

# CODER's GUIDE TO THE GALAXY\_v0.42

Autor: [Leonardo Costa Passos](#)

1.	Problemas Clássicos de Programação:	6
1.1.	Inverter uma String.....	6
1.1.1.	Python .....	6
1.1.2.	Node.js .....	7
1.1.3.	Java.....	10
1.2.	Busca Linear:.....	11
1.2.1.	Python .....	11
1.2.2.	Node.js .....	12
1.2.3.	Java.....	13
1.3.	Busca Binária (Lista Ordenada):.....	15
1.3.1.	Python .....	15
1.3.2.	Node.js .....	16
1.3.3.	Java.....	18
1.4.	Busca Binária (Lista Desordenada):.....	20
1.4.1.	Python .....	20
1.4.2.	Node.js .....	22
1.5.	FizzBuzz:.....	24
1.5.1.	Python .....	24
1.5.2.	Node.js .....	26
1.5.3.	Java.....	29
1.6.	Fibonacci (iterativo):.....	32
1.6.1.	Python .....	32
1.6.2.	Node.js .....	33
1.6.3.	Java.....	34
1.7.	Fibonacci (Recursivo):.....	35
1.7.1.	Python .....	35
1.7.2.	Node.js .....	35
1.7.3.	Java.....	36
1.8.	Fatorial:.....	37
1.8.1.	Python .....	37
1.8.2.	Node.js .....	38
1.8.3.	Java.....	38
1.9.	Números Primos:.....	39
1.9.1.	Python .....	39

1.9.2.	Node.js .....	40
1.9.3.	Java.....	41
1.10.	Palíndromo:.....	42
1.10.1.	Python .....	42
1.10.2.	Node.js .....	43
1.10.3.	Java.....	44
1.11.	Maior Divisor Comum (MDC): .....	45
1.11.1.	Python .....	45
1.11.2.	Node.js .....	45
1.11.3.	Java.....	46
1.12.	Mínimo Múltiplo Comum (MMC):.....	47
1.12.1.	Python .....	47
1.12.2.	Node.js .....	48
1.12.3.	Java.....	48
1.13.	Randomização:.....	49
1.13.1.	Python .....	49
1.13.2.	Node.js .....	50
1.13.3.	Java.....	51
1.14.	Bubble Sort.....	52
1.14.1.	Python .....	52
1.14.2.	Node.js .....	53
1.14.3.	Java.....	54
1.15.	Insertion Sort:.....	55
1.15.1.	Python .....	55
1.15.2.	Node.js .....	56
1.15.3.	Java.....	57
1.16.	Quick Sort:.....	58
1.16.1.	Python .....	58
1.16.2.	Node.js .....	59
1.17.	Mochila .....	60
1.17.1.	Python .....	60
1.17.2.	Node.js .....	61
1.17.3.	Java.....	62
1.18.	Torre de Hanói:.....	64
1.18.1.	Python .....	64
1.18.2.	Node.js .....	65
1.18.3.	Java.....	66
1.19.	Fila: .....	67
1.19.1.	Python .....	67

1.19.2. Node.js .....	69
1.19.3. Java.....	71
1.20. Pilha:.....	72
1.20.1. Python .....	72
1.20.2. Node.js .....	73
1.20.3. Java.....	74
2. Desenvolvimento de Programa com Especificação.....	76
2.1. Biblioteca .....	76
2.1.1. Java.....	76
2.1.2. Node.js .....	78
2.1.3. Python .....	79
2.2. Cinema.....	80
2.2.1. Java.....	80
2.2.2. Node.js .....	82
2.2.3. Python .....	84
2.3. Agenda.....	86
2.3.1. Java.....	86
2.3.2. Node.js .....	88
2.3.3. Python .....	90
2.4. Gerenciador de Tarefas.....	91
2.4.1. Java.....	91
2.4.2. Node.js .....	92
2.4.3. Python .....	93
2.5. Vendas Online.....	94
2.5.1. Java.....	94
2.5.2. Node.js .....	96
2.5.3. Python .....	97
3. Apresentação de Dados Calculados .....	98
3.1. Estatísticas Alunos .....	98
3.1.1. Java.....	98
3.1.2. Node.js .....	100
3.1.3. Python .....	101
3.2. GPS .....	102
3.2.1. Java.....	102
3.2.2. Node.js .....	103
3.2.3. Python .....	104
3.3. Meteorologia .....	105
3.3.1. Java.....	105
3.3.2. Node.js .....	107

3.3.3. Python .....	109
3.4. Estatísticas Futebol .....	110
3.4.1. Java.....	110
3.4.2. Node.js .....	112
3.4.3. Python .....	113
3.5. Estatísticas Tráfego .....	115
3.5.1. Java.....	115
3.5.2. Node.js .....	117
3.5.3. Python .....	118
4. Execução de Rotinas a partir de uma Especificação .....	119
4.1. Câmbio Moedas .....	119
4.1.1. Java.....	119
4.1.2. Node.js .....	121
4.1.3. Python .....	123
4.2. Classificação E-Mails:.....	124
4.2.1. Java.....	124
4.2.2. Node.js .....	126
4.2.3. Python .....	127
4.3. Análise de Logs: .....	128
4.3.1. Java.....	128
4.3.2. Node.js .....	129
4.3.3. Python .....	131
4.4. Rotas Delivery .....	132
4.4.1. Java.....	132
4.4.2. Node.js .....	134
4.4.3. Python .....	135
4.5. Rotina Backup.....	136
4.5.1. Node.js .....	136
5. Encontrar e Solucionar Erros.....	138
5.1. Alunos: erros .....	138
5.1.1. Java.....	138
5.1.2. Node.js .....	141
5.1.3. Python .....	143
5.2. Calculadora: erros.....	144
5.2.1. Enunciado (Parte A): .....	144
5.3. Soma: erros.....	148
5.3.1. Java.....	148
5.3.2. Node.js .....	149
5.3.3. Python .....	150

5.4.	Sistema Vendas: erros .....	152
5.4.1.	Java.....	152
5.4.2.	Node.js .....	154
5.4.3.	Python .....	154
5.5.	Public static void main(String args): erros .....	155
5.5.1.	Java.....	156
5.5.2.	Node.js .....	157
5.5.3.	Python .....	158
6.	Refatoração e Teste Unitário:.....	160
6.1.	Calculadora: .....	160
6.1.1.	Calculadora (REFATORAR): .....	160
6.1.2.	Calculadora (TESTE UNITÁRIO):.....	167
6.2.	Gerenciador Impressora .....	168
6.2.1.	Impressora (REFATORAR):.....	168
6.2.2.	Impressora (TESTE UNITÁRIO): .....	176
6.3.	Reserva Hotel .....	177
6.3.1.	Hotel (REFATORAÇÃO):.....	177
6.3.2.	Hotel (TESTE UNITÁRIO):.....	183
6.4.	Árvore Binária .....	185
6.4.1.	Árvore Binária (REFATORAÇÃO):.....	185
6.4.2.	Árvore Binária (TESTE UNITÁRIO): .....	195
6.5.	Stock Management.....	196
6.5.1.	Stocks (REFATORAÇÃO):.....	196
6.5.2.	Stocks (TESTE UNITÁRIO): .....	203

# 1. Problemas Clássicos de Programação:

## 1.1. Inverter uma String

### 1.1.1. Python

```
Coliseum > HAL2001 > inverte.py > ...
1  def inverter_string(s):
2      return s[::-1]
3
4  string_original = "Beeblebrox"
5  string_invertida = inverter_string(string_original)
6  print(f"String original: {string_original}")
7  print(f"String invertida: {string_invertida}")
```

```
String original: Beeblebrox
String invertida: xorbelbeeB
```

#### 1.1.1.1. Slicing

```
python
subsequencia = sequencia[inicio:fim:passo]
```

- **`inicio`**: É o índice inicial da subsecção. Se omitido, assume o valor 0.
- **`fim`**: É o índice que marca o fim da subsecção. Este índice não é incluído na subsequência resultante. Se omitido, assume o valor do tamanho da sequência.
- **`passo`**: É a diferença entre os índices na subsecção. Se omitido, assume o valor 1.

```
python
s = "Hello, World!"
substring = s[:3] # Resultado: "Hel"
```

Para obter os caracteres de índice 3 a 7:

```
python
substring = s[3:8] # Resultado: "lo, W"
```

```
python
Copy code

s = "Hello, World!"
string_invertida = s[::-1] # Resultado: "!dlroW ,olleH"
```

Neste exemplo, omitimos os valores de `inicio` e `fim`, então a operação de slicing considera a string completa. O valor de `passo` é `-1`, o que significa que a subseção será composta pelos elementos da string original em ordem inversa. Isso resulta em uma string invertida.

### 1.1.2. Node.js

```
Coliseum > HAL2001 > JS inverte.js > ...
1  function inverterString(s) {
2    return s.split('').reverse().join('');
3  }
4
5  const stringOriginal = "Infinite Improbability Drive";
6  const stringInvertida = inverterString(stringOriginal);
7  console.log(`String original: ${stringOriginal}`);
8  console.log(`String invertida: ${stringInvertida}`);

String original: Infinite Improbability Drive
String invertida: evirD ytilibaborpmI etinifnI
```

#### 1.1.1.1. Split

```
javascript
Copy code

str.split([separator[, limit]])
```

- `separator`: É uma string ou uma expressão regular que define o ponto onde cada divisão deve ocorrer.  
- `limit`: Um inteiro que especifica o número máximo de divisões que devem ser feitas.

- Dividir uma string por espaços:

```
javascript Copy code  
  
var texto = "Hello World!";  
var result = texto.split(" "); // Resultado: ["Hello", "World!"]
```

- Dividir uma string por cada caractere:

```
javascript Copy code  
  
var texto = "Hello";  
var result = texto.split(""); // Resultado: ["H", "e", "l", "l", "o"]
```

### 1.1.2.1. Join

```
javascript Copy code  
  
array.join([separator])
```

- `separator` : (Opcional) Especifica uma string para separar cada par de elementos adjacentes na array resultante. O separador é convertido em uma string, se necessário. Se omitido, os elementos da array são separados por uma vírgula (`, `).

#### 1. Utilizando o Separador Padrão (Vírgula)

```
javascript Copy code  
  
var frutas = ["Maçã", "Banana", "Cereja"];  
var resultado = frutas.join(); // Resultado: "Maçã,Banana,Cereja"
```

#### 2. Utilizando um Espaço como Separador

```
javascript Copy code  
  
var frutas = ["Maçã", "Banana", "Cereja"];  
var resultado = frutas.join(" "); // Resultado: "Maçã Banana Cereja"
```

#### 3. Utilizando um Hífen como Separador

```
javascript Copy code  
  
var frutas = ["Maçã", "Banana", "Cereja"];  
var resultado = frutas.join("-"); // Resultado: "Maçã-Banana-Cereja"
```

### 1.1.2.2. Reverse

**Método `reverse`**

O método `reverse` é usado para inverter os elementos de uma array. Ele modifica a array original.

**Exemplo Combinando `split` e `reverse`**

Se você deseja inverter uma string, pode combinar os métodos `split`, `reverse`, e `join`:

```
javascript
var texto = "Hello World!";
var result = texto.split("").reverse().join(""); // Resultado: "!dlroW olleH"
```

1. `split("")` - divide a string em uma array de caracteres.  
2. `reverse()` - inverte a ordem dos elementos na array.  
3. `join("")` - junta os elementos da array invertida em uma string.

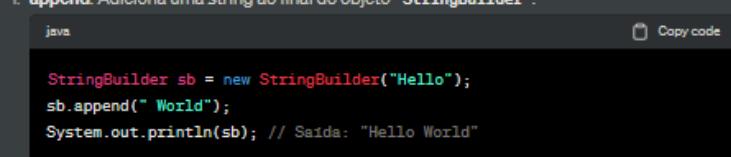
Assim, a combinação desses métodos permite inverter uma string em JavaScript de maneira bastante direta e elegante.

### 1.1.3. Java

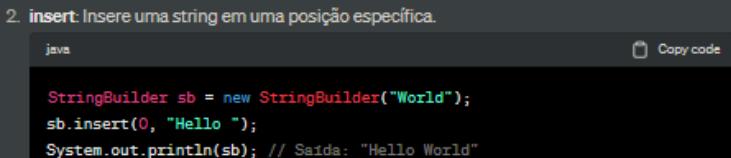
```
Coliseum > HAL2001 > J InverterString.java > ...
1 package Coliseum.HAL2001;
2
3 public class InverterString {
4     Run | Debug
5     public static void main(String[] args) {
6         String stringOriginal = "Dont Panic!";
7         String stringInvertida = inverterString(stringOriginal);
8         System.out.println("String original: " + stringOriginal);
9         System.out.println("String invertida: " + stringInvertida);
10    }
11
12    public static String inverterString(String s) {
13        StringBuilder stringBuilder = new StringBuilder(s);
14        return stringBuilder.reverse().toString();
15    }
16}
```

```
String original: Dont Panic!
String invertida: !cinaP tnoD
```

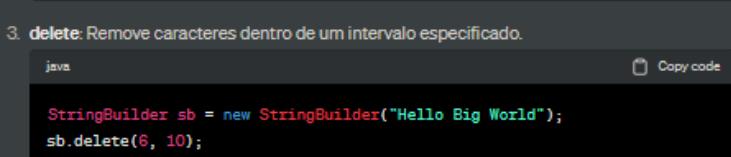
#### 1.1.3.1. StringBuilder

1. **append:** Adiciona uma string ao final do objeto `StringBuilder`.  


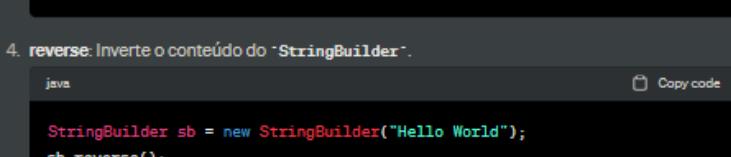
```
java
Copy code

StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
System.out.println(sb); // Saída: "Hello World"
```
2. **insert:** Insere uma string em uma posição específica.  


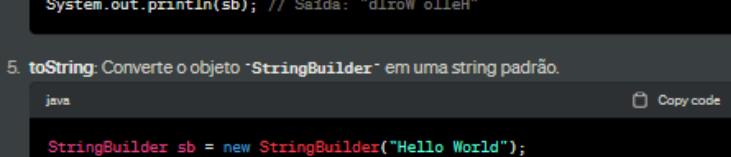
```
java
Copy code

StringBuilder sb = new StringBuilder("World");
sb.insert(0, "Hello ");
System.out.println(sb); // Saída: "Hello World"
```
3. **delete:** Remove caracteres dentro de um intervalo especificado.  


```
java
Copy code

StringBuilder sb = new StringBuilder("Hello Big World");
sb.delete(6, 10);
System.out.println(sb); // Saída: "Hello World"
```
4. **reverse:** Inverte o conteúdo do `StringBuilder`.  


```
java
Copy code

StringBuilder sb = new StringBuilder("Hello World");
sb.reverse();
System.out.println(sb); // Saída: "dlroW olleH"
```
5. **toString:** Converte o objeto `StringBuilder` em uma string padrão.  


```
java
Copy code

StringBuilder sb = new StringBuilder("Hello World");
String str = sb.toString();
```

## 1.2. Busca Linear:

Procurar sequencialmente um elemento em um array.

### 1.2.1. Python

```
Coliseum > HAL2001 > ⚡ busca_linear.py > ...
1  def busca_linear(lista, elemento_procurado):
2      for indice, elemento in enumerate(lista):
3          if elemento == elemento_procurado:
4              return indice
5      return -1
6
7  lista = [34, 56, 78, 12, 44, 89, 23, 42, 13]
8  elemento_procurado = 42
9
10 indice_encontrado = busca_linear(lista, elemento_procurado)
11
12 if indice_encontrado != -1:
13     print(f"Elemento {elemento_procurado} encontrado na posicao {indice_encontrado}.")
14 else:
15     print(f"Elemento {elemento_procurado} nao encontrado na lista.")
16
```

Elemento 42 encontrado na posicao 7.

#### 1.2.1.1. Enumerate

Suponha que você queira percorrer uma lista de frutas e imprimir cada fruta junto com seu índice. Sem `enumerate`, você pode fazer algo assim:

```
python
frutas = ['maçã', 'banana', 'cereja']
for i in range(len(frutas)):
    print(i, frutas[i])
```

Mas com `enumerate`, o código fica mais limpo e Pythonico:

```
python
frutas = ['maçã', 'banana', 'cereja']
for i, fruta in enumerate(frutas):
    print(i, fruta)
```

## 1.2.2. Node.js

```
Coliseum > HAL2001 > js busca_linear.js > ...
1  function buscaLinear(lista, elementoProcurado) {
2    for (let indice = 0; indice < lista.length; indice++) {
3      if (lista[indice] === elementoProcurado) {
4        return indice;
5      }
6    }
7    return -1;
8  }
9
10 const lista = [34, 56, 78, 12, 44, 89, 23, 42, 13];
11 const elementoProcurado = 42;
12
13 const indiceEncontrado = buscaLinear(lista, elementoProcurado);
14
15 if (indiceEncontrado !== -1) {
16   console.log(`Elemento ${elementoProcurado} encontrado na posição ${indiceEncontrado}.`);
17 } else {
18   console.log(`Elemento ${elementoProcurado} não encontrado na lista.`);
19 }
20
```

### 1.2.2.1. Operador “==”

```
javascript Copy code
5 === 5; // true, mesmo valor e mesmo tipo
5 === '5'; // false, mesmo valor, mas tipos diferentes

5 == 5; // true, mesmo valor e mesmo tipo
5 == '5'; // true, o operador == converte o '5' em número antes da comparação
```

#### Vantagens de Usar “==”

O uso de “==” é geralmente preferível ao “=” em muitos casos, pois ele evita resultados inesperados devido à coerção de tipo. Essa coerção automática pode levar a bugs sutis e difíceis de detectar em programas complexos.

Ao usar “==” você tem certeza de que está comparando tanto o tipo quanto o valor, o que tende a tornar o código mais previsível e compreensível.

### 1.2.3. Java

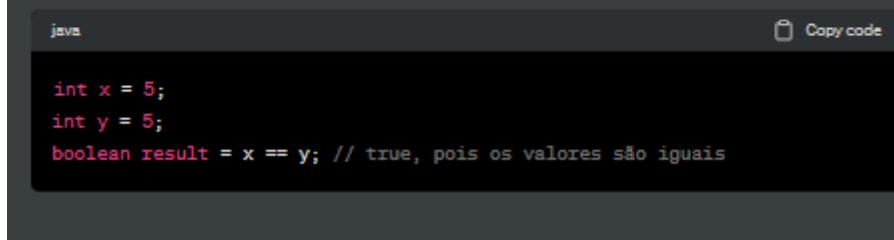
```
Coliseum > HAL2001 > J BuscaLinear.java > ...
1 package Coliseum.HAL2001;
2
3 public class BuscaLinear {
4     public static int buscaLinear(int[] lista, int elementoProcurado) {
5         for (int indice = 0; indice < lista.length; indice++) {
6             if (lista[indice] == elementoProcurado) {
7                 return indice; // Retorna o índice do elemento encontrado
8             }
9         }
10        return -1; // Retorna -1 se o elemento não for encontrado
11    }
12
13    Run | Debug
14    public static void main(String[] args) {
15        int[] lista = { 34, 56, 78, 12, 44, 89, 23, 42, 13 };
16        int elementoProcurado = 42;
17
18        int indiceEncontrado = buscaLinear(lista, elementoProcurado);
19
20        if (indiceEncontrado != -1) {
21            System.out.println("Elemento " + elementoProcurado + " encontrado na posição " + indiceEncontrado + ".");
22        } else {
23            System.out.println("Elemento " + elementoProcurado + " não encontrado na lista.");
24        }
25    }
26}
```

#### 1.2.3.1. Comparação

Em Java, o operador "`==`" é utilizado para comparar primitivos ou referências de objetos, e não há um operador "`====`" como em JavaScript.

##### Comparação de Primitivos

Quando usado com tipos primitivos (como `int`, `char`, `double`, etc.), o operador "`==`" compara os valores desses primitivos:



A screenshot of a Java code editor showing a simple comparison example. The code is as follows:

```
java
int x = 5;
int y = 5;
boolean result = x == y; // true, pois os valores são iguais
```

The code editor has a "Copy code" button in the top right corner.

## Comparação de Referências de Objetos

Quando usado com referências de objetos, o operador "==" compara se as duas referências apontam para o mesmo objeto na memória, e não se os conteúdos dos objetos são iguais.

```
java Copy code
String str1 = new String("example");
String str2 = new String("example");

boolean result = str1 == str2; // false, pois são diferentes objetos na memó
```

Neste exemplo, mesmo que os conteúdos das strings sejam iguais, o resultado da comparação é `false` porque `str1` e `str2` referenciam diferentes objetos na memória.

## Usando `equals`

Para comparar o conteúdo de dois objetos em Java, geralmente usamos o método `equals`. A implementação padrão desse método na classe `Object` é semelhante ao operador "==" (compara referências), mas muitas classes, como `String`, `Integer`, etc., sobrescrevem esse método para fornecer uma comparação baseada no conteúdo.

```
java Copy code
boolean result = str1.equals(str2); // true, pois o conteúdo das strings é :
```

### 1.3. Busca Binária (Lista Ordenada):

#### 1.3.1. Python

```
Coliseum > HAL2001 >(busca_binaria.py) ...
1  def binarySearch(valor, lista):
2      left = 0
3      right = len(lista) - 1
4      while left <= right:
5          mid = left + (right - left) // 2
6          if lista[mid] == valor:
7              return mid
8          elif lista[mid] > valor:
9              right = mid - 1
10         else:
11             left = mid + 1
12     return -1
13
14 arr = [13,33,77,66,9,8,4,42,5,6]
15 x = 42
16 index_find = binarySearch(x,arr)
17 if (index_find != -1):
18     print(f"O indice do elemento {x} eh {index_find}")
19 else:
20     print(f"O elemento {x} nao estah presente na lista")
0 indice do elemento 42 eh 7
```

### 1.3.2. Node.js

```
Coliseum > HAL2001 > js busca_binaria.js > ...
1  function binarySearch(valor, lista) {
2      var left = 0;
3      var right = lista.length - 1;
4      while (left <= right) {
5          var mid = left + Math.floor((right - left) / 2);
6          if (lista[mid] === valor) {
7              return mid;
8          } else if (lista[mid] > valor) {
9              right = mid - 1;
10         } else {
11             left = mid + 1;
12         }
13     }
14     return -1;
15 }
16
17 const arr = [13,33,77,66,9,8,4,42,5,6];
18 x = 99;
19 index_find = binarySearch(x,arr);
20 if (index_find != -1){
21     console.log(`O indice do elemento ${x} eh ${index_find}`);
22 }else{
23     console.log(`O elemento ${x} nao estah presente na lista`);
24 }
```

```
O elemento 99 nao estah presente na lista
```

#### 1.3.2.1. Math

##### Math.floor

O método `Math.floor(x)` retorna o maior número inteiro que é menor ou igual a um número `x`. Basicamente, arredonda o número para baixo.

Exemplo:

```
javascript
Copy code

var num = 5.7;
console.log(Math.floor(num)); // Saida: 5
```

## Math.ceil

O método `Math.ceil(x)` retorna o menor número inteiro que é maior ou igual a um número `x`. Ou seja, arredonda o número para cima.

Exemplo:

javascript

 Copy code

```
var num = 5.7;
console.log(Math.ceil(num)); // Saída: 6
```

## Math.round

O método `Math.round(x)` retorna o valor de um número arredondado para o número inteiro mais próximo. Ele arredonda para cima se a fração for 0.5 ou superior e arredonda para baixo se a fração for inferior a 0.5.

Exemplo:

javascript

 Copy code

```
var num1 = 5.7;
var num2 = 5.3;
console.log(Math.round(num1)); // Saída: 6
console.log(Math.round(num2)); // Saída: 5
```

## Math.pow

O método `Math.pow(x, y)` retorna a base `x` elevada à potência de `y`. Em outras palavras, multiplica `x` por si mesmo `y` vezes.

Exemplo:

javascript

 Copy code

```
var base = 2;
var exponente = 3;
console.log(Math.pow(base, exponente)); // Saída: 8
```

### 1.3.3. Java

```
Coliseum > HAL2001 > J binarySearch.java > ...
1 package Coliseum.HAL2001;
2
3 public class BinarySearch {
4
5     public static int b_Search(int valor, int[] lista) {
6         int left = 0;
7         int right = lista.length - 1;
8         while (left <= right) {
9             int mid = left + (right - left) / 2;
10            if (lista[mid] == valor) {
11                return mid;
12            } else if (lista[mid] > valor) {
13                right = mid - 1;
14            } else {
15                left = mid + 1;
16            }
17        }
18        return -1;
19    }
20
Run | Debug
21    public static void main(String[] args) {
22        int x = 42;
23        int[] arr = { 1, 3, 5, 8, 11, 13, 33, 42, 66, 77 };
24
25        int index_search = b_Search(x, arr);
26        if (index_search != -1) {
27            System.out.println("O índice de " + x + " é " + index_search + ".");
28        } else {
29            System.out.println("O elemento " + x + " não se encontra na lista.");
30        }
31    }
32 }
```

0 índice de 42 é 7.

### 1.3.3.1. Divisão de inteiros

Na implementação do algoritmo de busca binária, é comum calcular o índice do meio do array. Em linguagens como JavaScript, você pode encontrar o meio usando algo como `Math.floor((low + high) / 2)` para garantir que o resultado seja um número inteiro.

Em Java, no entanto, essa etapa adicional não é necessária. Isso se deve ao fato de que, em Java, a divisão de dois inteiros resulta em um inteiro. A parte fracionária é simplesmente descartada. Aqui está um exemplo:

```
java Copy code
int low = 0;
int high = 7;
int mid = (low + high) / 2; // Resulta em 3, não 3.5
```

Como os índices de array em Java são inteiros, a divisão de dois inteiros é adequada para calcular o índice do meio. A parte fracionária é automaticamente descartada, e o resultado é o mesmo que você obteria usando `Math.floor` em uma linguagem que não descarta a parte fracionária na divisão de inteiros.

## 1.4. Busca Binária (Lista Desordenada):

### 1.4.1. Python

```
Coliseum > HAL2001 > ⚙️ binary.py > ...
1  def binary(arr, x):
2      left = 0
3      right = len(arr) - 1
4      indexed_arr = [(index, value) for index, value in enumerate(arr)]
5      indexed_arr.sort(key=lambda tup: tup[1])
6      while left <= right:
7          mid = left + ((right-left)//2)
8          if indexed_arr[mid][1] == x:
9              return indexed_arr[mid][0]
10         elif indexed_arr[mid][1] > x:
11             right = mid - 1
12         else:
13             left = mid + 1
14     return -1
15
16 arr = [5,6,7,3,2,1,15,10,77,88,99,42,35,21,8,12,13]
17 x = 13
18
19 result = binary(arr, x)
20
21 if result != -1:
22     print(f"Elemento encontrado no indice {result}")
23 else:
24     print("Elemento nao encontrado na matriz")
```

```
Elemento encontrado no indice 16
```

#### 1.4.1.1. Sort

##### 1. Ordenar Strings por Comprimento:

Suponha que você queira ordenar uma lista de strings não por ordem alfabética, mas pelo comprimento de cada string. Você pode fazer isso usando o seguinte código:

```
python
Copy code

words = ["banana", "apple", "cherry", "date"]
words.sort(key=len)
print(words) # Saida: ['date', 'apple', 'cherry', 'banana']
```

## 2. Ordenar Objetos por um Atributo:

Imagine que você tenha uma lista de objetos representando pessoas, e você queira ordenar essa lista pelo atributo de idade. Aqui está um exemplo de como você pode fazer isso:

```
python Copy code
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

people = [Person("Alice", 30), Person("Bob", 25), Person("Eve", 35)]
people.sort(key=lambda person: person.age)
# A lista agora está ordenada por idade
```

## 3. Ordenar em Ordem Decrescente:

Você pode usar o parâmetro `key` em combinação com o parâmetro `reverse` para ordenar em ordem decrescente. Por exemplo, para ordenar uma lista de números pelo valor absoluto em ordem decrescente:

```
python Copy code
numbers = [-5, 3, 1, -2, -4]
numbers.sort(key=abs, reverse=True)
print(numbers) # Saída: [-5, -4, 3, -2, 1]
```

## 4. Ordenar Tuplas por Segundo Elemento:

Se você quiser ordenar uma lista de tuplas pelo segundo elemento de cada tupla, você pode fazer isso usando uma função lambda:

```
python Copy code
pairs = [(1, 9), (3, 3), (2, 5)]
pairs.sort(key=lambda pair: pair[1])
print(pairs) # Saída: [(3, 3), (2, 5), (1, 9)]
```

## 1.4.2. Node.js

```
Coliseum > HAL2001 > js binary.js > ...
1  function binary(arr, x) {
2      let right = arr.length - 1;
3      let left = 0;
4      let indexedArr = arr.map((value, index) => ({ index, value }));
5      indexedArr.sort((a, b) => a.value - b.value);
6      while (left <= right) {
7          let mid = left + Math.floor((right - left) / 2);
8          if (indexedArr[mid].value === x) {
9              return indexedArr[mid].index;
10         } else if (indexedArr[mid].value > x) {
11             right = mid - 1;
12         } else {
13             left = mid + 1;
14         }
15     }
16     return -1;
17 }
18
19 const arr = [5,6,7,3,2,1,15,10,77,88,99,42,35,21,8,12,13];
20 const x = 42;
21
22 const result = binary(arr, x);
23
24 if (result !== -1) {
25     console.log(`Elemento encontrado no índice ${result}`);
26 } else {
27     console.log("Elemento não encontrado na matriz");
28 }
```

```
Elemento encontrado no índice 11
```

### 1.4.2.1. Map

O resultado desse código é um novo array de objetos, onde cada objeto contém o valor original e o índice desse valor no array original.

Exemplo:

```
javascript Copy code
const arr = ['a', 'b', 'c'];
const result = arr.map((value, index) => ({ index, value }));
console.log(result);
// [
//   { index: 0, value: 'a' },
//   { index: 1, value: 'b' },
//   { index: 2, value: 'c' }
// ]
```

1. O valor do elemento atual sendo processado no array.
2. O índice do elemento atual sendo processado no array.
3. O array no qual o método `.map` foi chamado.

Portanto, se você inverter a ordem dos parâmetros `value` e `index`, o código não funcionará como esperado, pois os parâmetros receberão os valores errados.

Dado o código:

```
javascript      3     1     2      Copy code
indexed_arr = arr.map(({value, index}) => ({value, index}));
```

#### 1.4.2.2. Sort

```
javascript      Copy code
indexed_arr.sort((a, b) => (a.value - b.value));
```

Aqui, `indexed_arr` é presumivelmente um array de objetos, onde cada objeto tem uma propriedade `value`. A função de comparação recebe dois elementos adjacentes, `a` e `b`, e retorna o resultado de `(a.value - b.value)`.

O resultado dessa expressão determina a ordem em que os elementos `a` e `b` são colocados:

1. Se `(a.value - b.value)` for menor que 0, então `a` vem antes de `b`.
2. Se `(a.value - b.value)` for igual a 0, então `a` e `b` permanecem inalterados em relação uns aos outros, mas ordenados em relação a todos os outros elementos.
3. Se `(a.value - b.value)` for maior que 0, então `b` vem antes de `a`.

Essa função de comparação assegura que o array seja ordenado em ordem crescente com base na propriedade `value` dos objetos.

## 1.5. FizzBuzz:

O problema "FizzBuzz" é um clássico problema de programação, onde você precisa imprimir números de um intervalo, substituindo os múltiplos de 3 por "Fizz", os múltiplos de 5 por "Buzz", e os múltiplos de ambos (15) por "FizzBuzz".

### 1.5.1. Python

```
Coliseum > HAL2001 > ⚙ fizzbuzz.py > ...
1  def fizzbuzz(inicio, fim):
2      resultado = []
3      for i in range(inicio, fim + 1):
4          if i % 3 == 0 and i % 5 == 0:
5              resultado.append("FizzBuzz")
6          elif i % 3 == 0:
7              resultado.append("Fizz")
8          elif i % 5 == 0:
9              resultado.append("Buzz")
10         else:
11             resultado.append(i)
12     return resultado
13
14 inicio = 1
15 fim = 15
16 resultados = fizzbuzz(inicio, fim)
17 print(resultados)
```

```
[1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz']
```

#### 1.5.1.1. Funções de Array (Python)

1. `append(x)`: Adiciona um item ao fim da lista.
2. `extend(iterable)`: Estende a lista, adicionando todos os itens do iterável fornecido (por exemplo, outra lista).

3. `insert(i, x)`: Insere um item em uma posição específica.

python

 Copy code

```
lista = [1, 2, 4]
lista.insert(2, 3) # Resultado: [1, 2, 3, 4]
```

4. `remove(x)`: Remove o primeiro item da lista cujo valor é igual a x.

python

 Copy code

```
lista = [1, 2, 3, 2]
lista.remove(2) # Resultado: [1, 3, 2]
```

5. `pop([i])`: Remove o item na posição fornecida e retorna o seu valor. Se nenhum índice for especificado, remove e retorna o último item da lista.

python

 Copy code

```
lista = [1, 2, 3]
item = lista.pop(1) # Resultado: item = 2, lista = [1, 3]
```

6. `clear()`: Remove todos os itens da lista.

python

 Copy code

```
lista = [1, 2, 3]
lista.clear() # Resultado: []
```

7. `index(x[, start[, end]])`: Retorna o índice do primeiro item cujo valor é igual a x.

python

 Copy code

```
lista = [1, 2, 3, 2]
idx = lista.index(2) # Resultado: idx = 1
```

8. `count(x)`: Retorna o número de vezes que x aparece na lista.

python

 Copy code

```
lista = [1, 2, 3, 2]
count = lista.count(2) # Resultado: count = 2
```

9. `sort(key=None, reverse=False)`: Ordena os itens da lista.

python

 Copy code

```
lista = [3, 1, 2]
lista.sort() # Resultado: [1, 2, 3]
```

10. `reverse()`: Inverte a ordem dos elementos da lista.

python

 Copy code

```
lista = [1, 2, 3]
lista.reverse() # Resultado: [3, 2, 1]
```

11. `copy()`: Retorna uma cópia rasa da lista.

python

 Copy code

```
lista = [1, 2, 3]
copia = lista.copy() # Resultado: copia = [1, 2, 3]
```

### 1.5.2. Node.js

```
Coliseum > HAL2001 > JS fizzbuzz.js > ...
1  function fizzbuzz(inicio, fim) {
2      const resultado = [];
3      for (let i = inicio; i <= fim; i++) {
4          if (i % 3 === 0 && i % 5 === 0) {
5              resultado.push("FizzBuzz");
6          } else if (i % 3 === 0) {
7              resultado.push("Fizz");
8          } else if (i % 5 === 0) {
9              resultado.push("Buzz");
10         } else {
11             resultado.push(i);
12         }
13     }
14     return resultado;
15 }
16
17 const inicio = 15;
18 const fim = 30;
19 const resultados = fizzbuzz(inicio, fim);
20 console.log(resultados);
21
```

```
[
  'FizzBuzz', 16,
  17,           'Fizz',
  19,           'Buzz',
  'Fizz',       22,
  23,           'Fizz',
  'Buzz',       26,
  'Fizz',       28,
  29,           'FizzBuzz'
]
```

### 1.5.2.1. Funções de Array (Node.js)

1. `push(...items)`: Adiciona um ou mais elementos ao fim de um array e retorna o novo comprimento do array.

```
javascript Copy code
const arr = [1, 2, 3];
arr.push(4, 5); // Resultado: arr = [1, 2, 3, 4, 5]
```

2. `pop()`: Remove o último elemento de um array e o retorna.

```
javascript Copy code
const arr = [1, 2, 3];
const lastItem = arr.pop(); // Resultado: lastItem = 3, arr = [1, 2]
```

3. `unshift(...items)`: Adiciona um ou mais elementos ao início de um array e retorna o novo comprimento do array.

```
javascript Copy code
const arr = [1, 2, 3];
arr.unshift(0); // Resultado: arr = [0, 1, 2, 3]
```

4. `shift()`: Remove o primeiro elemento de um array e o retorna.

```
javascript Copy code
const arr = [1, 2, 3];
const firstItem = arr.shift(); // Resultado: firstItem = 1, arr = [2, 3]
```

5. `splice(start[, deleteCount[, ...items]])`: Modifica o conteúdo de um array, removendo ou substituindo elementos existentes e/ou adicionando novos elementos.

```
javascript Copy code
const arr = [1, 2, 3];
arr.splice(1, 1, 4); // Resultado: arr = [1, 4, 3]
```

6. `slice([begin[, end]])`: Retorna uma cópia rasa de uma parte do array em um novo objeto de array.

```
javascript Copy code
const arr = [1, 2, 3];
const sliced = arr.slice(1); // Resultado: sliced = [2, 3]
```

7. `reverse()`: Inverte os elementos do array no lugar.

```
javascript Copy code
const arr = [1, 2, 3];
arr.reverse(); // Resultado: arr = [3, 2, 1]
```

8. `sort([compareFunction])`: Ordena os elementos do array no lugar e retorna o array.

```
javascript Copy code
const arr = [3, 1, 2];
arr.sort(); // Resultado: arr = [1, 2, 3]
```

9. `map(callback(currentValue[, index[, array]])[, thisArg])`: Cria um novo array com os resultados da chamada de uma função fornecida em cada elemento do array.

```
javascript Copy code
const arr = [1, 2, 3];
const doubled = arr.map(x => x * 2); // Resultado: doubled = [2, 4, 6]
```

10. `filter(callback(element[, index[, array]])[, thisArg])`: Cria um novo array com todos os elementos que passam no teste implementado pela função fornecida.

```
javascript Copy code
const arr = [1, 2, 3, 4, 5];
const even = arr.filter(x => x % 2 === 0); // Resultado: even = [2, 4]
```

11. `reduce(callback(accumulator, currentValue[, index[, array]])[, initialValue])`: Executa uma função redutora em cada elemento do array, resultando em um único valor de saída.

```
javascript Copy code
const arr = [1, 2, 3];
const sum = arr.reduce((acc, val) => acc + val, 0); // Resultado: sum = 6
```

### 1.5.3. Java

```
Coliseum > HAL2001 > FizzBuzz.java > ...
1 package Coliseum.HAL2001;
2
3 import java.util.ArrayList;
4
5 public class FizzBuzz {
6     public static ArrayList<Object> fizzbuzz(int inicio, int fim) {
7         ArrayList<Object> resultado = new ArrayList<>();
8         for (int i = inicio; i <= fim; i++) {
9             if (i % 3 == 0 && i % 5 == 0) {
10                 resultado.add("FizzBuzz");
11             } else if (i % 3 == 0) {
12                 resultado.add("Fizz");
13             } else if (i % 5 == 0) {
14                 resultado.add("Buzz");
15             } else {
16                 resultado.add(i);
17             }
18         }
19         return resultado;
20     }
21
22     Run | Debug
23     public static void main(String[] args) {
24         int inicio = 1;
25         int fim = 15;
26         ArrayList<Object> resultados = fizzbuzz(inicio, fim);
27         System.out.println(resultados);
28     }
}
```

```
[1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz]
```

#### 1.5.3.1. Declaração ArrayList

##### 1. Declarando um ArrayList

A declaração de um `ArrayList` requer a importação da classe, e então você pode declarar a variável utilizando a sintaxe genérica.

```
java
import java.util.ArrayList;
ArrayList<Tipo> nomeDaLista = new ArrayList<>();
```

No código FizzBuzz que você forneceu, o tipo do `ArrayList` foi declarado como `Object` porque a lista pretendia armazenar diferentes tipos de valores, incluindo `String` (para "Fizz", "Buzz", e "FizzBuzz") e `Integer` (para os outros números).

Em Java, todas as classes herdam direta ou indiretamente da classe `Object`. Portanto, é possível usar `Object` como tipo genérico para uma coleção que pode conter diferentes tipos de objetos.

### 1.5.3.2. Métodos ArrayList

#### 2.1. Adicionar Elementos

```
java Copy code
nomeDaLista.add(elemento); // Adiciona o elemento ao final da lista
nomeDaLista.add(index, elemento); // Adiciona o elemento na posição especificada
```

#### 2.2. Remover Elementos

```
java Copy code
nomeDaLista.remove(index); // Remove o elemento na posição especificada
nomeDaLista.remove(elemento); // Remove a primeira ocorrência do elemento especificado
```

#### 2.3. Obter Elemento

```
java Copy code
Tipo elemento = nomeDaLista.get(index); // Obtém o elemento na posição especificada
```

#### 2.4. Alterar Elemento

```
java Copy code
nomeDaLista.set(index, elemento); // Substitui o elemento na posição especificada
```

#### 2.5. Tamanho da Lista

```
java Copy code
int tamanho = nomeDaLista.size(); // Retorna o número de elementos na lista
```

#### 2.6. Verificar se Contém

```
java Copy code
boolean contem = nomeDaLista.contains(elemento); // Verifica se a lista contém o elemento
```

## 2.7. Limpar a Lista

```
java
```

Copy code

```
nomeDaLista.clear(); // Remove todos os elementos da lista
```

## 2.8. Converter para Array

```
java
```

Copy code

```
Tipo[] array = nomeDaLista.toArray(new Tipo[0]); // Converte a lista em um array
```

## Exemplo Completo

```
java
```

Copy code

```
import java.util.ArrayList;

public class ExemploArrayList {
    public static void main(String[] args) {
        ArrayList<String> frutas = new ArrayList<>();
        frutas.add("Maçã");
        frutas.add("Banana");
        frutas.add(1, "Morango");
        frutas.remove("Banana");
        System.out.println(frutas.get(1)); // Saída: Morango
        frutas.set(0, "Pera");
        System.out.println(frutas.size()); // Saída: 2
        System.out.println(frutas.contains("Morango")); // Saída: true
        frutas.clear();
    }
}
```

## 1.6. Fibonacci (iterativo):

Gerar a série de Fibonacci até um determinado número.

### 1.6.1. Python

```
Coliseum > HAL2001 > ⚡ fibo_it.py > ...
1  def fibo_it(num):
2      fib = [0,1]
3      if (num <= 0):
4          fib = []
5          return fib
6      if (num <= 1):
7          return [fib[0]]
8      elif (num == 2):
9          return fib
10     else:
11         for i in range(num-2):
12             temp = fib[i] + fib[i+1]
13             fib.append(temp)
14     return fib
15
16 x = 13
17 sequencia_fib = fibo_it(x)
18 print(f"SEQUENCIA FIBONNACI: {sequencia_fib}")
19 if (len(sequencia_fib)>0):
20     print(f"Posicao {x} na sequencia fibonacci = {sequencia_fib[len(sequencia_fib) -1]}")
21 else:
22     print(f"Numero {x} INVALIDO!")
```

```
SEQUENCIA FIBONNACI: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
Posicao 13 na sequencia fibonacci = 144
```

## 1.6.2. Node.js

```
Coliseum > HAL2001 > js fibo_it.js > ...
1  function fibo_it(num){
2      const fib = [0,1];
3      if (num <= 0){
4          fib = [];
5      } else if (num === 1){
6          return fib[0]
7      }else if (num === 2){
8          return fib
9      }else{
10         for (let i = 0; i< num - 2; i++){
11             temp = fib[i] + fib[i+1];
12             fib.push(temp);
13         }
14     }
15     return fib;
16 }
17
18 x = 17;
19 const sequencia = fibo_it(x);
20 console.log(`SEQUENCIA FIBONNACI: ${sequencia}`);
21 if (sequencia.length > 0){
22     console.log(`Posição ${x} na sequencia fibonacci = ${sequencia[sequencia.length-1]}`);
23 }
24
```

```
SEQUENCIA FIBONNACI: 0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987
Posição 17 na sequencia fibonacci = 987
```

### 1.6.3. Java

```
Coliseum > HAL2001 > J Fibojava > Fibo
1 package Coliseum.HAL2001;
2
3 import java.util.ArrayList;
4
5 public class Fibo {
6
7     public static ArrayList<Integer> calcula_fibo(int num) {
8         ArrayList<Integer> fib = new ArrayList<>();
9         fib.add(0);
10        if (num <= 0) {
11            fib.clear();
12            return fib;
13        } else if (num == 1) {
14            return fib;
15        } else if (num >= 2) {
16            fib.add(1);
17            for (int i = 0; i < num - 2; i++) {
18                int next = fib.get(i) + fib.get(i + 1);
19                fib.add(next);
20            }
21        }
22    }
23 }
```

```
24 Run | Debug
25 public static void main(String[] args) {
26     int x = 10;
27     ArrayList<Integer> sequencia_fib = calcula_fibo(x);
28     System.out.println("SEQUENCIA FIBONNACI: " + sequencia_fib);
29     if (sequencia_fib.size() > 0) {
30         System.out.println(
31             "Posicao " + x + " na sequencia fibonacci = " + sequencia_fib.get(sequencia_fib.size() - 1));
32     } else {
33         System.out.println("Numero " + x + " INVALIDO!");
34     }
35 }
36 }
```

```
SEQUENCIA FIBONNACI: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Posicao 10 na sequencia fibonacci = 34
```

## 1.7. Fibonacci (Recursivo):

Use função recursiva para calcular o n-ésimo termo da sequência Fibonacci.

### 1.7.1. Python

```
Coliseum > HAL2001 > fibo_rec.py > ...
1  def fibo_rec(n):
2      if n <= 1:
3          return n
4      else:
5          return fibo_rec(n - 1) + fibo_rec(n - 2)
6
7  n = 10
8  result = fibo_rec(n)
9  print(f"O {n}º numero da serie de Fibonacci eh {result}")

O 10th numero da serie de Fibonacci eh 55
```

### 1.7.2. Node.js

```
Coliseum > HAL2001 > fibo_rec.js > ...
1  function fibonacci(n) {
2      if (n <= 1) {
3          return n;
4      } else {
5          return fibonacci(n - 1) + fibonacci(n - 2);
6      }
7
8
9  const n = 42;
10 const result = fibonacci(n);
11 console.log(`O ${n}º número da série de Fibonacci é ${result}`);

O 42º número da série de Fibonacci é 267914296
```

### 1.7.3. Java

```
Coliseum > HAL2001 > J Fibo_Rec.java > ...
1 package Coliseum.HAL2001;
2
3 public class Fibo_Rec {
4     public static int fibonacci(int n) {
5         if (n <= 1) {
6             return n;
7         } else {
8             return fibonacci(n - 1) + fibonacci(n - 2);
9         }
10    }
11
12    Run | Debug
13    public static void main(String[] args) {
14        int n = 10;
15        int result = fibonacci(n);
16        System.out.println("O " + n + "º número da série de Fibonacci é " + result);
17    }
18 }
```

```
O 10º número da série de Fibonacci é 55
```

## 1.8. Fatorial:

Calcular o fatorial de um número.

### 1.8.1. Python

```
Coliseum > HAL2001 > fatorial.py > ...
1  def fatorial(n):
2      resultado = 1
3      for i in range(1, n + 1):
4          resultado *= i
5      return resultado
6
7  numero = 5
8  print(f'O fatorial de {numero} eh {fatorial(numero)}')
O fatorial de 5 eh 120
```

## 1.8.2. Node.js

```
Coliseum > HAL2001 > JS fatorial.js > ...
1  function fatorial(n) {
2      let resultado = 1;
3      for (let i = 1; i <= n; i++) {
4          resultado *= i;
5      }
6      return resultado;
7  }
8
9  const numero = 10;
10 console.log(`O factorial de ${numero} é ${fatorial(numero)} `);
11
```

```
O factorial de 10 é 3628800
```

## 1.8.3. Java

```
Coliseum > HAL2001 > J Fatorial.java > ...
1  package Coliseum.HAL2001;
2
3  public class Fatorial {
4      public static int fatorial(int n) {
5          int resultado = 1;
6          for (int i = 1; i <= n; i++) {
7              resultado *= i;
8          }
9          return resultado;
10     }
11
12     Run | Debug
13     public static void main(String[] args) {
14         int numero = 13;
15         System.out.println("O factorial de " + numero + " é " + fatorial(numero));
16     }
17 }
```

```
O factorial de 13 é 1932053504
```

## 1.9. Números Primos:

Encontrar números primos em um determinado intervalo.

### 1.9.1. Python

```
Coliseum > HAL2001 > 📁 primo.py > ...
1  def numeros_primos_no_intervalo(inicio, fim):
2      primos = []
3      for num in range(inicio, fim + 1):
4          if num > 1:
5              isPrime = True
6              for i in range(2, num):
7                  if (num % i) == 0:
8                      isPrime = False
9                      break
10             if isPrime:
11                 primos.append(num)
12
13
14     inicio = 0
15     fim = 42
16     primos = numeros_primos_no_intervalo(inicio, fim)
17     print(f"Numeros primos no intervalo de {inicio} a {fim}: {primos}")
18
Numeros primos no intervalo de 0 a 42: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]
```

## 1.9.2. Node.js

```
Coliseum > HAL2001 > js primojs > ...
1  function numerosPrimosNoIntervalo(inicio, fim) {
2      const primos = [];
3      for (let num = inicio; num <= fim; num++) {
4          if (num > 1) {
5              let isPrime = true;
6              for (let i = 2; i < num; i++) {
7                  if (num % i === 0) {
8                      isPrime = false;
9                      break;
10                 }
11             }
12             if (isPrime) primos.push(num);
13         }
14     }
15     return primos;
16 }
17
18 const inicio = 10;
19 const fim = 50;
20 const primos = numerosPrimosNoIntervalo(inicio, fim);
21 console.log(`Números primos no intervalo de ${inicio} a ${fim}: ${primos}`);
```

```
Números primos no intervalo de 10 a 50: 11,13,17,19,23,29,31,37,41,43,47
```

### 1.9.3. Java

```
Coliseum > HAL2001 > J NumerosPrimos.java > ...
1  package Coliseum.HAL2001;
2
3  import java.util.ArrayList;
4
5  public class NumerosPrimos {
6      public static ArrayList<Integer> numerosPrimosNoIntervalo(int inicio, int fim) {
7          ArrayList<Integer> primos = new ArrayList<>();
8          for (int num = inicio; num <= fim; num++) {
9              if (num > 1) {
10                  boolean isPrime = true;
11                  for (int i = 2; i < num; i++) {
12                      if (num % i == 0) {
13                          isPrime = false;
14                          break;
15                      }
16                  }
17                  if (isPrime)
18                      primos.add(num);
19              }
20          }
21          return primos;
22      }
23
24      Run | Debug
25      public static void main(String[] args) {
26          int inicio = 42;
27          int fim = 100;
28          ArrayList<Integer> primos = numerosPrimosNoIntervalo(inicio, fim);
29          System.out.println("Números primos no intervalo de " + inicio + " a " + fim + ": " + primos);
30      }
31 }
```

```
Números primos no intervalo de 42 a 100: [43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

## 1.10. Palíndromo:

Verificar se uma string é um palíndromo.

### 1.10.1. Python

```
Coliseum > HAL2001 > ✎ palindromo.py > ...
1  def palindromo(palavra):
2      if len(palavra) < 3:
3          return '?'
4      invertida = palavra[::-1]
5      if palavra == invertida:
6          return 'PALINDROMO'
7      return 'N'
8
9  palavra = "anilina"
10 code=palindromo(palavra)
11 print(f"{palavra}: {code}")
12 palavra = "Zaphodhdpaz"
13 code=palindromo(palavra)
14 print(f"{palavra}: {code}")
15 palavra = "Marvin"
16 code=palindromo(palavra)
17 print(f"{palavra}: {code}")
18 palavra = "42"
19 code=palindromo(palavra)
20 print(f"{palavra}: {code}")
21
```

```
anilina: PALINDROMO
Zaphodhdpaz: PALINDROMO
Marvin: N
42: ?
```

## 1.10.2. Node.js

```
Coliseum > HAL2001 > js palindromo.js > ...
1  function palindromo(palavra){
2      palavra_lower = palavra.toLowerCase();
3      if (palavra.length <=3){
4          return "?";
5      }
6      const invertida = palavra_lower.split('').reverse().join('');
7      if (palavra_lower === invertida){
8          return "PALINDROMO";
9      }else{
10         return "N"
11     }
12 }
13
14 let word = "Malayalam";
15 let code = palindromo(word);
16 console.log(`#${word}: ${code}` );
17
```

```
Malayalam: PALINDROMO
```

### 1.10.3. Java

```
Coliseum > HAL2001 > J Palindromo.java > ...
1 package Coliseum.HAL2001;
2
3 public class Palindromo {
4     public static String palindromo(String palavra) {
5         String palavraLower = palavra.toLowerCase();
6
7         if (palavraLower.length() < 3) {
8             return "?";
9         }
10
11         String invertida = new StringBuilder(palavraLower).reverse().toString();
12
13         if (palavraLower.equals(invertida)) {
14             return "PALINDROMO";
15         }
16         return "N";
17     }
18
19     public static void main(String[] args) {
20         System.out.println("42: " + palindromo("42"));
21         System.out.println("Able was I ere I saw Elba: " + palindromo("Able was I ere I saw Elba"));
22         System.out.println("mama: " + palindromo("mama"));
23         System.out.println("anilina: " + palindromo("anilina"));
24     }
25 }
```

```
42: ?
Able was I ere I saw Elba: PALINDROMO
mama: N
anilina: PALINDROMO
```

## 1.11. Maior Divisor Comum (MDC):

Encontrar o maior divisor comum entre dois números.

### 1.11.1. Python

```
Coliseum > HAL2001 > 🐍 mdc.py > ...
1  def mdc(a, b):
2      menor_numero = min(a, b)
3      for divisor in range(menor_numero, 0, -1):
4          if a % divisor == 0 and b % divisor == 0:
5              return divisor
6
7  numero1 = 56
8  numero2 = 48
9
10 resultado = mdc(numero1, numero2)
11
12 print(f"O maior divisor comum entre {numero1} e {numero2} eh {resultado}")
                                O maior divisor comum entre 56 e 48 eh 8
```

### 1.11.2. Node.js

```
Coliseum > HAL2001 > JS mdc.js > ...
1  function mdc(a,b){
2      let menor = Math.min(a,b);
3      for (let divisor = menor; divisor > 0 ; divisor--){
4          if (a % divisor === 0 && b % divisor === 0){
5              return divisor
6          }
7      }
8  }
9
10 let numero1 = 42;
11 let numero2 = 98;
12
13 let resultado = mdc(numero1, numero2);
14
15 console.log(`O maior divisor comum entre ${numero1} e ${numero2} é ${resultado}`);
16
```

O maior divisor comum entre 42 e 98 é 14

### 1.11.3. Java

```
Coliseum > HAL2001 > MDC.java > ...
1 package Coliseum.HAL2001;
2
3 public class MDC {
4     public static int calcula_MDC(int a, int b) {
5         int menor = Math.min(a, b);
6         for (int divisor = menor; divisor > 0; divisor--) {
7             if (a % divisor == 0 && b % divisor == 0) {
8                 return divisor;
9             }
10        }
11    }
12 }
13
14 Run | Debug
15 public static void main(String[] args) {
16     System.out.println(calcula_MDC(48, 56));
17 }
18 }
```

## 1.12. Mínimo Múltiplo Comum (MMC):

Mínimo Múltiplo Comum (MMC) é o menor número que é múltiplo de dois ou mais números sem restante. Encontrar o mínimo múltiplo comum entre dois números.

### 1.12.1. Python

```
Coliseum > HAL2001 > mmc.py > ...
1  def mdc(a, b): # MANEIRA ALTERNATIVA DE CALCULAR MDC (ALGORITMO DE EUCLIDES):
2      while b != 0:
3          a, b = b, a % b
4      return a
5
6  def mmc(a, b):
7      return a * b // mdc(a, b)
8
9  numero1 = 12
10 numero2 = 18
11
12 resultado = mmc(numero1, numero2)
13 print(f'O MMC de {numero1} e {numero2} eh {resultado}')
14
```

```
1. Primeira iteração:
    - `a = 12`, `b = 18`
    - `a % b = 12`, então atualizamos `a = 18` e `b = 12`
2. Segunda iteração:
    - `a = 18`, `b = 12`
    - `a % b = 6`, então atualizamos `a = 12` e `b = 6`
3. Terceira iteração:
    - `a = 12`, `b = 6`
    - `a % b = 0`, então atualizamos `a = 6` e `b = 0`
4. Quarta iteração:
    - `b = 0`, então o loop termina, e `a = 6` é retornado

O MDC de 12 e 18 é 6, o que é o resultado esperado.
```

```
Coliseum > HAL2001 > mmc2.py > ...
1  def mdc(a,b):      #(ALGORITMO DE EUCLIDES)
2      while (b!=0):
3          temp = b
4          b = a % b
5          a = temp
6      return a
7
8  def mmc(a,b):
9      return ((a*b) // mdc(a,b))
10
11 num1 = 12
12 num2 = 18
13 print(f'O MDC entre {num1} e {num2} eh {mdc(num1,num2)}')
14 print(f'O MMC entre {num1} e {num2} eh {mmc(num1,num2)}')
15
```

```
O MDC entre 12 e 18 eh 6
O MMC entre 12 e 18 eh 36
```

### 1.12.2. Node.js

```
Coliseum > HAL2001 > js mmc.js > ...
1  function mdc(a,b){
2      while (b != 0){
3          let temp = b;
4          b = a % b;
5          a = temp;
6      } return a;
7
8 }
9
10 function mmc(a,b){
11     return (a * b / mdc(a,b))
12 }
13
14 let num1 = 12;
15 let num2 = 18;
16 console.log(`MDC de ${num1} e ${num2} é ${mdc(num1,num2)}.`)
17 console.log(`MMC de ${num1} e ${num2} é ${mmc(num1,num2)}.`)
18
```

```
MDC de 12 e 18 é 6.
MMC de 12 e 18 é 36.
```

### 1.12.3. Java

```
Coliseum > HAL2001 > J MMC.java > ...
1 package Coliseum.HAL2001;
2
3 public class MMC {
4     public static int calcula_MDC(int a, int b) {
5         while (b != 0) {
6             int temp = b;
7             b = a % b;
8             a = temp;
9         }
10        return a;
11    }
12
13    public static int calcula_MMC(int a, int b) {
14        return ((a * b) / calcula_MDC(a, b));
15    }
16
Run | Debug
17    public static void main(String[] args) {
18        int num1 = 12;
19        int num2 = 18;
20        System.out.println("O MDC entre " + num1 + " e " + num2 + " é : " + calcula_MDC(num1, num2));
21        System.out.println("O MMC entre " + num1 + " e " + num2 + " é : " + calcula_MMC(num1, num2));
22    }
23
24 }
```

```
O MDC entre 12 e 18 é : 6
O MMC entre 12 e 18 é : 36
```

## 1.13. Randomização:

O algoritmo procede percorrendo o array de trás para frente e trocando cada elemento com um elemento aleatório em um índice anterior (incluindo possivelmente ele mesmo). Isso garante que cada permutação possível seja igualmente provável, e assim, o rearranjo é verdadeiramente aleatório.

### 1.13.1. Python

```
Coliseum > HAL2001 > randomize.py > ...
1  import random
2
3  def randomize(arr):
4      n = len(arr)
5      for i in range(n - 1, 0, -1):
6          j = random.randint(0, i) # índice aleatório no intervalo [0, i]
7          arr[i], arr[j] = arr[j], arr[i]
8      return arr
9
10 nomes = ["VogonCode", "InfiniteImprobabilitySource", "DontPanic_Code", "MarvinScripts", "FordPrefect"]
11 result = randomize(nomes)
12 print(result)
13 numbers = [42,69,66,77,88,99999,756,33,45,13]
14 result = randomize(numbers)
15 print(result)

[ 'DontPanic_Code', 'VogonCode', 'InfiniteImprobabilitySource', 'MarvinScripts', 'FordPrefect' ]
[45, 88, 69, 99999, 66, 42, 77, 33, 13, 756]
```

## 1.13.2. Node.js

```
Coliseum > HAL2001 > js randomize.js > ...
1  function randomize (arr) {
2    const n = arr.length;
3    for (let i = n - 1; i > 0; i--) {
4      const j = Math.floor(Math.random() * (i + 1)); // Math.random() será igual a [0 até 0,9999...]
5      [arr[i], arr[j]] = [arr[j], arr[i]];
6    }
7    return arr;
8  };
9
10 const names = ["VogonCode", "InfiniteImprobabilitySource", "DontPanic_Code", "MarvinScripts", "FordPrefect"];
11 const randomizedNames = randomize(names);
12 console.log(randomizedNames);
13
14 const numbers = [42, 69, 66, 77, 88, 99999, 756, 33, 45, 13];
15 const randomizedNumbers = randomize(numbers);
16 console.log(randomizedNumbers);
```

```
[
  'MarvinScripts',
  'DontPanic_Code',
  'VogonCode',
  'InfiniteImprobabilitySource',
  'FordPrefect'
]
[
  88, 66, 756,
  42, 45, 33,
  77, 69, 13,
  99999
]
```

### 1.13.3. Java

```
Coliseum > HAL2001 > Randomize.java > ...
1 package Coliseum.HAL2001;
2
3 import java.util.Random;
4
5 public class Randomize {
6     public static void randomize(Object[] arr) {
7         int n = arr.length;
8         Random rand = new Random();
9         for (int i = n - 1; i > 0; i--) {
10             int j = rand.nextInt(i + 1);
11             Object temp = arr[i];
12             arr[i] = arr[j];
13             arr[j] = temp;
14         }
15     }
16
17     Run | Debug
18     public static void main(String[] args) {
19         String[] names = { "VogonCode", "InfiniteImprobabilitySource", "DontPanic_Code", "MarvinScripts", "FordPrefect" };
20         randomize(names);
21         for (String name : names) {
22             System.out.print(name + " ");
23         }
24         System.out.println();
25
26         Integer[] numbers = { 42, 69, 66, 77, 88, 99999, 756, 33, 45, 13 };
27         randomize(numbers);
28         for (Integer number : numbers) {
29             System.out.print(number + " ");
30         }
31         System.out.println();
32     }
}
```

```
VogonCode FordPrefect DontPanic_Code MarvinScripts InfiniteImprobabilitySource
69 13 33 45 99999 77 42 756 66 88
```

## 1.14. Bubble Sort

### 1.14.1. Python

```
Coliseum > HAL2001 > ⚙ bubble_sort.py > ...
1  def bubble_sort(arr):
2      n = len(arr)
3      for i in range(n):
4          for j in range(0, n-i-1):
5              if arr[j] > arr[j+1]:
6                  arr[j], arr[j+1] = arr[j+1], arr[j]
7      return arr
8
9  numbers = [42, 69, 66, 77, 88, 99999, 756, 33, 45, 13]
10 sorted_numbers = bubble_sort(numbers)
11 print("Sorted Numbers:", sorted_numbers)
12 names = ["Ford Prefect", "Zaphod Beeblebrox", "Arthur Dent", "Trillian", "Marvin"]
13 sorted_names = bubble_sort(names)
14 print(f"Sorted Names: {sorted_names}")
15
```

```
Sorted Numbers: [13, 33, 42, 45, 66, 69, 77, 88, 756, 99999]
Sorted Names: ['Arthur Dent', 'Ford Prefect', 'Marvin', 'Trillian', 'Zaphod Beeblebrox']
```

## 1.14.2. Node.js

```
Coliseum > HAL2001 > JS bubble_sort.js > ...
1  function bubble_sort(arr){
2      let n = arr.length;
3      for (let i = 0; i < n; i++){
4          for (let j=0; j < n-i-1;j++){
5              if (arr[j]>arr[j+1]){
6                  [arr[j],arr[j+1]] = [arr[j+1],arr[j]];
7              }
8          }
9      }
10     return arr;
11 }
12
13 const lista = [7,99,456,13,45,42];
14 console.log(bubble_sort(lista));
15
```

```
[ 7, 13, 42, 45, 99, 456 ]
```

### 1.14.3. Java

```
Coliseum > HAL2001 > J Bubble_sort.java > ...
1 package Coliseum.HAL2001;
2
3 import java.util.Arrays;
4
5 public class Bubble_sort {
6     public static int[] ordena_bubble(int[] arr) {
7         int n = arr.length;
8         for (int i = 0; i < n; i++) {
9             for (int j = 0; j < n - i - 1; j++) {
10                 if (arr[j] > arr[j + 1]) {
11                     int temp = arr[j];
12                     arr[j] = arr[j + 1];
13                     arr[j + 1] = temp;
14                 }
15             }
16         }
17         return arr;
18     }
19
20     Run | Debug
21     public static void main(String[] args) {
22         int[] lista = { 99, 77, 55, 13, 42, 666 };
23         System.out.println(Arrays.toString(ordena_bubble(lista)));
24     }
25 }
```

#### 1.14.3.1. Imprimir Array

```
int[] numbers = {3, 2, 1};
// Converter para string e imprimir
System.out.println(Arrays.toString(numbers));
```

## 1.15. Insertion Sort:

### 1.15.1. Python

```
Coliseum > HAL2001 > insertion_sort.py > ...
1  def insertion_sort(arr):
2      n = len(arr)
3      for i in range(1, n):
4          j = i
5          while j > 0 and numbers[j] < numbers[j - 1]:
6              numbers[j], numbers[j - 1] = numbers[j - 1], numbers[j]
7              j -= 1
8      return arr
9
10 numbers = [42, 13, 69, 5, 1, 8, 7, 3, 4, 10, 9,]
11 insertion_sort(numbers)
12 print(numbers)
13
```

```
[1, 3, 4, 5, 7, 8, 9, 10, 13, 42, 69]
```

## 1.15.2. Node.js

```
Coliseum > HAL2001 > JS insertion_sort.js > ...
1   function insertion_sort(arr){
2       let n = arr.length;
3       for (let i=1; i < n; i++){
4           let j = i;
5           while (j > 0 && arr[j] < arr[j-1]){
6               [arr[j], arr[j-1]] = [arr[j-1], arr[j]];
7               j--;
8           }
9       }
10      return arr;
11  }
12
13  numbers = [42, 13, 69, 5, 1, 8, 7, 3, 4, 10, 9,];
14  insertion_sort(numbers);
15  console.log(numbers);
16
```

```
[  
    1, 3, 4, 5, 7,  
    8, 9, 10, 13, 42,  
    69  
]
```

### 1.15.3. Java

```
Coliseum > HAL2001 > J Insertion_Sort.java > ...
1  package Coliseum.HAL2001;
2
3  import java.util.Arrays;
4
5  public class Insertion_Sort {
6      public static int[] ordena_insertion(int[] arr) {
7          int n = arr.length;
8          for (int i = 1; i < n; i++) {
9              int j = i;
10             while (j > 0 && arr[j] < arr[j - 1]) {
11                 int temp = arr[j];
12                 arr[j] = arr[j - 1];
13                 arr[j - 1] = temp;
14                 j--;
15             }
16         }
17         return arr;
18     }
19
20     Run | Debug
21     public static void main(String[] args) {
22         int[] numeros = { 42, 13, 69, 5, 1, 8, 7, 3, 4, 10, 9, };
23         ordena_insertion(numeros);
24         System.out.println(Arrays.toString(numeros));
25     }
26 }
```

[1, 3, 4, 5, 7, 8, 9, 10, 13, 42, 69]

## 1.16. Quick Sort:

### 1.16.1. Python

```
Coliseum > HAL2001 > ✎ quicksort.py > ...
1  def quicksort(arr):
2      if len(arr) <= 1:
3          return arr
4      pivot = arr[len(arr) // 2]
5      left = [x for x in arr if x < pivot]
6      middle = [x for x in arr if x == pivot]
7      right = [x for x in arr if x > pivot]
8      return quicksort(left) + middle + quicksort(right)
9
10 numbers = [42, 13, 69, 5, 99, 27, 19, 57, 71, 81, 11, 31, 8, 50, 64]
11 sorted_numbers = quicksort(numbers)
12 print("Array ordenado:", sorted_numbers)
13
Array ordenado: [5, 8, 11, 13, 19, 27, 31, 42, 50, 57, 64, 69, 71, 81, 99]
```

## 1.16.2. Node.js

```
Coliseum > HAL2001 > JS quicksort.js > ...
1  function quicksort(arr) {
2      if (arr.length <= 1) {
3          return arr;
4      }
5      const pivot = arr[Math.floor(arr.length / 2)];
6      const left = [];
7      const middle = [];
8      const right = [];
9
10     for (let i = 0; i < arr.length; i++) {
11         if (arr[i] < pivot) {
12             left.push(arr[i]);
13         } else if (arr[i] > pivot) {
14             right.push(arr[i]);
15         } else {
16             middle.push(arr[i]);
17         }
18     }
19
20     return [...quicksort(left), ...middle, ...quicksort(right)];
21 }
22
23 const numbers = [42, 13, 69, 5, 99, 27, 19, 57, 71, 81, 11, 31, 8, 50, 64];
24 const sortedNumbers = quicksort(numbers);
25 console.log("Array ordenado:", sortedNumbers);
26
```

### 1.16.2.1. Operador de Espalhamento/Spread("...")

O operador `...` é conhecido como o operador de espalhamento ("spread operator") em JavaScript. Ele permite que uma expressão seja expandida em locais onde múltiplos argumentos (para chamadas de função) ou múltiplos elementos (para arrays) são esperados.

Neste contexto específico, `[...,quicksort(left), ...,middle, ...,quicksort(right)]` faz o seguinte:

1. `...quicksort(left)`: A função `quicksort(left)` retorna um array de elementos que são menores que o pivô. O operador `...` espalha esses elementos no novo array.
2. `...middle`: Este array contém os elementos que são iguais ao pivô. O operador `...` espalha esses elementos no novo array, imediatamente após os elementos menores que o pivô.
3. `...quicksort(right)`: A função `quicksort(right)` retorna um array de elementos que são maiores que o pivô. O operador `...` espalha esses elementos no novo array, após os elementos que são iguais ao pivô.

O resultado final é um novo array que concatena os três conjuntos de elementos: aqueles menores que o pivô, aqueles iguais ao pivô, e aqueles maiores que o pivô, respectivamente. Este é um passo crucial para o algoritmo de Quick Sort, pois ele efetua a concatenação das partes ordenadas do array em um único array ordenado.

## 1.17. Mochila

Dado um conjunto de itens, cada um com um valor e um peso, e uma capacidade máxima para uma mochila, determine o valor máximo que pode ser alcançado selecionando alguns dos itens sem exceder a capacidade da mochila. A solução deve utilizar uma abordagem gulosa.

### 1.17.1. Python

```
Coliseum > HAL2001 > mochila.py > ...
1  def knapsack(values, weights, capacity):
2      n = len(values) # Número total de itens
3      # Calcula a relação valor/peso para cada item e armazena em uma lista de tuplas:
4      ratios = [(values[i] / weights[i], values[i], weights[i]) for i in range(n)]
5          # [(6.0, 60, 10), (4.0, 120, 30), (5.0, 100, 20)]
6
7      ratios.sort(reverse=True) # Ordena os itens pelo valor/peso em ordem decrescente
8
9      total_value = 0
10     current_weight = 0
11     selected_items = []
12
13     # Itera sobre os itens ordenados pela relação valor/peso
14     for ratio, value, weight in ratios:
15         if current_weight + weight <= capacity:
16             selected_items.append((value, weight))
17             total_value += value
18             current_weight += weight
19
20     return total_value, selected_items
21
22 item_values = [60, 120, 100]
23 item_weights = [10, 30, 20]
24 knapsack_capacity = 50
25
26 total_value, selected_items = knapsack(item_values, item_weights, knapsack_capacity)
27
28 print("Valor total na mochila:", total_value)
29 print("Itens selecionados:")
30 for value, weight in selected_items:
31     print(f"Item - Valor: {value}, Peso: {weight}")
32
```

```
Valor total na mochila: 160
Itens selecionados:
Item - Valor: 60, Peso: 10
Item - Valor: 100, Peso: 20
```

## 1.17.2. Node.js

```
Coliseum > HAL2001 > js mochilajs > ...
1  function mochila(values, weights, capacity) {
2      const n = values.length;
3      const ratios = values.map((value, i) => [value / weights[i], value, weights[i]]);
4      ratios.sort((a, b) => b[0] - a[0]); //ORDENA DECRESCENTE A PARTIR DO i(index)=0
5
6      let totalValue = 0;
7      let currentWeight = 0;
8      const selectedItems = [];
9
10     for (let i = 0; i < n; i++) {
11         if (currentWeight + weights[i] <= capacity) {
12             selectedItems.push([values[i], weights[i]]);
13             totalValue += values[i];
14             currentWeight += weights[i];
15         }
16     }
17
18     return { totalValue, selectedItems };
19 }
20
21 const itemValues = [60, 100, 120];
22 const itemWeights = [10, 20, 30];
23 const knapsackCapacity = 50;
24
25 const { totalValue, selectedItems } = mochila(itemValues, itemWeights, knapsackCapacity);
26
27 console.log(`Valor total na mochila: ${totalValue}`);
28 console.log("Itens selecionados:");
29 for (let i = 0; i < selectedItems.length; i++) {
30     const value = selectedItems[i][0];
31     const weight = selectedItems[i][1];
32     console.log("Item - Valor: " + value + ", Peso: " + weight);
33 }
```

```
Valor total na mochila: 160
Itens selecionados:
Item - Valor: 60, Peso: 10
Item - Valor: 100, Peso: 20
```

### 1.17.3. Java

```
Coliseum > J4V4 > mochila > KnapsackGreedy.java > KnapsackGreedy
 1 package Coliseum.J4V4.mochila;
 2
 3 import java.util.ArrayList;
 4 import java.util.Arrays;
 5 import java.util.Comparator;
 6
 7 public class KnapsackGreedy {
 8     public static class Item {
 9         double ratio;
10         int value;
11         int weight;
12
13         public Item(double ratio, int value, int weight) {
14             this.ratio = ratio;
15             this.value = value;
16             this.weight = weight;
17         }
18     }
19
20     public static class Result {
21         int totalValue;
22         ArrayList<Item> selectedItems;
23
24         public Result(int totalValue, ArrayList<Item> selectedItems) {
25             this.totalValue = totalValue;
26             this.selectedItems = selectedItems;
27         }
28     }
29
30     public static Result knapsackGreedy(int[] values, int[] weights, int capacity) {
31         int n = values.length;
32         Item[] ratios = new Item[n];
33         for (int i = 0; i < n; i++) {
34             ratios[i] = new Item((double) values[i] / weights[i], values[i], weights[i]);
35         }
36         Arrays.sort(ratios, Comparator.comparingDouble((Item item) -> item.ratio).reversed());
37
38         int totalValue = 0;
39         int currentWeight = 0;
40         ArrayList<Item> selectedItems = new ArrayList<>();
41
42         for (Item item : ratios) {
43             if (currentWeight + item.weight <= capacity) {
44                 selectedItems.add(item);
45                 totalValue += item.value;
46                 currentWeight += item.weight;
47             }
48         }
49
50         return new Result(totalValue, selectedItems);
51     }
}
```

```
Run | Debug
53 public static void main(String[] args) {
54     int[] itemValues = { 60, 100, 120 };
55     int[] itemWeights = { 10, 20, 30 };
56     int knapsackCapacity = 50;
57
58     Result result = knapsackGreedy(itemValues, itemWeights, knapsackCapacity);
59
60     System.out.println("Valor total na mochila: " + result.totalValue);
61     System.out.println("Itens selecionados:");
62     for (Item item : result.selectedItems) {
63         System.out.println("Item - Valor: " + item.value + ", Peso: " + item.weight);
64     }
65 }
66 }
```

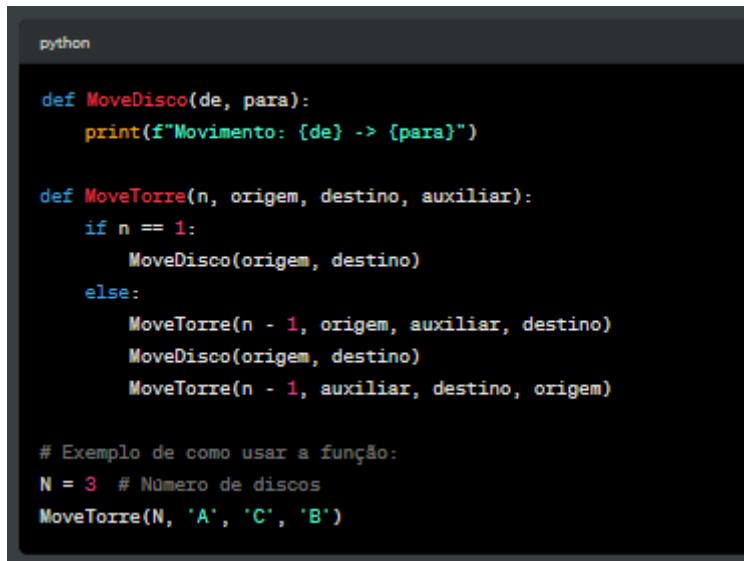
## 1.18. Torre de Hanói:

Resolver o problema da Torre de Hanói.

```
procedimento MoveTorre(N : natural; Orig, Dest, Aux : caracter)
início
  se N = 1 então
    MoveDisco(Orig, Dest)  senão
  início
    MoveTorre(N - 1, Orig, Aux, Dest)
    MoveDisco(Orig, Dest)
    MoveTorre(N - 1, Aux, Dest, Orig)
  fim
fim

procedimento MoveDisco(Orig, Dest : caracter)
início
  Escreva("Movimento: ", Orig, " -> ", Dest)
fim
```

### 1.18.1. Python



```
python

def MoveDisco(de, para):
    print(f"Movimento: {de} -> {para}")

def MoveTorre(n, origem, destino, auxiliar):
    if n == 1:
        MoveDisco(origem, destino)
    else:
        MoveTorre(n - 1, origem, auxiliar, destino)
        MoveDisco(origem, destino)
        MoveTorre(n - 1, auxiliar, destino, origem)

# Exemplo de como usar a função:
N = 3 # Número de discos
MoveTorre(N, 'A', 'C', 'B')
```

## 1.18.2. Node.js

```
javascript

function MoveDisco(de, para) {
    console.log("Movimento: " + de + " -> " + para);
}

function MoveTorre(n, origem, destino, auxiliar) {
    if (n === 1) {
        MoveDisco(origem, destino);
    } else {
        MoveTorre(n - 1, origem, auxiliar, destino);
        MoveDisco(origem, destino);
        MoveTorre(n - 1, auxiliar, destino, origem);
    }
}

// Exemplo de como usar a função:
var N = 3; // Número de discos
MoveTorre(N, 'A', 'C', 'B');
```

### 1.18.3. Java

```
java                                         Copy code

public class TorreDeHanoi {

    public static void MoveDisco(char de, char para) {
        System.out.println("Movimento: " + de + " -> " + para);
    }

    public static void MoveTorre(int n, char origem, char destino, char auxiliar) {
        if (n == 1) {
            MoveDisco(origem, destino);
        } else {
            MoveTorre(n - 1, origem, auxiliar, destino);
            MoveDisco(origem, destino);
            MoveTorre(n - 1, auxiliar, destino, origem);
        }
    }

    public static void main(String[] args) {
        int N = 3; // Número de discos
        MoveTorre(N, 'A', 'C', 'B');
    }
}
```

## 1.19. Fila:

### 1.19.1. Python

```
Coliseum > HAL2001 > fila.py > ...
1  class Fila:
2      def __init__(self):
3          self.fila = []
4
5      def enfileirar(self, item):
6          self.fila.append(item)
7          print(f"Item {item} adicionado a fila.")
8
9      def desenfileirar(self):
10         if not self.vazia():
11             item_removido = self.fila.pop(0)
12             print(f"Item {item_removido} removido da fila.")
13         else:
14             print("Fila está vazia.")
15
16     def vazia(self):
17         return len(self.fila) == 0
18
19     def mostrar(self):
20         if (self.vazia()):
21             print("Fila Vazia.")
22         else:
23             print("Fila atual:", self.fila)
24
25 queue = Fila()
26 queue.mostrar()
27 queue.enfileirar(13)
28 queue.enfileirar(42)
29 queue.enfileirar(69)
30 queue.mostrar()
31 queue.desenfileirar()
32 queue.mostrar()
```

```
Fila Vazia.
Item 13 adicionado a fila.
Item 42 adicionado a fila.
Item 69 adicionado a fila.
Fila atual: [13, 42, 69]
Item 13 removido da fila.
Fila atual: [42, 69]
```

### 1.19.1.1. Método “pop” (Python)

Em Python, o método `pop` pode ser utilizado tanto em listas quanto em outras estruturas de dados semelhantes, como dequeues da biblioteca `collections`. O método remove um elemento em um índice específico e retorna esse elemento. Se o índice não for fornecido, ele remove e retorna o último elemento da lista. É importante notar que esse método modifica a lista original e lança um `IndexError` se a lista estiver vazia ou o índice fornecido estiver fora do alcance.

Exemplos:

```
python
Copy code

lista = [1, 2, 3, 4, 5]
item = lista.pop() # Remove e retorna o último elemento, '5' neste caso.
item2 = lista.pop(1) # Remove e retorna o elemento no índice 1, '2' neste c
```

## 1.19.2. Node.js

```
Coliseum > HAL2001 > js fila.js > ...
1  class Fila {
2      constructor() {
3          this.fila = [];
4      }
5
6      enfileirar(item) {
7          this.fila.push(item);
8          console.log(`Item ${item} adicionado à fila.`);
9      }
10
11     desenfileirar() {
12         if (!this.isEmpty()) {
13             const removedItem = this.fila.shift();
14             console.log(`Item ${removedItem} removido da fila.`);
15         } else {
16             console.log("Fila está vazia.");
17         }
18     }
19
20     isEmpty() {
21         return this.fila.length === 0;
22     }
23
24     mostrar() {
25         if (this.isEmpty()){
26             console.log("Fila Vazia");
27         }else{
28             console.log("Fila atual:", this.fila);
29         }
30     }
31 }
32 }
33
34 let queue = new Fila();
35 queue.mostrar();
36 queue.enfileirar(13);
37 queue.enfileirar(42);
38 queue.enfileirar(69);
39 queue.mostrar();
40 queue.desenfileirar();
41 queue.mostrar();
42 }
```

```
Fila Vazia
Item 13 adicionado à fila.
Item 42 adicionado à fila.
Item 69 adicionado à fila.
Fila atual: [ 13, 42, 69 ]
Item 13 removido da fila.
Fila atual: [ 42, 69 ]
```

### 1.19.2.1. Métodos “shift” X “pop” (Node.js)

**Usando ` .shift()` para remover o primeiro elemento:**

```
javascript

let array = [1, 2, 3];
let primeiroElemento = array.shift();
console.log(primeiroElemento); // Saída: 1
console.log(array); // Saída: [2, 3]
```

**Usando ` .pop()` para remover o último elemento:**

```
javascript

let array = [1, 2, 3];
let ultimoElemento = array.pop();
console.log(ultimoElemento); // Saída: 3
console.log(array); // Saída: [1, 2]
```

### 1.19.3. Java

```
Coliseum > HAL2001 > J Fila.java > ...
1 package Coliseum.HAL2001;
2
3 import java.util.ArrayList;
4
5 public class Fila {
6     private ArrayList<Integer> fila;
7
8     public Fila() {
9         this.fila = new ArrayList<>();
10    }
11
12    public void enfileirar(int item) {
13        this.fila.add(item);
14        System.out.println("Item " + item + " adicionado à fila.");
15    }
16
17    public void desenfileirar() {
18        if (!this.isEmpty()) {
19            int removedItem = this.fila.remove(0);
20            System.out.println("Item " + removedItem + " removido da fila.");
21        } else {
22            System.out.println("Fila está vazia.");
23        }
24    }
25
26    public boolean isEmpty() {
27        return this.fila.size() == 0;
28    }
29
30    public void mostrar() {
31        if (this.isEmpty()) {
32            System.out.println("Fila Vazia");
33        } else {
34            System.out.println("Fila atual: " + this.fila);
35        }
36    }
37
38    Run | Debug
39    public static void main(String[] args) {
40        Fila queue = new Fila();
41        queue.mostrar();
42        queue.enfileirar(13);
43        queue.enfileirar(42);
44        queue.enfileirar(69);
45        queue.mostrar();
46        queue.desenfileirar();
47        queue.mostrar();
48    }
}
```

```
Fila Vazia
Item 13 adicionado à fila.
Item 42 adicionado à fila.
Item 69 adicionado à fila.
Fila atual: [13, 42, 69]
Item 13 removido da fila.
Fila atual: [42, 69]
```

## 1.20. Pilha:

### 1.20.1. Python

```
Coliseum > HAL2001 > pilha.py > ...
 1  class Pilha:
 2      def __init__(self):
 3          self.pilha = []
 4
 5      def empilhar(self,elemento):
 6          self.pilha.append(elemento)
 7          print(f"Adicionado elemento: {elemento}")
 8
 9      def desempilhar(self):
10          if (self.isEmpty()):
11              print ("Pilha Vazia")
12          else:
13              removido = self.pilha.pop(-1)
14              print (f" Removido elemento: {removido}")
15
16      def isEmpty(self):
17          return len(self.pilha) == 0
18
19      def mostrar(self):
20          if (self.isEmpty()):
21              print("Pilha Vazia.")
22          else:
23              print(self.pilha)
24
25
26      stack = Pilha()
27      stack.mostrar()
28      stack.empilhar(66)
29      stack.empilhar(69)
30      stack.empilhar(42)
31      stack.empilhar(81)
32      stack.mostrar()
33      stack.desempilhar()
34      stack.mostrar()
```

```
Fila Vazia.
Adicionado elemento: 66
Adicionado elemento: 69
Adicionado elemento: 42
Adicionado elemento: 81
[66, 69, 42, 81]
| Removido elemento: 81
[66, 69, 42]
```

## 1.20.2. Node.js

```
Coliseum > HAL2001 > js pilha.js > ...
1  class Pilha {
2      constructor() {
3          this.pilha = [];
4      }
5
6      empilhar(elemento) {
7          this.pilha.push(elemento);
8          console.log(`Elemento ${elemento} adicionado à Pilha.`);
9      }
10
11     desempilhar() {
12         if (this.isEmpty()) {
13             console.log(`Pilha Vazia`);
14         } else {
15             const elementoRemovido = this.pilha.pop();
16             console.log(`Elemento ${elementoRemovido} removido da Pilha.`);
17         }
18     }
19
20     mostrar() {
21         if (this.isEmpty()) {
22             console.log(`Pilha Vazia`);
23         } else {
24             console.log(this.pilha);
25         }
26     }
27
28     isEmpty() {
29         return this.pilha.length === 0;
30     }
31 }
32
33 let stack = new Pilha();
34 stack.mostrar();
35 stack.empilhar(66);
36 stack.empilhar(69);
37 stack.empilhar(42);
38 stack.empilhar(81);
39 stack.mostrar();
40 stack.desempilhar();
41 stack.mostrar();
42
```

```
Pilha Vazia
Elemento 66 adicionado à Pilha.
Elemento 69 adicionado à Pilha.
Elemento 42 adicionado à Pilha.
Elemento 81 adicionado à Pilha.
[ 66, 69, 42, 81 ]
Elemento 81 removido da Pilha.
[ 66, 69, 42 ]
```

### 1.20.3. Java

```
Coliseum > HAL2001 > J Pilha.java > ...
1  package Coliseum.HAL2001;
2
3  import java.util.ArrayList;
4
5  public class Pilha {
6      private ArrayList<Integer> pilha;
7
8      public Pilha() {
9          this.pilha = new ArrayList<>();
10     }
11
12     public void empilhar(int num) {
13         this.pilha.add(num);
14         System.out.println("Empilhado elemento: " + num);
15     }
16
17     public void desempilhar() {
18         if (this.isEmpty()) {
19             System.out.println("Pilha Vazia");
20         } else {
21             int removido = this.pilha.remove(this.pilha.size() - 1);
22             System.out.println("Desempilhado elemento: " + removido);
23         }
24     }
25
26
27     public boolean isEmpty() {
28         return pilha.size() == 0;
29     }
30
31     public void mostrar() {
32         if (this.isEmpty()) {
33             System.out.println("Pilha Vazia");
34         } else {
35             System.out.println("Pilha atual: " + this.pilha);
36         }
37     }
38
39     Run | Debug
40     public static void main(String[] args) {
41         Pilha stack = new Pilha();
42         stack.mostrar();
43         stack.empilhar(13);
44         stack.empilhar(69);
45         stack.empilhar(42);
46         stack.empilhar(77);
47         stack.mostrar();
48         stack.desempilhar();
49         stack.mostrar();
50     }
51 }
```

```
Pilha Vazia
Empilhado elemento: 13
Empilhado elemento: 69
Empilhado elemento: 42
Empilhado elemento: 77
Pilha atual: [13, 69, 42, 77]
Desempilhado elemento: 77
Pilha atual: [13, 69, 42]
```

## 2. Desenvolvimento de Programa com Especificação

### 2.1. Biblioteca

Desenvolva um programa utilizando Java (versão 20.0.2) que gerencie uma biblioteca. O programa deve ser capaz de adicionar, listar e remover livros, com ou sem acesso a um banco de dados fictício. O livro deve ter título, autor e ISBN.

#### 2.1.1. Java

```
Final > J4V4 > Livro11 > J Livro.java > ...
1  package Final.J4V4.Livro11;
2
3  public class Livro {
4      private String titulo;
5      private String autor;
6      private String isbn;
7
8      public Livro(String titulo, String autor, String isbn) {
9          this.titulo = titulo;
10         this.autor = autor;
11         this.isbn = isbn;
12     }
13
14     public String getTitulo() {
15         return titulo;
16     }
17
18     public String getAutor() {
19         return autor;
20     }
21
22     public String getIsbn() {
23         return isbn;
24     }
25
26     @Override
27     public String toString() {
28         return "Título: " + titulo + ", Autor: " + autor + ", ISBN: " + isbn;
29     }
30 }
```

```
Final > J4V4 > Livro11 > Biblioteca.java > ...
1 package Final.J4V4.Livro11;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Biblioteca {
7     private List<Livro> livros;
8
9     public Biblioteca() {
10         this.livros = new ArrayList<>();
11     }
12
13     public void adicionarLivro(Livro livro) {
14         livros.add(livro);
15     }
16
17     public void removerLivro(String isbn) {
18         livros.removeIf(livro -> livro.getIsbn().equals(isbn));
19     }
20
21     public List<Livro> listarLivros() {
22         return new ArrayList<>(livros);
23     }
24 }
```

```
Final > J4V4 > Livro11 > Main.java > ...
1 package Final.J4V4.Livro11;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         Biblioteca biblioteca = new Biblioteca();
7
8         // Adicionar livros
9         biblioteca.adicionarLivro(new Livro("Java Programming", "James Gosling", "123456789"));
10        biblioteca.adicionarLivro(new Livro("Effective Java", "Joshua Bloch", "987654321"));
11
12        // Listar livros
13        System.out.println("Livros na Biblioteca:");
14        for (Livro livro : biblioteca.listarLivros()) {
15            System.out.println(livro);
16        }
17
18        // Remover um livro pelo ISBN
19        biblioteca.removerLivro("123456789");
20
21        // Listar livros novamente
22        System.out.println("\nLivros na Biblioteca após remocao:");
23        for (Livro livro : biblioteca.listarLivros()) {
24            System.out.println(livro);
25        }
26    }
}
```

```
Livros na Biblioteca:
Título: Java Programming, Autor: James Gosling, ISBN: 123456789
Título: Effective Java, Autor: Joshua Bloch, ISBN: 987654321
```

```
Livros na Biblioteca após remocao:
Título: Effective Java, Autor: Joshua Bloch, ISBN: 987654321
```

## 2.1.2. Node.js

```
Final > Node > JS 11Livro.js > ...
1  // Livros 1.1
2
3  class Livro {
4      constructor(titulo, autor, isbn) {
5          this.titulo = titulo;
6          this.autor = autor;
7          this.isbn = isbn;
8      }
9
10     toString() {
11         return `Título: ${this.titulo}, Autor: ${this.autor}, ISBN: ${this.isbn}`;
12     }
13 }
14
15 class Biblioteca {
16     constructor() {
17         this.livros = [];
18     }
19
20     adicionarLivro(livro) {
21         this.livros.push(livro);
22     }
23
24     removerLivro(isbn) {
25         this.livros = this.livros.filter(livro => livro.isbn !== isbn);
26     }
27
28     listarLivros() {
29         return this.livros.slice();
30     }
31 }
32
33 const biblioteca = new Biblioteca();
34
35 // Adicionar livros
36 biblioteca.adicionarLivro(new Livro("Java Programming", "James Gosling", "123456789"));
37 biblioteca.adicionarLivro(new Livro("Effective Java", "Joshua Bloch", "987654321"));
38
39 // Listar livros
40 console.log("Livros na Biblioteca:");
41 biblioteca.listarLivros().forEach(livro => console.log(livro.toString()));
42
43 // Remover um livro pelo ISBN
44 biblioteca.removerLivro("123456789");
45
46 // Listar livros novamente
47 console.log("\nLivros na Biblioteca após remoção:");
48 biblioteca.listarLivros().forEach(livro => console.log(livro.toString()));
```

### 2.1.3. Python

```
Final > Python > 11Livros.py > ...
1  #Livros 1.1
2
3  class Livro:
4      def __init__(self, titulo, autor, isbn):
5          self.titulo = titulo
6          self.autor = autor
7          self.isbn = isbn
8
9      def __str__():
10         return f"Titulo: {self.titulo}, Autor: {self.autor}, ISBN: {self.isbn}"
11
12 class Biblioteca:
13     def __init__():
14         self.livros = []
15
16     def adicionar_livro(self, livro):
17         self.livros.append(livro)
18
19     def remover_livro(self, isbn):
20         self.livros = [livro for livro in self.livros if livro.isbn != isbn]
21
22     def listar_livros(self):
23         return self.livros[:]
24
25 biblioteca = Biblioteca()
26
27 # Adicionar livros
28 biblioteca.adicionar_livro(Livro("Java Programming", "James Gosling", "123456789"))
29 biblioteca.adicionar_livro(Livro("Effective Java", "Joshua Bloch", "987654321"))
30
31 # Listar livros
32 print("Livros na Biblioteca:")
33 for livro in biblioteca.listar_livros():
34     print(str(livro))
35
36 # Remover um livro pelo ISBN
37 biblioteca.remover_livro("123456789")
38
39 # Listar livros novamente
40 print("\nLivros na Biblioteca apos remocao:")
41 for livro in biblioteca.listar_livros():
42     print(str(livro))
43
```

## 2.2. Cinema

Utilizando Python (versão 3.11.4), projete um programa que gerencie um sistema de vendas de ingressos para um cinema. O programa deve ser capaz de exibir filmes em cartaz, selecionar assentos e calcular o preço total. Você pode simular o acesso a um banco de dados se desejar.

### 2.2.1. Java

```
Final > J4V4 > Cinema21 > J Cinema.java > Cinema
1 package Final.J4V4.Cinema21;
2
3 import java.util.Scanner;
4
5 public class Cinema {
6
7     static class Filme {
8         String nome;
9         double preco;
10
11         Filme(String nome, double preco) {
12             this.nome = nome;
13             this.preco = preco;
14         }
15     }
16
17     static class Sala {
18         String nome;
19         String[][] assentos;
20
21         Sala(String nome, int rows, int cols) {
22             this.nome = nome;
23             this.assentos = new String[rows][cols];
24             for (int i = 0; i < rows; i++) {
25                 for (int j = 0; j < cols; j++) {
26                     assentos[i][j] = "Livre";
27                 }
28             }
29         }
30     }
}
```

```
Run | Debug
32  public static void main(String[] args) {
33      Scanner scanner = new Scanner(System.in);
34      Filme[] filmes = { new Filme("Movie 1", 10), new Filme("Movie 2", 12) };
35      Sala[] salas = { new Sala("Sala 1", 5, 10), new Sala("Sala 2", 5, 10) };
36
37      System.out.println("Filmes em Cartaz:");
38      for (int i = 0; i < filmes.length; i++) {
39          System.out.println((i + 1) + ". " + filmes[i].nome + " - $" + filmes[i].preco);
40      }
41
42      System.out.print("Selecione o filme: ");
43      int filmeSelecionado = scanner.nextInt() - 1;
44
45      System.out.println("Salas Disponíveis:");
46      for (int i = 0; i < salas.length; i++) {
47          System.out.println((i + 1) + ". " + salas[i].nome);
48      }
49
50      System.out.print("Selecione a sala: ");
51      int salaSelecionada = scanner.nextInt() - 1;
52
53      Sala sala = salas[salaSelecionada];
54      System.out.println("Assentos:");
55      for (int i = 0; i < sala.assentos.length; i++) {
56          for (int j = 0; j < sala.assentos[0].length; j++) {
57              System.out.print(sala.assentos[i][j] + " | ");
58          }
59          System.out.println();
60      }
61
62      System.out.print("Selecione a fileira: ");
63      int row = scanner.nextInt() - 1;
64      System.out.print("Selecione a coluna: ");
65      int col = scanner.nextInt() - 1;
66
67      if (sala.assentos[row][col].equals("Livre")) {
68          sala.assentos[row][col] = "Ocupado";
69          double precoTotal = filmes[filmeSelecionado].preco;
70          System.out.println("Ingresso reservado. Preço Total: $" + precoTotal);
71      } else {
72          System.out.println("Assento já ocupado!");
73      }
74
75      scanner.close();
76  }
77 }
```

## 2.2.2. Node.js

```
Final > Node > js 21Cinema.js > ...
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const filmes = [
9    { nome: 'Movie 1', preco: 10 },
10   { nome: 'Movie 2', preco: 12 }
11 ];
12
13 const salas = [
14   { nome: 'Sala 1', assentos: Array(5).fill(Array(10).fill('Livre')) },
15   { nome: 'Sala 2', assentos: Array(5).fill(Array(10).fill('Livre')) }
16 ];
17
18 console.log('Filmes em Cartaz:');
19 filmes.forEach((filme, index) => {
20   console.log(` ${index + 1}. ${filme.nome} - ${filme.preco}`);
21 });
22
23 rl.question('Selecione o filme: ', filmeSelecionado => {
24   const filme = filmes[filmeSelecionado - 1];
25
26   console.log('Salas Disponíveis:');
27   salas.forEach((sala, index) => {
28     console.log(` ${index + 1}. ${sala.nome}`);
29   });
30
31   rl.question('Selecione a sala: ', salaSelecionada => {
32     const sala = salas[salaSelecionada - 1];
33
34     console.log('Assentos:');
35     sala.assentos.forEach((row, rowIndex) => {
36       console.log(row.map(seat => seat).join(' | '));
37     });
38   });
39 });
40
41 rl.on('close', () => process.exit());
```

```
38
39     rl.question('Selecione a fileira: ', row => {
40         rl.question('Selecione a coluna: ', col => {
41             const selectedRow = sala.assentos[row - 1];
42             if (selectedRow[col - 1] === 'Livre') {
43                 selectedRow[col - 1] = 'Ocupado';
44                 console.log(`Ingresso reservado. Preço Total: ${filme.preco}`);
45             } else {
46                 console.log('Assento já ocupado!');
47             }
48
49             rl.close();
50         });
51     });
52 });
53 })
```

### 2.2.3. Python

```
Final > Python > 21Cinema.py > Cinema
1  # Cinema 2.1
2  class Cinema:
3
4      def __init__(self):
5          self.filmes = [
6              {"nome": "Movie 1", "preco": 10},
7              {"nome": "Movie 2", "preco": 12},
8              {"nome": "Movie 3", "preco": 15}
9          ]
10         self.salas = [{"nome": "Sala 1", "assentos": self.create_seats()},
11                         {"nome": "Sala 2", "assentos": self.create_seats()}]
12
13     def create_seats(self):
14         return [[["Livre" for _ in range(10)] for _ in range(5)]]
15
16     def exibir_filmes(self):
17         print("Filmes em Cartaz:")
18         for idx, filme in enumerate(self.filmes):
19             print(f"{idx + 1}. {filme['nome']} - ${filme['preco']}")
20
21     def selecionar_assentos(self, sala_idx, row, col):
22         sala = self.salas[sala_idx]
23         if sala["assentos"][row][col] == "Livre":
24             sala["assentos"][row][col] = "Ocupado"
25             return True
26         else:
27             return False
28
29     def calcular_preco(self, filme_idx, quantidade):
30         return self.filmes[filme_idx]["preco"] * quantidade
31
32     def exibir_salas(self):
33         print("Salas Disponíveis:")
34         for idx, sala in enumerate(self.salas):
35             print(f"{idx + 1}. {sala['nome']}")
```

```
36
37     def exibir_assentos(self, sala_idx):
38         print("Assentos:")
39         for row in self.salas[sala_idx]["assentos"]:
40             print(" | ".join(row))
41
```

```
43 # Simulação
44 cinema = Cinema()
45 cinema.exibir_filmes()
46 filme_selecionado = int(input("Selecione o filme: ")) - 1
47 cinema.exibir_salas()
48 sala_selecionada = int(input("Selecione a sala: ")) - 1
49 cinema.exibir_assentos(sala_selecionada)
50 row = int(input("Selecione a fileira: "))
51 col = int(input("Selecione a coluna: "))
52 if cinema.selecionar_assentos(sala_selecionada, row, col):
53     preco_total = cinema.calcular_preco(filme_selecionado, 1)
54     print(f"Ingresso reservado. Preço Total: ${preco_total}")
55 else:
56     print("Assento já ocupado!")
```

## 2.3. Agenda

Utilizando Java (versão 20.0.2), crie um programa que simule uma agenda de contatos. A aplicação deve permitir adicionar, buscar, atualizar e excluir contatos. Cada contato deve ter nome, telefone e e-mail. Você pode optar por simular o acesso a um banco de dados.

### 2.3.1. Java

```
Final > J4V4 > Agenda31 > J ContactBook.java > ContactBook
1  package Final.J4V4.Agenda31;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class ContactBook {
7      private List<Contact> contacts;
8
9      public ContactBook() {
10         this.contacts = new ArrayList<>();
11     }
12
13     public void addContact(Contact contact) {
14         contacts.add(contact);
15     }
16
17     public Contact findContactByName(String name) {
18         for (Contact contact : contacts) {
19             if (contact.getName().equals(name)) {
20                 return contact;
21             }
22         }
23         return null;
24     }
25
26     public void updateContact(String name, Contact newContact) {
27         for (int i = 0; i < contacts.size(); i++) {
28             if (contacts.get(i).getName().equals(name)) {
29                 contacts.set(i, newContact);
30                 return;
31             }
32         }
33     }
34
35     public void deleteContact(String name) {
36         contacts.removeIf(contact -> contact.getName().equals(name));
37     }
38
39     public void listContacts() {
40         for (Contact contact : contacts) {
41             System.out.println(contact);
42         }
43     }
44 }
```

```

Final > J4V4 > Agenda31 > Contact.java > ...
1 package Final.J4V4.Agenda31;
2
3 public class Contact {
4     private String name;
5     private String phone;
6     private String email;
7
8     public Contact(String name, String phone, String email) {
9         this.name = name;
10        this.phone = phone;
11        this.email = email;
12    }
13
14    public String getName() {
15        return name;
16    }
17
18    public void setName(String name) {
19        this.name = name;
20    }
21
22    public String getPhone() {
23        return phone;
24    }
25
26    public void setPhone(String phone) {
27        this.phone = phone;
28    }
29
30    public String getEmail() {
31        return email;
32    }
33
34    public void setEmail(String email) {
35        this.email = email;
36    }
37
38    @Override
39    public String toString() {
40        return "Contact [name=" + name + ", phone=" + phone + ", email=" + email + "]";
41    }
42 }

```

```

Final > J4V4 > Agenda31 > Main.java > ...
1 package Final.J4V4.Agenda31;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         ContactBook contactBook = new ContactBook();
7
8         contactBook.addContact(new Contact("Alice", "123456789", "alice@email.com"));
9         contactBook.addContact(new Contact("Bob", "987654321", "bob@email.com"));
10
11         contactBook.listContacts(); // Listar todos os contatos
12
13         Contact alice = contactBook.findContactByName("Alice"); // Buscar contato por nome
14
15         contactBook.updateContact("Alice", new Contact("Alice", "111222333", "alice@newemail.com")); // Atualizar
16
17         contactBook.deleteContact("Bob"); // Excluir contato
18
19         contactBook.listContacts(); // Listar todos os contatos após as operações
20     }
21 }

```

```

Contact [name=Alice, phone=123456789, email=alice@email.com]
Contact [name=Bob, phone=987654321, email=bob@email.com]
Contact [name=Alice, phone=111222333, email=alice@newemail.com]

```

### 2.3.2. Node.js

```
Final > Node > js 31Contatojs > ...
1  // 3.1 - AGENDA
2
3  class Contact {
4      constructor(name, phone, email) {
5          this.name = name;
6          this.phone = phone;
7          this.email = email;
8      }
9  }
```

```
11 class ContactManager {
12     constructor() {
13         this.contacts = [];
14     }
15
16     addContact(contact) {
17         this.contacts.push(contact);
18     }
19
20     findContactByName(name) {
21         return this.contacts.find(contact => contact.name === name);
22     }
23
24     updateContact(name, newContact) {
25         const index = this.contacts.findIndex(contact => contact.name === name);
26         if (index !== -1) {
27             this.contacts[index] = newContact;
28         }
29     }
30
31     deleteContact(name) {
32         const index = this.contacts.findIndex(contact => contact.name === name);
33         if (index !== -1) {
34             this.contacts.splice(index, 1);
35         }
36     }
37
38     printContacts() {
39         console.log("Contatos:");
40         this.contacts.forEach(contact => console.log(`Nome: ${contact.name}, Telefone: ${contact.phone}, Email: ${contact.email}`));
41     }
42 }
```

```
44 const contact1 = new Contact("Alice", "123-456", "alice@example.com");
45 const contact2 = new Contact("Bob", "789-123", "bob@example.com");
46
47 const manager = new ContactManager();
48 manager.addContact(contact1);
49 manager.addContact(contact2);
50 manager.printContacts();
51
52 const alice = manager.findContactByName("Alice");
53 console.log("Encontrou:", alice);
54
55 manager.updateContact("Alice", new Contact("Alice", "111-222", "alice@new.com"));
56 manager.printContacts();
57
58 manager.deleteContact("Alice");
59 manager.printContacts();
60
```

```
Contatos:  
Nome: Alice, Telefone: 123-456, Email: alice@example.com  
Nome: Bob, Telefone: 789-123, Email: bob@example.com  
Encontrou: Contact { name: 'Alice', phone: '123-456', email: 'alice@example.com' }  
Contatos:  
Nome: Alice, Telefone: 111-222, Email: alice@new.com  
Nome: Bob, Telefone: 789-123, Email: bob@example.com  
Contatos:  
Nome: Bob, Telefone: 789-123, Email: bob@example.com
```

### 2.3.3. Python

```
Final > Python > 31Agenda.py > ...
1  # 3.1 - AGENDA
2  class Contact:
3      def __init__(self, name, phone, email):
4          self.name = name
5          self.phone = phone
6          self.email = email
7
8  class ContactManager:
9      def __init__(self):
10         self.contacts = []
11
12     def add_contact(self, contact):
13         self.contacts.append(contact)
14
15     def find_contact_by_name(self, name):
16         return next((contact for contact in self.contacts if contact.name == name), None)
17
18     def update_contact(self, name, new_contact):
19         index = next((i for i, contact in enumerate(self.contacts) if contact.name == name), None)
20         if index is not None:
21             self.contacts[index] = new_contact
22
23     def delete_contact(self, name):
24         index = next((i for i, contact in enumerate(self.contacts) if contact.name == name), None)
25         if index is not None:
26             self.contacts.pop(index)
27
28     def print_contacts(self):
29         print("Contatos:")
30         for contact in self.contacts:
31             print(f"Nome: {contact.name}, Telefone: {contact.phone}, Email: {contact.email}")
32
33 contact1 = Contact("Alice", "123-456", "alice@example.com")
34 contact2 = Contact("Bob", "789-123", "bob@example.com")
35
36 manager = ContactManager()
37 manager.add_contact(contact1)
38 manager.add_contact(contact2)
39 manager.print_contacts()
40
41 alice = manager.find_contact_by_name("Alice")
42 print("Encontrou:", alice.name, alice.phone, alice.email)
43
44 manager.update_contact("Alice", Contact("Alice", "111-222", "alice@new.com"))
45 manager.print_contacts()
46
47 manager.delete_contact("Alice")
48 manager.print_contacts()
```

```
Contatos:
Nome: Alice, Telefone: 123-456, Email: alice@example.com
Nome: Bob, Telefone: 789-123, Email: bob@example.com
Encontrou: Alice 123-456 alice@example.com
Contatos:
Nome: Alice, Telefone: 111-222, Email: alice@new.com
Nome: Bob, Telefone: 789-123, Email: bob@example.com
Contatos:
Nome: Bob, Telefone: 789-123, Email: bob@example.com
```

## 2.4. Gerenciador de Tarefas

Utilizando Node Javascript (versão 18.17.0), desenvolva um programa que simule um gerenciador de tarefas. Deve permitir adicionar, listar e excluir tarefas, podendo optar por simular o acesso a um banco de dados.

### 2.4.1. Java

```
Final > J4V4 > TaskManager41 > TaskManager.java > TaskManager
1 package Final.J4V4.TaskManager41;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class TaskManager {
7     private final Map<Integer, String> tasks; // Simulação de um banco de dados em memória
8
9     public TaskManager() {
10         this.tasks = new HashMap<>();
11     }
12
13     public void addTask(int id, String description) {
14         this.tasks.put(id, description);
15         System.out.println("Task added: " + description);
16     }
17
18     public void listTasks() {
19         System.out.println("Listing tasks:");
20         for (Map.Entry<Integer, String> entry : this.tasks.entrySet()) {
21             System.out.println("ID: " + entry.getKey() + ", Description: " + entry.getValue());
22         }
23     }
24
25     public void deleteTask(int id) {
26         if (this.tasks.containsKey(id)) {
27             this.tasks.remove(id);
28             System.out.println("Task with ID " + id + " deleted");
29         } else {
30             System.out.println("Task not found");
31         }
32     }
33
34 // Exemplo de uso
35 public static void main(String[] args) {
36     TaskManager taskManager = new TaskManager();
37     taskManager.addTask(1, "Task 1");
38     taskManager.addTask(2, "Task 2");
39     taskManager.listTasks();
40     taskManager.deleteTask(1);
41     taskManager.listTasks();
42 }
43
44 }
```

```
Task added: Task 1
Task added: Task 2
Listing tasks:
ID: 1, Description: Task 1
ID: 2, Description: Task 2
Task with ID 1 deleted
Listing tasks:
ID: 2, Description: Task 2
```

## 2.4.2. Node.js

```
Final > Node > JS 41TaskManager.js > ...
1  // 4.1 TaskManager
2
3  class TaskManager {
4      constructor() {
5          this.tasks = {}; // Simulação de um banco de dados em memória
6      }
7
8      addTask(id, description) {
9          this.tasks[id] = description;
10         console.log(`Task added: ${description}`);
11     }
12
13     listTasks() {
14         console.log("Listing tasks:");
15         for (const [id, description] of Object.entries(this.tasks)) {
16             console.log(`ID: ${id}, Description: ${description}`);
17         }
18     }
19
20     deleteTask(id) {
21         if (this.tasks[id]) {
22             delete this.tasks[id];
23             console.log(`Task with ID ${id} deleted`);
24         } else {
25             console.log("Task not found");
26         }
27     }
28 }
29
30 // Exemplo de uso
31 const taskManager = new TaskManager();
32 taskManager.addTask(1, "Task 1");
33 taskManager.addTask(2, "Task 2");
34 taskManager.listTasks();
35 taskManager.deleteTask(1);
36 taskManager.listTasks();
37
```

```
Task added: Task 1
Task added: Task 2
Listing tasks:
ID: 1, Description: Task 1
ID: 2, Description: Task 2
Task with ID 1 deleted
Listing tasks:
ID: 2, Description: Task 2
```

### 2.4.3. Python

```
Final > Python > 41TaskManager.py > ...
1  class TaskManager:
2      def __init__(self):
3          self.tasks = {} # Simulação de um banco de dados em memória
4
5      def add_task(self, id, description):
6          self.tasks[id] = description
7          print(f"Task added: {description}")
8
9      def list_tasks(self):
10         print("Listing tasks:")
11         for id, description in self.tasks.items():
12             print(f"ID: {id}, Description: {description}")
13
14     def delete_task(self, id):
15         if id in self.tasks:
16             del self.tasks[id]
17             print(f"Task with ID {id} deleted")
18         else:
19             print("Task not found")
20
21
22 # Exemplo de uso
23 if __name__ == "__main__":
24     task_manager = TaskManager()
25     task_manager.add_task(1, "Task 1")
26     task_manager.add_task(2, "Task 2")
27     task_manager.list_tasks()
28     task_manager.delete_task(1)
29     task_manager.list_tasks()
```

```
Task added: Task 1
Task added: Task 2
Listing tasks:
ID: 1, Description: Task 1
ID: 2, Description: Task 2
Task with ID 1 deleted
Listing tasks:
ID: 2, Description: Task 2
```

## 2.5. Vendas Online

Usando Java (versão 20.0.2), desenvolva um programa que simule um sistema de vendas para uma loja online, incluindo cálculo de descontos e total da compra. Opção por simular ou não o acesso a banco de dados fica a critério do candidato.

### 2.5.1. Java

```
Final > J4V4 > teste > LojaOnline.java > ...
1 package Final.J4V4.teste;
2
3 // QUESTÃO 5.1
4
5 public class LojaOnline {
6     Run | Debug
7     public static void main(String[] args) {
8         Produto produto1 = new Produto("Camiseta", 25.99, 10);
9         Produto produto2 = new Produto("Calça", 50.99, 5);
10
11         CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
12         carrinho.adicionarProduto(produto1, 3);
13         carrinho.adicionarProduto(produto2, 2);
14
15         System.out.println("Total: " + String.format("%.2f", carrinho.calcularTotal()));
16         System.out.println("Desconto: " + carrinho.calcularDesconto());
17         System.out.println("Total com desconto: " + carrinho.calcularTotalComDesconto());
18     }
}
```

```
Final > J4V4 > teste > Produto.java > ...
1 package Final.J4V4.teste;
2
3 public class Produto {
4     private String nome;
5     private double preco;
6     private int quantidade;
7
8     public Produto(String nome, double preco, int quantidade) {
9         this.nome = nome;
10        this.preco = preco;
11        this.quantidade = quantidade;
12    }
13
14    public double getPreco() {
15        return preco;
16    }
17
18    public int getQuantidade() {
19        return quantidade;
20    }
21 }
```

```
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class CarrinhoDeCompras {
7     private List<Produto> produtos;
8
9     public CarrinhoDeCompras() {
10         this.produtos = new ArrayList<>();
11     }
12
13     public void adicionarProduto(Produto produto, int quantidade) {
14         for (int i = 0; i < quantidade; i++) {
15             produtos.add(produto);
16         }
17     }
18
19     public double calcularTotal() {
20         double total = 0;
21         for (Produto produto : produtos) {
22             total += produto.getPreco();
23         }
24
25         return total;
26     }
27
28     public double calcularDesconto() {
29         // Desconto de 10% se o total for maior que 100
30         if (calcularTotal() > 100) {
31             return calcularTotal() * 0.1;
32         }
33         return 0;
34     }
35
36     public double calcularTotalComDesconto() {
37         return calcularTotal() - calcularDesconto();
38     }
39 }
```

```
Total: 179,95
Desconto: 17.995
Total com desconto: 161.955
```

## 2.5.2. Node.js

```
Final > Node > JS 51Compras.js > ...
1   class Produto {
2       constructor(nome, preco, quantidade) {
3           this.nome = nome;
4           this.preco = preco;
5           this.quantidade = quantidade;
6       }
7
8       getPreco() {
9           return this.preco;
10      }
11
12      getQuantidade() {
13          return this.quantidade;
14      }
15  }

18  class CarrinhoDeCompras {
19      constructor() {
20          this.produtos = [];
21      }
22
23      adicionarProduto(produto, quantidade) {
24          for (let i = 0; i < quantidade; i++) {
25              this.produtos.push(produto);
26          }
27      }
28
29      calcularTotal() {
30          return this.produtos.reduce((total, produto) => total + produto.getPreco(), 0);
31      }
32
33      calcularDesconto() {
34          return this.calcularTotal() > 100 ? this.calcularTotal() * 0.1 : 0;
35      }
36
37      calcularTotalComDesconto() {
38          return this.calcularTotal() - this.calcularDesconto();
39      }
40  }

43  const produto1 = new Produto('Camiseta', 25.99, 10);
44  const produto2 = new Produto('Calça', 50.99, 5);
45
46  const carrinho = new CarrinhoDeCompras();
47  carrinho.adicionarProduto(produto1, 3);
48  carrinho.adicionarProduto(produto2, 2);
49
50  console.log("Total: " + carrinho.calcularTotal().toFixed(2));
51  console.log('Desconto: ' + carrinho.calcularDesconto());
52  console.log('Total com desconto: ' + carrinho.calcularTotalComDesconto());
```

### 2.5.3. Python

```
Final > Python > 51Compras.py > ...
1  class Produto:
2      def __init__(self, nome, preco, quantidade):
3          self.nome = nome
4          self.preco = preco
5          self.quantidade = quantidade
6
7      def get_preco(self):
8          return self.preco
9
10     def get_quantidade(self):
11         return self.quantidade
12
13 class CarrinhoDeCompras:
14     def __init__(self):
15         self.produtos = []
16
17     def adicionar_produto(self, produto, quantidade):
18         for _ in range(quantidade):
19             self.produtos.append(produto)
20
21     def calcular_total(self):
22         return sum(produto.get_preco() for produto in self.produtos)
23
24     def calcular_desconto(self):
25         return self.calcular_total() * 0.1 if self.calcular_total() > 100 else 0
26
27     def calcular_total_com_desconto(self):
28         return self.calcular_total() - self.calcular_desconto()
29
30
31 produto1 = Produto('Camiseta', 25.99, 10)
32 produto2 = Produto('Calça', 50.99, 5)
33
34 carrinho = CarrinhoDeCompras()
35 carrinho.adicionar_produto(produto1, 3)
36 carrinho.adicionar_produto(produto2, 2)
37
38 print('Total:', round(carrinho.calcular_total(), 2))
39 print('Desconto:', carrinho.calcular_desconto())
40 print('Total com desconto:', carrinho.calcular_total_com_desconto())
```

### 3. Apresentação de Dados Calculados

#### 3.1. Estatísticas Alunos

Utilizando Python (versão 3.11.4), escreva um programa que calcule e apresente as estatísticas de uma turma de estudantes. O usuário deve inserir as notas e o programa deve calcular a média, a mediana e o desvio padrão.

##### 3.1.1. Java

```
Final > J4V4 > Alunos12 > EstatisticasDaTurma.java > EstatisticasDaTurma
1 package Final.J4V4.Alunos12;
2
3 import java.util.Arrays;
4 import java.util.Scanner;
5
6 public class EstatisticasDaTurma {
7
8     public static double calcularMedia(double[] notas) {
9         double soma = 0;
10        for (double nota : notas) {
11            soma += nota;
12        }
13        return soma / notas.length;
14    }
15
16    public static double calcularMediana(double[] notas) {
17        Arrays.sort(notas);
18        int meio = notas.length / 2;
19        if (notas.length % 2 == 0) {
20            return (notas[meio - 1] + notas[meio]) / 2;
21        } else {
22            return notas[meio];
23        }
24    }
25
26    public static double calcularDesvioPadrao(double[] notas, double media) {
27        double somaDasDiferenciasQuadradas = 0;
28        for (double nota : notas) {
29            somaDasDiferenciasQuadradas += Math.pow(nota - media, 2);
30        }
31        double variancia = somaDasDiferenciasQuadradas / notas.length;
32        return Math.sqrt(variancia);
33    }
}
```

```
Run | Debug
35  public static void main(String[] args) {
36      Scanner scanner = new Scanner(System.in);
37
38      System.out.print("Insira o número de alunos na turma: ");
39      int numeroDeAlunos = scanner.nextInt();
40      double[] notas = new double[numeroDeAlunos];
41
42      for (int i = 0; i < numeroDeAlunos; i++) {
43          System.out.print("Insira a nota do aluno " + (i + 1) + ": ");
44          notas[i] = scanner.nextDouble();
45      }
46
47      double media = calcularMedia(notas);
48      double mediana = calcularMediana(notas);
49      double desvioPadrao = calcularDesvioPadrao(notas, media);
50
51      System.out.println("Média da turma: " + String.format("%.2f", media));
52      System.out.println("Mediana da turma: " + String.format("%.2f", mediana));
53      System.out.println("Desvio Padrao da turma: " + String.format("%.2f", desvioPadrao));
54
55      scanner.close();
56  }
57 }
```

```
Insira o número de alunos na turma: 3
Insira a nota do aluno 1: 0
Insira a nota do aluno 2: 5
Insira a nota do aluno 3: 10
Média da turma: 5,00
Mediana da turma: 5,00
Desvio Padrao da turma: 4,08
```

### 3.1.2. Node.js

```
Final > Node > js 12Alunos.js > ...
1
2  function calcularMedia(notas) {
3    |  return notas.reduce((a, b) => a + b) / notas.length;
4  }
5
6  function calcularMediana(notas) {
7    |  notas.sort((a, b) => a - b);
8    |  const meio = Math.floor(notas.length / 2);
9    |  if (notas.length % 2 === 0) {
10    |    return (notas[meio - 1] + notas[meio]) / 2;
11    |  } else {
12    |    return notas[meio];
13    |  }
14  }
15
16  function calcularDesvioPadrao(notas, media) {
17    |  const somaDasDiferenciasQuadradas = notas.reduce((a, b) => a + Math.pow(b - media, 2), 0);
18    |  const variancia = somaDasDiferenciasQuadradas / notas.length;
19    |  return Math.sqrt(variancia);
20  }

21
22  const readlineSync = require('readline-sync');
23
24  function estatisticasTurma() {
25    |  const numAlunos = parseInt(readlineSync.question('Insira o numero de alunos na turma: '));
26    |  const notas = [];
27
28    |  for (let index = 0; index < numAlunos; index++) {
29    |    |  const nota = parseFloat(readlineSync.question('Insira a nota do aluno: '));
30    |    |  notas.push(nota);
31    |  }
32
33    |  const media = calcularMedia(notas);
34    |  const mediana = calcularMediana(notas);
35    |  const desvioPadrao = calcularDesvioPadrao(notas, media);
36
37    |  console.log(`Média da turma: ${media.toFixed(2)} `);
38    |  console.log(` Mediana da turma: ${mediana.toFixed(2)} `);
39    |  console.log(` Desvio Padrão da turma: ${desvioPadrao.toFixed(2)} `);
40  }
41
42  estatisticasTurma();
```

### 3.1.3. Python

```
Final > Python > 12Alunos.py > ...
1  #Alunos 1.2
2
3  def calcular_media(notas):
4      return sum(notas) / len(notas)
5
6  def calcular_mediana(notas):
7      notas_ordenadas = sorted(notas)
8      meio = len(notas) // 2
9      if len(notas) % 2 == 0:
10          return (notas_ordenadas[meio - 1] + notas_ordenadas[meio]) / 2
11      else:
12          return notas_ordenadas[meio]
13
14 def calcular_desvio_padrao(notas, media):
15     soma_das_diferencias_quadradas = sum((x - media) ** 2 for x in notas)
16     variancia = soma_das_diferencias_quadradas / len(notas)
17     return variancia ** 0.5
```

```
19 def estatisticas_turma():
20     notas = [float(input("Insira a nota do aluno: ")) for _ in range(int(input("Insira o número de alunos na turma: ")))]
21
22     media = calcular_media(notas)
23     mediana = calcular_mediana(notas)
24     desvio_padrao = calcular_desvio_padrao(notas, media)
25
26     print(f"Média da turma: {media:.2f}")
27     print(f"Mediana da turma: {mediana:.2f}")
28     print(f"Desvio Padrão da turma: {desvio_padrao:.2f}")
29
30 if __name__ == "__main__":
31     estatisticas_turma()
```

## 3.2. GPS

Desenvolva um programa em Java (versão 20.0.2) que receba as coordenadas de diversos pontos geográficos e calcule a distância total percorrida. O programa deve apresentar a distância entre cada par de pontos consecutivos e a distância total.

### 3.2.1. Java

```
Final > J4V4 > DistanciaGeografica22 > J DistanciaGeografica.java > ...
1  // Distância 2.2
2  package Final.J4V4.DistanciaGeografica22;
3
4  import java.util.Scanner;
5  import java.util.ArrayList;
6
7  class Ponto {
8      double x;
9      double y;
10
11     Ponto(double x, double y) {
12         this.x = x;
13         this.y = y;
14     }
15 }
16
17 public class DistanciaGeografica {
18     public static double calcularDistancia(Ponto a, Ponto b) {
19         return Math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y, 2));
20     }
21
22     Run | Debug
23     public static void main(String[] args) {
24         Scanner scanner = new Scanner(System.in);
25         ArrayList<Ponto> pontos = new ArrayList<>();
26
27         System.out.print("Informe o número de pontos: ");
28         int numeroDePontos = scanner.nextInt();
29
30         for (int i = 0; i < numeroDePontos; i++) {
31             System.out.print("Informe a coordenada X do ponto " + (i + 1) + ": ");
32             double x = scanner.nextDouble();
33             System.out.print("Informe a coordenada Y do ponto " + (i + 1) + ": ");
34             double y = scanner.nextDouble();
35             pontos.add(new Ponto(x, y));
36         }
37
38         double distanciaTotal = 0;
39         for (int i = 0; i < pontos.size() - 1; i++) {
40             Ponto pontoAtual = pontos.get(i);
41             Ponto proximoPonto = pontos.get(i + 1);
42             double distancia = calcularDistancia(pontoAtual, proximoPonto);
43             System.out.println("Distância do ponto " + (i + 1) + " ao ponto " + (i + 2) + ": " + distancia);
44             distanciaTotal += distancia;
45         }
46
47         System.out.println("Distância total percorrida: " + distanciaTotal);
48     }
49 }
```

### 3.2.2. Node.js

```
Final > Node > js 22Distancia.js > ...
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  class Ponto {
9      constructor(x, y) {
10         this.x = x;
11         this.y = y;
12     }
13 }
14
15 function calcularDistancia(a, b) {
16     return Math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y, 2));
17 }
18
19 const pontos = [];
20 let numeroDePontos;
21
22 const questoes = [
23     () => rl.question(`Informe o número de pontos: `, resposta => {
24         numeroDePontos = parseInt(resposta);
25         questoes[1](0);
26     }),
27     (i) => {
28         if (i < numeroDePontos) {
29             rl.question(`Informe a coordenada X do ponto ${i + 1}: `, x => {
30                 rl.question(`Informe a coordenada Y do ponto ${i + 1}: `, y => {
31                     pontos.push(new Ponto(parseFloat(x), parseFloat(y)));
32                     questoes[1](i + 1);
33                 });
34             });
35         } else {
36             questoes[2]();
37         }
38     },
39     () => {
40         let distanciaTotal = 0;
41         for (let i = 0; i < pontos.length - 1; i++) {
42             const distancia = calcularDistancia(pontos[i], pontos[i + 1]);
43             console.log(`Distância do ponto ${i + 1} ao ponto ${i + 2}: ${distancia}`);
44             distanciaTotal += distancia;
45         }
46
47         console.log(`Distância total percorrida: ${distanciaTotal}`);
48         rl.close();
49     }
50 ];
51
52 questoes[0]();
```

### 3.2.3. Python

```
Final > Python > 22Distancia.py > ...
1  #Distancia 2.2
2  import math
3
4  class Ponto:
5      def __init__(self, x, y):
6          self.x = x
7          self.y = y
8
9  def calcular_distancia(a, b):
10     return math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2)
11
12 numero_de_pontos = int(input("Informe o número de pontos: "))
13 pontos = []
14
15 for i in range(numero_de_pontos):
16     x = float(input(f"Informe a coordenada X do ponto {i + 1}: "))
17     y = float(input(f"Informe a coordenada Y do ponto {i + 1}: "))
18     pontos.append(Ponto(x, y))
19
20 distancia_total = 0
21 for i in range(len(pontos) - 1):
22     distancia = calcular_distancia(pontos[i], pontos[i + 1])
23     print(f"Distância do ponto {i + 1} ao ponto {i + 2}: {distancia}")
24     distancia_total += distancia
25
26 print(f"Distância total percorrida: {distancia_total}")
```

### 3.3. Meteorologia

Escreva um programa em Python (versão 3.11.4) que receba dados meteorológicos (temperatura, umidade, pressão) e calcule a média, mínima e máxima para um período fornecido. Apresente os dados calculados.

#### 3.3.1. Java

```
Final > J4V4 > Metereologia32 > WeatherData.java > WeatherData
 1 package Final.J4V4.Metereologia32;
 2
 3 public class WeatherData {
 4     private double[] temperatures;
 5     private double[] humidities;
 6     private double[] pressures;
 7
 8     public WeatherData(double[] temperatures, double[] humidities, double[] pressures) {
 9         this.temperatures = temperatures;
10         this.humidities = humidities;
11         this.pressures = pressures;
12     }
13
14     private void calculateStatistics(double[] data) {
15         double average = 0;
16         double minimum = data[0];
17         double maximum = data[0];
18
19         for (double value : data) {
20             average += value;
21             if (value < minimum)
22                 minimum = value;
23             if (value > maximum)
24                 maximum = value;
25         }
26
27         average /= data.length;
28
29         System.out.println("Média: " + average);
30         System.out.println("Mínima: " + minimum);
31         System.out.println("Máxima: " + maximum);
32     }
33
34     public void report() {
35         System.out.println("Estatísticas de Temperatura:");
36         calculateStatistics(temperatures);
37
38         System.out.println("\nEstatísticas de Umidade:");
39         calculateStatistics(humidities);
40
41         System.out.println("\nEstatísticas de Pressão:");
42         calculateStatistics(pressures);
43     }
44 }
```

```
Run | Debug  
45     public static void main(String[] args) {  
46         double[] temperatures = { 20, 22, 18, 19, 23 };  
47         double[] humidities = { 60, 55, 58, 57, 59 };  
48         double[] pressures = { 1000, 1005, 999, 1003, 1001 };  
49  
50         WeatherData weatherData = new WeatherData(temperatures, humidities, pressures);  
51         weatherData.report();  
52     }  
53 }
```

```
Estatísticas de Temperatura:  
Média: 20.4  
Mínima: 18.0  
Máxima: 23.0  
  
Estatísticas de Umidade:  
Média: 57.8  
Mínima: 55.0  
Máxima: 60.0  
  
Estatísticas de Pressão:  
Média: 1001.6  
Mínima: 999.0  
Máxima: 1005.0
```

### 3.3.2. Node.js

```
Final > Node > js 32Metereologia.js > WeatherData
 1 // 3.2 Metereologia
 2
 3 class WeatherData {
 4   constructor(temperatures, humidities, pressures) {
 5     this.temperatures = temperatures;
 6     this.humidities = humidities;
 7     this.pressures = pressures;
 8   }
 9
10   calculateStatistics(data) {
11     let average = 0;
12     let minimum = data[0];
13     let maximum = data[0];
14
15     for (const value of data) {
16       average += value;
17       if (value < minimum) minimum = value;
18       if (value > maximum) maximum = value;
19     }
20
21     average /= data.length;
22
23     console.log('Média:', average);
24     console.log('Mínima:', minimum);
25     console.log('Máxima:', maximum);
26   }
27
28   report() {
29     console.log('Estatísticas de Temperatura:');
30     this.calculateStatistics(this.temperatures);
31
32     console.log('\nEstatísticas de Umidade:');
33     this.calculateStatistics(this.humidities);
34
35     console.log('\nEstatísticas de Pressão:');
36     this.calculateStatistics(this.pressures);
37   }
38 }
39
40 const temperatures = [20, 22, 18, 19, 23];
41 const humidities = [60, 55, 58, 57, 59];
42 const pressures = [1000, 1005, 999, 1003, 1001];
43
44 const weatherData = new WeatherData(temperatures, humidities, pressures);
45 weatherData.report();
```

```
Estatísticas de Temperatura:  
Média: 20.4  
Mínima: 18  
Máxima: 23
```

```
Estatísticas de Umidade:  
Média: 57.8  
Mínima: 55  
Máxima: 60
```

```
Estatísticas de Pressão:  
Média: 1001.6  
Mínima: 999  
Máxima: 1005
```

### 3.3.3. Python

```
Final > Python > 32Metereologia.py > WeatherData
 1 # 3.2 Metereologia
 2
 3 class WeatherData:
 4     def __init__(self, temperatures, humidities, pressures):
 5         self.temperatures = temperatures
 6         self.humidities = humidities
 7         self.pressures = pressures
 8
 9     def calculate_statistics(self, data):
10         return {
11             "average": sum(data) / len(data),
12             "minimum": min(data),
13             "maximum": max(data),
14         }
15
16     def report(self):
17         temperature_stats = self.calculate_statistics(self.temperatures)
18         humidity_stats = self.calculate_statistics(self.humidities)
19         pressure_stats = self.calculate_statistics(self.pressures)
20
21         print("Estatisticas de Temperatura:")
22         print("Media:", temperature_stats["average"])
23         print("Minima:", temperature_stats["minimum"])
24         print("Maxima:", temperature_stats["maximum"])
25
26         print("\nEstatisticas de Umidade:")
27         print("Media:", humidity_stats["average"])
28         print("Minima:", humidity_stats["minimum"])
29         print("Maxima:", humidity_stats["maximum"])
30
31         print("\nEstatisticas de Pressao:")
32         print("Media:", pressure_stats["average"])
33         print("Minima:", pressure_stats["minimum"])
34         print("Maxima:", pressure_stats["maximum"])
35
36
37 # Exemplo de uso
38 temperatures = [20, 22, 18, 19, 23]
39 humidities = [60, 55, 58, 57, 59]
40 pressures = [1000, 1005, 999, 1003, 1001]
41
42 weather_data = WeatherData(temperatures, humidities, pressures)
43 weather_data.report()
```

```
Estatisticas de Temperatura:
Media: 20.4
Minima: 18
Maxima: 23

Estatisticas de Umidade:
Media: 57.8
Minima: 55
Maxima: 60

Estatisticas de Pressao:
Media: 1001.6
Minima: 999
Maxima: 1005
```

## 3.4. Estatísticas Futebol

Crie um programa em Python (versão 3.11.4) que calcule e apresente estatísticas sobre uma série de jogos de futebol. O programa deve aceitar a entrada de dados como gols marcados, cartões e faltas, e calcular médias, máximos e mínimos.

### 3.4.1. Java

```
Final > J4V4 > Football42 > J FootballStatistics.java > FootballStatistics
 1 package Final.J4V4.Football42;
 2
 3 import java.util.ArrayList;
 4
 5 public class FootballStatistics {
 6     private static class Game {
 7         int goals, cards, fouls;
 8
 9         Game(int goals, int cards, int fouls) {
10             this.goals = goals;
11             this.cards = cards;
12             this.fouls = fouls;
13         }
14     }
15
16     private ArrayList<Game> games;
17
18     public FootballStatistics() {
19         this.games = new ArrayList<>();
20     }
21
22     public void addGame(int goals, int cards, int fouls) {
23         this.games.add(new Game(goals, cards, fouls));
24     }
25
26     public void calculateStatistics() {
27         int totalGoals = 0, totalCards = 0, totalFouls = 0;
28         int maxGoals = Integer.MIN_VALUE, maxCards = Integer.MIN_VALUE, maxFouls = Integer.MIN_VALUE;
29         int minGoals = Integer.MAX_VALUE, minCards = Integer.MAX_VALUE, minFouls = Integer.MAX_VALUE;
30
31         for (Game game : games) {
32             totalGoals += game.goals;
33             totalCards += game.cards;
34             totalFouls += game.fouls;
35             maxGoals = Math.max(maxGoals, game.goals);
36             maxCards = Math.max(maxCards, game.cards);
37             maxFouls = Math.max(maxFouls, game.fouls);
38             minGoals = Math.min(minGoals, game.goals);
39             minCards = Math.min(minCards, game.cards);
40             minFouls = Math.min(minFouls, game.fouls);
41         }
42
43         System.out.println("Statistics:");
44         System.out.println("Average goals: " + (double) totalGoals / games.size());
45         System.out.println("Average cards: " + (double) totalCards / games.size());
46         System.out.println("Average fouls: " + (double) totalFouls / games.size());
47         System.out.println("Max goals: " + maxGoals);
48         System.out.println("Max cards: " + maxCards);
49         System.out.println("Max fouls: " + maxFouls);
50         System.out.println("Min goals: " + minGoals);
51         System.out.println("Min cards: " + minCards);
52         System.out.println("Min fouls: " + minFouls);
53     }
54 }
```

```
Run | Debug
55  public static void main(String[] args) {
56      FootballStatistics stats = new FootballStatistics();
57      stats.addGame(2, 4, 10);
58      stats.addGame(3, 5, 8);
59      stats.addGame(1, 2, 12);
60      stats.addGame(0, 3, 7);
61      stats.addGame(4, 2, 9);
62      stats.addGame(3, 4, 5);
63      stats.addGame(2, 1, 11);
64      stats.addGame(5, 3, 6);
65      stats.addGame(2, 4, 7);
66      stats.addGame(1, 5, 8);
67      stats.calculateStatistics();
68  }
69 }
70 }
```

```
Statistics:
Average goals: 2.3
Average cards: 3.3
Average fouls: 8.3
Max goals: 5
Max cards: 5
Max fouls: 12
Min goals: 0
Min cards: 1
Min fouls: 5
```

### 3.4.2. Node.js

```
Final > Node > js 42Football.js >  FootballStatistics >  calculateStatistics
1  // 4.2 - Football
2
3  class FootballStatistics {
4    constructor() {
5      this.games = [];
6    }
7
8    addGame(goals, cards, fouls) {
9      this.games.push({ goals, cards, fouls });
10   }
11
12   calculateStatistics() {
13     let totalGoals = 0, totalCards = 0, totalFouls = 0;
14     let maxGoals = Number.MIN_SAFE_INTEGER, maxCards = Number.MIN_SAFE_INTEGER, maxFouls = Number.MIN_SAFE_INTEGER;
15     let minGoals = Number.MAX_SAFE_INTEGER, minCards = Number.MAX_SAFE_INTEGER, minFouls = Number.MAX_SAFE_INTEGER;
16
17     for (let game of this.games) {
18       totalGoals += game.goals;
19       totalCards += game.cards;
20       totalFouls += game.fouls;
21       maxGoals = Math.max(maxGoals, game.goals);
22       maxCards = Math.max(maxCards, game.cards);
23       maxFouls = Math.max(maxFouls, game.fouls);
24       minGoals = Math.min(minGoals, game.goals);
25       minCards = Math.min(minCards, game.cards);
26       minFouls = Math.min(minFouls, game.fouls);
27     }
28
29     console.log("Statistics:");
30     console.log("Average goals:", totalGoals / this.games.length);
31     console.log("Average cards:", totalCards / this.games.length);
32     console.log("Average fouls:", totalFouls / this.games.length);
33     console.log("Max goals:", maxGoals);
34     console.log("Max cards:", maxCards);
35     console.log("Max fouls:", maxFouls);
36     console.log("Min goals:", minGoals);
37     console.log("Min cards:", minCards);
38     console.log("Min fouls:", minFouls);
39   }
40 }
41
42 const stats = new FootballStatistics();
43 stats.addGame(2, 4, 10);
44 stats.addGame(3, 5, 8);
45 stats.addGame(1, 2, 12);
46 stats.addGame(0, 3, 7);
47 stats.addGame(4, 2, 9);
48 stats.addGame(3, 4, 5);
49 stats.addGame(2, 1, 11);
50 stats.addGame(5, 3, 6);
51 stats.addGame(2, 4, 7);
52 stats.addGame(1, 5, 8);
53 stats.calculateStatistics();
```

### 3.4.3. Python

```
Final > Python > 42Football.py > FootballStatistics
 1  #4.2 Football
 2
 3  class FootballStatistics:
 4      def __init__(self):
 5          self.games = []
 6
 7      def add_game(self, goals, cards, fouls):
 8          game = {'goals': goals, 'cards': cards, 'fouls': fouls}
 9          self.games.append(game)
10
11     def calculate_statistics(self):
12         total_goals, total_cards, total_fouls = 0, 0, 0
13         max_goals = max_cards = max_fouls = float('-inf')
14         min_goals = min_cards = min_fouls = float('inf')
15
16         for game in self.games:
17             total_goals += game['goals']
18             total_cards += game['cards']
19             total_fouls += game['fouls']
20             max_goals = max(max_goals, game['goals'])
21             max_cards = max(max_cards, game['cards'])
22             max_fouls = max(max_fouls, game['fouls'])
23             min_goals = min(min_goals, game['goals'])
24             min_cards = min(min_cards, game['cards'])
25             min_fouls = min(min_fouls, game['fouls'])
26
27         avg_goals = total_goals / len(self.games)
28         avg_cards = total_cards / len(self.games)
29         avg_fouls = total_fouls / len(self.games)
30
31         print("Statistics:")
32         print("Average goals:", avg_goals)
33         print("Average cards:", avg_cards)
34         print("Average fouls:", avg_fouls)
35         print("Max goals:", max_goals)
36         print("Max cards:", max_cards)
37         print("Max fouls:", max_fouls)
38         print("Min goals:", min_goals)
39         print("Min cards:", min_cards)
40         print("Min fouls:", min_fouls)
41
```

```
43  # Exemplo de uso com 10 jogos
44  if __name__ == "__main__":
45      stats = FootballStatistics()
46      stats.add_game(2, 4, 10)
47      stats.add_game(3, 5, 8)
48      stats.add_game(1, 2, 12)
49      stats.add_game(0, 3, 7)
50      stats.add_game(4, 2, 9)
51      stats.add_game(3, 4, 5)
52      stats.add_game(2, 1, 11)
53      stats.add_game(5, 3, 6)
54      stats.add_game(2, 4, 7)
55      stats.add_game(1, 5, 8)
56      stats.calculate_statistics()
57
```

```
Statistics:
Average goals: 2.3
Average cards: 3.3
Average fouls: 8.3
Max goals: 5
Max cards: 5
Max fouls: 12
Min goals: 0
Min cards: 1
Min fouls: 5
```

### 3.5. Estatísticas Tráfego

Em Python (versão 3.11.4), escreva um programa que faça a análise estatística de dados de tráfego em um site, incluindo visitantes únicos, páginas mais visitadas e duração média da visita. Apresente os resultados calculados.

#### 3.5.1. Java

```
Final > J4V4 > Visitas52 > J AnaliseTrafegoSite.java > AnaliseTrafegoSite
1 package Final.J4V4.Visitats52;
2
3 import java.util.ArrayList;
4 import java.util.HashSet;
5 import java.util.HashMap;
6
7 public class AnaliseTrafegoSite {
8     static class Visita {
9         String visitorId;
10        String page;
11        int duration;
12
13        public Visita(String visitorId, String page, int duration) {
14            this.visitorId = visitorId;
15            this.page = page;
16            this.duration = duration;
17        }
18    }
}
```

```
Run | Debug
20 public static void main(String[] args) {
21     ArrayList<Visita> dadosVisitas = new ArrayList<>();
22     dadosVisitas.add(new Visita("v1", "/home", 5));
23     dadosVisitas.add(new Visita("v1", "/contact", 2));
24     dadosVisitas.add(new Visita("v2", "/home", 3));
25     dadosVisitas.add(new Visita("v3", "/products", 8));
26     dadosVisitas.add(new Visita("v3", "/home", 2));
27
28     // Análise de visitantes únicos
29     HashSet<String> visitantesUnicos = new HashSet<>();
30     for (Visita visita : dadosVisitas) {
31         visitantesUnicos.add(visita.visitorId);
32     }
33     System.out.println("Visitantes únicos: " + visitantesUnicos.size());
34
35     // Análise de páginas mais visitadas
36     HashMap<String, Integer> paginasVisitas = new HashMap<>();
37     for (Visita visita : dadosVisitas) {
38         paginasVisitas.put(visita.page, paginasVisitas.getOrDefault(visita.page, 0) + 1);
39     }
40     String paginaMaisVisitada = paginasVisitas.keySet().stream()
41         .max((a, b) -> paginasVisitas.get(a) - paginasVisitas.get(b).get());
42     System.out.println("Página mais visitada: " + paginaMaisVisitada + " (" + paginasVisitas.get(paginaMaisVisitada)
43         + " visitas)");
44
45     // Análise da duração média da visita
46     int duracaoTotal = dadosVisitas.stream().mapToInt(visita -> visita.duration).sum();
47     double duracaoMedia = (double) duracaoTotal / dadosVisitas.size();
48     System.out.printf("Duração média da visita: %.2f minutos%n", duracaoMedia);
49 }
50 }
```

### 3.5.2. Node.js

```
Final > Node > JS 52Visitas.js > ...
1  // Exemplo de dados (em um cenário real, você extrairia esses dados de suas fontes)
2  const dadosVisitas = [
3      { visitor_id: 'v1', page: '/home', duration: 5 },
4      { visitor_id: 'v1', page: '/contact', duration: 2 },
5      { visitor_id: 'v2', page: '/home', duration: 3 },
6      { visitor_id: 'v3', page: '/products', duration: 8 },
7      { visitor_id: 'v3', page: '/home', duration: 2 },
8  ];
9
10 // Análise de visitantes únicos
11 const visitantesUnicos = new Set(dadosVisitas.map(visita => visita.visitor_id));
12 console.log(`Visitantes únicos: ${visitantesUnicos.size}`);
13
14 // Análise de páginas mais visitadas
15 const paginasVisitadas = {};
16 for (const visita of dadosVisitas) {
17     const page = visita.page;
18     paginasVisitadas[page] = (paginasVisitadas[page] || 0) + 1;
19 }
20 const paginaMaisVisitada = Object.keys(paginasVisitadas).reduce((a, b) => paginasVisitadas[a] > paginasVisitadas[b] ? a : b);
21 console.log(`Página mais visitada: ${paginaMaisVisitada} (${paginasVisitadas[paginaMaisVisitada]} visitas`);
22
23 // Análise da duração média da visita
24 const duracaoTotal = dadosVisitas.reduce((acc, visita) => acc + visita.duration, 0);
25 const duracaoMedia = duracaoTotal / dadosVisitas.length;
26 console.log(`Duração média da visita: ${duracaoMedia.toFixed(2)} minutos`);
```

### 3.5.3. Python

```
Final > Python > 52Visitas.py > ...
1  # Exemplo de dados (em um cenário real, você extrairia esses dados de suas fontes)
2  dados_visitas = [
3      {'visitor_id': 'v1', 'page': '/home', 'duration': 5},
4      {'visitor_id': 'v1', 'page': '/contact', 'duration': 2},
5      {'visitor_id': 'v2', 'page': '/home', 'duration': 3},
6      {'visitor_id': 'v3', 'page': '/products', 'duration': 8},
7      {'visitor_id': 'v3', 'page': '/home', 'duration': 2},
8  ]
9
10 # Análise de visitantes únicos
11 visitantes_unicos = set([visita['visitor_id'] for visita in dados_visitas])
12 print(f"Visitantes únicos: {len(visitantes_unicos)}")
13
14 # Análise de páginas mais visitadas
15 paginas_visitadas = {}
16 for visita in dados_visitas:
17     page = visita['page']
18     paginas_visitadas[page] = paginas_visitadas.get(page, 0) + 1
19 pagina_mais_visitada = max(paginas_visitadas, key=paginas_visitadas.get)
20 print(f"Pagina mais visitada: {pagina_mais_visitada} ({paginas_visitadas[pagina_mais_visitada]} visitas)")
21
22 # Análise da duração média da visita
23 duracao_total = sum([visita['duration'] for visita in dados_visitas])
24 duracao_media = duracao_total / len(dados_visitas)
25 print(f"Duracao media da visita: {duracao_media:.2f} minutos")
```

## 4. Execução de Rotinas a partir de uma Especificação

### 4.1. Câmbio Moedas

Desenvolva um programa em Node Javascript (versão 18.17.0) que execute uma rotina de conversão de moedas. A rotina deve ser capaz de converter valores entre USD, EUR e BRL, utilizando uma taxa de câmbio fixa fornecida na especificação.

#### 4.1.1. Java

```
Final > J4V4 > Cambio13 > J ConversorDeMoedas.java > ConversorDeMoedas > main(String[])
1 package Final.J4V4.Cambio13;
2
3 import java.util.Scanner;
4
5 public class ConversorDeMoedas {
6
7     Run | Debug
8     public static void main(String[] args) {
9         Scanner scanner = new Scanner(System.in);
10
11         // Definindo taxas de câmbio fixas
12         double usdParaBrl = 5.2;
13         double eurParaBrl = 6.1;
14         double usdParaEur = eurParaBrl / usdParaBrl;
15
16         System.out.println("Informe o valor a ser convertido:");
17         double valor = scanner.nextDouble();
18
19         System.out.println("Informe a moeda de origem (USD, EUR, BRL):");
20         String moedaOrigem = scanner.next().toUpperCase();
21
22         System.out.println("Informe a moeda de destino (USD, EUR, BRL):");
23         String moedaDestino = scanner.next().toUpperCase();
24
25         double valorConvertido = 0.0;
```

```
26     // Convertendo o valor de acordo com as moedas selecionadas
27     if (moedaOrigem.equals("USD") && moedaDestino.equals("BRL")) {
28         valorConvertido = valor * usdParaBrl;
29     } else if (moedaOrigem.equals("USD") && moedaDestino.equals("EUR")) {
30         valorConvertido = valor * usdParaEur;
31     } else if (moedaOrigem.equals("EUR") && moedaDestino.equals("BRL")) {
32         valorConvertido = valor * eurParaBrl;
33     } else if (moedaOrigem.equals("EUR") && moedaDestino.equals("USD")) {
34         valorConvertido = valor / usdParaEur;
35     } else if (moedaOrigem.equals("BRL") && moedaDestino.equals("USD")) {
36         valorConvertido = valor / usdParaBrl;
37     } else if (moedaOrigem.equals("BRL") && moedaDestino.equals("EUR")) {
38         valorConvertido = valor / eurParaBrl;
39     }
40
41     System.out.println("Valor convertido: " + moedaDestino + " " + valorConvertido);
42
43     scanner.close();
44 }
45 }
```

#### 4.1.2. Node.js

```
Final > Node > JS 13Cambio.js > ...
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const taxasDeCambio = {
9    USD_BRL: 5.20,
10   EUR_BRL: 6.13,
11   EUR_USD: 1.18
12 };
13
14 const converterMoeda = (valor, origem, destino) => {
15   let valorEmBRL;
16
17   if (origem === 'USD') {
18     valorEmBRL = valor * taxasDeCambio.USD_BRL;
19   } else if (origem === 'EUR') {
20     valorEmBRL = valor * taxasDeCambio.EUR_BRL;
21   } else {
22     valorEmBRL = valor;
23   }
24
25   let valorConvertido;
26
27   if (destino === 'USD') {
28     valorConvertido = valorEmBRL / taxasDeCambio.USD_BRL;
29   } else if (destino === 'EUR') {
30     valorConvertido = valorEmBRL / taxasDeCambio.EUR_BRL;
31   } else {
32     valorConvertido = valorEmBRL;
33   }
34
35   return valorConvertido;
36 };
```

```
38   rl.question('Insira a moeda de origem (USD, EUR, BRL): ', (origem) => {
39     rl.question('Insira a moeda de destino (USD, EUR, BRL): ', (destino) => {
40       rl.question('Insira o valor para conversão: ', (valor) => {
41         const valorConvertido = converterMoeda(Number(valor), origem, destino);
42         console.log(`Valor convertido: ${destino} ${valorConvertido.toFixed(2)}`);
43         rl.close();
44       });
45     });
46   });

```

```
Insira a moeda de origem (USD, EUR, BRL): USD
Insira a moeda de destino (USD, EUR, BRL): BRL
Insira o valor para conversão: 5000
Valor convertido: BRL 26000.00
```

### 4.1.3. Python

```
Final > Python > 13Cambio.py > ...
1 # Definindo taxas de câmbio fixas
2 usd_para_brl = 5.2
3 eur_para_brl = 6.1
4 usd_para_eur = eur_para_brl / usd_para_brl
5
6 valor = float(input("Informe o valor a ser convertido: "))
7 moeda_origem = input("Informe a moeda de origem (USD, EUR, BRL): ").upper()
8 moeda_destino = input("Informe a moeda de destino (USD, EUR, BRL): ").upper()
9
10 valor_convertido = 0.0
11
12 # Convertendo o valor de acordo com as moedas selecionadas
13 if moeda_origem == "USD" and moeda_destino == "BRL":
14     valor_convertido = valor * usd_para_brl
15 elif moeda_origem == "USD" and moeda_destino == "EUR":
16     valor_convertido = valor * usd_para_eur
17 elif moeda_origem == "EUR" and moeda_destino == "BRL":
18     valor_convertido = valor * eur_para_brl
19 elif moeda_origem == "EUR" and moeda_destino == "USD":
20     valor_convertido = valor / usd_para_eur
21 elif moeda_origem == "BRL" and moeda_destino == "USD":
22     valor_convertido = valor / usd_para_brl
23 elif moeda_origem == "BRL" and moeda_destino == "EUR":
24     valor_convertido = valor / eur_para_brl
25
26 print(f"Valor convertido: {moeda_destino} {valor_convertido}")
```

## 4.2. Classificação E-Mails:

Utilizando Node Javascript (versão 18.17.0), escreva um programa que execute uma rotina de filtragem e classificação de e-mails. A rotina deve ser capaz de separar e-mails em categorias como "Pessoal", "Trabalho" e "Spam", com base em regras fornecidas na especificação.

### 4.2.1. Java

```
Final > J4V4 > Emails23 > ClassificadorEmails.java > ...
1  // 2.3 Emails
2  package Final.J4V4.Emails23;
3
4  import java.util.ArrayList;
5  import java.util.HashMap;
6
7  class Email {
8      String endereco;
9      String dominio;
10
11     public Email(String endereco, String dominio) {
12         this.endereco = endereco;
13         this.dominio = dominio;
14     }
15 }
16
17 public class ClassificadorEmails {
18
19     public static String categorizarEmail(Email email, HashMap<String, ArrayList<String>> regras) {
20         if (regras.get("Spam").contains(email.dominio)) {
21             return "Spam";
22         }
23         if (regras.get("Trabalho").contains(email.dominio)) {
24             return "Trabalho";
25         }
26         return "Pessoal";
27     }
28
29     public static HashMap<String, ArrayList<String>> classificarEmails(ArrayList<Email> emails,
30                         HashMap<String, ArrayList<String>> regras) {
31         HashMap<String, ArrayList<String>> categorias = new HashMap<>();
32         categorias.put("Pessoal", new ArrayList<>());
33         categorias.put("Trabalho", new ArrayList<>());
34         categorias.put("Spam", new ArrayList<>());
35
36         for (Email email : emails) {
37             String categoria = categorizarEmail(email, regras);
38             categorias.get(categoria).add(email.endereco);
39         }
40
41     return categorias;
42 }
```

```
Run | Debug
44  public static void main(String[] args) {
45      ArrayList<Email> emails = new ArrayList<>();
46      emails.add(new Email("john@example.com", "example.com"));
47      emails.add(new Email("sara@work.com", "work.com"));
48      emails.add(new Email("spam@mail.net", "mail.net"));
49
50      HashMap<String, ArrayList<String>> regras = new HashMap<>();
51      regras.put("Spam", new ArrayList<>());
52      regras.put("Trabalho", new ArrayList<>());
53      regras.get("Spam").add("mail.net");
54      regras.get("Trabalho").add("work.com");
55
56      HashMap<String, ArrayList<String>> classificacao = classificarEmails(emails, regras);
57      System.out.println(classificacao);
58  }
59 }
```

```
Pessoal: [ 'john@example.com' ],
Trabalho: [ 'sara@work.com' ],
Spam: [ 'spam@mail.net' ]
```

#### 4.2.2. Node.js

```
Final > Node > js 23Emails.js > ...
1  // 2.3 Emails
2  function categorizarEmail(email, regras) {
3      if (regras.spam.includes(email.dominio)) {
4          return "Spam";
5      }
6      if (regras.trabalho.includes(email.dominio)) {
7          return "Trabalho";
8      }
9      return "Pessoal";
10 }
11
12 function classificarEmails(emails, regras) {
13     const categorias = {
14         Pessoal: [],
15         Trabalho: [],
16         Spam: []
17     };
18
19     emails.forEach(email => {
20         const categoria = categorizarEmail(email, regras);
21         categorias[categoria].push(email.endereco);
22     });
23
24     return categorias;
25 }
26
27 const emails = [
28     { endereco: 'john@example.com', dominio: 'example.com' },
29     { endereco: 'sara@work.com', dominio: 'work.com' },
30     { endereco: 'spam@mail.net', dominio: 'mail.net' }
31 ];
32
33 const regras = {
34     spam: ['mail.net'],
35     trabalho: ['work.com']
36 };
37
38 const classificacao = classificarEmails(emails, regras);
39 console.log(classificacao);
```

#### 4.2.3. Python

```
Final > Python > 23Emails.py > ...
1  # 2.3 Emails
2  class Email:
3      def __init__(self, endereco, dominio):
4          self.endereco = endereco
5          self.dominio = dominio
6
7  def categorizar_email(email, regras):
8      if email.dominio in regras['Spam']:
9          return 'Spam'
10     if email.dominio in regras['Trabalho']:
11         return 'Trabalho'
12     return 'Pessoal'
13
14 def classificar_emails(emails, regras):
15     categorias = {'Pessoal': [], 'Trabalho': [], 'Spam': []}
16
17     for email in emails:
18         categoria = categorizar_email(email, regras)
19         categorias[categoria].append(email.endereco)
20
21     return categorias
22
23 if __name__ == "__main__":
24     emails = [
25         Email('john@example.com', 'example.com'),
26         Email('sara@work.com', 'work.com'),
27         Email('spam@mail.net', 'mail.net')
28     ]
29
30     regras = {
31         'Spam': ['mail.net'],
32         'Trabalho': ['work.com']
33     }
34
35     classificacao = classificar_emails(emails, regras)
36     print(classificacao)
```

### 4.3. Análise de Logs:

Desenvolva um programa em Node Javascript (versão 18.17.0) que execute uma rotina de análise de logs de servidor. A rotina deve ser capaz de filtrar eventos críticos e apresentar um relatório com base em uma especificação fornecida.

#### 4.3.1. Java

```
Final > J4V4 > Logs33 > J LogAnalyzer.java > LogAnalyzer
1 package Final.J4V4.Logs33;
2
3 //3.3 Logs
4
5 public class LogAnalyzer {
6     private String[] logs;
7
8     public LogAnalyzer(String[] logs) {
9         this.logs = logs;
10    }
11
12    public String[] filterCriticalEvents() {
13        return java.util.Arrays.stream(logs)
14            .filter(log -> log.startsWith("CRITICAL"))
15            .toArray(String[]::new);
16    }
17
18    public void generateReport() {
19        String[] criticalEvents = filterCriticalEvents();
20
21        if (criticalEvents.length == 0) {
22            System.out.println("Nenhum evento crítico encontrado.");
23            return;
24        }
25
26        System.out.println("Relatório de Eventos Críticos:");
27        for (String event : criticalEvents) {
28            System.out.println(event);
29        }
30    }
}
```

```
Run | Debug
32     public static void main(String[] args) {
33         String[] logs = {
34             "INFO: Usuário logado com sucesso.",
35             "CRITICAL: Falha na base de dados.",
36             "INFO: Arquivo atualizado.",
37             "WARNING: Tentativa de acesso não autorizado.",
38             "CRITICAL: Sistema de arquivos corrompido."
39         };
40
41         LogAnalyzer analyzer = new LogAnalyzer(logs);
42         analyzer.generateReport();
43     }
44 }
```

```
CRITICAL: Falha na base de dados.
CRITICAL: Sistema de arquivos corrompido.
```

#### 4.3.2. Node.js

```
Final > Node > js 33Logs.js > LogAnalyzer
1  // 3.3 - Logs
2  const logs = [
3      'INFO: Usuário logado com sucesso.',
4      'CRITICAL: Falha na base de dados.',
5      'INFO: Arquivo atualizado.',
6      'WARNING: Tentativa de acesso não autorizado.',
7      'CRITICAL: Sistema de arquivos corrompido.'
8 ];
9
10 class LogAnalyzer {
11     constructor(logs) {
12         this.logs = logs;
13     }
14
15     filterCriticalEvents() {
16         return this.logs.filter(log => log.startsWith('CRITICAL'));
17     }
}
```

```
18
19     generateReport() {
20         const criticalEvents = this.filterCriticalEvents();
21
22         if (criticalEvents.length === 0) {
23             console.log('Nenhum evento crítico encontrado.');
24             return;
25         }
26
27         console.log('Relatório de Eventos Críticos:');
28         for (const event of criticalEvents) {
29             console.log(event);
30         }
31     }
32 }
33
34 const analyzer = new LogAnalyzer(logs);
35 analyzer.generateReport();
```

```
Relatório de Eventos Críticos:
CRITICAL: Falha na base de dados.
CRITICAL: Sistema de arquivos corrompido.
```

### 4.3.3. Python

```
Final > Python > 33Emails.py > ...
1  #3.3 Emails
2
3  def filter_critical_events(logs):
4      return [log for log in logs if log.startswith("CRITICAL")]
5
6  def generate_report(logs):
7      critical_events = filter_critical_events(logs)
8
9      if not critical_events:
10         print("Nenhum evento critico encontrado.")
11         return
12
13     print("Relatorio de Eventos Criticos:")
14     for event in critical_events:
15         print(event)
16
17 if __name__ == "__main__":
18     logs = [
19         "INFO: Usuário logado com sucesso.",
20         "CRITICAL: Falha na base de dados.",
21         "INFO: Arquivo atualizado.",
22         "WARNING: Tentativa de acesso não autorizado.",
23         "CRITICAL: Sistema de arquivos corrompido.",
24     ]
25
26     generate_report(logs)
```

```
Relatorio de Eventos Criticos:
CRITICAL: Falha na base de dados.
CRITICAL: Sistema de arquivos corrompido.
```

## 4.4. Rotas Delivery

Utilizando Java (versão 20.0.2), escreva um programa que implemente um sistema de roteamento para entregas. A aplicação deve ser capaz de calcular rotas eficientes baseadas em diferentes critérios especificados, como menor distância ou menor tempo.

### 4.4.1. Java

```
Final > J4V4 > Delivery43 > J Main.java > ...
1  package Final.J4V4.Delivery43;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  class DeliveryRoute {
7      String start;
8      String destination;
9      int distance;
10     int time;
11
12     public DeliveryRoute(String start, String destination, int distance, int time) {
13         this.start = start;
14         this.destination = destination;
15         this.distance = distance;
16         this.time = time;
17     }
18 }
19
20 class RoutingSystem {
21     List<DeliveryRoute> routes;
22
23     public RoutingSystem() {
24         this.routes = new ArrayList<>();
25     }
26
27     public void addRoute(DeliveryRoute route) {
28         routes.add(route);
29     }
30
31     public DeliveryRoute calculateEfficientRoute(String start, String destination, String criteria) {
32         DeliveryRoute efficientRoute = null;
33
34         for (DeliveryRoute route : routes) {
35             if (route.start.equals(start) && route.destination.equals(destination)) {
36                 if (efficientRoute == null) {
37                     efficientRoute = route;
38                     continue;
39                 }
40                 if (criteria.equals("distance") && route.distance < efficientRoute.distance) {
41                     efficientRoute = route;
42                 } else if (criteria.equals("time") && route.time < efficientRoute.time) {
43                     efficientRoute = route;
44                 }
45             }
46         }
47
48         return efficientRoute;
49     }
50 }
```

```
52  public class Main {
53      Run|Debug
54      public static void main(String[] args) {
55          RoutingSystem system = new RoutingSystem();
56          system.addRoute(new DeliveryRoute("A", "B", 100, 60));
57          system.addRoute(new DeliveryRoute("A", "B", 120, 50));
58          system.addRoute(new DeliveryRoute("A", "C", 150, 80));
59
60          DeliveryRoute efficientRouteByDistance = system.calculateEfficientRoute("A", "B", "distance");
61          System.out.println("Efficient route by distance: " + efficientRouteByDistance.distance + " km");
62
63          DeliveryRoute efficientRouteByTime = system.calculateEfficientRoute("A", "B", "time");
64          System.out.println("Efficient route by time: " + efficientRouteByTime.time + " minutes");
65      }
66  }
```

```
Efficient route by distance: 100 km
Efficient route by time: 50 minutes
```

#### 4.4.2. Node.js

```
Final > Node > js 43Delivery.js > RoutingSystem
  1  class DeliveryRoute {
  2    constructor(start, destination, distance, time) {
  3      this.start = start;
  4      this.destination = destination;
  5      this.distance = distance;
  6      this.time = time;
  7    }
  8  }
  9
10 class RoutingSystem {
11   constructor() {
12     this.routes = [];
13   }
14
15   addRoute(route) {
16     this.routes.push(route);
17   }
18
19   calculateEfficientRoute(start, destination, criteria) {
20     let efficientRoute = null;
21
22     for (const route of this.routes) {
23       if (route.start === start && route.destination === destination) {
24         if (efficientRoute === null) {
25           efficientRoute = route;
26           continue;
27         }
28         if (criteria === "distance" && route.distance < efficientRoute.distance) {
29           efficientRoute = route;
30         } else if (criteria === "time" && route.time < efficientRoute.time) {
31           efficientRoute = route;
32         }
33       }
34     }
35
36     return efficientRoute;
37   }
38 }
39
40 const system = new RoutingSystem();
41 system.addRoute(new DeliveryRoute("A", "B", 100, 60));
42 system.addRoute(new DeliveryRoute("A", "B", 120, 50));
43 system.addRoute(new DeliveryRoute("A", "C", 150, 80));
44
45 const efficientRouteByDistance = system.calculateEfficientRoute("A", "B", "distance");
46 console.log(`Efficient route by distance: ${efficientRouteByDistance.distance} km`);
47
48 const efficientRouteByTime = system.calculateEfficientRoute("A", "B", "time");
49 console.log(`Efficient route by time: ${efficientRouteByTime.time} minutes`);
```

```
Efficient route by distance: 100 km
Efficient route by time: 50 minutes
```

#### 4.4.3. Python

```
Final > Python > 43Delivery.py > RoutingSystem
 1  class DeliveryRoute:
 2      def __init__(self, start, destination, distance, time):
 3          self.start = start
 4          self.destination = destination
 5          self.distance = distance
 6          self.time = time
 7
 8  class RoutingSystem:
 9      def __init__(self):
10          self.routes = []
11
12      def add_route(self, route):
13          self.routes.append(route)
14
15      def calculate_efficient_route(self, start, destination, criteria):
16          efficient_route = None
17
18          for route in self.routes:
19              if route.start == start and route.destination == destination:
20                  if efficient_route is None:
21                      efficient_route = route
22                      continue
23                  if criteria == "distance" and route.distance < efficient_route.distance:
24                      efficient_route = route
25                  elif criteria == "time" and route.time < efficient_route.time:
26                      efficient_route = route
27
28          return efficient_route
29
30 system = RoutingSystem()
31 system.add_route(DeliveryRoute("A", "B", 100, 60))
32 system.add_route(DeliveryRoute("A", "B", 120, 50))
33 system.add_route(DeliveryRoute("A", "C", 150, 80))
34
35 efficient_route_by_distance = system.calculate_efficient_route("A", "B", "distance")
36 print(f"Efficient route by distance: {efficient_route_by_distance.distance} km")
37
38 efficient_route_by_time = system.calculate_efficient_route("A", "B", "time")
39 print(f"Efficient route by time: {efficient_route_by_time.time} minutes")
40
```

## 4.5. Rotina Backup

Utilizando Node Javascript (versão 18.17.0), crie um programa que execute uma rotina de backup de dados conforme uma especificação fornecida, incluindo a compressão e criptografia dos dados.

### 4.5.1. Node.js

```
Final > Node > JS 53Backup.js > ↳ backupFile > ↳ fs.readFile() callback > ↳ zlib.gzip() callback
1  const fs = require('fs');
2  const zlib = require('zlib');
3  const crypto = require('crypto');
4
5  function backupFile(inputFilePath, outputPath, encryptionKey) {
6    // Ler o arquivo de entrada
7    fs.readFile(inputFilePath, (err, data) => {
8      if (err) {
9        console.error('Erro ao ler o arquivo:', err);
10       return;
11     }
12
13     // Comprimir os dados
14     zlib.gzip(data, (err, compressedData) => {
15       if (err) {
16         console.error('Erro ao comprimir os dados:', err);
17         return;
18     }
```

```

13 // Comprimir os dados
14 zlib.gzip(data, (err, compressedData) => {
15   if (err) {
16     console.error('Erro ao comprimir os dados:', err);
17     return;
18   }
19
20   // Gere um IV (Vetor de Inicialização) seguro e aleatório
21   const iv = crypto.randomBytes(16);
22
23   // Converta a chave de criptografia em um buffer
24   const key = crypto.scryptSync(encryptionKey, 'salt', 32);
25
26   // Criar o cifrador com o algoritmo, a chave e o IV
27   const cipher = crypto.createCipheriv('aes-256-cbc', key, iv);
28
29   // Criptografar os dados comprimidos
30   let encryptedData = cipher.update(compressedData);
31   encryptedData = Buffer.concat([iv, encryptedData, cipher.final()]);
32
33   // Escrever os dados criptografados em um arquivo de backup
34   fs.writeFile(outputFilePath, encryptedData, (err) => {
35     if (err) {
36       console.error('Erro ao escrever o arquivo de backup:', err);
37     } else {
38       console.log('Backup realizado com sucesso!');
39     }
40   });
41 });
42 });
43 }

45 // Exemplo de uso
46 const inputFilePath = 'data.txt'; // Caminho do arquivo original
47 const outputFilePath = 'backup.gz.enc'; // Caminho do arquivo de backup
48 const encryptionKey = 'chave-secreta'; // Chave de criptografia
49 backupFile(inputFilePath, outputFilePath, encryptionKey);

```

Backup realizado com sucesso!			
 backup.gz.enc	16/08/2023 20:18	ENC File	1 KB
 data.txt	16/08/2023 20:15	Documento de Te...	4 KB

## 5. Encontrar e Solucionar Erros

### 5.1. Alunos: erros

Um trecho de código-fonte em Java (versão 20.0.2) será fornecido com pelo menos cinco erros. Sua tarefa é encontrar e corrigir esses erros, detalhando cada erro encontrado e apresentando a solução necessária para cada um.

#### 5.1.1. Java

**Código Original com Erros:**

```
java

import jav.util.ArrayList;

public class StudentManagement {
    private ArraList<Student> students;

    public StudentManagement() {
        students = new ArrayList<Student>;
    }

    public void addStudnt(Student student) {
        student.add(student);
    }

    public ArrayList<Student> getStudent() {
        return students
    }
}

class Student {
    private Strng name;
    private int age;

    public Student(Strng name, int age) {
        this.nam = name;
        this.age = age;
    }
}
```

#### Erros e Correções:

- **Erro:** Importação errada para ArrayList.
- **Solução:** Corrigir a importação para import java.util.ArrayList;.
  
- **Erro:** Declaração incorreta de ArrayList como ArraList.
- **Solução:** Corrigir para private ArrayList<Student> students;.
  
- **Erro:** Falta de modificador de acesso public no construtor.
- **Solução:** Corrigir para public StudentManagement() { ... }.
  
- **Erro:** Nome do método errado na adição de estudante, e o uso de student.add(student); ao invés de students.add(student);.
- **Solução:** Corrigir para:

```
java

public void addStudent(Student student) {
    students.add(student);
}
```

- **Erro:** Falta de ponto e vírgula na declaração de retorno do método getStudent().
- **Solução:** Corrigir para return students;.
  
- **Erro:** Tipo de nome errado na classe Student, e a atribuição errada this.nam.
- **Solução:** Corrigir para:

```
java

private String name;

public Student(String name, int age) {
    this.name = name;
    ...
}
```

## Código Corrigido:

```
java

import java.util.ArrayList;

public class StudentManagement {
    private ArrayList<Student> students;

    public StudentManagement() {
        students = new ArrayList<Student>();
    }

    public void addStudent(Student student) {
        students.add(student);
    }

    public ArrayList<Student> getStudents() {
        return students;
    }
}

class Student {
    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

### 5.1.2. Node.js

```
Código Original com Erros (Node.js):  
javascript  
  
const studentManagement = {  
    students: []  
  
    addStudent: function(student) {  
        this.student.push(student);  
    },  
  
    getStudents function() {  
        return student;  
    }  
};  
  
class Studen {  
    constructor(name, age) {  
        this.nme = name;  
        this.age = age;  
    }  
}  
  
const john = new Student("John", 25);  
studentManagement.addStudent(john);
```

#### Erros e Correções:

- **Erro:** Falta de vírgula após a declaração do atributo students.  
▪ **Solução:** Adicionar uma vírgula após a declaração do atributo students.
- **Erro:** Nome incorreto na chamada this.student.push(student).  
▪ **Solução:** Corrigir para this.students.push(student);
- **Erro:** Sintaxe incorreta para declarar a função getStudents.  
▪ **Solução:** Adicionar dois pontos e corrigir para getStudents: function() {...}.
- **Erro:** Retorno incorreto no método getStudents.  
▪ **Solução:** Corrigir para return this.students;;
- **Erro:** Nome incorreto da classe como Studen.  
▪ **Solução:** Corrigir para class Student {...}.
- **Erro:** Atribuição incorreta do nome dentro do construtor da classe Student.  
▪ **Solução:** Corrigir para this.name = name;.

### Código Corrigido:

```
javascript

const studentManagement = {
    students: [],

    addStudent: function(student) {
        this.students.push(student);
    },

    getStudents: function() {
        return this.students;
    }
};

class Student {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}

const john = new Student("John", 25);
studentManagement.addStudent(john);
```

### 5.1.3. Python

#### Trecho de código com erros:

```
python

def calculate_area(radius
    pi = 3.14
    area = pi radius ** 2
    print("Area of the circle with radius", radius, "is", area
```

Erros e soluções:

- **Erro:** Parênteses não fechados na definição da função.  
**Solução:** Adicionar um parêntese fechado após o parâmetro radius.
- **Erro:** Sintaxe inválida na linha que calcula a área, falta o operador de multiplicação.  
**Solução:** Inserir o operador de multiplicação (\*) entre pi e radius.
- **Erro:** A função está faltando dois pontos (:) após a declaração da função.  
**Solução:** Adicionar dois pontos após o parâmetro da função.
- **Erro:** Parênteses não fechados na declaração print.  
**Solução:** Adicionar um parêntese fechado no final da declaração print.
- **Erro:** A função não está sendo chamada, então o código não terá saída.  
**Solução:** Chamar a função com um valor de raio, por exemplo, calculate\_area(5).

#### Código corrigido:

```
python

def calculate_area(radius):
    pi = 3.14
    area = pi * radius ** 2
    print("Area of the circle with radius", radius, "is", area)

calculate_area(5)
```

## 5.2. Calculadora: erros

### 5.2.1. Enunciado (Parte A):

Um trecho de código-fonte em Java (versão 20.0.2) será fornecido contendo vários erros. Sua tarefa é identificar, detalhar e corrigir esses erros, apresentando a solução necessária para cada um.

#### 5.2.1.1. Java

**Parte A: Detecção e Correção de Erros**

Suponha que o seguinte trecho de código-fonte em Java contenha erros:

```
java

public clas Calculator {
    public sttic void main(String args[]) {
        int a = 10;
        int b 5;
        int sum = a + b
        System.out.println("Sum: " + sum)
    }
}
```

#### Erros e Correções:

- **Erro:** Classe com nome incorreto "clas".  
**Correção:** Trocar "clas" por "class".
- **Erro:** Modificador estático escrito incorretamente como "sttic".  
**Correção:** Trocar "sttic" por "static".
- **Erro:** Falta de sinal de igual ao atribuir o valor a b.  
**Correção:** Adicionar sinal de igual b = 5;
- **Erro:** Falta de ponto e vírgula após a expressão int sum = a + b.  
**Correção:** Adicionar ponto e vírgula int sum = a + b;.
- **Erro:** Falta de ponto e vírgula após o System.out.println.  
**Correção:** Adicionar ponto e vírgula System.out.println("Sum:" + sum);.

### Código Corrigido:

```
java

public class Calculator {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        int sum = a + b;
        System.out.println("Sum: " + sum);
    }
}
```

### 5.2.1.2. Node.js

#### Parte A: Detecção e Correção de Erros

Suponha que o seguinte trecho de código-fonte em Node.js contenha erros:

```
javascript

function calculator() {
    const a = 10
    const b 5;
    const sum a + b;
    consol.log("Sum: " + sum);
}

calculator)
```

#### Erros e Correções:

- **Erro:** Falta de sinal de igual ao atribuir o valor a b.  
**Correção:** Adicionar sinal de igual const b = 5;
- **Erro:** Falta de sinal de igual na declaração de sum.  
**Correção:** Adicionar sinal de igual const sum = a + b;.
- **Erro:** Falta de ponto e vírgula após a declaração de a.  
**Correção:** Adicionar ponto e vírgula const a = 10;.
- **Erro:** Erro de digitação em "console", escrito como "consol".  
**Correção:** Corrigir para console.log("Sum: " + sum);.
- **Erro:** Falta de parênteses ao chamar a função calculator.  
**Correção:** Adicionar parênteses calculator();.

### Código Corrigido:

```
javascript

function calculator() {
    const a = 10;
    const b = 5;
    const sum = a + b;
    console.log("Sum:" + sum);
}

calculator();
```

### 5.2.1.3. Python

#### Parte A: Detecção e Correção de Erros

Suponha que o seguinte trecho de código-fonte em Python contenha erros:

```
python

def calculator()
    a = 10
    b 5
    sum a + b
    print("Sum:", sum
```

Erros e Correções:

- **Erro:** Falta de dois-pontos ao definir a função calculator.
- **Correção:** Adicionar dois-pontos def calculator():.
  
- **Erro:** Falta de sinal de igual ao atribuir o valor a b.
- **Correção:** Adicionar sinal de igual b = 5.
  
- **Erro:** Falta de sinal de igual na declaração de sum.
- **Correção:** Adicionar sinal de igual sum = a + b.
  
- **Erro:** Falta de parêntese de fechamento na linha do print.
- **Correção:** Adicionar parêntese de fechamento print("Sum:", sum).
  
- **Erro:** A função é definida, mas não é chamada em nenhum lugar.
- **Correção:** Chamar a função calculator().

#### Código Corrigido:

```
python

def calculator():
    a = 10
    b = 5
    sum = a + b
    print("Sum:", sum)

calculator()
```

### 5.3. Soma: erros

Um trecho de código-fonte em Python (versão 3.11.4) será fornecido com vários erros lógicos e de sintaxe. Encontre e corrija esses erros, detalhando cada um e apresentando a solução necessária para cada erro.

#### 5.3.1. Java

Código Original com Erros

```
java

public class AnalyzeErrors {
    public static void main(String args[]) { // Erro na palavra "static"
        int a = 5;
        int b = 10;
        int result = sum(a b); // Falta vírgula entre os argumentos

        System.ou.println("A soma é: " + result); // "out" está incorreto
    }

    public static int sum(int x, int y); { // Ponto e vírgula indevidos
        return x + y + z; // z não está definido
    }
}
```

#### Erros e Soluções

- **Erro de Sintaxe na Assinatura do Método main:** A palavra-chave static está escrita incorretamente.  
**Solução:** Corrija para static.
- **Erro de Sintaxe na Chamada da Função sum:** A vírgula está faltando entre os argumentos a e b.  
**Solução:** Adicione uma vírgula entre os argumentos.
- **Erro de Sintaxe na Chamada do Método println:** O método está escrito como ou em vez de out.  
**Solução:** Corrija para System.out.println.
- **Erro de Sintaxe na Declaração do Método sum:** O ponto e vírgula não são necessários após a assinatura do método.  
**Solução:** Remova o ponto e vírgula.
- **Erro Lógico na Função sum:** A variável z não está definida na função.  
**Solução:** Remova a referência a z ou defina-a conforme necessário.

## Código Corrigido

```
java

public class AnalyzeErrors {
    public static void main(String[] args) { // Corrigido
        int a = 5;
        int b = 10;
        int result = sum(a, b); // Corrigido

        System.out.println("A soma é: " + result); // Corrigido
    }

    public static int sum(int x, int y) { // Corrigido
        return x + y; // Corrigido
    }
}
```

### 5.3.2. Node.js

## Código Original com Erros

```
javascript

function sum(x, y) {
    return x + y + z; // z não está definido
}

var a = 5;
var b = 10;
var result sum(a, b); // Sinal de igual está faltando

console.log("A soma é: " + result); // "log" está escrito incorretamente
```

## Erros e Soluções

- **Erro Lógico na Função sum:** A variável z não está definida na função.
- **Solução:** Remova a referência a z ou defina-a conforme necessário.
- **Erro de Sintaxe na Atribuição de result:** O sinal de igual (=) está faltando na atribuição.
- **Solução:** Inclua o sinal de igual.
- **Erro de Sintaxe na Chamada do Método log:** O método está escrito como logg em vez de log.
- **Solução:** Corrija para console.log.

### Código Corrigido

```
javascript

function sum(x, y) {
    return x + y; // Corrigido
}

var a = 5;
var b = 10;
var result = sum(a, b); // Corrigido

console.log("A soma é: " + result); // Corrigido
```

### 5.3.3. Python

#### Código Original com Erros

```
python

def sum(a b): # Falta vírgula entre os parâmetros
    return a + b + c # c não está definido

x = [1,2,3
y = sum(1 2) # Falta vírgula e fechamento da lista

if x == 3 # Falta dois-pontos
    print("X é igual a 3")
else
    print("X é diferente de 3")
```

## Erros e Soluções

- **Erro de Sintaxe na Definição da Função:** Na definição da função sum, a vírgula está faltando entre os parâmetros a e b.
- **Solução:** Adicione uma vírgula entre os parâmetros.

- **Erro Lógico na Função:** A variável c não está definida na função.  
▪ **Solução:** Remova a referência a c ou defina-a conforme necessário.
- **Erro de Sintaxe na Definição da Lista:** A lista x está com o fechamento faltando.  
▪ **Solução:** Adicione o fechamento da lista.
- **Erro de Sintaxe na Chamada da Função:** A vírgula está faltando na chamada da função sum.  
▪ **Solução:** Adicione uma vírgula entre os argumentos.
- **Erro de Sintaxe no if-statement:** Os dois-pontos estão faltando após a condição if.  
▪ **Solução:** Adicione dois-pontos após a condição.
- **Erro de Sintaxe no else-statement:** Os dois-pontos estão faltando após o else.  
▪ **Solução:** Adicione dois-pontos após o else.

### Código Corrigido

```
python

def sum(a, b): # Corrigido
    return a + b # Corrigido

x = [1,2,3] # Corrigido
y = sum(1, 2) # Corrigido

if x == 3: # Corrigido
    print("X é igual a 3")
else: # Corrigido
    print("X é diferente de 3")
```

## 5.4. Sistema Vendas: erros

Você receberá um código-fonte em Node Javascript (versão 18.17.0) com vários erros. Sua tarefa é identificar e corrigir esses erros, fornecendo uma explicação detalhada para cada correção realizada.

### 5.4.1. Java

Código Original

```
java
public class TotalPriceCalculator {
    public static void main(String[] args) {
        Product product = new Product("Laptop", "1200");
        Product[] products = {product, product, product};
        System.out.println(calculateTotalPrice(products));
    }

    public static int calculateTotalPrice(Product[] items) {
        int total = 0;
        for (Product item : items) {
            total + item.getPrice();
        }
        return total;
    }
}

class Product {
    private String name;
    private String price;

    public Product(String name, String price) {
        this.name = name;
        this.price = price;
    }

    public String getPrice() {
        return price;
    }
}
```

Explicação:

- **Erro na Acumulação de Total:** A expressão "total + item.getPrice()" foi alterada para "total += item.getPrice()" para acumular o valor no total.
- **Tipo de Dado Incorreto:** A variável price foi alterada de string para número inteiro. Isto é necessário para realizar a soma correta. Os métodos relevantes, incluindo o construtor, também foram atualizados para usar o tipo correto.

## Código Corrigido e Explicação

java

 Copy code

```
public class TotalPriceCalculator {
    public static void main(String[] args) {
        Product product = new Product("Laptop", 1200); // Alterado o preço pa
        Product[] products = {product, product, product};
        System.out.println(calculateTotalPrice(products));
    }

    public static int calculateTotalPrice(Product[] items) {
        int total = 0;
        for (Product item : items) {
            total += item.getPrice(); // Alterado de "+" para "+=" para acumul
        }
        return total;
    }
}

class Product {
    private String name;
    private int price; // Alterado o tipo de dado para int

    public Product(String name, int price) { // Alterado o parâmetro para int
        this.name = name;
        this.price = price;
    }

    public int getPrice() { // Alterado o tipo de retorno para int
        return price;
    }
}
```

### 5.4.2. Node.js

Código Original

```
javascript
function calculateTotalPrice(items) {
  var total = 0
  for (item of items) {
    total + item.price
  }
  return total
}

const product = {name: 'Laptop', price: '1200'}
const products = [product, product, product]

console.log(calculateTotalPrice(products))
```

Explicação:

- **Erro na Declaração de Variável:** Adicionado "const" para declarar a variável "item" no loop for.
- **Erro na Acumulação de Total:** A expressão "total + item.price" foi alterada para "total += item.price" para acumular o valor no total.
- **Tipo de Dado Incorreto:** O preço foi alterado de string para número. Isto é necessário para realizar a soma correta.
- **Sintaxe Incorreta:** Adicionado o parêntesis fechando na chamada da função console.log.

Código Corrigido e Explicação

```
javascript
function calculateTotalPrice(items) {
  var total = 0;
  for (const item of items) { // Adicionado "const" para declarar a variável
    total += item.price; // Alterado de "+" para "+=" para acumular o valor no total
  }
  return total;
}

const product = {name: 'Laptop', price: 1200}; // Alterado o preço de string para número
const products = [product, product, product];

console.log(calculateTotalPrice(products)); // Adicionado o parêntesis fechando a chamada
```

### 5.4.3. Python

## Código Original

```
python                                         Copy code

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

def calculate_total_price(products):
    total = 0
    for product in products:
        total + product.price
    return total

product = Product("Laptop", "1200")
products = [product] * 3
print(calculate_total_price(products))
```

Explicação:

- **Erro na Acumulação de Total:** A expressão "total + product.price" foi alterada para "total += product.price" para acumular o valor no total.
- **Tipo de Dado Incorreto:** O preço foi inicializado como uma string, mas para a operação de adição, ele deve ser um inteiro. Fiz a correção para aceitar um inteiro, mas também poderia ser corrigido convertendo a string para um inteiro no construtor da classe Produto.

## Código Corrigido e Explicação

```
python                                         Copy code

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = int(price) # Converte o preço para inteiro

    def calculate_total_price(self, products):
        total = 0
        for product in products:
            total += product.price # Alterado de "+" para "+=" para acumular o v
        return total

product = Product("Laptop", 1200) # Alterado o preço para int
products = [product] * 3
print(calculate_total_price(product, products))
```

## 5.5. Public static void main(String args): erros

Você receberá um trecho de código-fonte em Java (versão 20.0.2) com vários erros. Identifique e solucione os erros, detalhando cada um e apresentando a solução necessária.

### 5.5.1. Java

Trecho de Código-Fonte com Erros (Java):

```
java

public clas Main {
    pbublic static void main(String args) {
        int x == 5;
        float y = 10.5

        Systm.out.println("A soma de x e y é: " + x + y);
        System.out.println(A soma de x e y é: " + x + y);
    }
}
```

Erros e Soluções:

- **Erro:** A palavra-chave class está digitada incorretamente como clas.  
**Solução:** Corrija a palavra-chave clas para class.
- **Erro:** A palavra-chave public está digitada incorretamente como pbuplic.  
**Solução:** Corrija a palavra-chave pbuplic para public.
- **Erro:** A variável args no método main está faltando os colchetes [].  
**Solução:** Corrija o parâmetro para String[] args.
- **Erro:** Uso incorreto do operador de comparação == na declaração da variável x.  
**Solução:** Mude o operador de comparação == para o operador de atribuição =.
- **Erro:** Falta um ponto e vírgula na declaração da variável y.  
**Solução:** Adicione um ponto e vírgula ao final da linha.
- **Erro:** A palavra-chave System está escrita incorretamente como Systm.  
**Solução:** Corrija para System.
- **Erro:** A string literal no segundo System.out.println está fora das aspas.  
**Solução:** Coloque a string literal entre aspas.

#### Trecho de Código-Fonte Corrigido:

```
java

public class Main {
    public static void main(String[] args) {
        int x = 5;
        float y = 10.5f;

        System.out.println("A soma de x e y é: " + (x + y));
        System.out.println("A soma de x e y é: " + (x + y));
    }
}
```

#### 5.5.2. Node.js

#### Trecho de Código-Fonte com Erros (Node.js):

```
javascript

const addNumbers = functin(a, b) {
    retun a + b
};

console.log('The sum of 5 and 10 is: ' addNumbers(5, 10);
```

#### Erros e Soluções:

- **Erro:** A palavra-chave `function` está digitada incorretamente como `functin`.  
**Solução:** Corrija a palavra-chave `functin` para `function`.
- **Erro:** A palavra-chave `return` está digitada incorretamente como `retun`.  
**Solução:** Corrija a palavra-chave `retun` para `return`.
- **Erro:** Falta um ponto e vírgula na declaração da função `addNumbers`.  
**Solução:** Adicione um ponto e vírgula ao final da linha.
- **Erro:** Na linha `console.log`, falta uma vírgula para separar os argumentos.  
**Solução:** Adicione uma vírgula entre a string literal e a chamada de função.
- **Erro:** Falta um parêntese fechando na chamada de `console.log`.  
**Solução:** Adicione um parêntese fechando após a chamada de função.

#### Trecho de Código-Fonte Corrigido:

```
javascript

const addNumbers = function(a, b) {
    return a + b;
};

console.log('The sum of 5 and 10 is:', addNumbers(5, 10));
```

#### 5.5.3. Python

#### Trecho de Código-Fonte com Erros (Python):

```
python

class MathOperations:
    def __init__(self, value1, value2)
        self.value1 = value1
        self.value2: value2

    def add(self)
        return self.value1 + value2

    def subtract(self):
        return self.value1 - self.value2

math_obj = MathOperations(10, 5
print("Addition:", math_obj.add())
print("Subtraction:", math_obj.subtract())
```

#### Erros e Soluções:

- **Erro:** Falta de dois pontos (:) após a definição do método `__init__`.  
**Solução:** Adicionar dois pontos após a definição do método.
- **Erro:** A atribuição `self.value2: value2` está errada; o operador de atribuição é `=`.  
**Solução:** Substituir o dois pontos por um sinal de igual.
- **Erro:** Falta de dois pontos (:) após a definição do método `add`.  
**Solução:** Adicionar dois pontos após a definição do método.
- **Erro:** Uso de `value2` em vez de `self.value2` no método `add`.  
**Solução:** Substituir `value2` por `self.value2`.
- **Erro:** Parênteses mal formados na definição do método `subtract`.  
**Solução:** Corrigir a abertura e fechamento dos parênteses.

- **Erro:** Parênteses mal formados na criação do objeto MathOperations.
- **Solução:** Adicionar um parêntese de fechamento.
- **Erro:** Falta de parêntese de fechamento no método print.
- **Solução:** Adicionar parêntese de fechamento.
- **Erro:** Uso incorreto do nome de método subtract em vez de subtract.
- **Solução:** Corrigir o nome do método.

#### Trecho de Código-Fonte Corrigido:

```
python

class MathOperations:
    def __init__(self, value1, value2):
        self.value1 = value1
        self.value2 = value2

    def add(self):
        return self.value1 + self.value2

    def subtract(self):
        return self.value1 - self.value2

math_obj = MathOperations(10, 5)
print("Addition:", math_obj.add())
print("Subtraction:", math_obj.subtract())
```

## 6. Refatoração e Teste Unitário:

### 6.1. Calculadora:

#### 6.1.1. Calculadora (REFATORAR):

Você receberá um código-fonte em Python (versão 3.11.4) que implementa uma calculadora simples. Refatore este código com o objetivo de melhorá-lo sem mudar sua funcionalidade.

##### 6.1.1.1. Java

```
Final > J4V4 > Calculator15A > SimpleCalculator.java > SimpleCalculator
1 //CÓDIGO ORIGINAL ANTES DA REFATORAÇÃO
2
3 package Final.J4V4.Calculator15A;
4
5 import java.util.Scanner;
6
7 public class SimpleCalculator {
8     public static double calculator(double a, double b, String operation) {
9         switch (operation) {
10             case "add":
11                 return a + b;
12             case "subtract":
13                 return a - b;
14             case "multiply":
15                 return a * b;
16             case "divide":
17                 if (b != 0) {
18                     return a / b;
19                 } else {
20                     System.out.println("Cannot divide by zero");
21                     return 0;
22                 }
23             default:
24                 System.out.println("Invalid operation");
25                 return 0;
26         }
27     }
}
```

```

29     Run | Debug
30     public static void main(String[] args) {
31         Scanner scanner = new Scanner(System.in);
32         System.out.print("Enter first number: ");
33         double a = scanner.nextDouble();
34
35         System.out.print("Enter second number: ");
36         double b = scanner.nextDouble();
37
38         System.out.print("Enter operation (add, subtract, multiply, divide): ");
39         String operation = scanner.next();
40
41         double result = calculator(a, b, operation);
42         System.out.println("Result: " + result);
43
44         scanner.close();
45     }
46 }
```

As mudanças incluem:

- **Uso de Enumeração:** Utilização de uma enumeração para representar as operações aritméticas.
- **Uso de Interfaces Funcionais:** Utilização de uma interface funcional BiFunction para representar cada operação, facilitando a extensão do código.
- **Tratamento de Erros:** Utilização de uma abordagem de tratamento de erro para lidar com operações inválidas.

```

Final > J4V4 > Calculator15A > J SimpleCalculator2.java > SimpleCalculator2
1 // CÓDIGO REFATORADO !!!
2
3 package Final.J4V4.Calculator15A;
4
5 import java.util.Scanner;
6 import java.util.function.BiFunction;
7
8 public class SimpleCalculator2 {
9     enum Operation {
10         ADD(Double::sum),
11         SUBTRACT((a, b) -> a - b),
12         MULTIPLY((a, b) -> a * b),
13         DIVIDE((a, b) -> b != 0 ? a / b : Double.NaN);
14
15     private final BiFunction<Double, Double, Double> operation;
16
17     Operation(BiFunction<Double, Double, Double> operation) {
18         this.operation = operation;
19     }
20
21     public double apply(double a, double b) {
22         return operation.apply(a, b);
23     }
24 }
```

```
Run | Debug
24  public static void main(String[] args) {
25      Scanner scanner = new Scanner(System.in);
26
27      System.out.print("Enter first number: ");
28      double a = scanner.nextDouble();
29
30      System.out.print("Enter second number: ");
31      double b = scanner.nextDouble();
32
33      System.out.print("Enter operation (add, subtract, multiply, divide): ");
34      String operationInput = scanner.next().toUpperCase();
35
36      try {
37          Operation operation = Operation.valueOf(operationInput);
38          double result = operation.apply(a, b);
39          if (Double.isNaN(result)) {
40              System.out.println("Cannot divide by zero");
41          } else {
42              System.out.println("Result: " + result);
43          }
44      } catch (IllegalArgumentException e) {
45          System.out.println("Invalid operation");
46      }
47
48      scanner.close();
49  }
50 }
```

### 6.1.1.2. Node.js

```
Final > Node > JS 15ACalculadora.js > ...
1 //CÓDIGO ORIGINAL ANTES DA REFATORAÇÃO
2 const readline = require('readline');
3
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout
7 });
8
9 function calculator(a, b, operation) {
10   switch (operation) {
11     case 'add':
12       return a + b;
13     case 'subtract':
14       return a - b;
15     case 'multiply':
16       return a * b;
17     case 'divide':
18       if (b !== 0) {
19         return a / b;
20       } else {
21         console.log('Cannot divide by zero');
22         return 0;
23       }
24     default:
25       console.log('Invalid operation');
26       return 0;
27   }
28 }
29
30 rl.question('Enter first number: ', (a) => {
31   rl.question('Enter second number: ', (b) => {
32     rl.question('Enter operation (add, subtract, multiply, divide): ', (operation) => {
33       const result = calculator(parseFloat(a), parseFloat(b), operation);
34       console.log('Result:', result);
35       rl.close();
36     });
37   });
38 });
```

#### Mudanças Realizadas:

- **Uso de Objeto para Operações:** As operações são armazenadas em um objeto onde a chave é o nome da operação e o valor é a função correspondente. Isso torna o código mais conciso e fácil de estender.
- **Verificação de Operação:** A refatoração inclui uma verificação para garantir que a operação inserida é válida. Se a operação não for encontrada no objeto operations, uma mensagem de erro é exibida.
- **Controle de Divisão por Zero:** A divisão por zero é tratada diretamente na função de divisão, retornando uma mensagem de erro se o divisor for zero.

```
Final > Node > JS 15ACalculadora_DEPOIS.js > ...
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const operations = {
9    add: (a, b) => a + b,
10   subtract: (a, b) => a - b,
11   multiply: (a, b) => a * b,
12   divide: (a, b) => b !== 0 ? a / b : 'Cannot divide by zero'
13 };
14
```

```
15  rl.question('Enter first number: ', (a) => {
16    rl.question('Enter second number: ', (b) => {
17      rl.question('Enter operation (add, subtract, multiply, divide): ', (operation) => {
18        const func = operations[operation];
19        if (func) {
20          const result = func(parseFloat(a), parseFloat(b));
21          console.log('Result:', result);
22        } else {
23          console.log('Invalid operation');
24        }
25        rl.close();
26      });
27    });
28  });
29
```

### 6.1.1.3. Python

```
Final > Python > 15Calculadora_ANTES.py > ...
1 # ANTES DA REFATORAÇÃO:
2
3 def calculator(a, b, operation):
4     if operation == 'add':
5         return a + b
6     elif operation == 'subtract':
7         return a - b
8     elif operation == 'multiply':
9         return a * b
10    elif operation == 'divide':
11        if b != 0:
12            return a / b
13        else:
14            return "Cannot divide by zero"
15    else:
16        return "Invalid operation"
17
18 a = float(input("Enter first number: "))
19 b = float(input("Enter second number: "))
20 operation = input("Enter operation (add, subtract, multiply, divide): ")
21
22 result = calculator(a, b, operation)
23 print("Result:", result)
```

```
Enter first number: 17
Enter second number: 25
Enter operation (add, subtract, multiply, divide): add
Result: 42.0
```

As mudanças aqui incluem:

- **Uso de Lambda Functions:** Utilização de expressões lambda para representar as operações aritméticas, tornando o código mais conciso.
- **Uso de Dicionário:** Utilização de um dicionário para mapear as operações, facilitando a extensão para novas operações no futuro.
- **Separando a Entrada:** Criação de uma função separada para obter a entrada do usuário, tornando o código mais modular e testável.

```
Final > Python > 15Calculadora_DEPOIS.py > ...
1  def calculator(a, b, operation):
2      operations = {
3          'add': lambda x, y: x + y,
4          'subtract': lambda x, y: x - y,
5          'multiply': lambda x, y: x * y,
6          'divide': lambda x, y: x / y if y != 0 else "Cannot divide by zero",
7      }
8
9      return operations.get(operation, "Invalid operation")(a, b)
10
11 def get_input():
12     a = float(input("Enter first number: "))
13     b = float(input("Enter second number: "))
14     operation = input("Enter operation (add, subtract, multiply, divide): ")
15     return a, b, operation
16
17 a, b, operation = get_input()
18 result = calculator(a, b, operation)
19 print("Result:", result)
```

```
Enter first number: 37
Enter second number: 5
Enter operation (add, subtract, multiply, divide): add
Result: 42.0
```

## 6.1.2. Calculadora (TESTE UNITÁRIO):

Elabore o teste unitário do código-fonte refatorado, utilizando a biblioteca de sua escolha, para garantir que as funcionalidades originais foram preservadas após a refatoração.

### 6.1.2.1. Python

```
Final > Python > 15BCalculadora.py > ...
1  def calculator(a, b, operation):
2      operations = {
3          "add": lambda x, y: x + y,
4          "subtract": lambda x, y: x - y,
5          "multiply": lambda x, y: x * y,
6          "divide": lambda x, y: x / y if y != 0 else "Cannot divide by zero"
7      }
8
9      func = operations.get(operation)
10
11     if func:
12         return func(a, b)
13     else:
14         return "Invalid operation"
15
16 import unittest
17
18 class TestCalculator(unittest.TestCase):
19
20     def test_add(self):
21         self.assertEqual(calculator(5, 3, "add"), 8)
22
23     def test_subtract(self):
24         self.assertEqual(calculator(9, 4, "subtract"), 5)
25
26     def test_multiply(self):
27         self.assertEqual(calculator(3, 3, "multiply"), 9)
28
29     def test_divide(self):
30         self.assertEqual(calculator(10, 2, "divide"), 5)
31         self.assertEqual(calculator(10, 0, "divide"), "Cannot divide by zero")
32
33     def test_invalid_operation(self):
34         self.assertEqual(calculator(5, 3, "modulus"), "Invalid operation")
35
36 if __name__ == "__main__":
37     unittest.main()
```

```
.....
-----
Ran 5 tests in 0.000s
OK
```

## 6.2. Gerenciador Impressora

### 6.2.1. Impressora (REFATORAR):

Você receberá um código-fonte em Python (versão 3.11.4) que realiza operações em uma fila de impressão. Sua tarefa é refatorar o código para melhorá-lo sem alterar sua funcionalidade original.

#### 6.2.1.1. Java

```
Final > J4V4 > Printer24 > J PrinterQueue.java > PrinterQueue
1 package Final.J4V4.Printer24;
2
3 // 2.4 - PRINTER - CÓDIGO ORIGINAL
4
5 import java.util.LinkedList;
6 import java.util.Queue;
7
8 public class PrinterQueue {
9     private Queue<String> queue;
10
11     public PrinterQueue() {
12         this.queue = new LinkedList<>();
13     }
14
15     public void addJob(String job) {
16         queue.add(job);
17     }
18
19     public void printNext() {
20         System.out.println("Printing: " + queue.poll());
21     }
22
23     public void viewQueue() {
24         System.out.println("Queue: " + queue);
25     }
```

```
Queue: [Document1, Document2]
Printing: Document1
Queue: [Document2]
```

#### Melhorias Realizadas

- Validação de Entrada:** Adicionada verificação para garantir que o job não seja vazio, nulo ou composto apenas de espaços em branco antes de adicioná-lo à fila.
- Melhoria nas Mensagens de Saída:** As mensagens de saída agora são mais informativas e específicas, incluindo o nome do job que está sendo processado.
- Verificação da Fila Vazia:** Adicionadas condições para lidar com a situação quando a fila está vazia, tanto para imprimir o próximo job quanto para visualizar a fila.

```

Final > J4V4 > Printer24 > J PrinterQueue2.java > PrinterQueue2
 1 package Final.J4V4.Printer24;
 2
 3 // 2.4 - PRINTER: APÓS REFATORAÇÃO
 4
 5 import java.util.LinkedList;
 6 import java.util.Queue;
 7
 8 public class PrinterQueue2 {
 9     private Queue<String> queue;
10
11     public PrinterQueue2() {
12         this.queue = new LinkedList<>();
13     }
14
15     public void addJob(String job) {
16         if (job != null && !job.trim().isEmpty()) {
17             queue.add(job);
18             System.out.println("Job '" + job + "' added to the queue!");
19         } else {
20             System.out.println("Invalid job. Please add a valid job.");
21         }
22     }
23
24     public void printNext() {
25         if (!queue.isEmpty()) {
26             System.out.println("Printing the job: '" + queue.poll() + "'");
27         } else {
28             System.out.println("No job in the queue!");
29         }
30     }
31
32     public void viewQueue() {
33         if (!queue.isEmpty()) {
34             System.out.println("Print Queue:");
35             queue.forEach(System.out::println);
36         } else {
37             System.out.println("Print queue is empty.");
38         }
39     }
40
41     Run | Debug
42     public static void main(String[] args) {
43         PrinterQueue2 printerQueue2 = new PrinterQueue2();
44         printerQueue2.addJob("Document1"); // Output: Job 'Document1' added to the queue!
45         printerQueue2.addJob("Document2"); // Output: Job 'Document2' added to the queue!
46         printerQueue2.viewQueue(); // Output: Print Queue: Document1, Document2
47         printerQueue2.printNext(); // Output: Printing the job: 'Document1'
48         printerQueue2.viewQueue(); // Output: Print Queue: Document2
49     }
}

```

```

Job 'Document1' added to the queue!
Job 'Document2' added to the queue!
Print Queue:
Document1
Document2
Printing the job: 'Document1'
Print Queue:
Document2

```

### 6.2.1.2. Node.js

```
Final > Node > js 24Printer.js > ...
1  // 2.4 CÓDIGO ORIGINAL
2  class PrinterQueue {
3      constructor() {
4          this.queue = [];
5      }
6
7      addJob(job) {
8          this.queue.push(job);
9      }
10
11     printNext() {
12         console.log('Printing:', this.queue.shift());
13     }
14
15     viewQueue() {
16         console.log('Queue:', this.queue);
17     }
18 }
19
20 // Exemplo de Utilização
21 const printerQueue = new PrinterQueue();
22 printerQueue.addJob('Document1');
23 printerQueue.addJob('Document2');
24 printerQueue.viewQueue(); // Saída: Queue: [ 'Document1', 'Document2' ]
25 printerQueue.printNext(); // Saída: Printing: Document1
26 printerQueue.viewQueue(); // Saída: Queue: [ 'Document2' ]
```

```
Queue: [ 'Document1', 'Document2' ]
Printing: Document1
Queue: [ 'Document2' ]
```

## Melhorias Realizadas

- **Validação de Entrada:** Adicionada verificação para garantir que o job não seja vazio ou undefined antes de adicioná-lo à fila.
- **Melhoria nas Mensagens de Saída:** As mensagens de saída agora são mais informativas e específicas, incluindo o nome do job que está sendo processado.
- **Verificação da Fila Vazia:** Adicionadas condições para lidar com a situação quando a fila está vazia, tanto para imprimir o próximo job quanto para visualizar a fila.

```
Final > Node > JS 24Printer_REFATORADO.js > PrinterQueue2
1  // 2.4 PRINTER - CÓDIGO REFATORADO
2
3  class PrinterQueue2 {
4      constructor() {
5          this.queue = [];
6      }
7
8      addJob(job) {
9          if (job) {
10              this.queue.push(job);
11              console.log(`Job '${job}' added to the queue!`);
12          } else {
13              console.log("Invalid job. Please add a valid job.");
14          }
15      }
16
17      printNext() {
18          if (this.queue.length > 0) {
19              console.log(`Printing the job: '${this.queue.shift()}'`);
20          } else {
21              console.log("No job in the queue!");
22          }
23      }
}
```

```
17     printNext() {
18         if (this.queue.length > 0) {
19             console.log(`Printing the job: ${this.queue.shift()}`);
20         } else {
21             console.log("No job in the queue!");
22         }
23     }
24
25     viewQueue() {
26         if (this.queue.length > 0) {
27             console.log("Print Queue:");
28             this.queue.forEach(job => console.log(job));
29         } else {
30             console.log("Print queue is empty.");
31         }
32     }
33 }
34
35 // Exemplo de Utilização
36 const printerQueue2 = new PrinterQueue2();
37 printerQueue2.addJob('Document1'); // Saída: Job 'Document1' added to the queue!
38 printerQueue2.addJob('Document2'); // Saída: Job 'Document2' added to the queue!
39 printerQueue2.viewQueue();        // Saída: Print Queue: Document1, Document2
40 printerQueue2.printNext();       // Saída: Printing the job: 'Document1'
41 printerQueue2.viewQueue();       // Saída: Print Queue: Document2
```

```
Job 'Document1' added to the queue!
Job 'Document2' added to the queue!
Print Queue:
Document1
Document2
Printing the job: 'Document1'
Print Queue:
Document2
|
```

### 6.2.1.3. Python

```
Final > Python > 24PrinterQueue.py > ...
1  # 2.4 PRINTER QUEUE
2  # CÓDIGO ORIGINAL ANTES DA REFATORAÇÃO
3
4  class PrinterQueue:
5      def __init__(self):
6          self.queue = []
7
8      def add_job(self, job):
9          self.queue.append(job)
10         print("Job adicionado!")
11
12     def print_next(self):
13         if len(self.queue) > 0:
14             job = self.queue.pop(0)
15             print("Imprimindo o job:", job)
16         else:
17             print("Nenhum job na fila!")
18
19     def view_queue(self):
20         print("Fila de impressão:")
21         for job in self.queue:
22             print(job)
23
24     # Exemplo de uso
25     printer_queue = PrinterQueue()
26     printer_queue.add_job("Documento 1")
27     printer_queue.add_job("Documento 2")
28     printer_queue.view_queue()
29     printer_queue.print_next()
30     printer_queue.print_next()
31     printer_queue.print_next()
```

```
Job adicionado!
Job adicionado!
Fila de impressão:
Documento 1
Documento 2
Imprimindo o job: Documento 1
Imprimindo o job: Documento 2
Nenhum job na fila!
```

## Melhorias Realizadas:

- **Uso de deque:** Troquei a lista pelo deque, que é uma estrutura de dados feita para filas e pilhas. Isso melhora a eficiência de adicionar e remover elementos do início da fila.
- **Validação de Entrada:** Adicionei uma verificação para garantir que o job não seja vazio ou None antes de adicioná-lo à fila.
- **Melhoria nas Mensagens de Saída:** As mensagens de saída agora são mais informativas e específicas, incluindo o nome do job que está sendo processado.
- **Verificação da Fila Vazia:** Adicionei condições para lidar com a situação quando a fila está vazia, tanto para imprimir o próximo job quanto para visualizar a fila.

```
Final > Python > 24PrinterQueue2_REFATORADO.py > PrinterQueue2
1  # 2.4 CÓDIGO REFATORADO
2
3  from collections import deque
4
5  class PrinterQueue2:
6      def __init__(self):
7          self.queue = deque()
8
9      def add_job(self, job):
10         if job:
11             self.queue.append(job)
12             print(f"Job '{job}' adicionado na fila!")
13         else:
14             print("Job inválido. Por favor, adicione um job válido.")
15
16     def print_next(self):
17         if self.queue:
18             job = self.queue.popleft()
19             print(f"Imprimindo o job: '{job}'")
20         else:
21             print("Nenhum job na fila!")
22
23     def view_queue(self):
24         if self.queue:
25             print("Fila de impressão:")
26             for job in self.queue:
27                 print(job)
28         else:
29             print("Fila de impressão vazia.")
```

```
31 # Exemplo de uso
32 printer_queue = PrinterQueue2()
33 printer_queue.add_job("Documento 1")
34 printer_queue.add_job("Documento 2")
35 printer_queue.view_queue()
36 printer_queue.print_next()
37 printer_queue.print_next()
38 printer_queue.print_next()
```

```
Job 'Documento 1' adicionado na fila!
Job 'Documento 2' adicionado na fila!
Fila de impressão:
Documento 1
Documento 2
Imprimindo o job: 'Documento 1'
Imprimindo o job: 'Documento 2'
Nenhum job na fila!
```

## 6.2.2. Impressora (TESTE UNITÁRIO):

Utilizando Python (versão 3.11.4), elabore testes unitários para um código-fonte fornecido que implementa um sistema de agendamento. Certifique-se de que os testes cubram todas as funcionalidades principais.

### 6.2.2.1. Python

```
Final > Python > 25bScheduler.py > ...
1  class Scheduler:
2      def __init__(self):
3          self.appointments = []
4
5      def add_appointment(self, date, description):
6          self.appointments.append({"date": date, "description": description})
7
8      def remove_appointment(self, index):
9          self.appointments.pop(index)
10
11     def get_appointment(self, index):
12         return self.appointments[index]
13
14     def list_appointments(self):
15         return self.appointments
16
17
18     import unittest
19
20     class SchedulerTest(unittest.TestCase):
21
22         def setUp(self):
23             self.scheduler = Scheduler()
24
25         def test_add_appointment(self):
26             self.scheduler.add_appointment('2023-08-16', 'Meeting')
27             self.assertEqual(self.scheduler.get_appointment(0), {'date': '2023-08-16', 'description': 'Meeting'})
28
29         def test_remove_appointment(self):
30             self.scheduler.add_appointment('2023-08-16', 'Meeting')
31             self.scheduler.remove_appointment(0)
32             self.assertEqual(self.scheduler.list_appointments(), [])
33
34         def test_list_appointments(self):
35             self.scheduler.add_appointment('2023-08-16', 'Meeting')
36             self.scheduler.add_appointment('2023-08-17', 'Conference')
37             self.assertEqual(self.scheduler.list_appointments(), [
38                 {'date': '2023-08-16', 'description': 'Meeting'},
39                 {'date': '2023-08-17', 'description': 'Conference'}
40             ])
41
42         def test_get_appointment(self):
43             self.scheduler.add_appointment('2023-08-16', 'Meeting')
44             self.assertEqual(self.scheduler.get_appointment(0), {'date': '2023-08-16', 'description': 'Meeting'})
45
46     if __name__ == '__main__':
47         unittest.main()
```

```
.....
-----
Ran 4 tests in 0.000s

OK
```

## 6.3. Reserva Hotel

### 6.3.1. Hotel (REFATORAÇÃO):

Receberá um código-fonte em Java (versão 20.0.2) que implementa um sistema de reservas de hotel. Refatore este código para melhorá-lo sem mudar sua funcionalidade.

#### 6.3.1.1. Java

```
Final > J4V4 > Hotel > HotelReservation.java > HotelReservation
1 package Final.J4V4.Hotel;
2
3 class HotelReservation {
4     public String bookRoom(String roomType, int days, boolean isVip) {
5         String result;
6         double price = 0.0;
7
8         if (roomType.equals("Single")) {
9             price = 100;
10        } else if (roomType.equals("Double")) {
11            price = 200;
12        } else if (roomType.equals("Suite")) {
13            price = 300;
14        }
15
16        if (isVip) {
17            price = price * 0.9; // Desconto para VIP
18        }
19
20        price = price * days;
21
22        if (price > 0) {
23            result = "Reserva efetuada: " + roomType + " por " + days + " dias. Preço: " + price;
24        } else {
25            result = "Tipo de quarto inválido.";
26        }
27
28        return result;
29    }
30
31
32    public static void main(String[] args) {
33        HotelReservation reservation = new HotelReservation();
34        System.out.println(reservation.bookRoom("Single", 3, true));
35    }
36}
```

Modificações:

- **Constantes:** As taxas para diferentes tipos de quartos e o desconto VIP foram movidos para constantes para melhorar a legibilidade e tornar a manutenção mais fácil.
- **Método Privado:** Criei um método privado calculatePrice para separar a lógica de cálculo do preço. Isso torna o código mais modular e mais fácil de testar.
- **Switch em Vez de If-Else:** Substituí a estrutura de if-else por uma instrução switch para melhorar a legibilidade na seleção do tipo de quarto.
- **Operador de Multiplicação Compacto:** Utilizei o operador \*= no cálculo do desconto VIP para tornar o código um pouco mais conciso.

```
Final > J4V4 > Hotel > J HotelReservation2.java >  HotelReservation2
1 package Final.J4V4.Hotel;
2
3 class HotelReservation2 {
4     private static final double SINGLE_ROOM_PRICE = 100.0;
5     private static final double DOUBLE_ROOM_PRICE = 200.0;
6     private static final double SUITE_ROOM_PRICE = 300.0;
7     private static final double VIP_DISCOUNT = 0.9;
8
9     public String bookRoom(String roomType, int days, boolean isVip) {
10         double price = calculatePrice(roomType, isVip);
11         if (price <= 0) {
12             return "Tipo de quarto inválido.";
13         }
14         return "Reserva efetuada: " + roomType + " por " + days + " dias. Preço: " + (price * days);
15     }
16
17     private double calculatePrice(String roomType, boolean isVip) {
18         double price = 0.0;
19         switch (roomType) {
20             case "Single":
21                 price = SINGLE_ROOM_PRICE;
22                 break;
23             case "Double":
24                 price = DOUBLE_ROOM_PRICE;
25                 break;
26             case "Suite":
27                 price = SUITE_ROOM_PRICE;
28                 break;
29         }
30         if (isVip) {
31             price *= VIP_DISCOUNT;
32         }
33         return price;
34     }
35
36     public static void main(String[] args) {
37         HotelReservation2 reservation = new HotelReservation2();
38         System.out.println(reservation.bookRoom("Single", 3, true));
39     }
40 }
```

```
Reserva efetuada: Single por 3 dias. Preço: 270.0
```

### 6.3.1.2. Node.js

```
Final > Node > JS 35Hotel.js > ...
1  //3.5 Hotel
2
3  class HotelReservation {
4      bookRoom(roomType, days, isVip) {
5          let price = 0;
6          if (roomType === "Single") {
7              price = 100;
8          } else if (roomType === "Double") {
9              price = 200;
10         } else if (roomType === "Suite") {
11             price = 300;
12         } else {
13             return "Tipo de quarto inválido.";
14         }
15         if (isVip) {
16             price *= 0.9;
17         }
18         return "Reserva efetuada: " + roomType + " por " + days + " dias. Preço: " + (price * days);
19     }
20 }
21
22 const reservation = new HotelReservation();
23 console.log(reservation.bookRoom("Single", 3, true));
```

```
Reserva efetuada: Single por 3 dias. Preço: 270
```

#### Modificações:

- **Constantes:** Defini as taxas para diferentes tipos de quartos e o desconto VIP como propriedades estáticas da classe para melhorar a legibilidade.
- **Método Privado:** Criei um método separado calculatePrice para calcular o preço. Isso torna o código mais modular e facilita a manutenção.
- **Switch em Vez de If-Else:** Alterei a estrutura de if-else por uma instrução switch para melhorar a legibilidade.
- **Template Strings:** Utilizei template strings para a concatenação de strings, o que torna o código mais legível.

```
Final > Node > JS 35Hotel_Refatorado.js > HotelReservation2
1  //3.5 Hotel
2
3  //CÓDIGO REFATORADO
4
5  class HotelReservation2 {
6      static SINGLE_ROOM_PRICE = 100;
7      static DOUBLE_ROOM_PRICE = 200;
8      static SUITE_ROOM_PRICE = 300;
9      static VIP_DISCOUNT = 0.9;
10
11     bookRoom(roomType, days, isVip) {
12         let price = this.calculatePrice(roomType, isVip);
13         if (price <= 0) {
14             return "Tipo de quarto inválido.";
15         }
16         return `Reserva efetuada: ${roomType} por ${days} dias. Preço: ${price * days}`;
17     }
18
19     calculatePrice(roomType, isVip) {
20         let price = 0;
21         switch (roomType) {
22             case "Single":
23                 price = HotelReservation2.SINGLE_ROOM_PRICE;
24                 break;
25             case "Double":
26                 price = HotelReservation2.DOUBLE_ROOM_PRICE;
27                 break;
28             case "Suite":
29                 price = HotelReservation2.SUITE_ROOM_PRICE;
30                 break;
31         }
32         if (isVip) {
33             price *= HotelReservation2.VIP_DISCOUNT;
34         }
35         return price;
36     }
37 }
38
39 const reservation = new HotelReservation2();
40 console.log(reservation.bookRoom("Single", 3, true));
41
```

### 6.3.1.3. Python

```
Final > Python > 35Hotel.py > ...
1  #3.5 HOTEL ORIGINAL - ANTES DA REFATORAÇÃO
2
3  class HotelReservation:
4      def book_room(self, room_type, days, is_vip):
5          price = 0
6          if room_type == "Single":
7              price = 100
8          elif room_type == "Double":
9              price = 200
10         elif room_type == "Suite":
11             price = 300
12         else:
13             return "Tipo de quarto invalido."
14
15         if is_vip:
16             price *= 0.9
17
18         return f"Reserva efetuada: {room_type} por {days} dias. Preco: {price * days}"
19
20 reservation = HotelReservation()
21 print(reservation.book_room("Single", 3, True))
22
```

#### Modificações:

- **Constantes:** Defini as taxas para diferentes tipos de quartos e o desconto VIP como propriedades de classe para melhorar a legibilidade.
- **Método Privado:** Criei um método separado calculate\_price para calcular o preço, tornando o código mais modular.
- **Utilização de Dicionário:** Substituí a lógica de if-else por um dicionário para mapear os tipos de quarto aos seus preços, melhorando a legibilidade e escalabilidade do código.

```
Final > Python > 35Hotel_REFATORADO.py > ...
1 #3.5 HOTEL REFATORADO
2
3 class HotelReservation2:
4     SINGLE_ROOM_PRICE = 100
5     DOUBLE_ROOM_PRICE = 200
6     SUITE_ROOM_PRICE = 300
7     VIP_DISCOUNT = 0.9
8
9     def book_room(self, room_type, days, is_vip):
10         price = self.calculate_price(room_type, is_vip)
11         if price <= 0:
12             return "Tipo de quarto inválido."
13         return f"Reserva efetuada: {room_type} por {days} dias. Preço: {price * days}"
14
15     def calculate_price(self, room_type, is_vip):
16         price_mapping = {
17             "Single": HotelReservation2.SINGLE_ROOM_PRICE,
18             "Double": HotelReservation2.DOUBLE_ROOM_PRICE,
19             "Suite": HotelReservation2.SUITE_ROOM_PRICE
20         }
21         price = price_mapping.get(room_type, 0)
22         if is_vip:
23             price *= HotelReservation2.VIP_DISCOUNT
24         return price
25
26 reservation = HotelReservation2()
27 print(reservation.book_room("Single", 3, True))
28
```

### 6.3.2. Hotel (TESTE UNITÁRIO):

Utilizando Python (versão 3.11.4), elabore os testes unitários para um código-fonte fornecido que gere uma árvore genealógica. Garanta que os testes verifiquem a correta funcionalidade das principais características do código.

#### 6.3.2.1. Python

```
Final > Python > 35B_Arvore_Genealogica.py > ...
1 #3.5B ARVORE GENEALÓGICA
2
3 class Person:
4     def __init__(self, name, parent=None):
5         self.name = name
6         self.parent = parent
7         self.children = []
8
9     def add_child(self, child):
10        self.children.append(child)
11        child.parent = self
12
13    def get_ancestors(self):
14        ancestors = []
15        current_person = self.parent
16        while current_person:
17            ancestors.append(current_person.name)
18            current_person = current_person.parent
19        return ancestors
20
```

#### Explicação

- **Classe Person:** A classe representa uma pessoa com um nome, referência ao pai e uma lista de filhos.
- **Método add\_child:** Adiciona uma criança à lista de filhos e define o pai da criança.
- **Método get\_ancestors:** Retorna uma lista com os nomes dos ancestrais.

```
22 import unittest
23
24 class TestPerson(unittest.TestCase):
25     def test_add_child(self):
26         parent = Person("Parent")
27         child = Person("Child")
28         parent.add_child(child)
29
30         self.assertEqual(child.parent, parent)
31         self.assertIn(child, parent.children)
32
33     def test_get_ancestors(self):
34         grandparent = Person("Grandparent")
35         parent = Person("Parent", grandparent)
36         child = Person("Child", parent)
37
38         self.assertEqual(child.get_ancestors(), ["Parent", "Grandparent"])
39         self.assertEqual(parent.get_ancestors(), ["Grandparent"])
40         self.assertEqual(grandparent.get_ancestors(), [])
41
42 if __name__ == '__main__':
43     unittest.main()
```

```
--
```

```
Ran 2 tests in 0.000s
```

```
OK
```

#### Testes Unitários:

- **test\_add\_child:** Verifica se o método add\_child funciona corretamente.
- **test\_get\_ancestors:** Verifica se o método get\_ancestors retorna a lista correta de ancestrais.

## 6.4. Árvore Binária

### 6.4.1. Árvore Binária (REFATORAÇÃO):

Um código-fonte em Python (versão 3.11.4) que realiza operações em uma estrutura de dados de árvore será fornecido. Refatore este código para melhorá-lo, sem alterar a funcionalidade original.

#### 6.4.1.1. Java

```
Final > J4V4 > TreeNode45 > J Main.java > BinaryTreeNode
1 package Final.J4V4.TreeNode45;
2
3 // CÓDIGO ORIGINAL
4
5 class TreeNode {
6     int value;
7     TreeNode left, right;
8
9     TreeNode(int item) {
10         value = item;
11         left = right = null;
12     }
13 }
14
15 class BinaryTree {
16     TreeNode root;
17
18     void insert(int value) {
19         root = insertRec(root, value);
20     }
21
22     TreeNode insertRec(TreeNode root, int value) {
23         if (root == null) {
24             root = new TreeNode(value);
25             return root;
26         }
27         if (value < root.value) {
28             root.left = insertRec(root.left, value);
29         } else {
30             root.right = insertRec(root.right, value);
31         }
32         return root;
33     }
}
```

```
35     void inorderTraversal() {
36         inorderTraversalRec(root);
37     }
38
39     void inorderTraversalRec(TreeNode root) {
40         if (root != null) {
41             inorderTraversalRec(root.left);
42             System.out.println(root.value);
43             inorderTraversalRec(root.right);
44         }
45     }
46 }
47
48 public class Main {
49     Run|Debug
50     public static void main(String[] args) {
51         BinaryTree tree = new BinaryTree();
52         tree.insert(50);
53         tree.insert(30);
54         tree.insert(70);
55         tree.inorderTraversal();
56     }
57 }
```

#### Modificações:

- **Renomeação de Classes:** As classes foram renomeadas para seguir um padrão mais claro e consistente.
- **Refatoração da Inserção:** O método de inserção foi ligeiramente simplificado.

Final > J4V4 > TreeNode45 > **J** Main2.java > BinaryTree2

```
1  package Final.J4V4.TreeNode45;
2
3  // CÓDIGO REFATORADO
4
5  class TreeNode2 {
6      int value;
7      TreeNode2 left, right;
8
9      TreeNode2(int item) {
10         value = item;
11         left = right = null;
12     }
13 }
14
15 class BinaryTree2 {
16     TreeNode2 root;
17
18     void insert(int value) {
19         root = insertRec(root, value);
20     }
21
22     TreeNode2 insertRec(TreeNode2 root, int value) {
23         if (root == null) {
24             return new TreeNode2(value);
25         }
26         if (value < root.value) {
27             root.left = insertRec(root.left, value);
28         } else {
29             root.right = insertRec(root.right, value);
30         }
31         return root;
32     }
}
```

```
33     void inorderTraversal() {
34         inorderTraversalRec(root);
35     }
36
37     void inorderTraversalRec(TreeNode2 root) {
38         if (root != null) {
39             inorderTraversalRec(root.left);
40             System.out.println(root.value);
41             inorderTraversalRec(root.right);
42         }
43     }
44 }
45
46 public class Main2 {
47     Run | Debug
48     public static void main(String[] args) {
49         BinaryTree2 tree = new BinaryTree2();
50         tree.insert(50);
51         tree.insert(30);
52         tree.insert(70);
53         tree.inorderTraversal();
54     }
55 }
56 }
```

### 6.4.1.2. Node.js

```
Final > Node > js 45TreeNode.js >  BinaryTree
1  // CÓDIGO ORIGINAL
2
3  class TreeNode {
4      constructor(value) {
5          this.value = value;
6          this.left = null;
7          this.right = null;
8      }
9  }
10
11 class BinaryTree {
12     constructor() {
13         this.root = null;
14     }
15
16     insert(value) {
17         let current = this.root;
18         if (current === null) {
19             this.root = new TreeNode(value);
20             return;
21         }
22         while (true) {
23             if (value < current.value) {
24                 if (current.left === null) {
25                     current.left = new TreeNode(value);
26                     return;
27                 }
28                 current = current.left;
29             } else {
30                 if (current.right === null) {
31                     current.right = new TreeNode(value);
32                     return;
33                 }
34                 current = current.right;
35             }
36         }
37     }
38
39     inorderTraversal(node) {
40         if (node !== null) {
41             this.inorderTraversal(node.left);
42             console.log(node.value);
43             this.inorderTraversal(node.right);
44         }
45     }
46
47     traverse() {
48         this.inorderTraversal(this.root);
49     }
50 }
51
52 const tree = new BinaryTree();
53 tree.insert(50);
54 tree.insert(30);
55 tree.insert(70);
56 tree.traverse();
```

## Modificações

- **Refatoração da inserção:** O método de inserção foi refatorado para usar uma função recursiva, tornando o código mais legível.
- **Renomeação de Classes e Métodos:** As classes e métodos foram renomeados para seguir um padrão mais claro e consistente.

```
Final > Node > JS 45TreeNode2_refatorado.js >  BinaryTree2
1  // CÓDIGO REFATORADO
2
3  class TreeNode2 {
4      constructor(value) {
5          this.value = value;
6          this.left = null;
7          this.right = null;
8      }
9  }
10
11 class BinaryTree2 []
12 constructor() {
13     this.root = null;
14 }
15
16 insert(value) {
17     this.root = this._insert(this.root, value);
18 }
19
20 _insert(node, value) {
21     if (node == null) {
22         return new TreeNode2(value);
23     }
24     if (value < node.value) {
25         node.left = this._insert(node.left, value);
26     } else {
27         node.right = this._insert(node.right, value);
28     }
29     return node;
30 }
```

```
32     inorderTraversal() {
33         this._inorderTraversal(this.root);
34     }
35
36     _inorderTraversal(node) {
37         if (node != null) {
38             this._inorderTraversal(node.left);
39             console.log(node.value);
40             this._inorderTraversal(node.right);
41         }
42     }
43 }
44
45 const tree = new BinaryTree2();
46 tree.insert(50);
47 tree.insert(30);
48 tree.insert(70);
49 tree.inorderTraversal();
50
```

#### 6.4.1.3. Python

```
Final > Python > 45TreeNode.py > ...
1  #CÓDIGO ORIGINAL ANTES DA REFATORAÇÃO
2
3  class TreeNode:
4      def __init__(self, value):
5          self.value = value
6          self.left = None
7          self.right = None
8
9      def insert(node, value):
10         if not node:
11             return TreeNode(value)
12         if value < node.value:
13             node.left = insert(node.left, value)
14         else:
15             node.right = insert(node.right, value)
16         return node
17
18     def inorder_traversal(node):
19         if node:
20             inorder_traversal(node.left)
21             print(node.value)
22             inorder_traversal(node.right)
23
24     root = None
25     root = insert(root, 50)
26     root = insert(root, 30)
27     root = insert(root, 70)
28     inorder_traversal(root)
29
```

Explicação:

- O código foi refatorado para encapsular as funções e variáveis relacionadas a uma árvore binária dentro de uma classe `BinaryTree`. Isso facilita o gerenciamento e expansão do código no futuro.
- Adicionei um método `find` que permite buscar um valor na árvore. Ele utiliza uma pesquisa binária recursiva para localizar o valor e retornar o nó correspondente ou `None` se o valor não for encontrado.

Final > Python > 45TreeNode\_Refatorado.py > BinaryTree2

```
1 # CÓDIGO REFATORADO
2
3 class BinaryTree2:
4     class TreeNode2:
5         def __init__(self, value):
6             self.value = value
7             self.left = None
8             self.right = None
9
10    def __init__(self):
11        self.root = None
12
13    def insert(self, value):
14        self.root = self._insert(self.root, value)
15
16    def _insert(self, node, value):
17        if not node:
18            return self(TreeNode2(value))
19        if value < node.value:
20            node.left = self._insert(node.left, value)
21        else:
22            node.right = self._insert(node.right, value)
23        return node
24
25    def inorder_traversal(self):
26        self._inorder_traversal(self.root)
27
28    def _inorder_traversal(self, node):
29        if node:
30            self._inorder_traversal(node.left)
31            print(node.value)
32            self._inorder_traversal(node.right)
```

```
28     def _inorder_traversal(self, node):
29         if node:
30             self._inorder_traversal(node.left)
31             print(node.value)
32             self._inorder_traversal(node.right)
33
34     def find(self, value):
35         return self._find(self.root, value)
36
37     def _find(self, node, value):
38         if not node:
39             return None
40         if node.value == value:
41             return node
42         if value < node.value:
43             return self._find(node.left, value)
44         return self._find(node.right, value)
45
46 tree = BinaryTree2()
47 tree.insert(50)
48 tree.insert(30)
49 tree.insert(70)
50 tree.inorder_traversal()
51 found_node = tree.find(30)
52 if found_node:
53     print(f"Found node with value: {found_node.value}")
54 else:
55     print("Node not found.")
```

## 6.4.2. Árvore Binária (TESTE UNITÁRIO):

Utilizando Python, elabore testes unitários para um código-fonte fornecido que gerencia um sistema de biblioteca. Os testes devem abranger todas as funcionalidades essenciais.

### 6.4.2.1. Python

```
Final > Python > 45B_Library.py > ...
1
2
3 class LibrarySystem:
4     def __init__(self):
5         self.books = {}
6
7     def add_book(self, title, author):
8         self.books[title] = author
9
10    def remove_book(self, title):
11        if title in self.books:
12            del self.books[title]
13
14    def find_book_by_title(self, title):
15        return self.books.get(title, None)
16
17    def list_books(self):
18        return self.books
19
20
21 import unittest
22
23 class TestLibrarySystem(unittest.TestCase):
24
25     def setUp(self):
26         self.library = LibrarySystem()
27         self.library.add_book("The Hobbit", "J.R.R. Tolkien")
28         self.library.add_book("Dune", "Frank Herbert")
29
30     def test_add_book(self):
31         self.library.add_book("Foundation", "Isaac Asimov")
32         self.assertEqual(self.library.find_book_by_title("Foundation"), "Isaac Asimov")
33
34     def test_remove_book(self):
35         self.library.remove_book("The Hobbit")
36         self.assertIsNone(self.library.find_book_by_title("The Hobbit"))
37
38     def test_find_book_by_title(self):
39         self.assertEqual(self.library.find_book_by_title("Dune"), "Frank Herbert")
40
41     def test_list_books(self):
42         books = {
43             "The Hobbit": "J.R.R. Tolkien",
44             "Dune": "Frank Herbert"
45         }
46         self.assertEqual(self.library.list_books(), books)
47
48 if __name__ == '__main__':
49     unittest.main()
```

## 6.5. Stock Management

### 6.5.1. Stocks (REFATORAÇÃO):

Um código-fonte em Node Javascript (versão 18.17.0) que realiza operações de gerenciamento de estoque será fornecido. Refatore este código para melhorá-lo sem mudar a funcionalidade.

#### 6.5.1.1. Java

```
Final > J4V4 > Stock55 > StockManagement.java > StockManagement
1 package Final.J4V4.Stock55;
2
3 import java.util.HashMap;
4
5 // CÓDIGO ORIGINAL A SER REFATORADO
6
7 public class StockManagement {
8     private HashMap<String, Integer> stock;
9
10    public StockManagement() {
11        this.stock = new HashMap<>();
12    }
13
14    public void addProduct(String productName, int quantity) {
15        if (stock.containsKey(productName)) {
16            stock.put(productName, stock.get(productName) + quantity);
17        } else {
18            stock.put(productName, quantity);
19        }
20    }
21
22    public void removeProduct(String productName, int quantity) {
23        if (stock.containsKey(productName) && stock.get(productName) >= quantity) {
24            stock.put(productName, stock.get(productName) - quantity);
25        } else {
26            System.out.println("Not enough stock or product not found");
27        }
28    }
29
30    public int getStock(String productName) {
31        return stock.getOrDefault(productName, -1); // -1 represents "Product not found"
32    }
33
34    // Usage
35    Run | Debug
36    public static void main(String[] args) {
37        StockManagement stockManager = new StockManagement();
38        stockManager.addProduct("Apple", 10);
39        stockManager.removeProduct("Orange", 5);
40        System.out.println(stockManager.getStock("Apple")); // Outputs 10
41        System.out.println(stockManager.getStock("Orange")); // Outputs -1
42    }
}
```

## Lista das Correções Realizadas:

- **Método addProduct:** Simplificado usando o método `getOrDefault` para reduzir a verificação de existência e manipulação do valor atual.
- **Método removeProduct:** Alterado para retornar um valor booleano para indicar o sucesso ou falha na remoção do produto. Isso permite que o chamador do método lide com a situação, em vez de imprimir diretamente no console.
- **Modificador Final:** Adicionado o modificador final ao atributo `stock` para garantir que não será modificado após a inicialização. Isso ajuda a tornar a classe mais robusta.
- **Encapsulamento de Mensagem de Erro:** A mensagem de erro foi movida para o método principal, colocando a lógica de apresentação fora da classe de gerenciamento de estoque.
- **Convenção de Nomenclatura:** Ajustado o nome da classe e outros elementos para seguir as convenções padrão de nomenclatura Java.

```
Final > J4V4 > Stock55 > StockManagement2.java > StockManagement2
1 package Final.J4V4.Stock55;
2
3 import java.util.HashMap;
4
5 // CÓDIGO REFATORADO
6
7 public class StockManagement2 {
8     private final HashMap<String, Integer> stock;
9
10    public StockManagement2() {
11        this.stock = new HashMap<>();
12    }
13
14    public void addProduct(String productName, int quantity) {
15        stock.put(productName, stock.getOrDefault(productName, 0) + quantity);
16    }
17
18    public boolean removeProduct(String productName, int quantity) {
19        int currentStock = stock.getOrDefault(productName, 0);
20        if (currentStock >= quantity) {
21            stock.put(productName, currentStock - quantity);
22            return true;
23        } else {
24            return false;
25        }
26    }
}
```

```
28     public int getStock(String productName) {
29         return stock.getOrDefault(productName, -1); // -1 represents "Product not found"
30     }
31
32     // Usage
33     Run | Debug
34     public static void main(String[] args) {
35         StockManagement2 stockManager = new StockManagement2();
36         stockManager.addProduct("Apple", 10);
37         if (!stockManager.removeProduct("Orange", 5)) {
38             System.out.println("Not enough stock or product not found");
39         }
40         System.out.println(stockManager.getStock("Apple")); // Outputs 10
41         System.out.println(stockManager.getStock("Orange")); // Outputs -1
42     }
43 }
```

### 6.5.1.2. Node.js

#### Parte A - Código Original:

```
javascript Copy code  
  
class StockManagement {  
    constructor() {  
        this.stock = {};  
    }  
  
    addProduct(productName, quantity) {  
        if(this.stock[productName]) {  
            this.stock[productName] += quantity;  
        } else {  
            this.stock[productName] = quantity;  
        }  
    }  
  
    removeProduct(productName, quantity) {  
        if(this.stock[productName] && this.stock[productName] >= quantity) {  
            this.stock[productName] -= quantity;  
        } else {  
            console.log('Not enough stock or product not found');  
        }  
    }  
  
    getStock(productName) {  
        return this.stock[productName] || 'Product not found';  
    }  
}  
  
// Usage  
const stockManager = new StockManagement();  
stockManager.addProduct('Apple', 10);  
stockManager.removeProduct('Orange', 5);  
console.log(stockManager.getStock('Apple')) // Outputs 10
```

#### Refatorações Realizadas:

- Substituição do objeto literal pelo objeto Map para gerenciar o estoque;
- Separação da lógica de erro em um método separado (handleError), permitindo uma manipulação de erro mais sofisticada no futuro;
- Melhoria na legibilidade, simplificando a lógica de adicionar e remover produtos.

Essas mudanças mantêm a funcionalidade original, mas tornam o código mais claro e fácil de manter.

```
javascript

class StockManagement {
    constructor() {
        this.stock = new Map();
    }

    addProduct(productName, quantity) {
        const currentQuantity = this.stock.get(productName) || 0;
        this.stock.set(productName, currentQuantity + quantity);
    }

    removeProduct(productName, quantity) {
        const currentQuantity = this.stock.get(productName);

        if (currentQuantity && currentQuantity >= quantity) {
            this.stock.set(productName, currentQuantity - quantity);
        } else {
            this.handleError('Not enough stock or product not found');
        }
    }

    getStock(productName) {
        return this.stock.get(productName) || 'Product not found';
    }

    handleError(message) {
        console.log(message);
    }
}

// Usage
const stockManager = new StockManagement();
stockManager.addProduct('Apple', 10);
stockManager.removeProduct('Orange', 5);
console.log(stockManager.getStock('Apple')) // Outputs 10
```

### 6.5.1.3. Python

```
Final > Python > 55StockManagement.py > ...
1
2 # CÓDIGO ORIGINAL PARA SER REFATORADO
3
4 class StockManagement:
5     def __init__(self):
6         self.stock = {}
7
8     def add_product(self, product_name, quantity):
9         if product_name in self.stock:
10             self.stock[product_name] += quantity
11         else:
12             self.stock[product_name] = quantity
13
14     def remove_product(self, product_name, quantity):
15         if product_name in self.stock and self.stock[product_name] >= quantity:
16             self.stock[product_name] -= quantity
17         else:
18             print("Not enough stock or product not found")
19
20     def get_stock(self, product_name):
21         if product_name in self.stock:
22             return self.stock[product_name]
23         else:
24             return "Product not found"
25
26
27 # Usage
28 stock_manager = StockManagement()
29 stock_manager.add_product("Apple", 10)
30 stock_manager.remove_product("Orange", 5)
31 print(stock_manager.get_stock("Apple")) # Outputs 10
32 print(stock_manager.get_stock("Orange")) # Outputs "Product not found"
```

Lista das Correções Realizadas:

- **Simplificação do add\_product:** Utilizando o método get, podemos simplificar a adição de produtos, sem necessidade de verificar se o produto já existe.
- **Alteração do remove\_product:** O método agora retorna um valor booleano para indicar o sucesso ou falha na remoção do produto, permitindo que o chamador do método lide com a situação.
- **Encapsulamento de Mensagem de Erro:** A mensagem de erro foi movida para fora da classe, melhor separando as responsabilidades de apresentação ao usuário.

```
Final > Python > 55StockManagement2.py > ...
1
2 # CÓDIGO REFATORADO
3
4 class StockManagement2:
5     def __init__(self):
6         self.stock = {}
7
8     def add_product(self, product_name, quantity):
9         self.stock[product_name] = self.stock.get(product_name, 0) + quantity
10
11    def remove_product(self, product_name, quantity):
12        current_stock = self.stock.get(product_name, 0)
13        if current_stock >= quantity:
14            self.stock[product_name] = current_stock - quantity
15            return True
16        else:
17            return False
18
19    def get_stock(self, product_name):
20        return self.stock.get(product_name, "Product not found")
21
22
23 # Usage
24 stock_manager = StockManagement2()
25 stock_manager.add_product("Apple", 10)
26 if not stock_manager.remove_product("Orange", 5):
27     print("Not enough stock or product not found")
28 print(stock_manager.get_stock("Apple")) # Outputs 10
29 print(stock_manager.get_stock("Orange")) # Outputs "Product not found"
```

#### **6.5.2. Stocks (TESTE UNITÁRIO):**

Em Python (versão 3.11.4), elabore os testes unitários para um código-fonte fornecido que lide com manipulação de datas e horários. Os testes devem garantir a funcionalidade correta do código.

### 6.5.2.1. Python

```
Final > Python > 55BDateManipulation.py > ...
1  from datetime import datetime, timedelta
2
3  class DateManipulation:
4
5      def __init__(self, date_str):
6          self.date = datetime.strptime(date_str, '%Y-%m-%d %H:%M:%S')
7
8      def add_days(self, days):
9          self.date += timedelta(days=days)
10         return self.date.strftime('%Y-%m-%d %H:%M:%S')
11
12     def subtract_days(self, days):
13         self.date -= timedelta(days=days)
14         return self.date.strftime('%Y-%m-%d %H:%M:%S')
15
16     def get_day_of_week(self):
17         days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
18         return days_of_week[self.date.weekday()]
```

```
20 import unittest
21
22
23 class TestDateManipulation(unittest.TestCase):
24
25     def test_add_days(self):
26         date_manipulation = DateManipulation('2023-08-16 00:00:00')
27         self.assertEqual(date_manipulation.add_days(5), '2023-08-21 00:00:00')
28
29     def test_subtract_days(self):
30         date_manipulation = DateManipulation('2023-08-16 00:00:00')
31         self.assertEqual(date_manipulation.subtract_days(3), '2023-08-13 00:00:00')
32
33     def test_get_day_of_week(self):
34         date_manipulation = DateManipulation('2023-08-16 00:00:00')
35         self.assertEqual(date_manipulation.get_day_of_week(), 'Wednesday')
36
37 if __name__ == '__main__':
38     unittest.main()
```

Esses testes unitários abordam diferentes aspectos da classe DateManipulation:

- Testam o método `add_days`, que adiciona dias à data;
  - Testam o método `subtract_days`, que subtrai dias da data;
  - Testam o método `get_day_of_week`, que retorna o dia da semana para a data.