Universidade Federal do Ceará - UFC
Centro de Ciências - Física e Matemática
Departamento de Computação
Disciplina: Fundamentos de Programação - FP
5ª Lista de Exercício no Laboratório - Prof: Caminha - Agosto de 2024

1. Como Gerar Números Aleatórios em *Python*

Podemos na linguagem de programação *Python* gerar números aleatórios. Criar números aleatórios conforme determinados critérios. Sem dúvuida, é de grande utilidade a ferramento que possibilita gerar números aleatórios.

Podemos gerar números aleatórios para fazer simulações, principalmente científicas e de outros cunhos acadêmicos, é necessário usar uma enormidade de números aleatórios.

Agora vamos aprender a usar a biblioteca random, **random module** em Python. Nesse módulo pode ser utilizada a função "**randint**", que tem a finalidade de gerar Número Inteiro Aleatório.

Vamos usar a função **randint**, da biblioteca padrão do módulo "*random*" do Python. Assim, no início do programa devemos importar esse módulo:

import random

Essa função tem dois parâmetros, vamos chamar de inicio e fim:

random.randint (inicio, fim)

Essa função retorna um número inteiro de 'inicio' até 'fim' (inclusive eles).

Por exemplo:

randint(1,10) ➡ Gera qualquer número do 1 até o 10

randint(1,1000) ➡ Resultar qualquer número 1,2, 3, ..., 1000

Exemplo de uso da função randint()

Um programa em *Python* que simula o resultado de um dado, ou seja, gera números aleatórios de 1 até 6, quantas vezes o usuário desejar.

Para utilizar a função "***randint***", deve-se primeiro importar a biblioteca padrão "*random*", do *Python*.

Exemplo 1:

```
import random
controle = 1
while controle == 1:
    NumGerado = random.randint(1,6)
    Print (‘Numero Gerado = “, NumGerado)
    Continuar = int(input("Gerar novamente?  \n1.Sim\ n0.Não\n")
Print (“F i m   d o   T r a b a l h o”)
```

Exemplo 2:

```
import random
d1 = random.randrange(1, 10)
d2 = random.randrange(11, 20)
d3 = random.randrange(21, 30)
d4 = random.randrange(31, 40)
d5 = random.randrange(41, 50)
d6 = random.randrange(51, 60)
print(d1, d2, d3, d4, d5, d6)
```

18ª Questão:

Codificar um programa em linguagem *Python* para processar **N elementos** de uma Lista (conjunto).

No caso deve conhecer e apresentar os valores de cada elemento e calcular a média aritmética.

Determinar a **quantidade** de elementos cujo valor é maior ou igual a média e menor que o valor de **Limite**. O valor de Limite deve ser informado.

Imprimir todos os elementos da Lista que atendem as condições (>= Média e < Limite).

O resultado deve ser **apresentado** conforme modelo indicado a seguir:

```
Digite o valor de Limite: xx
Digite o número de elementos do Conjunto: xx
Digite o elemento [0] do conjunto: xx
Digite o elemento [1] do conjunto: xx
.........
Digite o elemento [N-1] do conjunto: xx


UFC - Programação – 2024.1

Valores da Lista: [ xx, xx, ....., xx, xx ]
Média = xx.x
Limite = xxx
Quantidade de Elementos >= a Média e < Limite = xx
Elementos >= a Média e < Limite: [ xx, xx, ....., xx, xx ]

Fim do Trabalho
```

Programa: 1ª Versão onde utiliza uma Lista.

Resultado: da 1ª Versão que utiliza uma Lista.

Programa: 2ª Versão onde utiliza duas Listas.

Resultado: da 2ª Versão que utiliza duas Listas.

19ª Questão:

Codificar um programa em linguagem *Python* para **gerar** os valores a serem lançados em um vetor de 1000 elementos. Os valores serão gerados de forma aleatórios.

Para tanto, utilizar a função **randint (x, xx)** do módulo random para gerar os valores inteiros aleatórios.

A informação "**xx**" na função **randint** corresponde ao valor limite máximo dos valores que serão gerados. No caso desse programa, o valor de "xx" poderá ser 2000, por exemplo: randint (1, 2000).

Então o programa deve gerar 1000 números aleatórios cujos valores estarão no intervalo de 0 a 2000.

O programa deve na sequência colocar os valores desse conjunto em ordem crescente.

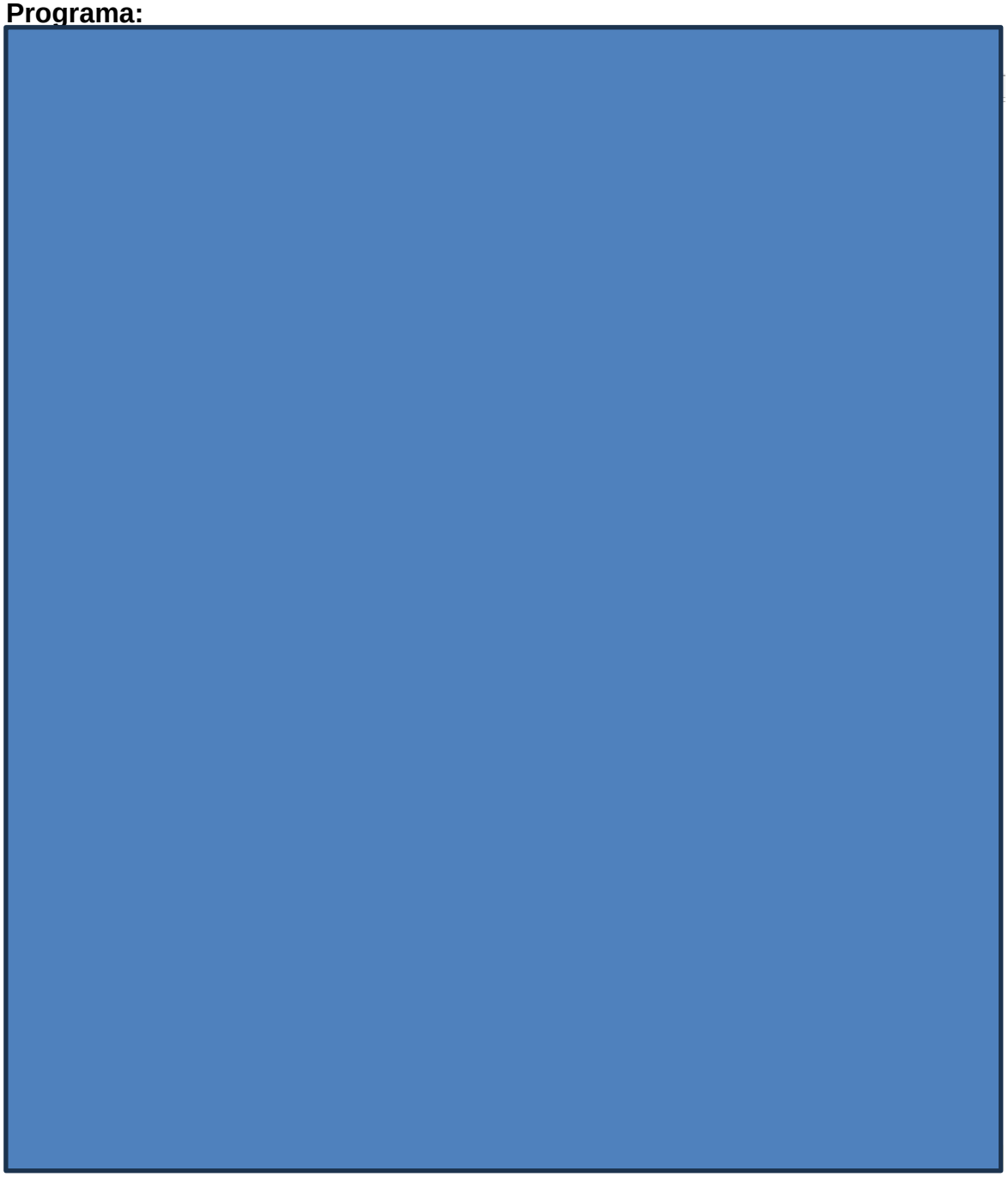
Utilizar o modelo de ordenação por trocas.

UFC - Programação - 2024.1
Nome: x----------x

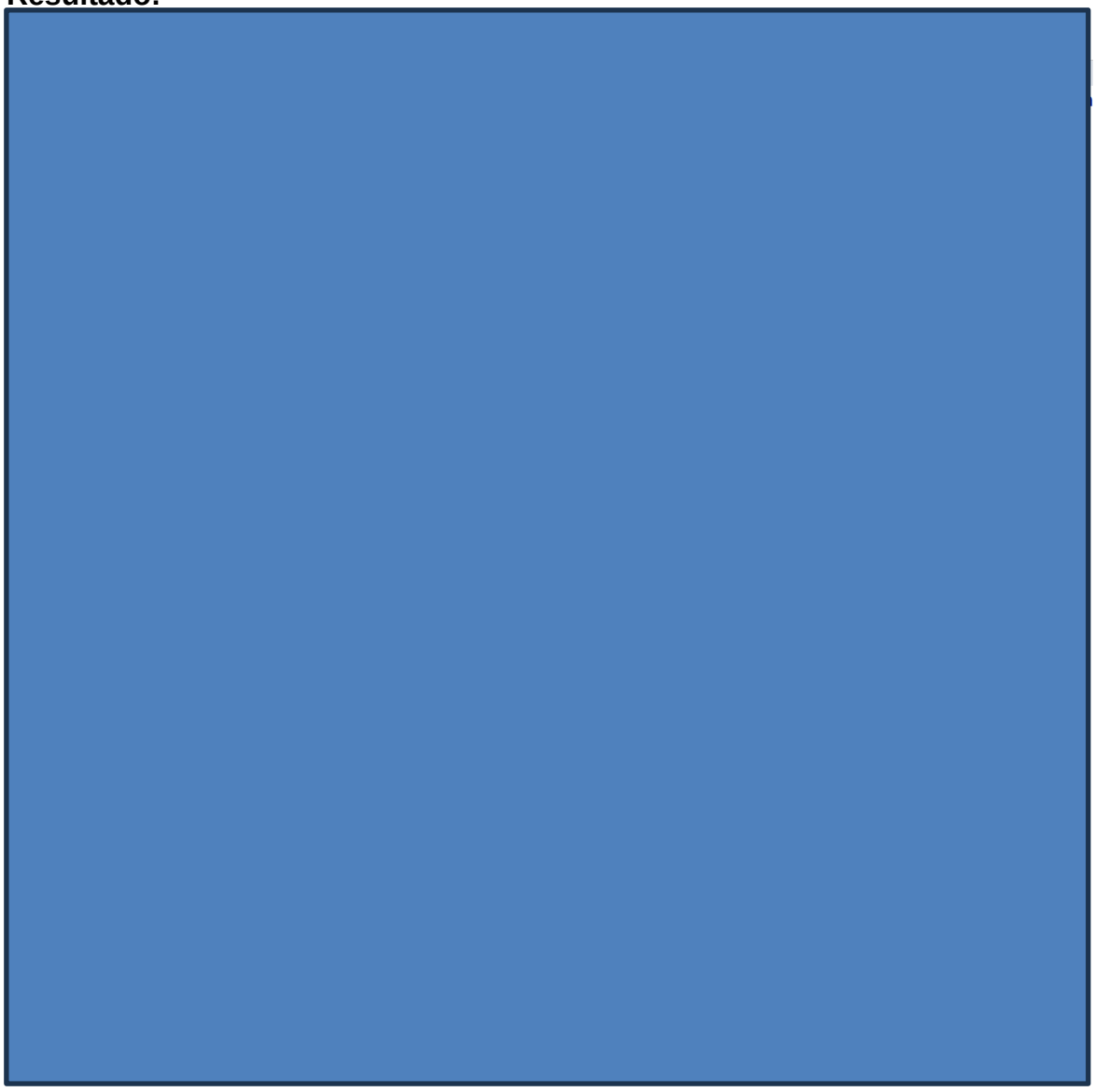
Vetor - Ordem Original:
pppp - vvvv pppp - vvvv ... pppp - vvvv
(p é posição e v é valor)

Vetor - Ordenado:
pppp - vvvv pppp - vvvv ... pppp - vvvv
(p é posição e v é valor)

Programa:



Resultado:



20ª Questão:

Elaborar um programa em *Python* para realizar **pesquisa binária** de um determinado valor em um vetor.

O vetor tem 1.000 elementos numéricos inteiros dispostos em ordem crescente.

A base de dados deve ter 1.000 valores numéricos gerados aleatoriamente (função **randint (1, valor)** do módulo random da linguagem *Python*, onde "*valor*" é o limite do maior valor a ser gerado) no intervalo de 1 até 2000. Esses valores devem ser ordenados, classificados em ordem crescente, conforme modelo utilizado na questão 19ª.

Além realizar a pesquisa de um valor no vetor, o programa deve também calcular e informar quantas comparações foram realizadas até localizar o valor informado.

```
UFC - Programação - 2024.1
Nome: X.....X

Vetor - Ordem Original:
pppp - vvvv    pppp - vvvv   ...   pppp - vvvv
(p é posição e v é valor)

Vetor - Ordenado:
pppp - vvvv    pppp - vvvv   ...   pppp - vvvv
(p é posição e v é valor)

Valor Pesquisado: XXXX
POSIÇÃO: XXX
Número de Iterações = xx
```

A **"pesquisa binária"** é um procedimento utilizado para **localizar** um dado "valor" dentre os valores de um vetor. Os valores deste vetor devem, necessariamente, estarem ordenados. O algoritmo de "pesquisa binária" segue o paradigma de divisão e conquista.

Como o pressuposto inicial é de que **o vetor está ordenado**, então o procedimento é realizar sucessivas **divisões do vetor** (ou espaço de busca), sempre comparando o valor do elemento buscado (a chave) com o valor do **elemento no meio** do vetor (ou espaço de busca).

Se o valor do elemento do meio do vetor (ou espaço de busca) for igual ao valor ao valor do elemento buscado (chave), a busca termina com sucesso.

Caso contrário, se o valor do **elemento do meio** do vetor (ou espaço de busca) for **menor** que o valor do elemento buscado, então a busca continua na **metade posterior** do vetor. Ou, se o **elemento do meio** do vetor (ou espaço de busca) for **maior** que o valor do elemento buscado, então a busca continua na **metade anterior** do vetor.

Esse procedimento deve continuar até localizar o valor buscado ou concluir que ele não faz parte do vetor.

Pesquisa Binária: Exemplo.

Valor de Busca: **vb**.

	Vetor									
	v1	**v2**	**v3**	**v4**	**v5**	**v6**	**v7**	**v8**	**v9**	**v10**
índices	1	2	3	4	5	6	7	8	9	10

Os elementos do Vetor já estão ordenados conforme os valores.

Definir o elemento do meio do vetor:

- índice médio = (índice inicial + índice final) / 2
- índice médio = (1 + 10) / 2
- **índice médio = 5**

compara o **vb** com **v5** que é o elemento da posição média do vetor. se o valor de **v5** for menor que o valor de **vb**, então a pesquisa continua na metade posterior do vetor.

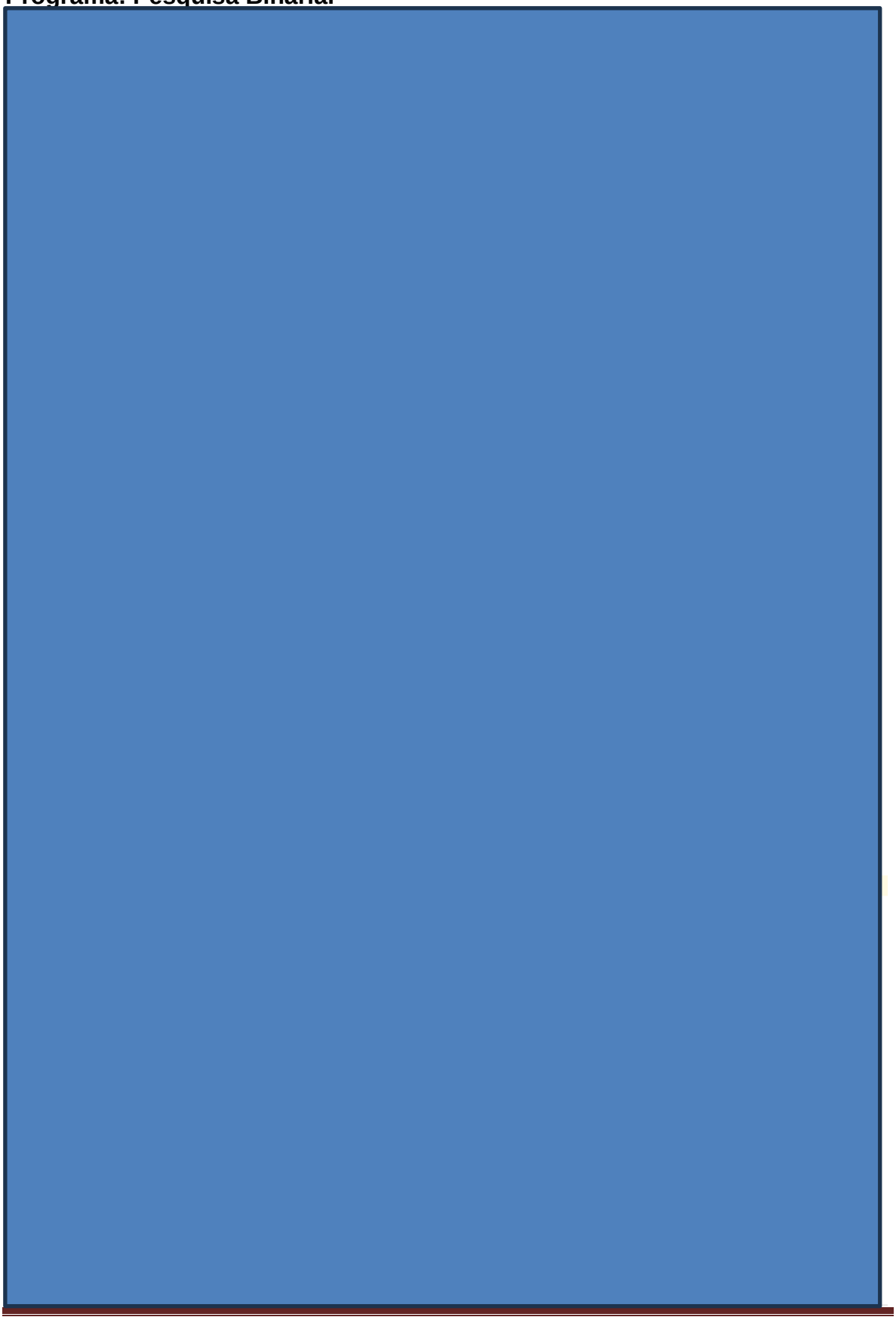
	v6	**v7**	**v8**	**v9**	**v10**
índices	6	7	8	9	10

Definir o elemento do meio do vetor (ou espaço de busca):

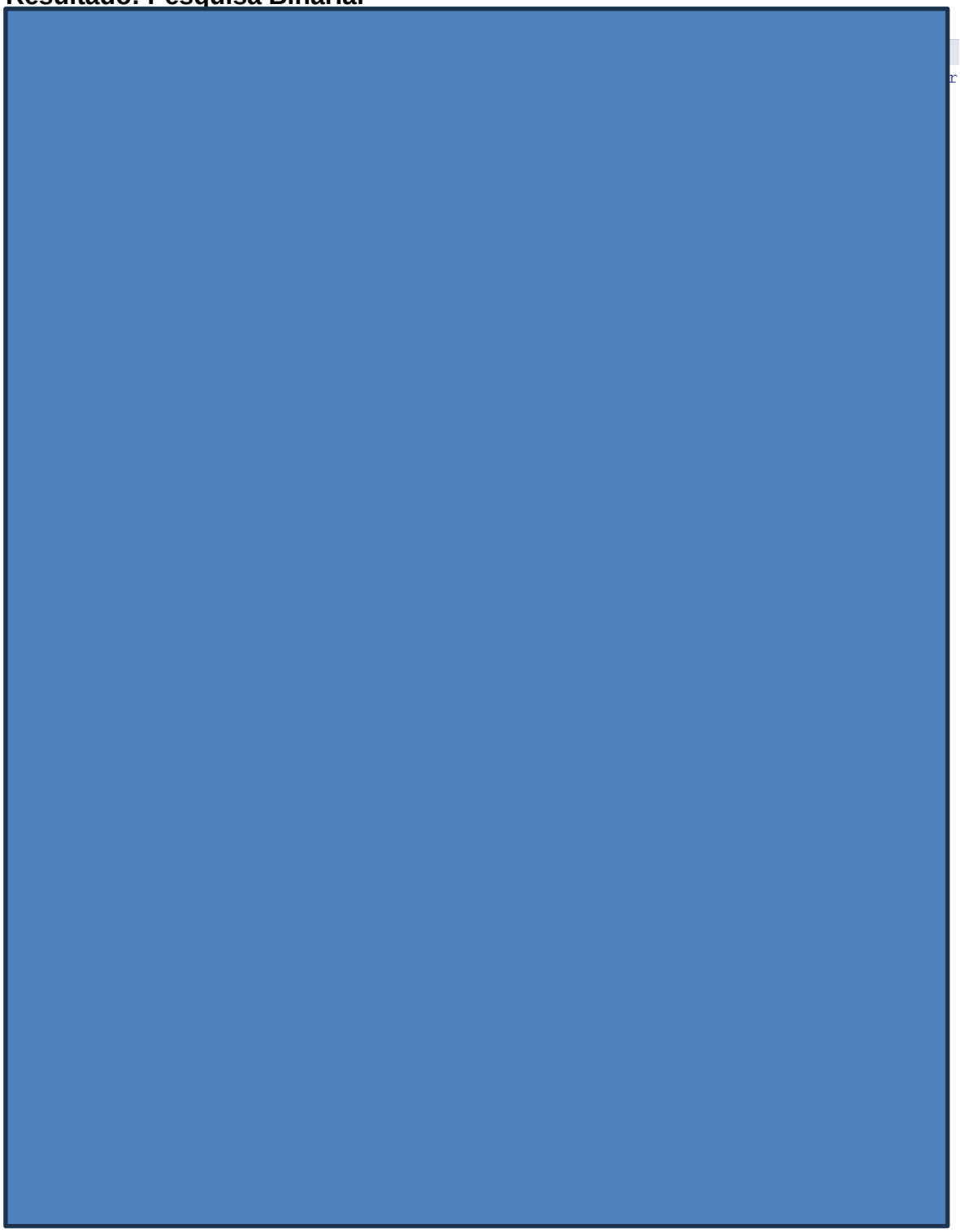
- índice médio = (índice inicial + índice final) / 2
- índice médio = (6 + 10) / 2
- **índice médio = 8**

Compara o **vb** com **v8** que é o elemento da posição média do vetor. se o valor de **v8** for maior que o valor de **vb**, então a pesquisa continua na metade anterior do vetor.

Programa: Pesquisa Binária.



Resultado: Pesquisa Binária.



21ª Questão:
Codificar um programa em linguagem *Python* gerar e imprimir os valores do **Triângulo de Pascal**.

Propriedades do Triângulo de Pascal:

(i) O triângulo de Pascal é dividido por linhas e colunas, começando sempre da linha zero e da coluna zero;

(ii) Cada linha que forma o Triângulo de Pascal sempre inicia e termina com o número 1;

(iii) Todo Número que faz parte do triângulo é um resultado da soma dos dois números que estão acima dele, na linha anterior.

Os valores do Triângulo de Pascal devem ser impressos até a primeira linha que tenha pelo menos um valor maior ou igual a N.

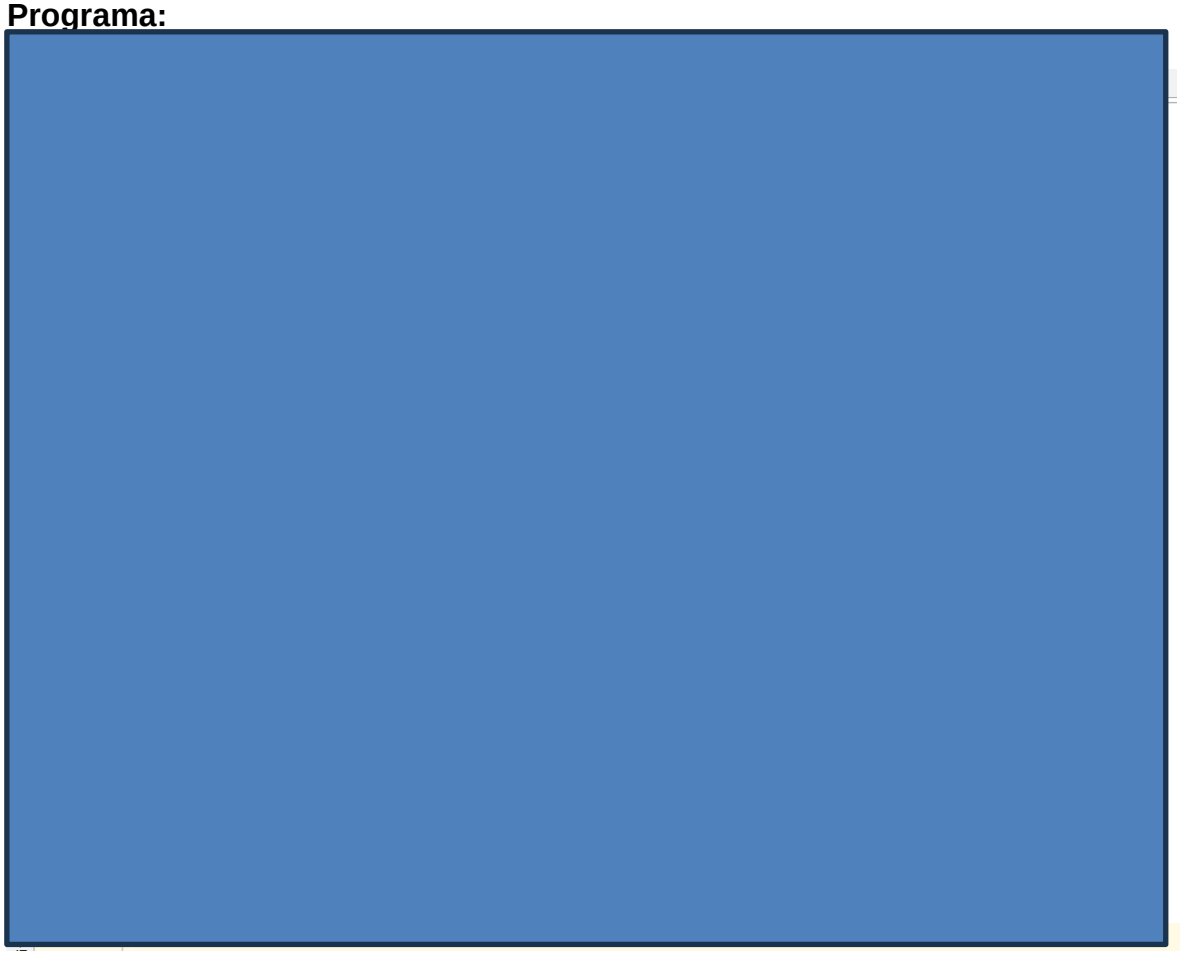
O resultado deve ser apresentado conforme modelo indicado a seguir:

```
UFC - Programação  -  2024.1

Triângulo de Pascal:

1
1  1
1  2   1
1  3   3   1
1  4   6   4   1
1  5  10  10   5   1
1  6  15  20  15   6   1
1  7  21  35  35  21   7   1
.....................................
.....................N.....................
```

Programa:



Resultado:

22ª Questão:

Alguns números inteiros possuem a capacidade de se auto elogiarem através de seus dígitos. Esses são números que formam a família dos **Números Narcisistas**.

A **característica** que identifica os Números Narcisistas clássicos corresponde aquela onde o ***Número Informado*** é igual à soma de cada um dos dígitos que compõe esse *Número Informado*, elevados à potência do valor igual a quantidade de dígitos que compõe o *Número Informado*.

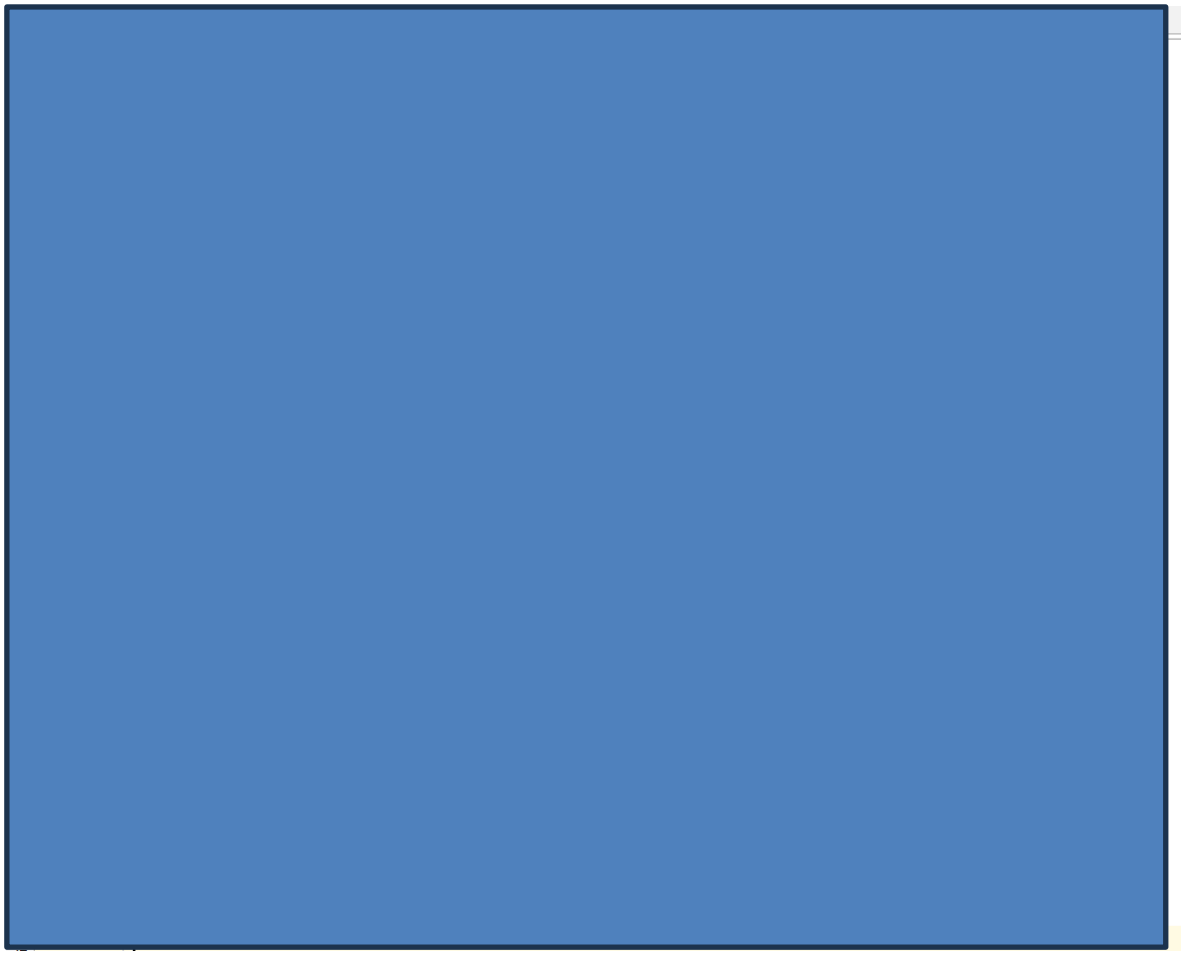
Por exemplo, o número 153 é um narcisista clássico porque a soma de cada um de seus dígitos elevados ao cubo (total de dígitos que compõe o número 153), é exatamente 153.

$$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

Elaborar um programa na linguagem *Python* para listar todos os **Números Narcisistas** no intervalo de 10 até N.

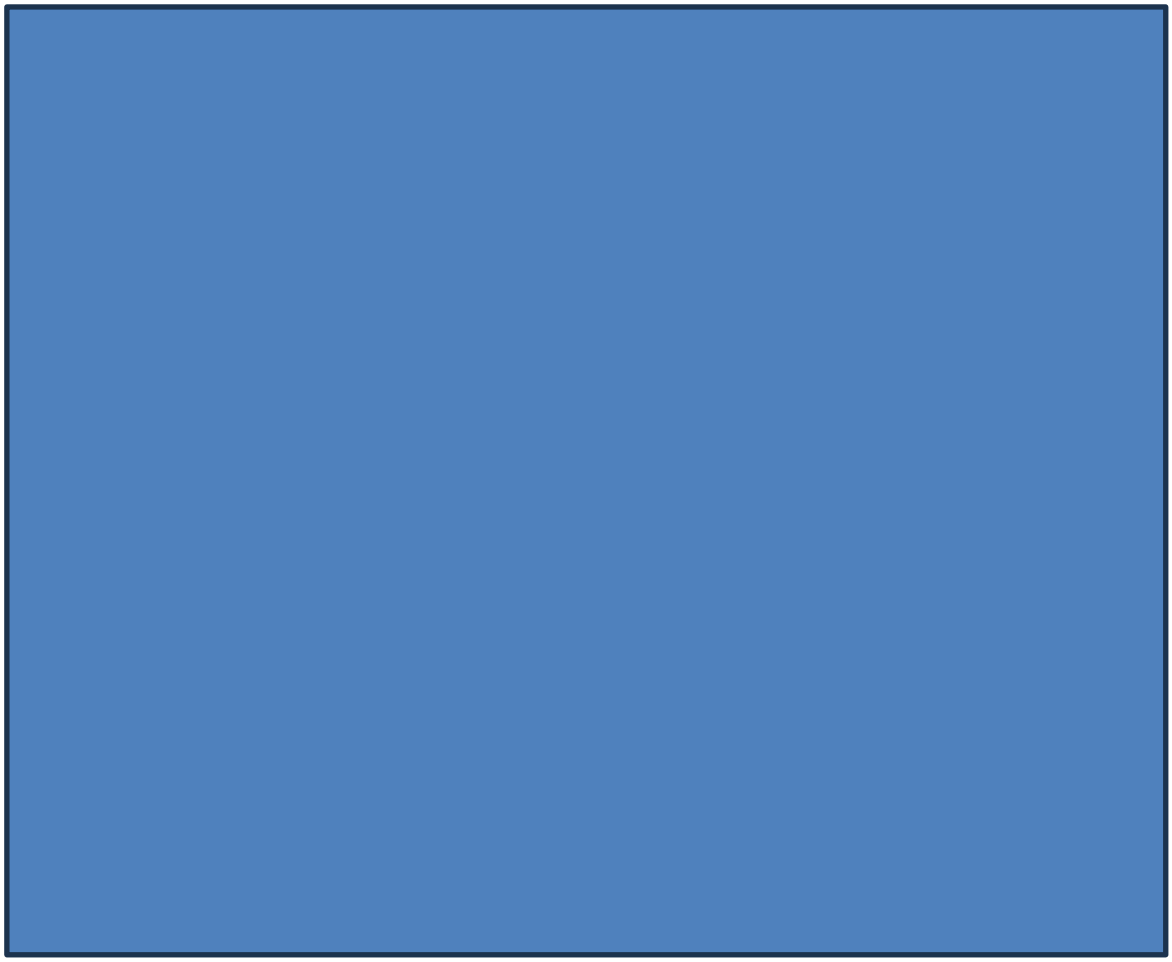
Apresentar resultado em layout definido pelo aluno, mas semelhante ao utilizado em outras questões deste trabalho.

Programa: 1ª Versão Verifica se um Valor Informado é Narcisista.



Resultado: 1ª Versão Verifica se um Valor Informado é Narcisista.

Programa: 2ª Versão Identifica Valores Narcisista no Intervalo de 10 até N.



Resultado: 2ª Versão Identifica Valores Narcisista no Intervalo de 10 até N.

X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X