilhar())
8             if (ch == '}' and chx != '{') or (ch == ']' and chx != '[') or (ch == ')' and
9                 chx != '('):
10                print('Erro: ', ch, ' na posição ', i)
11                break
12            else:
13                print('Erro: ', ch, ' na posição ', i)
14 if not pilha.pilha_vazia():
15     print('Erro!')
```

[10]

✓ 0.0s

Python

... Erro:] na posição 5
Erro!

A[B{C}D]E}

```
▷ 
1 for i in range(len(expressao)):
2     ch = expressao[i]
3     if ch == '{' or ch == '[' or ch == '(':
4         pilha.empilhar(ch)
5     elif ch == '}' or ch == ']' or ch == ')':
6         if not pilha.pilha_vazia():
7             chx = str(pilha.desempilhar())
8             if (ch == '}' and chx != '{') or (ch == ']' and chx != '[') or (ch == ')' and
9                 chx != '('):
10                print('Erro: ', ch, ' na posição ', i)
11                break
12            else:
13                print('Erro: ', ch, ' na posição ', i)
14 if not pilha.pilha_vazia():
15     print('Erro!')
```

[12] ✓ 0.0s Python

... Erro: } na posição 9

A{B(C)

```
1 for i in range(len(expressao)):
2     ch = expressao[i]
3     if ch == '{' or ch == '[' or ch == '(':
4         pilha.empilhar(ch)
5     elif ch == '}' or ch == ']' or ch == ')':
6         if not pilha.pilha_vazia():
7             chx = str(pilha.desempilhar())
8             if (ch == '}' and chx != '{') or (ch == ']' and chx != '[') or (ch == ')' and
9                 chx != '('):
10                print('Erro: ', ch, ' na posição ', i)
11                break
12            else:
13                print('Erro: ', ch, ' na posição ', i)
14 if not pilha.pilha_vazia():
15     print('Erro!')
```

[14] ✓ 0.0s Python

... Erro!

EXERCÍCIO! IMPLEMENTAÇÃO BÁSICA DE PILHA

IMPLEMENTE UMA CLASSE DE PILHA EM PYTHON QUE SUPORTE AS OPERAÇÕES BÁSICAS DE UMA PILHA: **PUSH**, **POP**, E **PEEK**. ALÉM DISSO, ADICIONE UM MÉTODO PARA **VERIFICAR** SE A PILHA ÉSTA **VAZIA**.

REQUISITOS:

- **PUSH:** ESTE MÉTODO DEVE ACEITAR UM VALOR E COLOCÁ-LO NO TOPO DA PILHA.
- **POP:** ESTE MÉTODO DEVE REMOVER O VALOR DO TOPO DA PILHA E RETORNÁ-LO. SE A PILHA ESTIVER VAZIA, RETORNE UMA MENSAGEM DE ERRO OU EXCEÇÃO.
- **PEEK:** ESTE MÉTODO DEVE RETORNAR O VALOR DO TOPO DA PILHA SEM REMOVÉ-LO. SE A PILHA ESTIVER VAZIA, RETORNE UMA MENSAGEM DE ERRO OU EXCEÇÃO.
- **ISEMPTY:** ESTE MÉTODO DEVE RETORNAR TRUE SE A PILHA ESTIVER VAZIA, CASO CONTRÁRIO FALSE.

EXERCÍCIO! IMPLEMENTAÇÃO BÁSICA DE PILHA

EXEMPLO DE SAÍDA

```
Pilha criada. Estado atual: []
Empilhando: 5
Estado atual da pilha: [5]
Empilhando: 10
Estado atual da pilha: [5, 10]
Elemento do topo: 10
Desempilhando: 10
Estado atual da pilha: [5]
A pilha está vazia? Não
```

SOLUÇÃO

```
1 class Pilha:
2     def __init__(self):
3         # Inicializa a pilha como uma lista vazia.
4         self.itens = []
5
6     def push(self, item):
7         # Adiciona um item ao topo da pilha.
8         # Em uma lista, isso pode ser feito usando o método append.
9         self.itens.append(item)
10        print(f"Empilhando: {item}")
11        print(f"Estado atual da pilha: {self.itens}")
12
13    def pop(self):
14        # Remove o item do topo da pilha e o retorna.
15        # Primeiro, verificamos se a pilha está vazia para evitar erros.
16        if self.isEmpty():
17            print("Erro: tentativa de desempilhar uma pilha vazia")
18            return None
19        else:
20            item = self.itens.pop() # Remove e retorna o último item da lista
21            print(f"Desempilhando: {item}")
22            print(f"Estado atual da pilha: {self.itens}")
23            return item
24
```

SOLUÇÃO

```
24
25     def peek(self):
26         # Retorna o item no topo da pilha sem removê-lo.
27         # Novamente, verificamos primeiro se a pilha está vazia.
28         if self.isEmpty():
29             print("Erro: pilha vazia")
30             return None
31         else:
32             return self.itens[-1] # Retorna o último item da lista sem remover
33
34     def isEmpty(self):
35         # Retorna True se a pilha estiver vazia, caso contrário, retorna False.
36         return len(self.itens) == 0
```

✓ 0.0s

SOLUÇÃO

```
1 # Criando e usando a pilha  
2 pilha = Pilha() # Cria uma pilha vazia
```

[2] ✓ 0.0s

```
1 pilha.push(5) # Empilha 5  
2 pilha.push(10) # Empilha 10
```

[3] ✓ 0.0s

```
... Empilhando: 5  
Estado atual da pilha: [5]  
Empilhando: 10  
Estado atual da pilha: [5, 10]
```

SOLUÇÃO

```
1 print(f"Elemento do topo: {pilha.peek()}") # Mostra o elemento do topo  
[4] ✓ 0.0s
```

```
... Elemento do topo: 10
```

```
1 pilha.pop() # Remove o elemento do topo  
[5] ✓ 0.0s
```

```
... Desempilhando: 10  
Estado atual da pilha: [5]
```

```
... 10
```

```
▷ ▾  
1 print(  
2 |   f"A pilha está vazia? {'Sim' if pilha.isEmpty() else 'Não'}"  
3 ) # Verifica se a pilha está vazia
```

```
[6] ✓ 0.0s
```

```
... A pilha está vazia? Não
```

That's all Folks!

ATÉ A PRÓXIMA!