Recurso Educacional:

Contagem com Python: Uma Abordagem pelo Conjunto de Resultados na Análise Combinatória

SIDNEY DA SILVA FERREIRA

JONES COLOMBO

Niterói - 2026

Lista de Figuras

1 Árvore de Possibilidades da Atividade 1. 30

2 Saída do Código 2.1. 31

3 Árvore de Possibilidades da Atividade 2. 35

4 Saída do Código 2.2. 36

5 Árvore de Possibilidades da Atividade 3. 40

6 Saída do Código 2.3. 41

7 Árvore de Possibilidades da Atividade 4. 47

8 Saída do Código 2.4. 48

9 Árvore de Possibilidades da Atividade 5. 53

10 Saída do Código 2.5. 54

11 Saída do Código 2.6. 60

12 Saída do Código 2.7. 67

13 Saída do Código 2.8. 71

14 Saída do Código 2.9. 77

15 Saída do Código 2.10. 78

16 Saída do Código 2.10. 84

17 Saída do Código 2.11 usando a palavra AMOR. 84

18 Saída do Código 2.11 usando a palavra AMAR. 86

19 Saída do Código 2.11 usando a palavra BABA. 87

20 Saída do Código 2.11 usando a palavra AAZA. 88

21 Saída do Código 2.12 usando a palavra AMAR. 91

22 Saída do Código 2.12 usando a palavra AAZA. 92

23 Resumo da Saída do Código 2.12 usando a palavra SOSSEGO. 93

24 Saída do Código 2.13. 97

25 Saída do Código 2.14. 102

Lista de Códigos

2.1 Subindo e descendo o morro 8

2.2 Subindo e descendo o morro por caminhos diferentes 9

2.3 Sequências de cara e coroa 9

2.4 Colorindo a bandeira 10

2.5 Números pares com dois algarismos 12

2.6 Números pares com dois algarismos distintos 13

2.7 Três pessoas em cinco cadeiras em fila 14

2.8 Código da Atividade 9 16

2.9 Anagramas com restrições 18

2.10 Anagramas de NÚMERO e recursividade 19

2.11 Permutações simples e recursividade 19

2.12 Permutações com elementos repetidos 21

2.13 Filas com três pessoas dadas cinco pessoas 23

2.14 Código da Atividade 16 25

3.1 Código da Atividade 8 68

Sumário

1 Introdução **5**

2 Sequência Didática **7**

2.1 Primeiro encontro 7

2.2 Segundo encontro 12

2.3 Terceiro encontro 16

2.4 Quarto encontro 21

3 Resultados Esperados **28**

3.1 Primeiro encontro 28

3.2 Segundo encontro 51

3.3 Terceiro encontro 69

3.4 Quarto encontro 89

Referências **107**

1 Introdução

Este recurso educacional é resultado da dissertação "A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python" (FERREIRA, 2026), e consiste de uma sequência didática composta por dezesseis atividades, separadas em quatro encontros, sobre Análise Combinatória. A ideia central é usar a linguagem de programação Python para tornar visível aquilo que normalmente fica invisível na resolução de problemas de contagem: o conjunto de todos os resultados possíveis.

As atividades foram organizadas de forma progressiva avançando gradualmente para a abstração matemática, tendo o Princípio Fundamental da Contagem ou Princípio Multiplicativo como principal norteador para obter os resultados almejados. Pretende-se, assim, levar o aluno a pensar de verdade sobre o problema, não apenas a aplicar uma receita. Esta intervenção está fundamentada na seguinte triangulação teórica:

- A **Didática dos Princípios de Contagem (DPC)** que é utilizada em (CERIOLI; VIANA, 2012), a fim de apresentar uma abordagem analítica para a resolução de problemas de contagem.
- O **Modelo Cognitivo de Lockwood** desenvolvido na tese de doutorado (LOCKWOOD, 2011), que mostra o papel fundamental desempenhado pelo conjunto de resultados de um problema de contagem, papel esse percebido, principalmente, no trabalho (LOCKWOOD; GIBSON, 2016). A partir disso, utilizamos a enumeração gerada pelo código em Python para conectar o processo de contagem à fórmula.
- O **Modelo dos Campos Semânticos (MCS)** de Lins (2012), que fornece a base para analisar a produção de significado. O Modelo dos Campos Semânticos nos mostra se o aluno está apenas aplicando regras ou se ele realmente está consciente da maneira que está resolvendo o problema.

As atividades propostas no primeiro encontro trás os seguintes objetivos principais: reconhecer os objetos dos problemas; reconhecer as configurações desejadas; construir a árvore de possibilidades; familiarizar-se com o código em Python; reconhecer a função do laço *for* e do condicional *if* ; enunciar o Princípio Fundamental da Contagem.

No segundo encontro temos os seguintes objetivos: utilizar o Princípio Aditivo junto com o Princípio Fundamental da Contagem, quando a resolução do problema divide-se em casos; determinar o número de arranjos por uma aplicação direta do Princípio Fundamental da Contagem.

No terceiro encontro, propomos problemas de contagem onde as configurações que estamos contando são permutações simples e utilizamos código Python recursivo para contar permutações simples.

E, no quarto encontro, o Princípio Fundamental da Contagem gera um excesso na resolução dos problemas propostos, onde os objetos que serão permutados não são todos distintos e quando da determinação da quantidade de grupos(conjuntos), logo, o objetivo principal desse encontro é corrigir a superestimação, construída pelo Princípio Fundamental da Contagem, utilizando o Princípio k para 1.

Cada encontro deve ser aplicado em três aulas com cinquenta minutos cada. Propomos que tais atividades possam ser aplicadas em turmas do $2^{\underline{o}}$ ou $3^{\underline{o}}$ anos do Ensino Médio, dependendo do currículo de cada estado.

Vale ressaltar que as atividades propostas devem ser aplicadas após os alunos terem contato com o mínimo sobre programação com Python. Sugerimos a utilização do app *Pydroid 3* para dispositivos Android que permite escrever e executar código Python no celular ou tablet. O *Pydroid 3* possui um interpretador Python 3 offline, o que significa que é possível executar programas Python sem conexão com a internet.

2 Sequência Didática

A seguir estão as atividades, dividimo-as em quatro encontros, cada encontro com duração de 150 minutos. Vale ressaltar que tais atividades devem ser aplicadas após os alunos terem contato com o mínimo sobre programação com Python. Acreditamos que pode ser muito proveitoso aos professores que desejem aplicá-las identificar possíveis adaptações e/ou desenvolvimento de atividades complementares buscando assim oferecer o material que melhor se adeque aos seus alunos.

As atividades propostas têm como objetivo desenvolver as seguintes habilidades presentes em (BRASIL, 2018):

(EM13MAT310) Resolver e elaborar problemas de contagem envolvendo agrupamentos ordenáveis ou não de elementos, por meio dos princípios multiplicativo e aditivo, recorrendo a estratégias diversas, como o diagrama de árvore.

(EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

De acordo com as habilidades propostas, esta atividade é destinada ao 2^{o} ou 3^{o} anos do Ensino Médio, dependendo do currículo de cada estado, e requer o conhecimento básico de programação em Python, como já mencionado.

2.1 Primeiro encontro

Objetivos: As atividades propostas no primeiro encontro têm como objetivos principais: reconhecer os objetos dos problemas; reconhecer as configurações desejadas; construir a árvore de possibilidades; familiarizar com o código em Python; reconhecer a função do laço *for* e do condicional *if*; enunciar o Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 1:** Considere o seguinte problema: "Quatro estradas A, B, C e D conduzem

ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
1  estradas = ['A','B','C','D']
2  contagem = 0
3
4  print('Passeio' '(','Subida',',','Descida',')',':')
5
6  for subida in estradas:
7      for descida in estradas:
8          print('(',subida, ',',descida,')')
9          contagem = contagem + 1
10
11 print('Quantidade de passeios: ', contagem)
```

Código 2.1: Subindo e descendo o morro

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.1.

g) Observe o Código 2.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

***Atividade* 2:** Considere o seguinte problema: "Suponhamos que na Atividade 1 a pessoa não queira descer o morro pela estrada que subiu. De quantos modos diferentes ela pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
estradas = ['A','B','C','D']
contagem = 0

print('Passeio' '(','Subida',',','Descida',')',':')

for subida in estradas:
    for descida in estradas:
        if subida != descida:
            print('(',subida, ',',descida,')')
            contagem = contagem + 1

print('Quantidade de passeios: ', contagem)
```

Código 2.2: Subindo e descendo o morro por caminhos diferentes

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.2.

g) Observe o Código 2.2, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if subida != descida*?

***Atividade* 3:** Considere o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
moeda = ['CARA','COROA']
contagem = 0

```

```
print('Resultados',':  ' )

for lancamento1 in moeda:
    for lancamento2 in moeda:
        for lancamento3 in moeda:
            print('(',lancamento1, ',',lancamento2, ',', lancamento3,')')
            contagem = contagem + 1

print('Quantidade de sequências: ', contagem)
```

Código 2.3: Sequências de cara e coroa

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.3.

g) Observe o Código 2.3, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

***Atividade* 4:** Considere o seguinte problema: "Uma bandeira é formada por quatro listras, que devem ser coloridas usando-se apenas as cores amarelo, rosa e verde, não devendo listras adjacentes ter a mesma cor. De quantos modos pode ser colorida a bandeira?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
cores = ['amarelo','rosa', 'verde']
contagem = 0

for cor1 in cores:
    for cor2 in cores:
        if cor2 != cor1:
            for cor3 in cores:
                if cor3 != cor2:
                    for cor4 in cores:
                        if cor4 != cor3:
                            print('(',cor1, ',',cor2, ',', cor3, ',', cor4,')')
                            contagem = contagem + 1
```

```
13
14 print('Quantidade de maneiras de colorir a bandeira: ', contagem)
```

Código 2.4: Colorindo a bandeira

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.4.

g) Observe o Código 2.4, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo das linhas de código *if cor2 != cor1*, *if cor3 != cor2* e *if cor4 != cor3*?

De acordo com as respostas obtidas ao resolver os problemas das atividades do Primeiro Encontro, chegamos no enunciado do Princípio Fundamental da Contagem, conforme encontramos em (CERIOLI; VIANA, 2012):

Princípio Fundamental da Contagem(PFC). *Seja A um conjunto finito. Se cada elemento de A pode ser formado pela tomada de n decisões, $d_1, d_2, \ldots, d_n$, de maneira que:*

- *a decisão d_1 pode ser tomada de m_1 maneiras;*
- *para cada maneira como a decisão d_1 pode ser tomada, a decisão d_2 pode ser tomada de m_2 maneiras;*
- *para cada maneira como as decisões d_1 e d_2 podem ser tomadas (nesta ordem), a decisão d_3 pode ser tomada de m_3 maneiras;*

  ...;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}$ podem ser tomadas (nesta ordem), a decisão d_i pode ser tomada de m_i maneiras;*

  ...;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}, d_i, \ldots, d_{n-1}$ podem ser tomadas (nesta ordem), a decisão d_n pode ser tomada de m_n maneiras.*

Então, $n(A) = m_1 \times m_2 \times m_3 \times \ldots \times m_i \times \ldots \times m_{n-1} \times m_n$, onde $n(A)$ indica a quantidade de elementos do conjunto A.

2.2 Segundo encontro

Objetivos: As atividades propostas no segundo encontro têm como objetivos principais: reconhecer o "funcionamento" do Princípio Fundamental da Contagem no código em Python; utilizar o Princípio Aditivo junto com o Princípio Fundamental da Contagem, quando a resolução do problema divide-se em casos; determinar o número de arranjos por uma aplicação direta do Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 5:** Considere o seguinte problema: "Quantos números pares de dois algarismos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos: ', contagem)
```

Código 2.5: Números pares com dois algarismos

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Observe o Código 2.5, qual é o objetivo das linhas de código *if dezenas!= 0* e *if unidades % 2 = 0*?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

***Atividade* 6:** Considere o seguinte problema: "Quantos números pares de dois algarismos distintos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual?

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0 and dezenas != unidades:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos distintos: ', contagem)
```

Código 2.6: Números pares com dois algarismos distintos

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Observe o Código 2.6, qual é o objetivo das linhas de código *if unidades % 2 = 0* e *if dezenas!= 0 and dezenas != unidades*?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

Você deve ter notado, observando as configurações obtidas no item c) do problema da Atividade 6, que há a necessidade da utilização de um outro Princípio de Contagem, junto com o Princípio Fundamental da Contagem, para encontrar a quantidade de configurações. Tal Princípio é conhecido como Princípio Aditivo e pode ser enunciado da seguinte forma, conforme podemos encontrar em (CERIOLI; VIANA, 2012):

Sejam A um conjunto finito e $A_1, A_2, \cdots, A_n$ subconjuntos não vazios de A.

1. Dizemos que $A_1, A_2, \cdots, A_n$ *exaurem* A se, para cada elemento a de A, existe i, com $1 \leq i \leq n$, tal que $a \in A_i$; ou seja, se $A_1 \cup A_2 \cup \cdots \cup A_n = A$.

2. Dizemos que $A_1, A_2, \cdots, A_n$ são *disjuntos dois a dois* se, quando examinados aos pares, eles não possuem elementos em comum; ou seja, $A_i \cap A_j = \emptyset$, para todos i, j com $1 \leq i \neq j \leq n$.

3. Dizemos que $A_1, A_2, \cdots, A_n$ são uma partição de A se $A_1, A_2, \cdots, A_n$ exaurem A e são disjuntos dois a dois.

Princípio Aditivo. *Seja A um conjunto finito.*

Se $A_1, A_2, \cdots, A_n$ são uma partição de A, então $n(A) = n(A_1) + n(A_2) + \cdots + n(A_n)$.

***Atividade* 7:** Considere o seguinte problema: "De quantos modos 3 pessoas podem sentar-se em 5 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```python
cadeiras = ['c1','c2','c3','c4','c5']
cadeiras_ocupadas = []

for cadeira_p1 in cadeiras:
    for cadeira_p2 in cadeiras:
            if cadeira_p2 != cadeira_p1:
                for cadeira_p3 in cadeiras:
                    if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
                        cadeira_ocupada = cadeira_p1,cadeira_p2,cadeira_p3
                        cadeiras_ocupadas.append(cadeira_ocupada)

colunas = 5

print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
    print(f"{cadeira_ocupada}", end=",")
    if (i + 1) % colunas == 0:
        print()

if len(cadeiras_ocupadas) % colunas != 0:
    print()

print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código 2.7: Três pessoas em cinco cadeiras em fila

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

***Atividade* 8:** Considere o seguinte problema: "De quantos modos 6 pessoas podem sentar-se em 10 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido. Se estiver e inseguro, em relação à sua solução, construa um código em Python para resolver o problema.

Observe que as configurações obtidas nos problemas das Atividades 7 e 8 são bem parecidos. Cerioli e Viana (2012) afirmam que: "Quando, para resolver um problema de contagem, podemos aplicar o Princípio Fundamental da Contagem de modo que todas as escolhas envolvidas são feitas em um mesmo conjunto e, a cada escolha a ser feita, os objetos já escolhidos anteriormente não podem mais ser considerados, dizemos que a configuração que estamos contando é um *arranjo simples*."

Será que nos problemas das atividades anteriores já contamos arranjos simples? Se sim, identifique-os.

Definição. Seja A um conjunto com n elementos e $k \in \mathbb{N}$ tal que $1 \leq k \leq n$. Um *arranjo simples* dos n elementos de A, tomados k a k, é uma **sequência** de k elementos distintos de A.

Denotando por $A_{n,k}$ o número de arranjos simples dos n elementos de A, tomados k a k. Temos que:

$$A_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)$$

.

Note que nem a noção de arranjo simples nem a fórmula para a determinação da quantidade de arranjos simples são muito importantes na resolução dos problemas de

contagem, com esse tipo de configuração, vistos até aqui. Como vimos, podemos determinar o número de arranjos simples por uma aplicação direta do Princípio Fundamental da Contagem.

2.3 Terceiro encontro

Objetivos: As atividades propostas no terceiro encontro têm como objetivos principais: resolver problemas de contagem onde as configurações que estamos contando são permutações simples; utilizar código Python recursivo para contar permutações simples.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 9:** Considere o seguinte código em Python que resolve um determinado problema de contagem e execute-o:

```
letras = ['N','U','M','E','R','O']
anagramas = []

for letra_1 in letras:
    for letra_2 in letras:
            if letra_2 != letra_1:
                for letra_3 in letras:
                    if letra_3 != letra_2 and letra_3 != letra_1:
                        for letra_4 in letras:
                            if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 !=
    letra_1:
                                for letra_5 in letras:
                                    if letra_5 != letra_4 and letra_5 != letra_3 and
    letra_5 != letra_2 and letra_5 != letra_1:
                                        for letra_6 in letras:
                                            if letra_6 != letra_5 and letra_6!= letra_4
    and letra_6 != letra_3 and letra_6 != letra_2 and letra_6 != letra_1:
                                                anagrama = (letra_1 + letra_2 + letra_3 +
     letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)


colunas = 20
largura_coluna = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
    print()

```

```
31 print('Quantidade de anagramas: ', len(anagramas))
```

Código 2.8: Código da Atividade 9

a) Pesquise o que são anagramas.

b) Quais são os objetos básicos do problema?

c) Quais são as configurações formadas?

d) Quantas decisões foram tomadas para resolver o problema? Quais foram essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão foi tomada?

e) Quantas configurações foram obtidas? Com isso, em que consiste o problema resolvido pelo Código 2.8?

f) Utilizando o Princípio Fundamental da Contagem, obtenha a solução do problema resolvido pelo Código 2.8.

No problema da Atividade 9, queremos determinar o número de arranjos simples de 6 letras, tomadas 6 a 6, quando isso ocorre dizemos que a configuração que estamos contando é uma *permutação simples*.

Definição. Seja A um conjunto com n elementos. Uma *permutação simples* dos n elementos de A é uma **sequência** sem repetições de todos os n elementos de A.

Denotando por P_n o número de permutações simples dos n elementos de A. Temos que:

$$P_n = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

.

O número $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$ é denotado por $n!$ e é chamado *fatorial* de n.

***Atividade* 10:** Considere o seguinte problema: "Quantos são os anagramas da palavra MOSTRA que começam por vogal e terminam por consoante?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
letras = ['M','O','S','T','R','A']
vogais = ['A','O']
consoantes = ['M','R','S','T']
anagramas = []

for letra_1 in letras:
    if letra_1 in vogais:
        for letra_6 in letras:
            if letra_6 in consoantes:
                for letra_2 in letras:
                    if letra_2 != letra_1 and letra_2 != letra_6:
                        for letra_3 in letras:
                            if letra_3 != letra_2 and letra_3 != letra_1 and
    letra_3 != letra_6:
                                for letra_4 in letras:
                                    if letra_4 != letra_3 and letra_4 != letra_2
    and letra_4 != letra_1 and letra_4 != letra_6:
                                        for letra_5 in letras:
                                            if letra_5 != letra_4 and letra_5 !=
    letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:
                                                anagrama = (letra_1 + letra_2 +
    letra_3 + letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)

colunas = 12

print(f"\nAnagramas da palavra MOSTRA que começam por vogal e teminam por consoante
    : ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(anagramas))
```

Código 2.9: Anagramas com restrições

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

***Atividade* 11:** Considere o seguinte código recursivo em Python que resolve um determinado problema de contagem:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = 'NUMERO'

todas_permutacoes = permutacoes(palavra)

colunas = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(todas_permutacoes) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código 2.10: Anagramas de NÚMERO e recursividade

a) Execute o código acima. O que esse código fornece como resultado?

b) Compare-o com o Código 2.9. Qual é a diferença?

c) Considere uma palavra com uma quantidade menor de letras, por exemplo ALO, e execute o Código 2.10 passo a passo manualmente, para entender o seu funcionamento.

***Atividade* 12:** Considere a variação do Código 2.10 em Python que determina os anagramas de uma palavra:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

```

```
    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = str(input("Digite uma palavra: ")).upper()

todas_permutacoes = permutacoes(palavra)

colunas = 8

print(f"\nAnagramas da palavra {palavra}: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(todas_permutacoes) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código 2.11: Permutações simples e recursividade

a) Execute o código acima, usando como exemplo a palavra AMOR.

b) Usando o Princípio Fundamental da Contagem, determine o número de anagramas da palavra AMOR. Compare o seu resultado com o aquele fornecido pelo Código 2.11.

c) Execute o código usando, agora, como exemplo a palavra AMAR.

d) Analise, atentamente, a saída de execução do código. Você percebeu alguma coisa estranha? O quê?

e) Então, quantos são os anagramas da palavra AMAR?

f) Repita os itens c), d) e e) considerando a palavra BABA.

g) Repita os itens c), d) e e) considerando a palavra AAZA(força em arábico).

h) Por quê o Código 2.11 não funciona com as palavras AMAR, BABA e AAZA?

2.4 Quarto encontro

Objetivos: As atividades propostas no quarto encontro têm como objetivos principais: mostrar que o Princípio Fundamental da Contagem gera um excesso na resolução de problemas onde os objetos que serão permutados não são todos distintos e quando da determinação da quantidade de grupos(conjuntos); diferenciar uma sequência de um conjunto; corrigir a superestimação, construída pelo Princípio Fundamental da Contagem, utilizando o Princípio k para 1.

Tempo para execução: 3 aulas de 50 minutos.

Na Atividade 12, deve-se ter notado que o código que calcula os anagramas de uma palavra só funciona corretamente quando a mesma possui letras distintas, ou seja, o código só calcula a quantidade de permutações simples.

O código da Atividade 13 tem como objetivo corrigir esse problema, isto é, calcular o número de anagramas de qualquer palavra.

***Atividade* 13:** Considere o seguinte código recursivo em Python que determina os anagramas de uma palavra mas, agora, de uma forma mais organizada, resolvendo os problemas que tivemos na Atividade 12:

```
def permutacoes(palavra):

    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado


def remover_duplicatas_sort(lista):

    lista_ordenada = sorted(lista)

    lista_de_listas = []
    grupo_atual = []
    for i in range(len(lista_ordenada)):
        if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
            grupo_atual.append(lista_ordenada[i])
        else:
```

```
27              lista_de_listas.append(grupo_atual)
28              grupo_atual = [lista_ordenada[i]]
29      lista_de_listas.append(grupo_atual)
30
31      for i, uplas in enumerate(lista_de_listas, 1):
32          print(f"{i:3}. {uplas}")
33      print()
34      print("Número de anagramas: ", len(lista_ordenada))
35      print()
36      print("Número de anagramas repetidos em cada caixa: ", len(grupo_atual))
37      print()
38
39
40      resultado = [lista_ordenada[0]]
41
42      for i in range(1, len(lista_ordenada)):
43          if lista_ordenada[i] != lista_ordenada[i-1]:
44              resultado.append(lista_ordenada[i])
45      return resultado
46
47 palavra = str(input("Digite uma palavra qualquer: ")).upper()
48
49 todas_permutacoes = permutacoes(palavra)
50 permutacoes_com_elemetos_repetidos = remover_duplicatas_sort(todas_permutacoes)
51
52
53 print(f"\nAnagramas da palavra {palavra}: ")
54 for i, perm in enumerate(permutacoes_com_elemetos_repetidos, 1):
55      print(f"{i:3}. {perm}")
56 print(f"Quantidade de anagramas da palavra {palavra}: {len(
      permutacoes_com_elemetos_repetidos)}")
```

Código 2.12: Permutações com elementos repetidos

a) Execute o código acima, usando como exemplo a palavra AMAR.

b) O resultado obtido é, de fato, a quantidade de anagramas da palavra AMAR?

c) Execute o código usando, agora, como exemplo a palavra AAZA.

d) O resultado obtido é, de fato, a quantidade de anagramas da palavra AAZA?

e) Execute o Código 2.12, usando a palavra SOSSEGO. Quantos anagramas essa palavra possui?

Na execução do código da Atividade 13, perceba que para descobrir a quantidade de anagramas das palavras com letras repetidas, houve necessidade de "jogar fora" excessos. Note que, em palavras com letras repetidas, para k anagramas obtidos(repetidos) apenas

1 é considerado anagrama da palavra em questão. Temos, assim, a utilização do Princípio k para 1 em conjunto com o Princípio Fundamental da Contagem.

Definição. Sejam A e B conjuntos finitos. Uma *função k para 1* de A em B associa elementos de A a elementos de B, de modo que:

- Cada k elementos de A estão associados a 1 elemento de B.
- Cada elemento de B é o associado de exatamente k elementos de A.

Princípio k para 1. *Sejam A e B conjuntos finitos. Se existe uma função k para 1 de A em B, então $n(A) = k \cdot n(B)$.*

Com isso, podemos concluir, que a quantidade de permutações de n objetos nem todos distintos, em que um deles aparece n_1 vezes(fazendo com que contemos cada anagrama $n_1!$ vezes), outro n_2 vezes(fazendo com que contemos cada anagrama $n_2!$ vezes), e assim por diante, é dada por

$$P_n^{n_1,n_2,\cdots} = \frac{n!}{n_1! \cdot n_2! \cdot \ldots}.$$

f) Determine o número de anagramas das palavras MANIA, CORRER e PASSISTA, usando o Princípio Fundamental da Contagem e o Princípio k para 1.

***Atividade* 14:** Vamos ao seguinte problema: "Considere 5 pessoas A, B, C, D e E. Quantas filas com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

```

```
    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado



pessoas = ['A','B','C','D','E']

filas_obtidas = formar_filas(pessoas)
```

Código 2.13: Filas com três pessoas dadas cinco pessoas

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

f) Esse problema é parecido com algum problema que já foi resolvido? Qual?

g) E quantas filas com 4 pessoas podemos formar?

***Atividade* 15:** Agora, vamos a esse problema: "Considere 5 pessoas A, B, C, D e E. Quantos grupos com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) A solução desse problema é igual à solução do problema da Atividade 14? Ou seja, a quantidade de filas é igual à quantidade de grupos? Justifique.

d) Se a resposta do item anterior for NÃO, corrija a resposta, obtendo, assim, a resposta correta.

No problema da Atividade 14, estamos, claramente, buscando a quantidade de arranjos simples de 5 pessoas tomadas 3 a 3. Já no problema da Atividade 15, mais uma vez, se utilizarmos o Código 2.13 teremos excesso e, para corrigir a solução, necessitamos utilizar o Princípio k para 1.

No problema da Atividade 15 temos 5 pessoas e devemos efetuar uma escolha na qual 3 pessoas são tomados simultaneamente, dizemos, então, que a configuração que estamos contando é uma *combinação simples*.

Antes de definir formalmente *combinação simples*, vejamos a Atividade 16.

***Atividade* 16:** Considere o seguinte código em Python que resolve um determinado problema de contagem:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado

def ordenar_e_imprimir_filas(filas):

    print("\n" + "=" * 60)
    print("PASSO 2: Ordena cada fila individualmente")
    print("=" * 60)

    filas_ordenadas = []
    ordenadas = []

    print("\nTodas as filas ORDENADAS:")
    print("-" * 60)

    for idx, fila in enumerate(filas, 1):
        fila_ordenada = sorted(fila)
        filas_ordenadas.append(fila_ordenada)
        print(f"Fila {idx}: {fila_ordenada}")
```

```
39
40      return filas_ordenadas
41
42  def remover_duplicatas_sort(lista):
43
44      print("\n" + "=" * 60)
45      print("PASSO 3: Coloca as filas iguais em uma mesma caixa")
46      print("=" * 60)
47
48      lista_ordenada = sorted(lista)
49
50      lista_de_listas = []
51      grupo_atual = []
52      for i in range(len(lista_ordenada)):
53          if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
54              grupo_atual.append(lista_ordenada[i])
55          else:
56              lista_de_listas.append(grupo_atual)
57              grupo_atual = [lista_ordenada[i]]
58
59      lista_de_listas.append(grupo_atual)
60
61      for i, uplas in enumerate(lista_de_listas, 1):
62          print(f"{i:3}. {uplas}")
63      print()
64      print("Número de filas: ", len(lista_ordenada))
65      print()
66      print("Depois que as filas são ordenadas, temos: ", len(grupo_atual) , " filas em
        cada caixa.")
67      print()
68
69      resultado = [lista_ordenada[0]]
70
71      for i in range(1, len(lista_ordenada)):
72          if lista_ordenada[i] != lista_ordenada[i-1]:
73              resultado.append(lista_ordenada[i])
74      return resultado
75
76
77  pessoas = ['A','B','C','D','E']
78
79  filas_obtidas = formar_filas(pessoas)
80  ordenadas = ordenar_e_imprimir_filas(filas_obtidas)
81  grupos = remover_duplicatas_sort(ordenadas)
82
83  print("Todas os grupos:")
84  for i, perm in enumerate(grupos, 1):
85      print(f"{i:3}. {perm}")
86  print(f"Número de grupos: {len(grupos)}")
```

Código 2.14: Código da Atividade 16

a) Execute o código acima.

b) O que esse código fornece como resultado?

c) Compare-o com o Código 2.13. Qual é a diferença?

d) Esse código fornece a solução do problema da Atividade 15?

e) Usando a ideia do código acima, dadas cinco pessoas quantos grupos com quatro pessoas podemos formar? E, dadas dez pessoas quantos grupos com quatro pessoas podemos formar?

Finalmente, temos a definição formal para Combinações Simples.

Definição. Seja A um conjunto com n elementos e $k \in \mathbb{N}$ tal que $1 \leq k \leq n$. Uma *combinação simples* dos n elementos de A, tomados k a k, é um **conjunto** de k elementos distintos de A.

Pela definição, como uma combinação é um **conjunto** com elementos de A. A ordem em que os elementos estão listados não é levada em conta na determinação da combinação.

Denotando por $C_{n,k}$ o número de combinações simples dos n elementos de A, tomados k a k. O número de combinações simples dos n elementos de A, tomados k a k, é dado pela fórmula

$$C_{n,k} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-(k-1))}{k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1} = \frac{A_{n,k}}{P_k}.$$

3 Resultados Esperados

3.1 Primeiro encontro

***Atividade* 1:** Considere o seguinte problema: "Quatro estradas A, B, C e D conduzem ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?".

> Geralmente, a palavra "diferentes" no enunciado do problema, pode causar um "estranhamento", mas, tal palavra refere-se ao par completo, subida e descida, e não necessariamente às estradas individuais. Assim, espera-se a seguinte Leitura Plausível, por exemplo, subir por B e descer por B é um "modo diferente" de subir por B e descer por D.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

> Nesse item temos a identificação dos elementos que formam o "mundo" do problema.
>
> - Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 4 estradas: $\{A, B, C, D\}$.
> - Configurações: A configuração é um trajeto de subida e descida. Matematicamente, é um par ordenado $(Subida, Descida)$.
>
> Ao resolver este item, o aluno é convidado a colocar-se no papel da pessoa que deve realizar a ação solicitada pelo problema, ou seja, o aluno deve se colocar no papel da pessoa que deve subir e descer o morro.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras

cada decisão pode ser tomada?

Precisamos tomar 2 decisões:1ª)Escolher a estrada de subida; 2ª) Escolher a estrada de descida.
Não há restrição. O enunciado diz "subir e descer", mas não diz "descer por uma estrada diferente". Se o aluno assumir que não pode repetir a estrada, ele inseriu uma regra que não existe no texto.
Temos, então:
1ª decisão(subida): 4 maneiras $\rightarrow \{A, B, C, D\}$;
2ª decisão(descida): 4 maneiras $\rightarrow \{A, B, C, D\}$.
Com a postura exercida pelo aluno, descrita no item anterior, ele consegue vislumbrar que o problema pode ser dividido em dois problemas menores, primeiro ele resolve o problema "escolher a estrada para subir o morro" e depois, já tendo subido o morro, resolve o segundo problema "escolher a estrada para descer do morro".

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Agora é o momento de montar os trajetos. Geralmente, o aluno fixa a estrada de subida e, a partir daí, escolhe a estrada de descida:

- **subida** $\rightarrow A \Rightarrow$ **Possíveis configurações:** $(A,A),(A,B),(A,C),(A,D)$.
- **subida** $\rightarrow B \Rightarrow$ **Possíveis configurações:** $(B,A),(B,B),(B,C),(B,D)$.
- **subida** $\rightarrow C \Rightarrow$ **Possíveis configurações:** $(C,A),(C,B),(C,C),(C,D)$.
- **subida** $\rightarrow D \Rightarrow$ **Possíveis configurações:** $(D,A),(D,B),(D,C),(D,D)$.

A partir daí, temos a seguinte árvore de possibilidades.

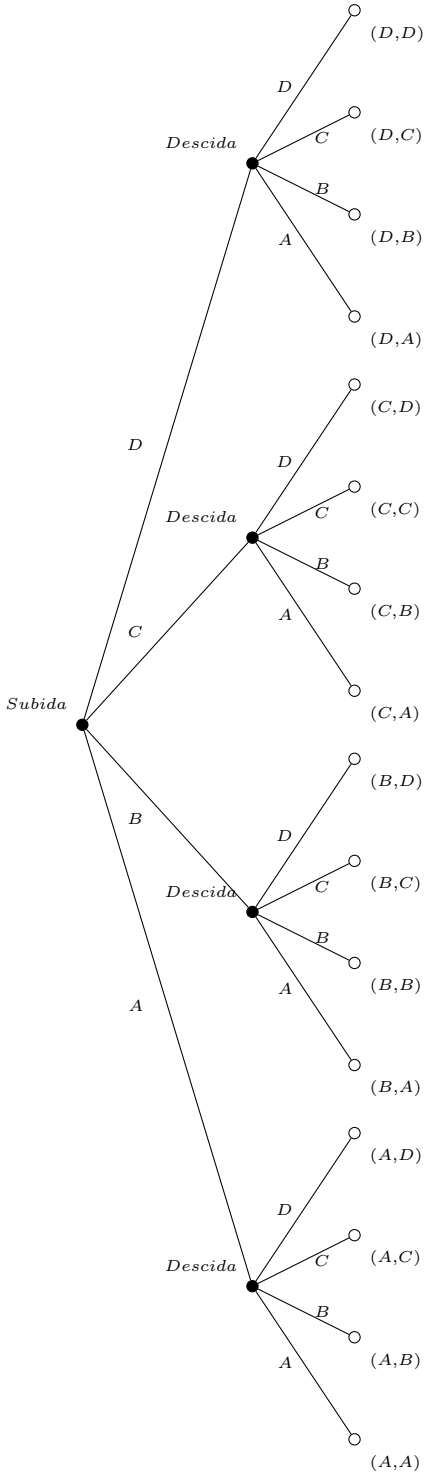

Figura 1: Árvore de Possibilidades da Atividade 1.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 16 configurações.

Para esse item poderemos obter as seguintes enunciações como respostas:

$$4 + 4 + 4 + 4 = 16 \text{ ou } 4 \times 4 = 16.$$

e) Agora, execute o código 2.1.

Temos o seguinte resultado ao executar o Código 2.1.

```
Passeio( Subida , Descida ) :
( A , A )
( A , B )
( A , C )
( A , D )
( B , A )
( B , B )
( B , C )
( B , D )
( C , A )
( C , B )
( C , C )
( C , D )
( D , A )
( D , B )
( D , C )
( D , D )
Quantidade de passeios:  16
>>>
```

Figura 2: Saída do Código 2.1.
Fonte: Autoria própria(2026).

Mas, executar não é simplesmente "rodar" o código. Há a necessidade de analisar o que o código faz:

I. **Linha 1:** *estradas = ['A','B','C','D']* → Define os objetos básicos do problema.

II. **Linha 6:** *for subida in estradas:* → Representa a 1ª Decisão. O computador "escolhe" uma estrada e "fixa" ela.

III. **Linha 7:** *for descida in estradas:* → Representa a 2ª Decisão. Para cada subida que está "fixada", o computador testa todas as descidas possíveis.

O fato de um *for* estar dentro do outro (aninhado) é a representação computacional da multiplicação matemática (4×4) e da ramificação da árvore.

IV. **Linha 8:** *print(...)* $\rightarrow$ Materializa a configuração (o par ordenado).

V. **Linha 9:** *contagem = contagem + 1* $\rightarrow$ Realiza a contagem passo a passo.

VI. **Resultado da execução do código:** O programa imprimirá linha por linha:*('A','A'),('A','B')... até ...('D','D').*

E ao final: *Quantidade de passeios: 16*

Visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

- **Laço externo:** Escolhe *subida = 'A'*, o computador "fixa" o A.
- **Entra no Laço interno:**
  - **Laço interno:**Escolhe *descida = 'A'* $\rightarrow$ Imprime (A, A).
  - **Laço interno:** Escolhe *descida = 'B'* $\rightarrow$ Imprime (A, B). (Note que a subida continua $'A'$).
  - **Laço interno:** Escolhe *descida = 'C'* $\rightarrow$ Imprime (A, C).
  - **Laço interno:** Escolhe *descida = 'D'* $\rightarrow$ Imprime (A, D).
- **Fim do Laço interno:** Acabaram as opções de descida para o $'A'$.
- **Laço externo:** Escolhe *subida = 'B'*.
- **O Laço interno recomeça do zero:**
  - **Laço interno:** Escolhe *descida = 'A'* $\rightarrow$ Imprime (B, A).
  - **Laço interno:** Escolhe *descida = 'B'* $\rightarrow$ Imprime (B, B)
  - ... e assim por diante.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.1.

O aluno deve entender que "Subir por A e Descer por B" é um significado diferente de "Subir por B e Descer por A".
A Árvore mostra visualmente todos os passeios. O Python "prova" a resposta gerando concretamente todas as 16 possibilidades, usando a mesma ideia utilizada pela maioria dos alunos descrita no item c).

g) Observe o Código 2.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

Foram utilizados 2 laços for. E cada um desses laços representa uma decisão.
O laço externo *for subida in estradas:* representa a 1ª Decisão. Na árvore de possibilidades, isso representa os galhos principais que saem do tronco.
O laço interno *for descida in estradas:* representa a 2ª Decisão. Na árvore de possibilidades, estes são os ramos menores que saem de cada galho principal.

***Atividade* 2:** Considere o seguinte problema: "Suponhamos que na Atividade 1 a pessoa não queira descer o morro pela estrada que subiu. De quantos modos diferentes ela pode subir e descer este morro?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 4 estradas: $\{A, B, C, D\}$.
- Configurações: A configuração é um trajeto de subida e descida. Matematicamente, é um par ordenado $(Subida, Descida)$.

  Diferente da Atividade 1, agora o par (A, A), por exemplo, tornou-se ilegítimo. Ou seja, devemos ter que $Subida \neq Descida$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Diferente do problema anterior, temos aqui a introdução de uma restrição de dependência (“não descer pela mesma estrada”).
Temos, então:
1ª decisão(subida): 4 maneiras $\rightarrow \{A, B, C, D\}$;
2ª decisão(descida): O número de maneiras depende da escolha anterior. Temos então, 3 maneiras. Por exemplo, se a estrada de subida é a A então a estrada de descida pode ser $\{B, C, D\}$.
Quando o professor perguntar “Por que 3?”, a Justificação do aluno será: “Porque eu já usei uma na subida e não posso repetir”. Essa fala explicita a restrição do problema.

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Fixando a estrada de subida e, a partir daí, escolhendo a estrada de descida:

- **subida** $\rightarrow A \Rightarrow$ **Possíveis configurações:** $(A,B),(A,C),(A,D)$.
- **subida** $\rightarrow B \Rightarrow$ **Possíveis configurações:** $(B,A),(B,C),(B,D)$.
- **subida** $\rightarrow C \Rightarrow$ **Possíveis configurações:** $(C,A),(C,B),(C,D)$.
- **subida** $\rightarrow D \Rightarrow$ **Possíveis configurações:** $(D,A),(D,B),(D,C)$.

A partir daí, temos a seguinte árvore de possibilidades.

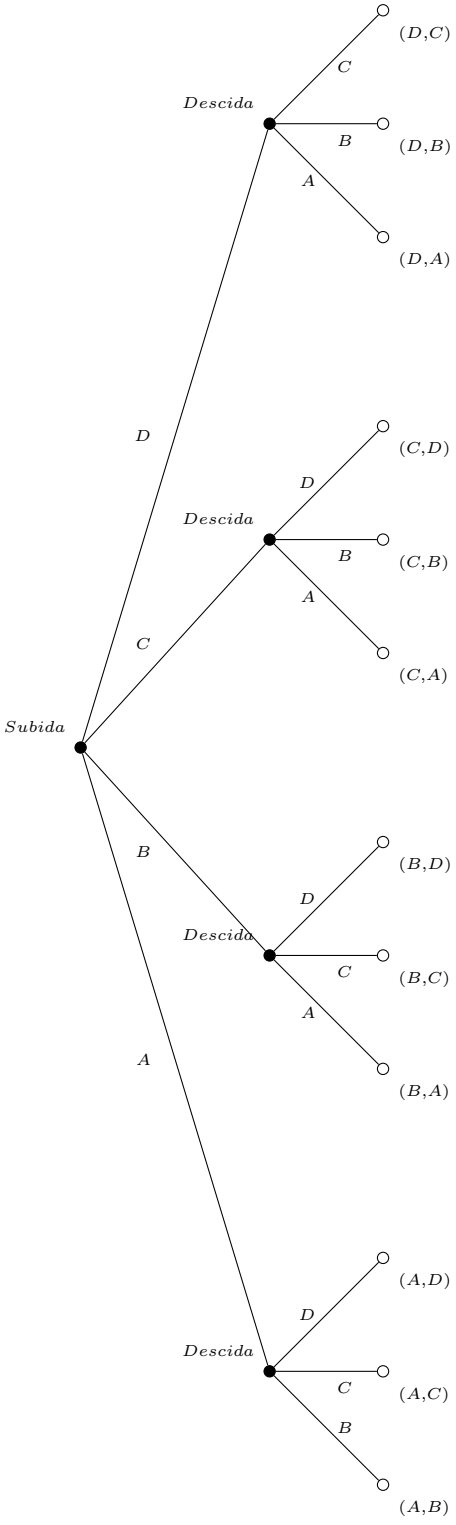

Figura 3: Árvore de Possibilidades da Atividade 2.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 12 configurações.

Para esse item poderemos obter as seguintes enunciações como respostas:

$$3+3+3+3=12 \text{ ou } 4\times 3=12.$$

e) Agora, execute o código 2.2

Obtemos o seguinte resultado.

```
Passeio( Subida , Descida ) :
( A , B )
( A , C )
( A , D )
( B , A )
( B , C )
( B , D )
( C , A )
( C , B )
( C , D )
( D , A )
( D , B )
( D , C )
Quantidade de passeios:  12
>>>
```

Figura 4: Saída do Código 2.2.
Fonte: Autoria própria(2026).

I. **Linha 1:** *estradas = ['A','B','C','D']* → Define os objetos básicos do problema.

II. **Linha 6:** *for subida in estradas:* → Representa a 1ª Decisão. O computador "escolhe" uma estrada e "fixa" ela.

III. **Linha 7:** *for descida in estradas:* → Representa a 2ª Decisão. Para cada subida que está "fixada", o computador testa todas as descidas possíveis.

IV. **Linha 8:** *if subida != descida* → Esta linha é a materialização da restrição de que a repetição é proibida. Ela atua como um filtro. Ela diz: "Eu gerei o par (A,A), mas a restrição do problema diz que ele não serve. Descarte-o."

V. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VI. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VII. **Resultado da execução do código:** O programa imprimirá linha por linha:*('A','B')... até ...('D','C')*.

E ao final: *Quantidade de passeios: 12*

Visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

- **Laço externo:** Escolhe *subida = 'A'*, o computador "fixa" o A.
- **Entra no Laço interno:**
  - **Laço interno:**Escolhe *descida = 'A'* → Não Imprime (A, A), devido a restrição.
  - **Laço interno:** Escolhe *descida = 'B'* → Imprime (A, B). (Note que a subida continua $'A'$).
  - **Laço interno:** Escolhe *descida = 'C'* → Imprime (A, C).
  - **Laço interno:** Escolhe *descida = 'D'* → Imprime (A, D).
- **Fim do Laço interno:** Acabaram as opções de descida para o $'A'$.
- **Laço externo:** Escolhe *subida = 'B'*.
- **O Laço interno recomeça do zero:**
  - **Laço interno:** Escolhe *descida = 'A'* → Imprime (B, A).
  - **Laço interno:** Escolhe *descida = 'B'* → Não Imprime (B, B), devido a restrição.
  - ... e assim por diante.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.2.

A Árvore mostra visualmente todos os passeios. O Python "prova" a resposta gerando concretamente todas as 12 possibilidades, usando a mesma ideia utilizada pela maioria dos alunos descrita no item c).

g) Observe o Código 2.2, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if subida != descida*?

Foram utilizados 2 laços for. E cada um desses laços representa uma decisão. O laço externo *for subida in estradas:* representa a 1ª Decisão. Na árvore de possibilidades, isso representa os galhos principais que saem do tronco.
O laço interno *for descida in estradas:* representa a 2ª Decisão. Na árvore de possibilidades, estes são os ramos menores que saem de cada galho principal.
É importante notar que o código está percorrendo um espaço de $4 \times 4 = 16$ possibilidades totais, mas, a linha *if subida != descida*, como visto na resolução do item e), é a materialização da restrição de que a repetição é proibida. Essa linha exclui as 4 possibilidades (A,A),(B,B),(C,C),(D,D) na resolução do problema.
Enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 2ª Decisão(3 maneiras), no código Python a restrição está escrita e visível na linha *if subida != descida:*.

***Atividade* 3:** Considere o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 2 faces da moeda: $\{CARA, COROA\}$.
- Configurações: A configuração é um resultado do 1º lançamento, do 2º lançamento e do 3º lançamento. Matematicamente, é uma tripla ordenada $(lancamento1, lancamento2, lancamento3)$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Precisamos tomar 3 decisões:1ª)Escolher a face vista no 1º lançamento; 2ª) Escolher a face vista no 2º lançamento; 3ª) Escolher a face vista no 3º lançamento.

Não há restrição no problema. Se o aluno assumir que não pode repetir a face vista em um lançamento anterior, ele inseriu uma regra que não existe no texto.

Temos, então:

1ª decisão(1º lançamento): 2 maneiras $\rightarrow \{CARA, COROA\}$;

2ª decisão(2º lançamento): 2 maneiras $\rightarrow \{CARA, COROA\}$;

3ª decisão(3º lançamento): 2 maneiras $\rightarrow \{CARA, COROA\}$.

Com a postura ativa assumida pelo aluno no item anterior, ele consegue vislumbrar que o problema pode ser dividido em três problemas menores (dividir para conquistar), primeiro ele resolve o problema "escolher a possível face ao lançar a moeda pela primeira vez", já tendo feito a primeira escolha, resolve o segundo problema "escolher a possível face ao lançar a moeda pela segunda vez" e, feito isso, resolve o terceiro problema "escolher a possível face ao lançar a moeda pela terceira vez".

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Podemos formar as seguintes configurações:

1ª e 2ª faces iguais a *cara* $\rightarrow$ (*cara,cara,cara*), (*cara,cara,coroa*);

1ª face *cara* e 2ª face *coroa* $\rightarrow$ (*cara,coroa,cara*), (*cara,coroa,coroa*);

1ª face *coroa* e 2ª face *cara* $\rightarrow$ (*coroa,cara,cara*), (*coroa,cara,coroa*);

1ª e 2ª faces iguais a *coroa* $\rightarrow$ (*coroa,coroa,cara*), (*coroa,coroa,coroa*).

Que nos fornece a seguinte árvore de possibilidades:

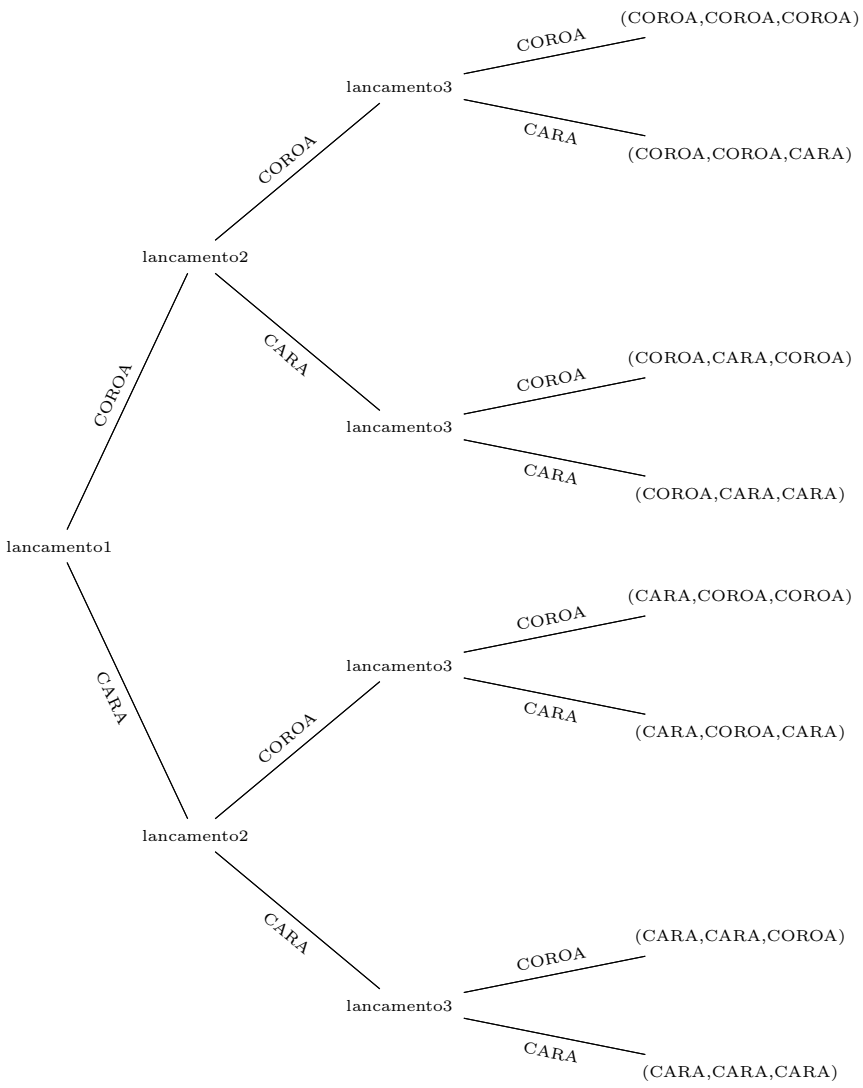

Figura 5: Árvore de Possibilidades da Atividade 3.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 8 configurações.

Para esse item poderemos obter as seguintes enunciações como respostas:

$$2 + 2 + 2 + 2 = 8 \text{ ou } 2 \times 2 \times 2 = 12.$$

e) Agora, execute o código 2.3.

```
Resultados :
( CARA , CARA , CARA )
( CARA , CARA , COROA )
( CARA , COROA , CARA )
( CARA , COROA , COROA )
( COROA , CARA , CARA )
( COROA , CARA , COROA )
( COROA , COROA , CARA )
( COROA , COROA , COROA )
Quantidade de sequências:  8
```

Figura 6: Saída do Código 2.3.
Fonte: Autoria própria(2026).

Analisando o que o código faz:

I. **Linha 1:** *moeda = ['CARA','COROA']* → Define os objetos básicos do problema.

II. **Linha 6:** *for lancamento1 in moeda:* → Representa a 1ª Decisão. O computador "escolhe" uma face e "fixa" ela.

III. **Linha 7:** *for lancamento2 in moeda:* → Representa a 2ª Decisão. Para cada face que está "fixada" na 1ª decisão, o computador "escolhe" uma face e "fixa".

IV. **Linha 8:** *for lancamento3 in moeda:* → Representa a 3ª Decisão. Para cada face que será "fixada" na 2ª decisão, o computador testa todas as outras faces possíveis.

V. **Linha 9:** *print(...)* → Materializa a configuração (a tripla ordenada).

VI. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VI. **Resultado da execução do código:** O programa imprimirá linha por linha:*('CARA','CARA','CARA'),('CARA','CARA','COROA')...até ...('COROA','COROA','COROA').*

E ao final: *Quantidade de sequências: 8*

Visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

- **Laço externo:** Escolhe *lancamento1 = 'CARA'*, o computador "fixa" $CARA$.
- **Entra no 1º Laço interno:**
  - **Laço interno:** Escolhe *lancamento2 = 'CARA'*,o computador "fixa" $CARA$.
  - **Entra no 2º Laço interno:**
    - **Laço interno:** Escolhe *lancamento3 = 'CARA'* $\rightarrow$ Imprime ($CARA$,$CARA$,$CARA$)
    - **Laço interno:** Escolhe *lancamento3 = 'COROA'* $\rightarrow$ Imprime ($CARA$,$CARA$,$COROA$)
  - **Fim do 2º laço interno:** Acabaram as opções do 3º lançamento para $'CARA'$.
  - **Laço interno:** Escolhe *lancamento2 = 'COROA'*.
  - **Entra no 2º Laço interno:**
    - **Laço interno:** Escolhe *lancamento3 = 'CARA'* $\rightarrow$ Imprime ($CARA$,$COROA$,$CARA$)
    - **Laço interno:** Escolhe *lancamento3 = 'COROA'* $\rightarrow$ Imprime ($CARA$,$COROA$,$COROA$)
  - **Fim do 2º laço interno:** Acabaram as opções do 3º lançamento para $'COROA'$.
- **Fim do 1º Laço interno.**
- **Laço externo:** Escolhe *lancamento1 = 'COROA'*.
- **Entra no 1º Laço interno:**
  - **Recomeça todo o processo.**

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.3.

> A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 8 resultados.

g) Observe o Código 2.3, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

> Foram utilizados 3 laços for. E cada um desses laços representa uma decisão. O laço externo *for lancamento1 in moeda:* representa a 1ª Decisão. Na árvore de possibilidades, isso representa os galhos principais que saem do tronco.
> O 1º laço interno *for lancamento2 in moeda:* representa a 2ª Decisão. Na árvore de possibilidades, estes são os galhos menores que saem de cada galho principal.
> O 2º laço interno *for lancamento3 in moeda:* representa a 3ª Decisão. Na árvore de possibilidades, estes são os ramos menores que saem de cada galho menor.
> Se, por acaso, a moeda fosse lançada mais uma vez, teríamos no problema quatro decisões e o nosso código, como consequência, teria quatro comandos *for*.

***Atividade* 4:** Considere o seguinte problema: "Uma bandeira é formada por quatro listras, que devem ser coloridas usando-se apenas as cores amarelo, rosa e verde, não devendo listras adjacentes ter a mesma cor. De quantos modos pode ser colorida a bandeira?".

> Esta atividade, a princípio, produz um estranhamento em alguns alunos: "Como vou colorir a bandeira com quatro listras, se possuo apenas três cores para usar?". Ou seja, eles acham que uma das listras ficará sem colorir.
> Por isso, para esse problema, é interessante propor que o aluno desenhe uma bandeira com quatro listras e pinte-a, obedecendo as regras do problema. Com isso, eles percebem que, pelo menos, uma das cores pode se repetir.
> Então, o aluno começa a produzir o significado de que a *listra 1* e a *listra 3*, por exemplo, podem ter a mesma cor, mas a *listra 1* e a *listra 2* não.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 3 cores disponíveis: $\{amarelo, rosa, verde\}$.
- Configurações: A configuração é uma bandeira inteira pintada corretamente, respeitando a regra de que listras adjacentes possuem cores diferentes. Matematicamente, é uma quádrupla ordenada $(cor1, cor2, cor3, cor4)$, onde $cor1 \neq cor2$, $cor2 \neq cor3$ e $cor3 \neq cor4$.

O aluno deve se colocar no papel da pessoa que está pintando a bandeira.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Precisamos tomar 4 decisões:1ª) Escolher a cor da primeira listra a ser colorida; 2ª) Escolher a cor da segunda listra a ser colorida; 3ª) Escolher a cor da terceira listra a ser colorida; 4ª) Escolher a cor da quarta listra a ser colorida.
Há uma restrição no problema. O enunciado diz "não devendo listras adjacentes ter a mesma cor". Isto é, há uma restrição de dependência local.
Sem perda de generalidade, escolheremos a seguinte bandeira para representar a bandeira que queremos colorir:

1ª listra
2ª listra
3ª listra
4ª listra

Agora, precisamos começar a colorir a bandeira.
O aluno poderia, agora, fazer a seguinte enunciação:"Vou colorir primeiro a 1ª listra da bandeira e depois a 3ª listra, não estou querendo me preocupar com a restrição do problema agora, e, depois vou colorir a 2ª e 4ª listras."

Nesse caso, o aluno pode enunciar que:
"1ª decisão(cor da primeira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;
3ª decisão(cor da segunda listra): 1 maneira → $\{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;
4ª decisão(cor da quarta listra): 2 maneiras → $\{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."
Mas, também, poderíamos ter a seguinte enunciação:
"1ª decisão(cor da primeira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
3ª decisão(cor da segunda listra): 2 maneiras → $\{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;
4ª decisão(cor da quarta listra): 2 maneiras → $\{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."
A partir das duas enunciações, observemos que a resposta da 3ª decisão depende da cor escolhida para colorir a 3ª listra, a realização da 2ª decisão. Com isso, temos dois casos a resolver: o caso em que as cores das 1ª e 3ª listras são iguais e o caso em que as cores dessas listras são diferentes.
A estratégia escolhida pelo aluno resolve o problema mas, chega nessa indecisão, obrigando que o problema seja resolvido em dois casos, torna-se mais trabalhoso. Em momento oportuno, voltaremos a esse problema e utilizaremos a estratégia acima para resolvê-lo.
Mas, tem como contornar, por enquanto, essa coisa "desagradável". Para isso, Morgado *et al.* (2006) recomendam a seguinte estratégia: "Pequenas dificuldades adiadas costumam transformar-se em grandes dificuldades. Se alguma decisão é mais complicada que as demais, ela deve ser tomada em primeiro lugar."

Ou seja, usando a estratégia acima, temos que a primeira listra a ser colorida na bandeira pode ser qualquer uma das quatro existentes. Devemos ter atenção na escolha das demais listras, a segunda listra escolhida, por exemplo, deve ser adjacente à listra anteriormente colorida. Então, podemos ter o seguinte:
1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$;
2ª decisão(cor da segunda listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da primeira listra é *rosa* então a cor da segunda listra pode ser $\{amarelo, verde\}$.
Quando o professor perguntar "Por que 2?", a Justificação do aluno será: "Porque eu não posso usar a mesma cor que usei na listra anterior, que é adjacente a listra que estou colorindo agora". Essa fala explicita a restrição do problema.
3ª decisão(cor da terceira listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da segunda listra é *verde* então a cor da terceira listra pode ser $\{amarelo, rosa\}$.
4ª decisão(cor da quarta listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da terceira listra é *amarelo* então a cor da quarta listra pode ser $\{verde, rosa\}$.

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Usando as mesmas estratégias utilizadas nas atividades anteriores, obtemos as configurações elencadas na árvore de possibilidades da Figura 7.

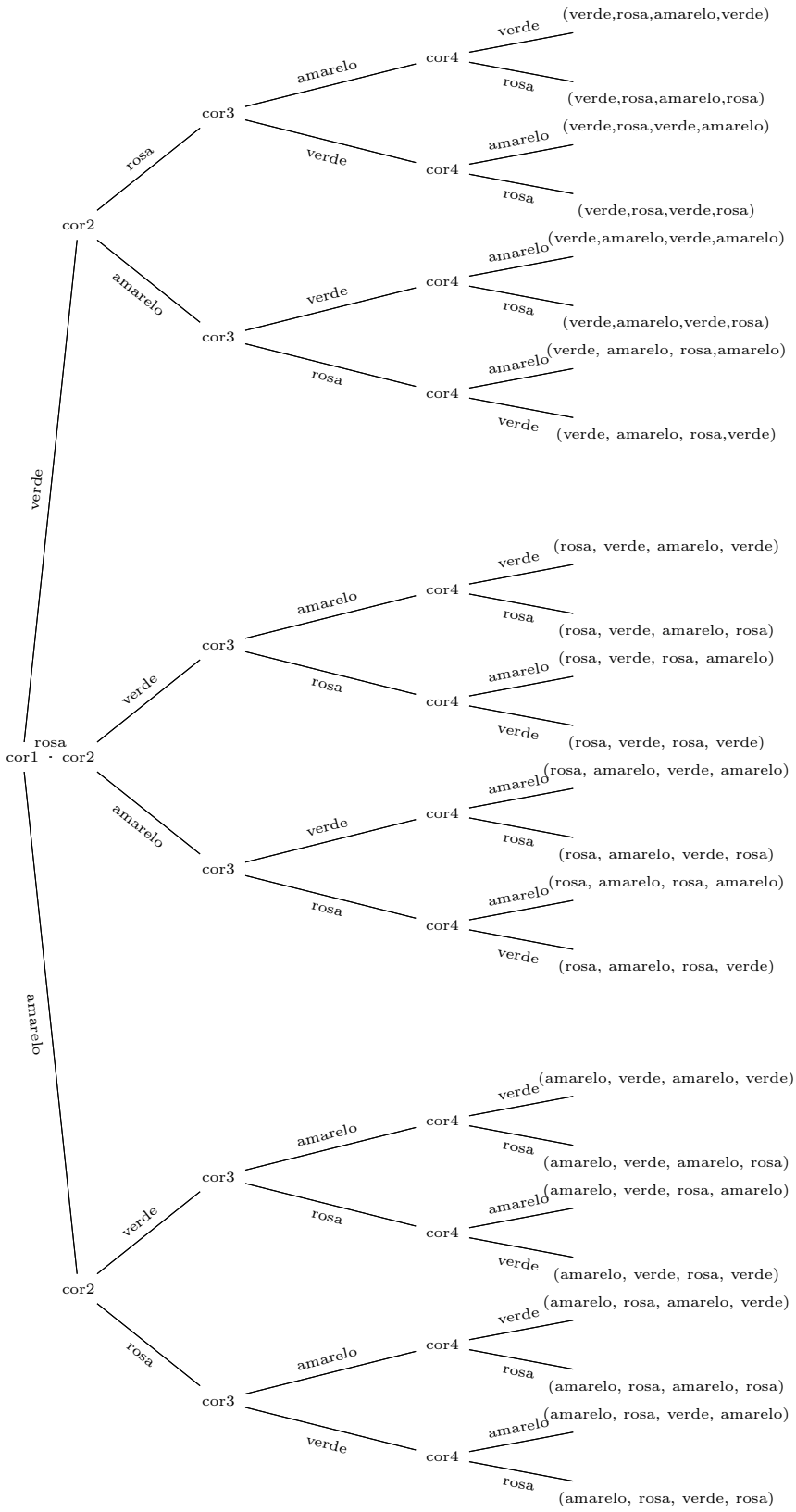

Figura 7: Árvore de Possibilidades da Atividade 4.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 24 configurações.

e) Agora, execute o código 2.4.

```
( amarelo , rosa , amarelo , rosa )
( amarelo , rosa , amarelo , verde )
( amarelo , rosa , verde , amarelo )
( amarelo , rosa , verde , rosa )
( amarelo , verde , amarelo , rosa )
( amarelo , verde , amarelo , verde )
( amarelo , verde , rosa , amarelo )
( amarelo , verde , rosa , verde )
( rosa , amarelo , rosa , amarelo )
( rosa , amarelo , rosa , verde )
( rosa , amarelo , verde , amarelo )
( rosa , amarelo , verde , rosa )
( rosa , verde , amarelo , rosa )
( rosa , verde , amarelo , verde )
( rosa , verde , rosa , amarelo )
( rosa , verde , rosa , verde )
( verde , amarelo , rosa , amarelo )
( verde , amarelo , rosa , verde )
( verde , amarelo , verde , amarelo )
( verde , amarelo , verde , rosa )
( verde , rosa , amarelo , rosa )
( verde , rosa , amarelo , verde )
( verde , rosa , verde , amarelo )
( verde , rosa , verde , rosa )
Quantidade de maneiras de colorir a bandeira:  24
```

Figura 8: Saída do Código 2.4.
Fonte: Autoria própria(2026).

Analisando o que o código faz:

I. **Linha 1:** *cores = ['amarelo','rosa','verde']* → Define os objetos básicos do problema.

II. **Linha 4:** *for cor1 in cores:* → Representa a 1ª Decisão. O computador "escolhe" uma cor para a primeira listra e "fixa" ela.

III. **Linha 5:** *for cor2 in cores:* → Representa a 2ª Decisão. Para cada cor que está "fixada" na 1ª decisão, o computador testa todas as outras cores possíveis.

IV. **Linha 6:** *if cor2 != cor1* → Esta linha é a materialização da restrição de que a cor da segunda listra não pode ser a mesma cor da primeira listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*amarelo*), mas a restrição do problema diz que ele não serve. Descarte-o."

V. **Linha 7:** *for cor3 in cores:* → Representa a 3ª Decisão. Para cada cor que está "fixada" na 2ª decisão, o computador testa todas as outras cores possíveis.

VI. **Linha 8:***if cor3 != cor2* → Esta linha é a materialização da restrição de que a cor da terceira listra não pode ser a mesma cor da segunda listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*verde*,*verde*), mas a restrição do problema diz que ele não serve. Descarte-o."

VII. **Linha 9:** *for cor4 in cores:* → Representa a 4ª Decisão. Para cada cor que está "fixada" na 3ª decisão, o computador testa todas as outras cores possíveis.

VIII. **Linha 10:***if cor4 != cor3* → Esta linha é a materialização da restrição de que a cor da quarta listra não pode ser a mesma cor da terceira listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*verde*,*amarelo*,*amarelo*), mas a restrição do problema diz que ele não serve. Descarte-o."

IX. **Linha 11:** *print(...)* → Materializa a configuração (a quádrupla ordenada).

X. **Linha 12:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

XI. **Resultado da execução do código:** O programa imprimirá linha por linha:

('amarelo','rosa','amarelo','rosa'),('amarelo','rosa','amarelo','verde')...
até ...('verde','rosa','verde','rosa').

E ao final: *Quantidade de maneiras de colorir a bandeira: 24*

Resumidamente, visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

1) $Cor1 = Amarelo$

2) $Cor2 = Amarelo \rightarrow if$ **bloqueia (iguais). Descarte.**

2) $Cor2 = Rosa \rightarrow if$ **permite**.

3) $Cor3 = Rosa \rightarrow if$ **bloqueia**.

3) $Cor3 = Verde \rightarrow if$ **permite**.

4) $Cor4 = Verde \rightarrow if$ **bloqueia**.

4) $Cor4 = Amarelo \rightarrow if$ **permite** $\rightarrow (amarelo, rosa, verde, amarelo)$.

4) $Cor4 = Rosa \rightarrow if$ **permite** $\rightarrow (amarelo, rosa, verde, rosa) \ldots$ e assim por diante até testar tudo.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.4.

A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 24 resultados.

g) Observe o Código 2.4, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo das linhas de código *if cor2 != cor1*, *if cor3 != cor2* e *if cor4 != cor3*?

Foram utilizados 4 laços for. E cada um desses laços representa uma decisão.
O laço externo *for cor1 in cores:* representa a 1ª Decisão.
O 1º laço interno *for cor2 in cores:* representa a 2ª Decisão.
O 2º laço interno *for cor3 in cores:* representa a 3ª Decisão.
O 3º laço interno *for cor4 in cores:* representa a 3ª Decisão.
Nesses quatro comandos for aninhados, o algoritmo se propõe a visitar todas as configurações, independentemente de serem válidas ou não, ou seja, o código está percorrendo um espaço de $3 \times 3 \times 3 \times 3 = 81$ possibilidades totais.

Mas, as linhas de comando *if* são a materialização da restrição de que a repetição de cores em listras adjacentes é proibida.
Por exemplo, enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 2ª Decisão(2 maneiras), no código Python a restrição está escrita e visível na linha *if cor2 != cor1:*.

3.2 Segundo encontro

***Atividade* 5:** Considere o seguinte problema: "Quantos números de dois algarismos podem ser formados no sistema decimal?".

Esta atividade é particularmente interessante porque mostra que nem sempre a restrição do problema aparece de forma explícita.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são os algarismos do sistema numérico decimal: $\{0, 1,2,3,4,5,6,7,8,9\}$.
- Configurações: A configuração é um número com dois algarismos. Matematicamente, é um par ordenado $(Dezena, Unidade)$ que forma um número válido com dois algarismos.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Como queremos formar números com dois algarismos, temos que tomar duas decisões para resolver esse problema. Como visto acima, precisamos formar o par ordenado $(Dezena, Unidade)$.

Temos uma restrição nesse problema. O par $(0, 7)$, por exemplo, representa o número 7, que é um número formado por apenas um algarismo. Portanto, o par ordenado $(0, a)$ não é uma configuração válida neste problema, isto é, o algarismo das dezenas não pode ser igual a zero.

Como no problema anterior, usaremos a estratégia de *Não adiar dificuldades*. Então, teremos:

1ª decisão: Escolher o algarismo das dezenas(como o algarismo das dezenas não pode ser igual a zero, essa restrição nos força a escolher o algarismo das dezenas primeiro);

2ª decisão: Escolher o algarismo das unidades.

Portanto,

1ª decisão(algarismo das dezenas): 9 maneiras $\rightarrow \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$;

2ª decisão(algarismo das unidades): 10 maneiras $\rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Quando o professor perguntar "Por que 9?", a Justificação do aluno será: "Porque se o algarismo das dezenas for igual a zero, o número só tem um algarismo.". Essa fala explicita a restrição do problema. E, se a pergunta for "Por que 10 na 2ª decisão?", a Justificação do aluno será: "Porque o problema não diz que os algarismos que formam o número devem ser distintos."

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

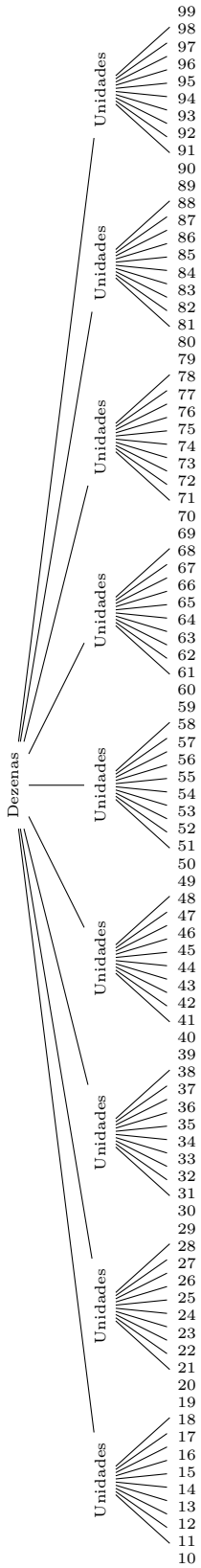

Figura 9: Árvore de Possibilidades da Atividade 5.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

> O conjunto de resultados desse problema já começa a dar trabalho para ser obtido manualmente. A maioria dos alunos, já percebendo uma certa regularidade ao resolver os problemas anteriores, não montam a árvore de possibilidades toda, simplesmente, chegam a conclusão que há $9+9+9+9+9+9+9+9+9+9 = 9 \times 10 = 90$ configurações, pois para cada algarismo utilizado nas dezenas há 10 possibilidades de escolhas para as unidades.

e) Agora, execute o código 2.5.

```
Números formados:
1 0
1 1
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
2 0
2 1
2 2
2 3
2 4
2 5
2 6
2 7
2 8
2 9
3 0
3 1
3 2
3 3
3 4
3 5
3 6
3 7
3 8
3 9
4 0
4 1
4 2
4 3
4 4
4 5
4 6
4 7
4 8
4 9
5 0
5 1
5 2
5 3
5 4
5 5
5 6
5 7
5 8
5 9
6 0
6 1
6 2
6 3
6 4
6 5
6 6
6 7
6 8
6 9
7 0
7 1
7 2
7 3
7 4
7 5
7 6
7 7
7 8
7 9
8 0
8 1
8 2
8 3
8 4
8 5
8 6
8 7
8 8
8 9
9 0
9 1
9 2
9 3
9 4
9 5
9 6
9 7
9 8
9 9
Quantidade de números com dois algarismos:  90
>>>
```

Figura 10: Saída do Código 2.5.
Fonte: Autoria própria(2026).

I. **Linha 1:** *algarismos = [0,1,2,3,4,5,6,7,8,9]* → Define os objetos básicos do problema.

II. **Linha 6:** *for dezenas in algarismos:* → Representa a 1ª Decisão. O computador "escolhe" um algarismo para as dezenas e "fixa" ele.

III. **Linha 7:** *if dezenas != 0* → Esta linha é a materialização da restrição de que o algarismo das dezenas não pode ser igual a zero. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par $(0,)$, mas a restrição do problema diz que ele não serve. Descarte-o."

IV. **Linha 8:** *for unidades in algarismos:* → Representa a 2ª Decisão. Para cada algarismo que está "fixado" na 1ª decisão, o computador testa todos os outros algarismos possíveis para as unidades.

V. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VI. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VII. **Resultado da execução do código:** O programa imprimirá linha por linha:10,11,12,...,98,99.

E ao final: *Quantidade de números com dois algarismos: 90*

Resumidamente, visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

1) $dezenas = 0 \rightarrow if$ **bloqueia. Descarte.**

1) $dezenas = 1 \rightarrow if$ **Permite.**

2) $unidades = 0 \rightarrow if$ **Permite** $\rightarrow 10$

2) $unidades = 1 \rightarrow if$ **Permite** $\rightarrow 11 \ldots$ e assim por diante até testar tudo.

Ou seja, se o algarismo das dezenas for igual a 1, o algarismo das unidades pode ser 0,1,2,3,4,5,6,7,8 ou 9, o que lhe fornece 10 números diferentes:

10,11,12,13,14,15,16,17,18,19

Se o algarismo das dezenas for 2, há novamente 10 maneiras diferentes de escolher o algarismo das unidades:

20,21,22,23,24,25,26,27,28,29

Analogamente, se o algarismo das dezenas for 3:

30,31,32,33,34,35,36,37,38,39

E, o mesmo ocorre se o algarismo das dezenas for 4, 5, 6, 7, 8, ou 9.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.5.

A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 90 resultados.

g) Observe o Código 2.5, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if dezenas!= 0*?

Foram utilizados 2 laços for. E cada um desses laços representa uma decisão. O laço externo *for dezenas in algarismos:* representa a 1ª Decisão. O laço interno *for unidades in algarismos:* representa a 2ª Decisão. Nesses dois comandos for aninhados, o algoritmo se propõe a visitar todas as configurações, independentemente de serem válidas ou não, ou seja, o código está percorrendo um espaço de $10 \times 10 = 100$ possibilidades totais. Mas, a linha de comando *if dezenas != 0* é a materialização da restrição de que o algarismos das dezenas não pode ser igual a zero. Por exemplo, enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 1ª Decisão(9 maneiras), no código Python a restrição está escrita e visível na linha *if dezenas != 0:*.

h) O que é mais trabalhoso, responder ao item c) ou construir o Código 2.5?

> O aluno começa a notar, na resolução desse problema, que desenhar 90 ramos é uma tarefa cansativa e propensa ao erro humano.
> Com isso, mesmo exigindo um certo nível de abstração, a escrita do código é operacionalmente mais eficiente. Além disso, faz com o aluno construa a lógica (os laços for e o if) que gera a prova que são 90 números com dois algarismos.

***Atividade* 6:** Considere o seguinte problema: "Quantos números pares de dois algarismos distintos podem ser formados no sistema decimal?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

> - Objetos básicos: são os algarismos do sistema numérico decimal: $\{0, 1,2,3,4,5,6,7,8,9\}$.
> - Configurações: A configuração é um número par com dois algarismos distintos. Matematicamente, é um par ordenado $(Dezena, Unidade)$ de modo que a Unidade é igual a 0,2,4,6 ou 8 e $Dezena \neq Unidade$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual?

> Temos três restrições nesse problema.
>
> - **Primeira restrição**: O par ordenado $(0, a)$ não é uma configuração válida neste problema, isto é, o algarismo das dezenas não pode ser igual a zero.
> - **Segunda restrição**: O algarismo das unidades deve ser um múltiplo de 2.
> - **Terceira restrição**: Os algarismos são distintos.
>
> Precisamos tomar 2 decisões:
> 1ª decisão: Escolher o algarismo das unidades;
> 2ª decisão: Escolher o algarismo das dezenas.

c) Execute o código 2.6

Analisando o código:

I. **Linha 1:** *algarismos = [0,1,2,3,4,5,6,7,8,9]* → Define os objetos básicos do problema.

II. **Linha 5:** *for unidades in algarismos:* → Representa a 1ª Decisão. O computador "escolhe" um algarismo para as unidades e "fixa" ele.

III. **Linha 6:** *if unidades % 2 == 0:* → Esta linha é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (?,1), mas a restrição do problema diz que ele não serve. Descarte-o."

IV. **Linha 7:** *for dezenas in algarismos:* → Representa a 2ª Decisão. Para cada algarismo que está "fixado" na 1ª decisão, o computador testa todos os outros algarismos possíveis para as dezenas.

V. **Linha 8:** *if dezenas != 0 and dezenas != unidades:* → Esta linha é a materialização de duas restrições, de que o algarismo das dezenas não pode ser igual a zero e de que o algarismo da dezenas não pode ser igual ao algarismo escolhido para as unidades. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (0,2), mas a restrição do problema diz que ele não serve. Descarte-o."E, "Eu estou prestes a gerar o par (4,4), mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VII. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VIII. **Resultado da execução do código:** O programa imprimirá linha por linha:10,20,30, . . . ,78,98.

E ao final: *Quantidade de números pares com dois algarismos: 41*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $unidades = 0 \rightarrow if$ **Permite.**

2) $dezenas = 0 \rightarrow if$ **Bloqueia. Descarte.**

2) $dezenas = 1 \rightarrow if$ **Permite.** $\rightarrow 10$

2) $dezenas = 2 \rightarrow if$ **Permite.** $\rightarrow 20$

2) $dezenas = 3 \rightarrow if$ **Permite.** $\rightarrow 30 \ldots$ e assim por diante até testar tudo.

Ou seja, se o algarismo das unidades for igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que lhe fornece 9 números diferentes:

10,20,30,40,50,60,70,80,90

Se o algarismo das unidades for 2, há 8 maneiras diferentes de escolher o algarismo das dezenas:

12,32,42,52,62,72,82,92

Analogamente, se o algarismo das unidades for 4:

14,24,34,54,64,74,84,94

E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.
Com isso, temos o seguinte resultado ao executar o Código 2.6.

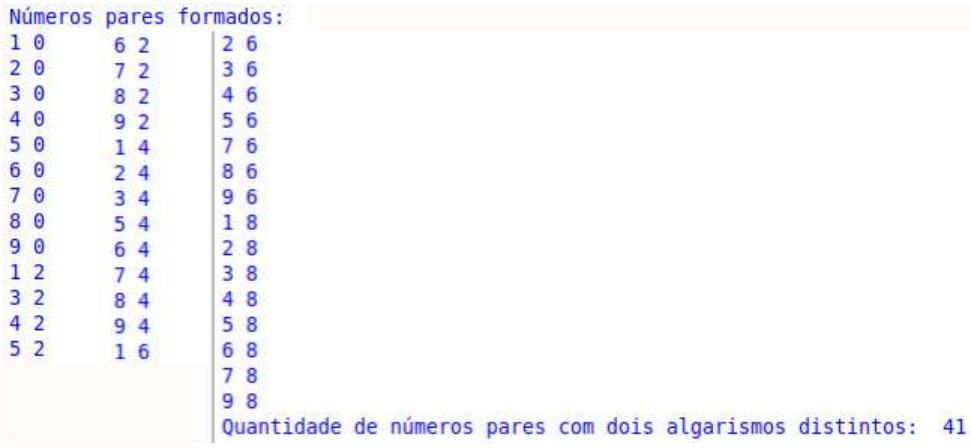

Figura 11: Saída do Código 2.6.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações satisfazem as condições do problema.
> Obtemos 41 configurações.

d) Observe o Código 2.6, qual é o objetivo das linhas de código *if unidades % 2 = 0* e *if dezenas!= 0 and dezenas != unidades*?

> A linha *if unidades % 2 == 0* é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois.
> A linha *if dezenas != 0 and dezenas != unidades* é a materialização de duas restrições: o algarismo das dezenas não pode ser igual a zero e o algarismo da dezenas não pode ser igual ao algarismo escolhido para as unidades.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

Vimos que, observando a execução do Código 2.6, temos que:

- quando o algarismo das unidades é igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que fornece 9 números pares diferentes:

  10,20,30,40,50,60,70,80,90;

- quando o algarismo das unidades for 2, o algarismo das dezenas 1,3,4,5,6,7,8 ou 9, o que fornece 8 números pares diferentes:

  12,32,42,52,62,72,82,92

  Analogamente, quando o algarismo das unidades for 4, teremos:

  14,24,34,54,64,74,84,94

  E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.

Logo, não é possível obter a resposta para esse problema usando, apenas, o Princípio Fundamental da Contagem.
A execução do Código 2.6 mostra-nos que existem $9 + 4 \cdot 8 = 9 + 32 = 41$ números pares com 2 algarismos distintos. Tal resultado nos mostra que há "dois tipos" de números pares formados pelo código, aqueles em que o zero ocupa as unidades e aqueles que o zero não ocupa o algarismo das unidades.
Então, para aplicar o Princípio Fundamental da Contagem na resolução do problema, precisamos resolver dois problemas, que são:

- Quantos números pares com dois algarismos distintos terminam em zero?

  Para esse problema, temos 1 única maneira de tomar a decisão d_1, que é escolher o zero, uma vez tomada a decisão d_1, temos 9 maneiras de tomar a decisão d_2. Logo, pelo Princípio Fundamental da Contagem, há $1 \cdot 9 = 9$ números desse tipo.

- Quantos números pares com dois algarismos distintos não terminam em zero?

  Para esse problema, temos 4 maneiras de tomar a decisão d_1, que é escolher um dos algarismos 2, 4, 6 ou 8, uma vez tomada a decisão d_1, temos 8 maneiras de tomar a decisão d_2, pois o zero não pode ocorrer nas dezenas e não podemos usar o algarismo usado nas unidades. Logo, pelo Princípio Fundamental da Contagem, há $4 \cdot 8 = 32$ desse tipo.

Portanto, há $9 + 32 = 41$ números pares com dois algarismos distintos.

Você deve ter notado, observando as configurações obtidas no item c) do problema da Atividade 6, que há a necessidade da utilização de um outro Princípio de Contagem, junto com o Princípio Fundamental da Contagem, para encontrar a quantidade de configurações. Tal Princípio é conhecido como Princípio Aditivo e pode ser enunciado da seguinte forma, conforme podemos encontrar em (CERIOLI; VIANA, 2012):

Sejam A um conjunto finito e $A_1, A_2, \cdots, A_n$ subconjuntos não vazios de A.

1. Dizemos que $A_1, A_2, \cdots, A_n$ *exaurem* A se, para cada elemento a de A, existe i, com $1 \leq i \leq n$, tal que $a \in A_i$; ou seja, se $A_1 \cup A_2 \cup \cdots \cup A_n = A$.

2. Dizemos que $A_1, A_2, \cdots, A_n$ são *disjuntos dois a dois* se, quando examinados aos pares, eles não possuem elementos em comum; ou seja, $A_i \cap A_j = \emptyset$, para todos i, j com $1 \leq i \neq j \leq n$.

3. Dizemos que $A_1, A_2, \cdots, A_n$ são uma partição de A se $A_1, A_2, \cdots, A_n$ exaurem A e são disjuntos dois a dois.

Princípio Aditivo. *Seja A um conjunto finito.*

Se $A_1, A_2, \cdots, A_n$ são uma partição de A, então $n(A) = n(A_1) + n(A_2) + \cdots + n(A_n)$.

No problema da Atividade 6, o conjunto A é formado pelos números pares com algarismos distintos, o conjunto A_1 é formados pelos números pares com algarismos distintos com o algarismo das unidades igual a zero e o conjunto A_2 é formados pelos números pares com algarismos distintos com o algarismo das unidades diferente de zero. Então, $n(A) = n(A_1) + n(A_2) = 9 + 32 = 41$.

Agora, podemos voltar a Enunciação feita por um aluno ao resolver o problema da Atividade 4(colorir a bandeira).

O aluno enunciou que:

"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;

3ª decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;

4ª decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."

Mas, também, poderíamos ter a seguinte enunciação:

"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

3ª decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;

4ª decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."

A partir das duas enunciações, observemos que a resposta da 3ª decisão depende da cor escolhida para colorir a 3ª listra, a realização da 2ª decisão. Com isso, temos dois casos a resolver:

1º caso: as cores das 1ª e 3ª listras são iguais;

1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

$2^{\underline{a}}$ decisão(cor da terceira listra): 1 maneira $\rightarrow \{rosa\}$, vamos supor que o aluno escolheu *rosa*;
$3^{\underline{a}}$ decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas $1^{\underline{a}}$ e $3^{\underline{a}}$ listras, vamos supor que o aluno escolheu *amarelo*;
$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra.
Pelo Princípio Fundamental da Contagem, conseguimos colorir de $3 \times 1 \times 2 \times 2 = 12$ maneiras diferentes de colorir a bandeira.
$2^{\underline{o}}$ caso: as cores das $1^{\underline{a}}$ e $3^{\underline{a}}$ listras são diferentes e as cores das $2^{\underline{a}}$ e $4^{\underline{a}}$ listras são iguais.
$1^{\underline{a}}$ decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
$2^{\underline{a}}$ decisão(cor da terceira listra): 2 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;
$3^{\underline{a}}$ decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas $1^{\underline{a}}$ e $3^{\underline{a}}$ listras;
$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa,verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra.
Pelo Princípio Fundamental da Contagem, conseguimos colorir de $3 \times 2 \times 1 \times 2 = 12$ maneiras diferentes de colorir a bandeira.
Portanto, pelo Princípio Aditivo, há $12 + 12 = 24$ maneiras de colorir a bandeira, conforme determinamos no problema da Atividade 4.

***Atividade* 7:** Considere o seguinte problema: "De quantos modos 3 pessoas podem sentar-se em 5 cadeiras em fila?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são as pessoas: $\{p1,p2,p3\}$ e as cadeiras: $\{c1,c2,c3,c4,c5\}$.
- Configurações: Cada configuração é formada por 3 pessoas sentadas em 3 cadeiras e 2 cadeiras vazias, ou seja, cada configuração é formada por uma tripla ordenada onde cada coordenada é uma cadeira ocupada pelas pessoas $p1$, $p2$ e $p3$, nessa ordem.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma cadeira para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a cadeira que a pessoa $p1$ vai sentar; 2ª) Escolher a cadeira que a pessoa $p2$ vai sentar; 3ª) Escolher a cadeira que a pessoa $p3$ vai sentar.
Temos, então:
1ª decisão: 5 maneiras de escolher uma cadeira para a pessoa $p1$;
2ª decisão: 4 maneiras de escolher uma cadeira para a pessoa $p2$;
3ª decisão: 3 maneiras de escolher uma cadeira para a pessoa $p3$.

c) Execute o código 2.7

Analisando o código:

I. **Linha 1:** *cadeiras* = [$c1$,$c2$,$c3$,$c4$,$c5$] → Define o objeto básico cadeiras.

II. **Linha 2:** *cadeiras_ocupadas* = [] → Define uma lista onde cada elemento será as cadeiras ocupadas pelas pessoas $p1$, $p2$ e $p3$.

III. **Linha 4:** *for cadeira_p1 in cadeiras:* → Representa a 1ª Decisão. O computador "escolhe" uma cadeira para a pessoa $p1$ e "fixa" ela.

IV. **Linha 5:** *for cadeira_p2 in cadeiras:* → Representa a 2ª Decisão. O computador "escolhe" uma cadeira para a pessoa $p2$ e "fixa" ela.

V. **Linha 6:** *if cadeira_p2 != cadeira_p1 :* → Esta linha é a materialização da restrição de que cada pessoa senta em uma única cadeira. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar a tripla ($c1$,$c1$,?), mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 7:** *for cadeira_p3 in cadeiras:* → Representa a 3ª Decisão. Para cada cadeira que está "fixada" na 1ª e 2ª decisões, o computador testa todas as outras cadeiras possíveis para a pessoa $p3$.

VII. **Linha 8:** *if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:* → Esta linha é a materialização da restrição de que que cada pessoa senta em uma única cadeira. Ela diz: "Eu estou prestes a gerar a tripla ($c1$,$c2$,$c1$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."E, "Eu estou prestes a gerar a tripla ($c1$,$c2$,$c2$), mas a restrição do problema diz que ele não serve. Descarte-o."

VIII. **Linha 9:** $cadeira\_ocupada = cadeira\_p1$,$cadeira\_p2$,$cadeira\_p3$ → Materializa a configuração (a tripla ordenada).

IX. **Linha 10:** *cadeiras_ocupadas.append(cadeira_ocupada)* → Realiza a inserção da tripla formada na lista *cadeiras_ocupadas*.

X. **Resultado da execução do código:** O programa imprimirá as triplas formadas.

E ao final: *Quantidade de maneiras de organizar as 3 pessoas: 60*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $cadeira\_p1 = c1$.

2) $cadeira\_p2 = c1 \rightarrow if$ **Bloqueia. Descarte.**

2) $cadeira\_p2 = c2 \rightarrow if$ **Permite.**

3) $cadeira\_p3 = c1 \rightarrow if$ **Bloqueia. Descarte.**

3) $cadeira\_p3 = c2 \rightarrow if$ **Bloqueia. Descarte.**

3) $cadeira\_p3 = c3 \rightarrow if$ **Permite.** → ($c1$,$c2$,$c3$) . . . e assim por diante até testar tudo.

Obtemos, então, a seguinte saída:

```
Cadeiras ocupadas pelas 3 pessoas:
('c1', 'c2', 'c3'),('c1', 'c2', 'c4'),('c1', 'c2', 'c5'),('c1', 'c3', 'c2'),('c1', 'c3', 'c4'),
('c1', 'c3', 'c5'),('c1', 'c4', 'c2'),('c1', 'c4', 'c3'),('c1', 'c4', 'c5'),('c1', 'c5', 'c2'),
('c1', 'c5', 'c3'),('c1', 'c5', 'c4'),('c2', 'c1', 'c3'),('c2', 'c1', 'c4'),('c2', 'c1', 'c5'),
('c2', 'c3', 'c1'),('c2', 'c3', 'c4'),('c2', 'c3', 'c5'),('c2', 'c4', 'c1'),('c2', 'c4', 'c3'),
('c2', 'c4', 'c5'),('c2', 'c5', 'c1'),('c2', 'c5', 'c3'),('c2', 'c5', 'c4'),('c3', 'c1', 'c2'),
('c3', 'c1', 'c4'),('c3', 'c1', 'c5'),('c3', 'c2', 'c1'),('c3', 'c2', 'c4'),('c3', 'c2', 'c5'),
('c3', 'c4', 'c1'),('c3', 'c4', 'c2'),('c3', 'c4', 'c5'),('c3', 'c5', 'c1'),('c3', 'c5', 'c2'),
('c3', 'c5', 'c4'),('c4', 'c1', 'c2'),('c4', 'c1', 'c3'),('c4', 'c1', 'c5'),('c4', 'c2', 'c1'),
('c4', 'c2', 'c3'),('c4', 'c2', 'c5'),('c4', 'c3', 'c1'),('c4', 'c3', 'c2'),('c4', 'c3', 'c5'),
('c4', 'c5', 'c1'),('c4', 'c5', 'c2'),('c4', 'c5', 'c3'),('c5', 'c1', 'c2'),('c5', 'c1', 'c3'),
('c5', 'c1', 'c4'),('c5', 'c2', 'c1'),('c5', 'c2', 'c3'),('c5', 'c2', 'c4'),('c5', 'c3', 'c1'),
('c5', 'c3', 'c2'),('c5', 'c3', 'c4'),('c5', 'c4', 'c1'),('c5', 'c4', 'c2'),('c5', 'c4', 'c3'),
Quantidade de maneiras de organizar as 3 pessoas:  60
>>>
```

Figura 12: Saída do Código 2.7.
Fonte: Autoria própria(2026).

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações satisfazem as condições do problema.
> Obtemos 60 configurações.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

> Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma cadeira para a pessoa $p1$ e, feito isso, há quatro maneiras de escolher uma cadeira para a pessoa $p2$ e, feito isso, há três maneiras de escolher uma cadeira para a pessoa $p3$, temos que, podemos organizar as três pessoas de $5 \times 4 \times 3 = 60$ maneiras diferentes. O Código 2.7 é a materialização algorítmica do Princípio Fundamental da Contagem.

***Atividade* 8:** Considere o seguinte problema: "De quantos modos 6 pessoas podem sentar-se em 10 cadeiras em fila?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

> - Objetos básicos: são as cadeiras: $\{c1,c2,c3,c4,c5,c6,c7,c8,c9,c10\}$ e as pessoas: $\{p1,p2,p3,p4,p5,p6\}$.
> - Configurações: Cada configuração é formada por 6 pessoas sentadas em 6 cadeiras e 4 cadeiras vazias, ou seja, cada configuração é formada por uma 6-upla ordenada onde cada coordenada é uma cadeira ocupada pelas pessoas $p1$, $p2$, $p3$, $p4$, $p5$ e $p6$, nessa ordem.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma cadeira para cada pessoa. Precisamos tomar 6 decisões: 1ª) Escolher a cadeira que a pessoa $p1$ vai sentar; 2ª) Escolher a cadeira que a pessoa $p2$ vai sentar; 3ª) Escolher a cadeira que a pessoa $p3$ vai sentar; 4ª) Escolher a cadeira que a pessoa $p4$ vai sentar; 5ª) Escolher a cadeira que a pessoa $p5$ vai sentar e 6ª) Escolher a cadeira que a pessoa $p6$ vai sentar.

Temos, então:

1ª decisão: 10 maneiras de escolher uma cadeira para a pessoa $p1$;

2ª decisão: 9 maneiras de escolher uma cadeira para a pessoa $p2$;

3ª decisão: 8 maneiras de escolher uma cadeira para a pessoa $p3$;

4ª decisão: 7 maneiras de escolher uma cadeira para a pessoa $p4$;

5ª decisão: 6 maneiras de escolher uma cadeira para a pessoa $p5$;

6ª decisão: 5 maneiras de escolher uma cadeira para a pessoa $p6$.

c) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido. Se estiver e inseguro, em relação à sua solução, construa um código em Python para resolver o problema.

Pelo Princípio Fundamental da Contagem, como há 10 maneiras de escolher uma cadeira para a pessoa $p1$ e, feito isso, há 9 maneiras de escolher uma cadeira para a pessoa $p2$ e, feito isso, há 8 maneiras de escolher uma cadeira para a pessoa $p3$ e, feito isso, há 7 maneiras de escolher uma cadeira para a pessoa $p4$ e, feito isso, há 6 maneiras de escolher uma cadeira para a pessoa $p5$ e, feito isso, há 5 maneiras de escolher uma cadeira para a pessoa $p6$, temos que, podemos organizar as seis pessoas de $10 \times 9 \times 8 \times 7 \times 6 \times 5 = 151200$ maneiras diferentes.

O código abaixo fornece as configurações e a quantidade de configurações.

```
1 cadeiras = ['c1','c2','c3','c4','c5','c6','c7','c8','c9','c10']
2 cadeiras_ocupadas = []
3
4 for cadeira_p1 in cadeiras:
5     for cadeira_p2 in cadeiras:
6             if cadeira_p2 != cadeira_p1:
7                 for cadeira_p3 in cadeiras:
```

```
8                       if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
9                           for cadeira_p4 in cadeiras:
10                              if cadeira_p4 != cadeira_p3 and cadeira_p4 !=
    cadeira_p2 and cadeira_p4 != cadeira_p1:
11                                  for cadeira_p5 in cadeiras:
12                                      if cadeira_p5 != cadeira_p4 and cadeira_p5 !=
    cadeira_p3 and cadeira_p5 != cadeira_p2 and cadeira_p5 != cadeira_p1:
13                                          for cadeira_p6 in cadeiras:
14                                              if cadeira_p6 != cadeira_p5 and
    cadeira_p6 != cadeira_p4 and cadeira_p6 != cadeira_p3 and cadeira_p6 !=
    cadeira_p2 and cadeira_p6 != cadeira_p1:
15                                                  cadeira_ocupada = cadeira_p1,
    cadeira_p2,cadeira_p3,cadeira_p4,cadeira_p5,cadeira_p6
16                                                  cadeiras_ocupadas.append(
    cadeira_ocupada)
17
18 colunas = 4
19
20 print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
21 for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
22     print(f"{cadeira_ocupada}", end=",")
23     if (i + 1) % colunas == 0:
24         print()
25
26 if len(cadeiras_ocupadas) % colunas != 0:
27     print()
28
29 print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código 3.1: Código da Atividade 8

3.3 Terceiro encontro

***Atividade* 9:** Considere o código 2.8 que resolve um determinado problema de contagem e execute-o:

a) Pesquise o que são anagramas.

> As várias ordenações de letras de uma determinada palavra são chamadas de **Anagramas**. Seriam como palavas, não necessariamente com sentido, que podem ser formadas a partir de um dado conjunto de letras.(SPREAFICO; SILVA, 2021)

b) Quais são os objetos básicos do problema?

> Observando a linha 1 do código, vemos que são as letras: $\{N,U,M,E,R,O\}$.

c) Quais são as configurações formadas?

Pelas linhas 2 e 22, as configurações são os anagramas da palavra NÚMERO.

d) Quantas decisões foram tomadas para resolver o problema? Quais foram essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão foi tomada?

Pelo o Código 2.8, observemos que há seis linhas de comando *for*. Logo, seis decisões foram tomadas.
Então, dadas as letras N,U,M,E,R,O, temos as seguintes decisões: 1ª) Escolher a 1ª letra do anagrama(*for letra_1 in letras*); 2ª) Escolher a 2ª letra do anagrama(*for letra_2 in letras*); 3ª) Escolher a 3ª letra do anagrama(*for letra_3 in letras*); 4ª) Escolher a 4ª letra do anagrama(*for letra_4 in letras*); 5ª) Escolher a 5ª letra do anagrama(*for letra_5 in letras*) e 6ª) Escolher a 6ª letra do anagrama(*for letra_6 in letras*).
Temos uma restrição no problema, cada letra só pode ser usada uma única vez. Basta ver as linhas de código *if* presentes no Código 2.8.
Temos, então:
1ª decisão: 6 maneiras, pois antes de iniciarmos a formação do anagrama, temos 6 letras disponíveis em *letras*;
2ª decisão: 5 maneiras, pois após a 1ª decisão ter sido realizada, para qualquer letra escolhida na 1ª decisão , temos 5 letras disponíveis(*if letra_2 != letra_1:*);
3ª decisão: 4 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª e 2ª decisões, temos 4 letras disponíveis(*if letra_3 != letra_2 and letra_3 != letra_1:*);
4ª decisão: 3 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª e 3ª decisões, temos 3 letras disponíveis(*if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 != letra_1:*);
5ª decisão: 2 maneiras, pois após a 1ª, 2ª, 3ª e 4ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª e 4ª decisões, temos 2 letras disponíveis(*if letra_5 != letra_4 and letra_5 != letra_3 and letra_5 != letra_2 and letra_5 != letra_1:*);

> 6ª decisão: 1 maneira, pois após a 1ª, 2ª, 3ª, 4ª e 5ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, temos 1 letra disponível(*if letra_6 != letra_5 and letra_6 != letra_4 and letra_6 != letra_3 and letra_6 != letra_2 and letra_6 != letra_1:*).

e) Quantas configurações foram obtidas? Com isso, em que consiste o problema resolvido pelo Código 2.8?

> Após a execução do Código 2.8, obtemos os anagramas da palavra NÚMERO e a quantidade dos anagramas.

```
Anagramas da palavra NÚMERO:
NUMERO,NUMEOR,NUMREO,NUMROE,NUMOER,NUMORE,NUEMRO,NUEMOR,NUERMO,NUEROM,NUEOMR,NUEORM,NURMEO,NURMOE,NUREMO,NUREOM,NUROME,NUROEM,NUOMER,NUOMRE,
NUOEMR,NUOERM,NUORME,NUOREM,NMUERO,NMUEOR,NMUREO,NMUROE,NMUOER,NMUORE,NMEURO,NMEUOR,NMERUO,NMEROU,NMEOUR,NMEORU,NMRUEO,NMRUOE,NMREUO,NMREOU,
NMROUE,NMROEU,NMOUER,NMOURE,NMOEUR,NMOERU,NMORUE,NMOREU,NEUMRO,NEUMOR,NEURMO,NEUROM,NEUOMR,NEUORM,NEMURO,NEMUOR,NEMRUO,NEMROU,NEMOUR,NEMORU,
NERUMO,NERUOM,NERMUO,NERMOU,NEROUM,NEROMU,NEOUMR,NEOURM,NEOMUR,NEOMRU,NEORUM,NEORMU,NRUMEO,NRUMOE,NRUEMO,NRUEOM,NRUOME,NRUOEM,NRMUEO,NRMUOE,
NRMEUO,NRMEOU,NRMOUE,NRMOEU,NREUMO,NREUOM,NREMUO,NREMOU,NREOUM,NREOMU,NROUME,NROUEM,NROMUE,NROMEU,NROEUM,NROEMU,NOUMER,NOUMRE,NOUEMR,NOUERM,
NOURME,NOUREM,NOMUER,NOMURE,NOMEUR,NOMERU,NOMRUE,NOMREU,NOEUMR,NOEURM,NOEMUR,NOEMRU,NOERUM,NOERMU,NORUME,NORUEM,NORMUE,NORMEU,NOREUM,NOREMU,
UNMERO,UNMEOR,UNMREO,UNMROE,UNMOER,UNMORE,UNEMRO,UNEMOR,UNERMO,UNEROM,UNEOMR,UNEORM,UNRMEO,UNRMOE,UNREMO,UNREOM,UNROME,UNROEM,UNOMER,UNOMRE,
UNOEMR,UNOERM,UNORME,UNOREM,UMNERO,UMNEOR,UMNREO,UMNROE,UMNOER,UMNORE,UMENRO,UMENOR,UMERNO,UMERON,UMEONR,UMEORN,UMRNEO,UMRNOE,UMRENO,UMREON,
UMRONE,UMROEN,UMONER,UMONRE,UMOENR,UMOERN,UMORNE,UMOREN,UENMRO,UENMOR,UENRMO,UENROM,UENOMR,UENORM,UEMNRO,UEMNOR,UEMRNO,UEMRON,UEMONR,UEMORN,
UERNMO,UERNOM,UERMNO,UERMON,UERONM,UEROMN,UEONMR,UEONRM,UEOMNR,UEOMRN,UEORNM,UEORMN,URNMEO,URNMOE,URNEMO,URNEOM,URNOME,URNOEM,URMNEO,URMNOE,
URMENO,URMEON,URMONE,URMOEN,URENMO,URENOM,UREMNO,UREMON,UREONM,UREOMN,URONME,URONEM,UROMEN,UROMNE,UROEMN,UROENM,UONMER,UONMRE,UONEMR,UONERM,
UONRME,UONREM,UOMNER,UOMNRE,UOMENR,UOMERN,UOMRNE,UOMREN,UOENMR,UOENRM,UOEMNR,UOEMRN,UOERNM,UOERMN,UORNME,UORNEM,UORMNE,UORMEN,UORENM,UOREMN,
MNUERO,MNUEOR,MNUREO,MNUROE,MNUOER,MNUORE,MNEURO,MNEUOR,MNERUO,MNEROU,MNEOUR,MNEORU,MNRUEO,MNRUOE,MNREUO,MNREOU,MNROUE,MNROEU,MNOUER,MNOURE,
MNOEUR,MNOERU,MNORUE,MNOREU,MUNERO,MUNEOR,MUNREO,MUNROE,MUNOER,MUNORE,MUENRO,MUENOR,MUERNO,MUERON,MUEONR,MUEORN,MURNEO,MURNOE,MURENO,MUREON,
MURONE,MUROEN,MUONER,MUONRE,MUOENR,MUOERN,MUORNE,MUOREN,MENURO,MENUOR,MENRUO,MENROU,MENOUR,MENORU,MEUNRO,MEUNOR,MEURNO,MEURON,MEUONR,MEUORN,
MERNUO,MERNOU,MERUNO,MERUON,MERONU,MEROUN,MEONUR,MEONRU,MEOUNR,MEOURN,MEORNU,MEORUN,MRNUEO,MRNUOE,MRNEUO,MRNEOU,MRNOUE,MRNOEU,MRUNEO,MRUNOE,
MRUENO,MRUEON,MRUONE,MRUOEN,MRENUO,MRENOU,MREUNO,MREUON,MREONU,MREOUN,MRONUE,MRONEU,MROUNE,MROUEN,MROENU,MROEUN,MONUER,MONURE,MONEUR,MONERU,
MONRUE,MONREU,MOUNER,MOUNRE,MOUENR,MOUERN,MOURNE,MOUREN,MOENUR,MOENRU,MOEUNR,MOEURN,MOERNU,MOERUN,MORNUE,MORNEU,MORUNE,MORUEN,MORENU,MOREUN,
ENUMRO,ENUMOR,ENURMO,ENUROM,ENUOMR,ENUORM,ENMURO,ENMUOR,ENMRUO,ENMROU,ENMOUR,ENMORU,ENRUMO,ENRUOM,ENRMUO,ENRMOU,ENROUM,ENROMU,ENOUMR,ENOURM,
ENOMUR,ENOMRU,ENORUM,ENORMU,EUNMRO,EUNMOR,EUNRMO,EUNROM,EUNOMR,EUNORM,EUMNRO,EUMNOR,EUMRNO,EUMRON,EUMONR,EUMORN,EURNMO,EURNOM,EURMNO,EURMON,
EURONM,EUROMN,EUONMR,EUONRM,EUOMNR,EUOMRN,EUORNM,EUORMN,EMNURO,EMNUOR,EMNRUO,EMNROU,EMNOUR,EMNORU,EMUNRO,EMUNOR,EMURNO,EMURON,EMUONR,EMUORN,
EMRNUO,EMRNOU,EMRUNO,EMRUON,EMRONU,EMROUN,EMONUR,EMONRU,EMOUNR,EMOURN,EMORNU,EMORUN,ERNUMO,ERNUOM,ERNMUO,ERNMOU,ERNOUM,ERNOMU,ERUNMO,ERUNOM,
ERUMNO,ERUMON,ERUONM,ERUOMN,ERMNUO,ERMNOU,ERMUNO,ERMUON,ERMONU,ERMOUN,ERONUM,ERONMU,EROUNM,EROUMN,EROMNU,EROMUN,EONURM,EONURM,EONMUR,EONMRU,
EONRUM,EONRMU,EOUNMR,EOUNRM,EOUMNR,EOUMRN,EOURNM,EOURMN,EOMNUR,EOMNRU,EOMUNR,EOMURN,EOMRNU,EOMRUN,EORNUM,EORNMU,EORUNM,EORUMN,EORMNU,EORMUN,
RNUMEO,RNUMOE,RNUEMO,RNUEOM,RNUOME,RNUOEM,RNMUEO,RNMUOE,RNMEUO,RNMEOU,RNMOUE,RNMOEU,RNEUMO,RNEUOM,RNEMUO,RNEMOU,RNEOUM,RNEOMU,RNOUME,RNOUEM,
RNOMUE,RNOMEU,RNOEUM,RNOEMU,RUNMEO,RUNMOE,RUNEMO,RUNEOM,RUNOME,RUNOEM,RUMNEO,RUMNOE,RUMENO,RUMEON,RUMONE,RUMOEN,RUENMO,RUENOM,RUEMNO,RUEMON,
RUEONM,RUEOMN,RUONME,RUONEM,RUOMNE,RUOMEN,RUOENM,RUOEMN,RMNUEO,RMNUOE,RMNEUO,RMNEOU,RMNOUE,RMNOEU,RMUNEO,RMUNOE,RMUENO,RMUEON,RMUONE,RMUOEN,
RMENUO,RMENOU,RMEUNO,RMEUON,RMEONU,RMEOUN,RMONUE,RMONEU,RMOUNE,RMOUEN,RMOENU,RMOEUN,RENUMO,RENUOM,RENMUO,RENMOU,RENOUM,RENOMU,REUNMO,REUNOM,
REUMNO,REUMON,REUONM,REUOMN,REMNUO,REMNOU,REMUNO,REMUON,REMONU,REMOUN,REONUM,REONMU,REOUNM,REOUMN,REOMNU,REOMUN,RONUME,RONUEM,RONMUE,RONMEU,
RONEUM,RONEMU,ROUNME,ROUNEM,ROUMNE,ROUMEN,ROUENM,ROUEMN,ROMNUE,ROMNEU,ROMUNE,ROMUEN,ROMENU,ROMEUN,ROENUM,ROENMU,ROEUNM,ROEUMN,ROEMNU,ROEMUN,
ONUMER,ONUMRE,ONUEMR,ONUERM,ONURME,ONUREM,ONMUER,ONMURE,ONMEUR,ONMERU,ONMRUE,ONMREU,ONEUMR,ONEURM,ONEMUR,ONEMRU,ONERUM,ONERMU,ONRUME,ONRUEM,
ONRMUE,ONRMEU,ONREUM,ONREMU,OUNMER,OUNMRE,OUNEMR,OUNERM,OUNRME,OUNREM,OUMNER,OUMNRE,OUMENR,OUMERN,OUMRNE,OUMREN,OUENMR,OUENRM,OUEMNR,OUEMRN,
OUERNM,OUERMN,OURNME,OURNEM,OURMNE,OURMEN,OURENM,OUREMN,OMNUER,OMNURE,OMNEUR,OMNERU,OMNRUE,OMNREU,OMUNER,OMUNRE,OMUENR,OMUERN,OMURNE,OMUREN,
OMENUR,OMENRU,OMEUNR,OMEURN,OMERNU,OMERUN,OMRNUE,OMRNEU,OMRUNE,OMRUEN,OMRENU,OMREUN,OENUMR,OENURM,OENMUR,OENMRU,OENRUM,OENRMU,OEUNMR,OEUNRM,
OEUMNR,OEUMRN,OEURNM,OEURMN,OEMNUR,OEMNRU,OEMUNR,OEMURN,OEMRNU,OEMRUN,OERNUM,OERNMU,OERUNM,OERUMN,OERMNU,OERMUN,ORNUME,ORNUEM,ORNMUE,ORNMEU,
ORNEUM,ORNEMU,ORUNME,ORUNEM,ORUMNE,ORUMEN,ORUENM,ORUEMN,ORMNUE,ORMNEU,ORMUNE,ORMUEN,ORMENU,ORMEUN,ORENUM,ORENMU,OREUNM,OREUMN,OREMNU,OREMUN,
Quantidade de anagramas:  720
```

Figura 13: Saída do Código 2.8.
Fonte: Autoria própria(2026).

> Portanto, o Código 2.8 resolve o seguinte problema: "Quantos são os anagramas da palavra NÚMERO?"

f) Utilizando o Princípio Fundamental da Contagem, obtenha a solução do problema resolvido pelo Código 2.8.

Pelo Princípio Fundamental da Contagem, como há 6 maneiras de escolher uma letra para ser 1ª letra do anagrama e, feito isso, há 5 maneiras de escolher uma letra para ser a 2ª letra do anagrama e, feito isso, há 4 maneiras de escolher uma letra para ser a 3ª letra do anagrama e, feito isso, há 3 maneiras de escolher uma letra para ser a 4ª letra do anagrama e, feito isso, há 2 maneiras de escolher uma letra para ser a 5ª letra do anagrama e, feito isso, há 1 maneira de escolher uma letra para ser a 6ª letra do anagrama, temos que, podemos formar $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$ anagramas.

***Atividade* 10:** Considere o seguinte problema: "Quantos são os anagramas da palavra MOSTRA que começam por vogal e terminam por consoante?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são as letras: $\{M,O,S,T,R,A\}$.
- Configurações: Cada configuração é um anagrama da palavra MOSTRA que começa por vogal e termina por consoante.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Como cada configuração é um anagrama da palavra MOSTRA, precisamos tomar 6 decisões.
Temos três restrições no problema: cada letra só pode ser usada uma única vez; a 1ª letra do anagrama deve ser uma vogal; a 6ª letra do anagrama deve ser uma consoante.
Levando em conta a estratégia, *"se alguma decisão é mais complicada que as demais, ela deve ser tomada em primeiro lugar"*, dadas as letras M,O,S,T,R,A, temos as seguintes decisões:

1ª) Escolher a 1ª letra do anagrama(**que deve ser vogal**);
2ª) Escolher a 6ª letra do anagrama(**que deve ser consoante**);
3ª) Escolher a 2ª letra do anagrama;
4ª) Escolher a 3ª letra do anagrama;
5ª) Escolher a 4ª letra do anagrama;
6ª) Escolher a 5ª letra do anagrama.
Temos, então:
1ª decisão: 2 maneiras, pois como essa letra deve ser vogal e nenhuma vogal foi selecionada, temos 2 letras disponíveis;
2ª decisão: 4 maneiras, pois como essa letra deve ser consoante e nenhuma consoante foi selecionada, temos 4 letras disponíveis;
3ª decisão: 4 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª e 2ª decisões, temos 4 letras disponíveis;
4ª decisão: 3 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª e 3ª decisões, temos 3 letras disponíveis;
5ª decisão: 2 maneiras, pois após a 1ª, 2ª, 3ª e 4ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª e 4ª decisões, temos 2 letras disponíveis;
6ª decisão: 1 maneira, pois após a 1ª, 2ª, 3ª, 4ª e 5ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, temos 1 letra disponível.

c) Execute o código 2.9.

Analisando o código:

I. **Linha 1:** *letras* = [*M*,*O*,*S*,*T*,*R*,*A*] → Define o objeto básico.

II. **Linhas 2 e 3:** *vogais* = [*A*,*O*] e *consoantes* = [*M*,*R*,*S*,*T*] → Diferencia as letras do objeto básico em vogais e consoantes.

III. **Linha 4:** *anagramas* = [] → Define uma lista onde cada elemento será um anagrama.

IV. **Linha 6:** *for letra_1 in letras:* → Representa a 1ª Decisão. O computador "escolhe" uma letra para ser a 1ª do anagrama e "fixa" ela.

V. **Linha 7:** *if letra_1 in vogais* → Esta linha é a materialização da restrição de que a 1ª letra do anagrama deve ser uma vogal. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama, por exemplo, M?????, mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 8:** *for letra_6 in letras:* → Representa a 2ª Decisão. O computador "escolhe" uma letra para ser a 6ª letra do anagrama e "fixa" ela.

VII. **Linha 9:** *if letra_6 in consoantes* → Esta linha é a materialização da restrição de que a 6ª letra do anagrama deve ser uma consoante. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama, por exemplo, $O????A$, mas a restrição do problema diz que ele não serve. Descarte-o."

VIII. **Linha 10:** *for letra_2 in letras:* → Representa a 3ª Decisão. O computador "escolhe" uma letra para ser a 2ª letra do anagrama e "fixa" ela.

IX. **Linha 11:** *if letra_2 != letra_1 and letra_2 != letra_6* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama $OM???M$, mas a restrição do problema diz que ele não serve. Descarte-o."

X. **Linha 12:** *for letra_3 in letras:* → Representa a 4ª Decisão. O computador "escolhe" uma letra para ser a 3ª letra do anagrama e "fixa" ela.

XI. **Linha 13:** *if letra_3 != letra_2 and letra_3 != letra_1 and letra_3 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMO??S$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XII. **Linha 14:** *for letra_4 in letras:* → Representa a 5ª Decisão. O computador "escolhe" uma letra para ser a 4ª letra do anagrama e "fixa" ela.

XIII. **Linha 15:** *if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 != letra_1 and letra_4 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMRM?S$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XIV. **Linha 16:** *for letra_5 in letras:* → Representa a 6ª Decisão. O computador "escolhe" uma letra para ser a 5ª letra do anagrama e para cada letra que está "fixada" nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, o computador testa todas as outras letras possíveis para ser a 5ª letra.

XV. **Linha 17:** *if letra_5 != letra_4 and letra_5 != letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMRTRS$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XVI. **Linha 18:** *anagrama = letra_1 + letra_2 + letra_3 + letra_4 + letra_5 + letra_6* → Materializa a configuração (o anagrama).

XVII. **Linha 19:** *anagramas.append(anagrama)* → Realiza a inserção do anagrama formado na lista *anagramas*.

XVIII. **Resultado da execução do código:** O programa imprimirá os anagramas.

E ao final: *Quantidade de anagramas: 192*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $letra\_1 = M \rightarrow if$ **Bloqueia. Descarte.**

1) $letra\_1 = O \rightarrow if$ **Permite.**

2) $letra\_6 = M \rightarrow if$ **Permite.**

3) $letra\_2 = M \rightarrow if$ **Bloqueia. Descarte.**

3) $letra\_2 = O \rightarrow if$ **Bloqueia. Descarte.**

3) $letra\_2 = S \rightarrow if$ **Permite.**

4) $letra\_3 = M \rightarrow if$ **Bloqueia. Descarte.**

4) $letra\_3 = O \rightarrow if$ **Bloqueia. Descarte.**

4) $letra\_3 = S \rightarrow if$ **Bloqueia. Descarte.**

4) $letra\_3 = T \rightarrow if$ **Permite.**

5) $letra\_4 = M \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = O \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = S \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = T \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = R \rightarrow if$ **Permite.**

6) $letra\_5 = M \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = O \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = S \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = T \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = R \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = A \rightarrow if$ **Permite.** $\rightarrow OSTRAM \ldots$ e assim por diante até testar tudo.

Obtendo, a saída abaixo:

```
Anagramas da palavra MOSTRA que começam por vogal e teminam por consoante:
OSTRAM,OSTARM,OSRTAM,OSRATM,OSATRM,OSARTM,OTSRAM,OTSARM,OTRSAM,OTRASM,OTASRM,OTARSM,
ORSTAM,ORSATM,ORTSAM,ORTASM,ORASTM,ORATSM,OASTRM,OASRTM,OATSRM,OATRSM,OARSTM,OARTSM,
OMTRAS,OMTARS,OMRTAS,OMRATS,OMATRS,OMARTS,OTMRAS,OTMARS,OTRMAS,OTRAMS,OTAMRS,OTARMS,
ORMTAS,ORMATS,ORTMAS,ORTAMS,ORAMTS,ORATMS,OAMTRS,OAMRTS,OATMRS,OATRMS,OARMTS,OARTMS,
OMSRAT,OMSART,OMRSAT,OMRAST,OMASRT,OMARST,OSMRAT,OSMART,OSRMAT,OSRAMT,OSAMRT,OSARMT,
ORMSAT,ORMAST,ORSMAT,ORSAMT,ORAMST,ORASMT,OAMSRT,OAMRST,OASMRT,OASRMT,OARMST,OARSMT,
OMSTAR,OMSATR,OMTSAR,OMTASR,OMASTR,OMATSR,OSMTAR,OSMATR,OSTMAR,OSTAMR,OSAMTR,OSATMR,
OTMSAR,OTMASR,OTSMAR,OTSAMR,OTAMSR,OTASMR,OAMSTR,OAMTSR,OASMTR,OASTMR,OATMSR,OATSMR,
AOSTRM,AOSRTM,AOTSRM,AOTRSM,AORSTM,AORTSM,ASOTRM,ASORTM,ASTORM,ASTROM,ASROTM,ASRTOM,
ATOSRM,ATORSM,ATSORM,ATSROM,ATROSM,ATRSOM,AROSTM,AROTSM,ARSOTM,ARSTOM,ARTOSM,ARTSOM,
AMOTRS,AMORTS,AMTORS,AMTROS,AMROTS,AMRTOS,AOMTRS,AOMRTS,AOTMRS,AOTRMS,AORMTS,AORTMS,
ATMORS,ATMROS,ATOMRS,ATORMS,ATRMOS,ATROMS,ARMOTS,ARMTOS,AROMTS,AROTMS,ARTMOS,ARTOMS,
AMOSRT,AMORST,AMSORT,AMSROT,AMROST,AMRSOT,AOMSRT,AOMRST,AOSMRT,AOSRMT,AORMST,AORSMT,
ASMORT,ASMROT,ASOMRT,ASORMT,ASRMOT,ASROMT,ARMOST,ARMSOT,AROMST,AROSMT,ARSMOT,ARSOMT,
AMOSTR,AMOTSR,AMSOTR,AMSTOR,AMTOSR,AMTSOR,AOMSTR,AOMTSR,AOSMTR,AOSTMR,AOTMSR,AOTSMR,
ASMOTR,ASMTOR,ASOMTR,ASOTMR,ASTMOR,ASTOMR,ATMOSR,ATMSOR,ATOMSR,ATOSMR,ATSMOR,ATSOMR,
Quantidade de anagramas:  192
```

Figura 14: Saída do Código 2.9.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações obtidas satisfazem as condições do problema.
> Obtemos 192 configurações.

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

> Pelo Princípio Fundamental da Contagem, como há 2 maneiras de escolher uma letra para ser 1ª letra do anagrama(pois deve ser vogal) e, feito isso, há 4 maneiras de escolher uma letra para ser a 6ª letra do anagrama(pois deve ser consoante) e, feito isso, há 4 maneiras de escolher uma letra para ser a 2ª letra do anagrama e, feito isso, há 3 maneiras de escolher uma letra para ser a 3ª letra do anagrama e, feito isso, há 2 maneiras de escolher uma letra para ser a 4ª letra do anagrama e, feito isso, há 1 maneira de escolher uma letra para ser a 5ª letra do anagrama, temos que, podemos formar $2 \times 4 \times 4 \times 3 \times 2 \times 1 = 192$ anagramas da palavra MOSTRA que começam por vogal e terminam por consoante.

***Atividade* 11:** Considere o código recursivo 2.10 que resolve um determinado problema de contagem:

a) Execute o código acima. O que esse código fornece como resultado?

Ao, simplesmente, executar o Código 2.10, ele listará os 720 anagramas da palavra NÚMERO.

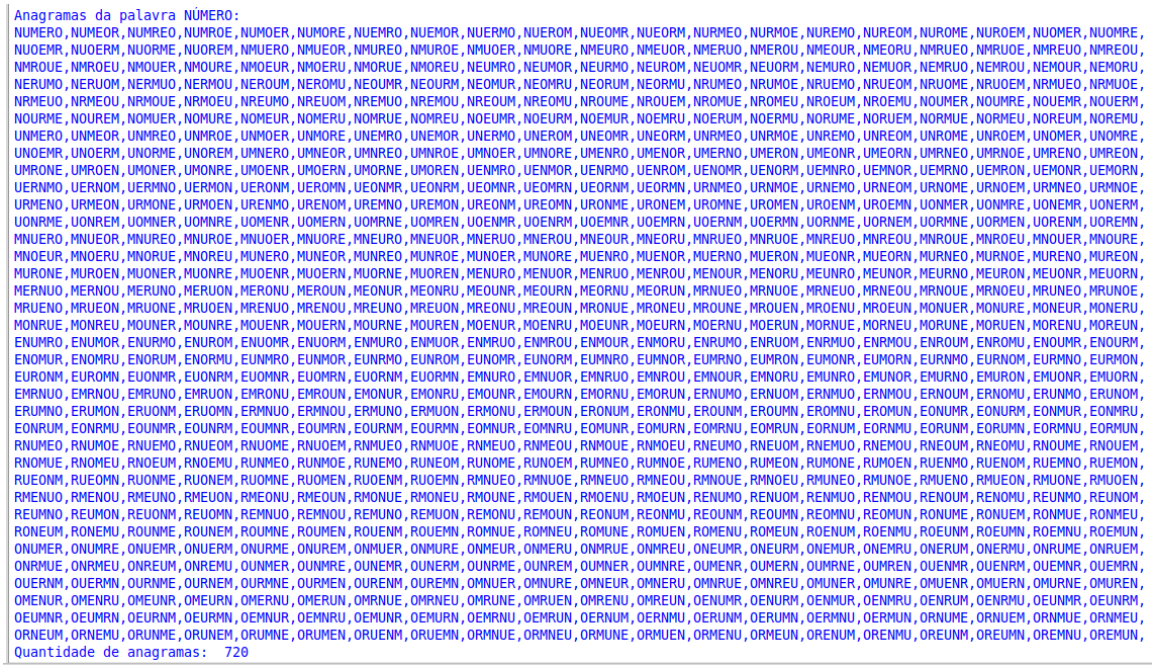

Figura 15: Saída do Código 2.10.
Fonte: Autoria própria(2026).

b) Compare-o com o Código 2.9. Qual é a diferença?

Diferente do Código 2.9 que usava vários *for* um dentro do outro, esse código introduz o conceito computacional conhecido como *Recursividade*.

c) Considere uma palavra com uma quantidade menor de letras, por exemplo ALO, e execute o Código 2.10 passo a passo manualmente, para entender o seu funcionamento.

Analisando o código:

I. **Linha 1:** *def permutacoes(palavra):* → Define uma função chamada *permutacoes* que aceita um texto (*palavra*) como entrada.

II. **Linhas 2 e 3:** $if len(palavra) == 1 :$ e $return[palavra] \rightarrow$ Se a palavra tiver apenas 1 letra, por exemplo *A*, qual é o anagrama dela? É ela mesma. Toda recursão precisa de um ponto de parada, o chamado *Caso Base*. Sem isso, o computador ficaria num *loop* infinito tentando dividir a palavra para sempre.

III. **Linha 5:** $resultado = [] \rightarrow$ Cria uma lista vazia para guardar os anagramas encontrados.

IV. **Linha 7:** *for i in range(len(palavra)):* $\rightarrow$ O laço *for* vai passar por cada letra da palavra original, uma por uma.

V. **Linha 8:** $char\_atual = palavra[i] \rightarrow$ "Fixa" a letra da vez. Por exemplo, se a palavra for "ALO" e $i = 0$, fixa o "A".

VI. **Linha 9:** $resto = palavra[: i] + palavra[i + 1 :] \rightarrow$ Cria uma nova palavra com tudo o que sobrou. Por exemplo, se tirou "A" de "ALO", o *resto* é "LO".

VII. **Linha 10:** *for p in permutacoes(resto):* $\rightarrow$ A função chama ela mesma, mas agora passando apenas o *resto*. É como se dissesse: "Eu não sei resolver "ALO" inteiro agora, mas se eu fixar o "A", você resolve "LO" para mim?"

VIII. **Linha 11:** $resultado.append(char\_atual + p) \rightarrow$ Concatena a letra que estava fixa ($char\_atual$) na frente de cada anagrama que voltou da recursão (p).

IX. **Linhas 15 e 17:** $palavra =$ *"ALO"* e $todas\_permutacoes = permutacoes(palavra) \rightarrow$ O usuário define a palavra "ALO" e dispara o processo.

X. **Resultado da execução do código:** O programa imprimirá os anagramas.

E ao final: *Quantidade de anagramas: 6*

Visualizemos o que acontece na "cabeça" do Python. Para o melhor entendimento da recursão, imagine que cada vez que a função *permutacoes* é chamada, abre-se um novo "nível" dentro da anterior. O computador pausa o "nível" de fora para resolver

o de dentro.

Início) **Nível 1**

- *permutacoes("ALO")*
- A função verifica: $len(\text{"ALO"}) == 1$? Não. $len(\text{"ALO"}) = 3$.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 1**→ (Fixa a letra "A")

- $i = 0$;
- $char\_atual = \text{"A"}$;
- $resto = \text{"LO"}$;
- Chamada para $permutacoes(\text{"LO"})$.

Início) **Nível 2**

- *permutacoes("LO")*
- A função verifica: $len(\text{"LO"}) == 1$? Não. $len(\text{"LO"}) = 2$.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "L")

- $i = 0$;
- $char\_atual = \text{"L"}$;
- $resto = \text{"O"}$;
- Chamada para $permutacoes(\text{"O"})$;

Início) **Nível 3**

* *permutacoes("O")*
* A função verifica: $len(\text{"O"}) == 1$? Sim.
* $resultado = [\text{"O"}]$;
* Fecha o Nível 3.

- Concatena $\text{"L"} + \text{"O"}$;
- $resultado = [\text{"LO"}]$.

2) **2ª iteração do Nível 2** → (Fixa a letra "O")

- $i = 1$;
- $char\_atual = \text{"O"}$;

- *resto* = *“L”*;
- Chamada para *permutacoes*(*“L”*);

Início) **Nível 3**

* *permutacoes(“L”)*
* A função verifica: *len*(*“L”*) == 1? Sim.
* *resultado* = [*“L”*];
* Fecha o Nível 3.

- Concatena *“O”* + *“L”*;
- *resultado* = [*“OL”*].

Fim) **Nível 2**

- *resultado* = [*“LO”*,*“OL”*].
- Fecha o Nível 2.

- Concatena *“A”* + *“LO”*;
- Concatena *“A”* + *“OL”*;
- *resultado* = [*“ALO”*,*“AOL”*].

2) **2ª iteração do Nível 1**→ (Fixa a letra “L”)

- $i = 1$;
- *char_atual* = *“L”*;
- *resto* = *“AO”*;
- Chamada para *permutacoes*(*“AO”*).

Início) **Nível 2**

- *permutacoes(“AO”)*
- A função verifica: *len*(*“AO”*) == 1? Não. *len*(*“AO”*) = 2.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra “A”)

- $i = 0$;
- *char_atual* = *“A”*;
- *resto* = *“O”*;
- Chamada para *permutacoes*(*“O”*);

Início) **Nível 3**

* *permutacoes(“O”)*

        * A função verifica: $len(\text{“}O\text{”}) == 1$? Sim.
        * $resultado = [\text{“}O\text{”}]$;
        * Fecha o Nível 3.
    - Concatena *“A”* + *“O”*;
    - $resultado = [\text{“}AO\text{”}]$.

2) **2ª iteração do Nível 2** → (Fixa a letra “O”)

    - $i = 1$;
    - $char\_atual = \text{“}O\text{”}$;
    - $resto = \text{“}A\text{”}$;
    - Chamada para *permutacoes(“A”)*;

Início) **Nível 3**

    * *permutacoes(“A”)*
    * A função verifica: $len(\text{“}A\text{”}) == 1$? Sim.
    * $resultado = [\text{“}A\text{”}]$;
    * Fecha o Nível 3.
    - Concatena *“O”* + *“A”*;
    - $resultado = [\text{“}OA\text{”}]$.

Fim) **Nível 2**

    - $resultado = [\text{“}AO\text{”},\text{“}OA\text{”}]$.
    - Fecha o Nível 2.

- Concatena *“L”* + *“AO”*;
- Concatena *“L”* + *“OA”*;
- $resultado = [\text{“}ALO\text{”},\text{“}AOL\text{”},\text{“}LAO\text{”},\text{“}LOA\text{”}]$.

3) **3ª iteração do Nível 1**→ (Fixa a letra “O”)

- $i = 2$;
- $char\_atual = \text{“}O\text{”}$;
- $resto = \text{“}AL\text{”}$;
- Chamada para *permutacoes(“AL”)*.

Início) **Nível 2**

- *permutacoes(“AL”)*
- A função verifica: $len(\text{“}AL\text{”}) == 1$? Não. $len(\text{“}AL\text{”}) = 2$.

- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "A")
   - $i = 0$;
   - $char\_atual$ = *"A"*;
   - $resto$ = *"L"*;
   - Chamada para $permutacoes$(*"L"*);

Início) **Nível 3**
   - $permutacoes$(*"L"*)
   - A função verifica: len(*"L"*) == 1? Sim.
   - $resultado$ = [*"L"*];
   - Fecha o Nível 3.
   - Concatena *"A"* + *"L"*;
   - $resultado$ = [*"AL"*].

2) **2ª iteração do Nível 2** → (Fixa a letra "L")
   - $i = 1$;
   - $char\_atual$ = *"L"*;
   - $resto$ = *"A"*;
   - Chamada para $permutacoes$(*"A"*);

Início) **Nível 3**
   - $permutacoes$(*"A"*)
   - A função verifica: len(*"A"*) == 1? Sim.
   - $resultado$ = [*"A"*];
   - Fecha o Nível 3.
   - Concatena *"L"* + *"A"*;
   - $resultado$ = [*"LA"*].

Fim) **Nível 2**
   - $resultado$ = [*"AL"*,*"LA"*].
   - Fecha o Nível 2.

- Concatena *"O"* + *"AL"*;
- Concatena *"O"* + *"LA"*;
- $resultado$ = [*"ALO"*,*"AOL"*,*"LAO"*,*"LOA"*,*"OAL"*,*"OLA"*].

Fim) **Nível 1**

- Fecha o Nível 1.
- $resultado = [“ALO”,“AOL”,“LAO”,“LOA”,“OAL”,“OLA”]$.

Temos o seguinte resultado ao executar o Código 2.10, utilizando como entrada a palavra ALO:

```
Anagramas da palavra ALO:
ALO,AOL,LAO,LOA,OAL,OLA,
Quantidade de anagramas:  6
>>>
```

Figura 16: Saída do Código 2.10.
Fonte: Autoria própria(2026).

Chegamos, assim, ao fim da iteração do Código 2.10. Notemos que para resolver o problema grande $permutacoes(“ALO”)$, o computador “empilhou” problemas menores até chegar no caso trivial (uma letra só), e depois veio “desempilhando” e montando as respostas.
O código prova de forma mecânica a fórmula $n! = n \times (n-1)!$.

***Atividade* 12:** Considere o código 2.11, uma variação do Código 2.10, em Python que determina os anagramas de uma palavra:

a) Execute o código acima, usando como exemplo a palavra AMOR.

Ao executar o Código 2.11, que é idêntico ao Código 2.10, obtemos os anagramas da palavra AMOR e a quantidade dos mesmos.

```
Anagramas da palavra AMOR:
AMOR,AMRO,AOMR,AORM,ARMO,AROM,MAOR,MARO,
MOAR,MORA,MRAO,MROA,OAMR,OARM,OMAR,OMRA,
ORAM,ORMA,RAMO,RAOM,RMAO,RMOA,ROAM,ROMA,
Quantidade de anagramas:  24
>
```

Figura 17: Saída do Código 2.11 usando a palavra AMOR.
Fonte: Autoria própria(2026).

b) Usando o Princípio Fundamental da Contagem, determine o número de anagramas da palavra AMOR. Compare o seu resultado com o aquele fornecido pelo Código 2.11.

- **Objetos básicos.** letras: $\{A,M,O,R\}$.
- **Configurações.** Cada configuração é formada por um anagrama com as letras da palavra AMOR.
- **Restrições.** Temos uma única restrição, cada letra só pode ser usada uma única vez.
- **Decisões.** Há quatro decisões a serem tomadas: $1^{\underline{a}}$) Escolher a primeira letra do anagrama; $2^{\underline{a}}$) Escolher a segunda letra do anagrama; $3^{\underline{a}}$) Escolher a terceira letra do anagrama; $4^{\underline{a}}$) Escolher a quarta letra do anagrama.

  Temos, então:

  $1^{\underline{a}}$ decisão: 4 maneiras, pois antes de iniciarmos a formação do anagrama, temos 4 letras disponíveis;

  $2^{\underline{a}}$ decisão: 3 maneiras, pois após a $1^{\underline{a}}$ decisão ter sido realizada, para qualquer letra escolhida na $1^{\underline{a}}$ decisão , temos 3 letras disponíveis;

  $3^{\underline{a}}$ decisão: 2 maneiras, pois após a $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões terem sido feitas, para quaisquer letras escolhidas nas $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões, temos 2 letras disponíveis;

  $4^{\underline{a}}$ decisão: 1 maneira, pois após a $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões terem sido feitas, para quaisquer letras escolhidas nas $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões, temos 1 letra disponível.

- **Pelo Princípio Fundamental da Conagem**, a palavra AMOR possui $4 \times 3 \times 2 \times 1 = 4! = 24$ anagramas.

Obtemos, assim, o mesmo resultado encontrado na execução do Código 2.11 quando usamos a palavra AMOR como entrada.

c) Execute o código usando, agora, como exemplo a palavra AMAR.

Ao executar o Código 2.11, obtemos os anagramas da palavra AMAR e a quantidade dos mesmos.

```
Anagramas da palavra AMAR:
AMAR,AMRA,AAMR,AARM,ARMA,ARAM,MAAR,MARA,
MAAR,MARA,MRAA,MRAA,AAMR,AARM,AMAR,AMRA,
ARAM,ARMA,RAMA,RAAM,RMAA,RMAA,RAAM,RAMA,
Quantidade de anagramas:  24
```

Figura 18: Saída do Código 2.11 usando a palavra AMAR.
Fonte: Autoria própria(2026).

d) Analise, atentamente, a saída de execução do código. Você percebeu alguma coisa estranha? O quê?

Ao analisar a lista de anagramas, percebemos que há anagramas repetidos. Por exemplo, o primeiro anagrama, AMAR, também é o décimo quinto anagrama na nossa lista.
O computador ficou maluco? Por que ele escreveu a mesma palavra duas vezes?
Devemos notar que trocar o A do início com o A do meio da palavra AMAR não gera uma nova palavra.
Então, o Código gerou anagramas em excesso. Vamos corrigir?

e) Então, quantos são os anagramas da palavra AMAR?

Comecemos colocando os anagramas iguais em uma mesma "caixa":
[AMAR,AMAR]
[AMRA,AMRA]
[AAMR,AAMR]
[AARM,AARM]
[ARMA,ARMA]
[ARAM,ARAM]
[MAAR,MAAR]

[MARA,MARA]
[MRAA,MRAA]
[RAMA,RAMA]
[RAAM,RAAM]
[RMAA,RMAA]
Observemos que cada "caixa" possui dois anagramas iguais(como os "A" são iguais, contamos cada anagrama $2! = 2$ vezes), ou seja, para cada anagrama da palavra AMAR produzido, dois são repetidos. Com isso, podemos concluir que a palavra AMAR possui $\frac{24}{2} = 12$ anagramas.
Anagramas da palavra AMAR = [AMAR, AMRA, AAMR, AARM, ARMA, ARAM, MAAR, MARA, MRAA, RAMA, RAAM, RMAA].

f) Repita os itens c), d) e e) considerando a palavra BABA.

Ao executar o Código 2.11, obtemos os anagramas da palavra BABA e a quantidade dos mesmos.

```
Anagramas da palavra BABA:
BABA,BAAB,BBAA,BBAA,BAAB,BABA,ABBA,ABAB,
ABBA,ABAB,AABB,AABB,BBAA,BBAA,BABA,BAAB,
BABA,BAAB,ABAB,ABBA,AABB,AABB,ABBA,ABAB,
Quantidade de anagramas:  24
```

Figura 19: Saída do Código 2.11 usando a palavra BABA.
Fonte: Autoria própria(2026).

Ao analisar a lista de anagramas, percebamos que há anagramas repetidos. Por exemplo, o primeiro anagrama, BABA, também é o sexto, décimo quinto e décimo sétimo anagramas na nossa lista.
Devemos notar que trocar o B do início com o B do meio e o A do meio com o A do final da palavra BABA não gera uma nova palavra.
Dessa vez parece que o excesso foi maior. Vamos corrigir?
Comecemos colocando os anagramas iguais em uma mesma "caixa":
[BABA,BABA,BABA,BABA]
[BAAB,BAAB,BAAB,BAAB]
[BBAA,BBAA,BBAA,BBAA]

[ABBA,ABBA,ABBA,ABBA]
[ABAB,ABAB,ABAB,ABAB]
[AABB,AABB,AABB,AABB]
Observemos que cada "caixa" possui quatro anagramas iguais(como os "A" são iguais, contamos cada anagrama 2! vezes; analogamente, como os "B" são iguais, contamos cada anagrama 2! vezes), ou seja, para cada anagrama da palavra BABA produzido, quatro($2! \times 2!$) são repetidos. Com isso, podemos concluir que a palavra BABA possui $\frac{24}{4} = 6$ anagramas.
Anagramas da palavra BABA = [BABA, BAAB, BBAA, ABBA, ABAB, AABB].

g) Repita os itens c), d) e e) considerando a palavra AAZA(força em arábico).

Ao executar o Código 2.11, obtemos os anagramas da palavra AAZA e a quantidade dos mesmos.

```
Anagramas da palavra AAZA:
AAZA,AAAZ,AZAA,AZAA,AAAZ,AAZA,AAZA,AAAZ,
AZAA,AZAA,AAAZ,AAZA,ZAAA,ZAAA,ZAAA,ZAAA,
ZAAA,ZAAA,AAAZ,AAZA,AAAZ,AAZA,AZAA,AZAA,
Quantidade de anagramas:  24
```

Figura 20: Saída do Código 2.11 usando a palavra AAZA.
Fonte: Autoria própria(2026).

Ao analisar a lista de anagramas, percebamos que há anagramas repetidos. Por exemplo, o primeiro anagrama, AAZA, também é o sexto, sétimo, décimo segundo, vigésimo e vigésimo segundo anagramas na nossa lista.
Devemos notar que trocar o A do início com o A do meio ou com o A do final da palavra AAZA não gera uma nova palavra.
Vamos corrigir?
Comecemos colocando os anagramas iguais em uma mesma "caixa":
[AAZA,AAZA,AAZA,AAZA,AAZA,AAZA]
[AAAZ,AAAZ,AAAZ,AAAZ,AAAZ,AAAZ]
[AZAA,AZAA,AZAA,AZAA,AZAA,AZAA]
[ZAAA,ZAAA,ZAAA,ZAAA,ZAAA,ZAAA]

Observemos que cada "caixa" possui seis anagramas iguais(como os "A" são iguais, contamos cada anagrama $3! = 6$ vezes), ou seja, para cada anagrama da palavra AAZA produzido, seis são repetidos. Com isso, podemos concluir que a palavra AAZA possui $\frac{24}{6} = 4$ anagramas.
Anagramas da palavra AAZA = [AAZA, AAAZ, AZAA,ZAAA].

h) Por quê o Código 2.11 não funciona com as palavras AMAR, BABA e AAZA?

O computador não "lê" letras; ele "lê" índices de memória. Para o código recursivo, a *string* "AMAR", por exemplo, é tratada internamente como um vetor de posições: $[0, 1, 2, 3]$.
O algoritmo permuta os índices.

- **Permutação A:** Índices $[0, 1, 2, 3] \rightarrow$ Letras $A_1MA_2R \rightarrow$ "AMAR".
- **Permutação B:** Índices $[2, 1, 0, 3] \rightarrow$ Letras $A_2MA_1R \rightarrow$ "AMAR"

Para o computador, a **Permutação A** e a **Permutação B** são objetos diferentes porque vieram de índices diferentes. Ele não verifica o valor do conteúdo.

3.4 Quarto encontro

Na Atividade 12, deve-se ter notado que o código que calcula os anagramas de uma palavra só funciona corretamente quando a mesma possui letras distintas, ou seja, o código só calcula a quantidade de permutações simples.

O código da Atividade 13 tem como objetivo corrigir esse problema, isto é, calcular o número de anagramas de qualquer palavra.

***Atividade* 13:** Considere o código recursivo 2.12 em Python que determina os anagramas de uma palavra mas, agora, de uma forma mais organizada, resolvendo os problemas que tivemos na Atividade 12:

A ideia por trás desse Código é a mesma que utilizamos nos itens e), f) e g) da Atividade 12.

- **Linhas 1 a 15:** *def permutacoes(palavra):* → O código gera todos os "anagramas" da palavra dada;
- **Linhas 17 a 45:** *def remover_duplicatas_sort(lista):* → Agora, o código começará a remover os excessos.
  - **Linha 19:** *lista_ordenada = sorted(lista)* → Para encontrar os excessos mais rápido, o primeiro passo dado é ordenar a lista de anagramas;
  - **Linhas 24 a 29:** *for i in range(len(lista_ordenada)):* → Neste bloco, o código agrupa e mostra as duplicatas antes de removê-las;
  - **Linhas 31 a 37:** *for i, uplas in enumerate(lista_de_listas,1):* → Neste bloco, o código imprime os grupos formados, mostrando que o computador contou em excesso;
  - **Linhas 40 a 45:** *resultado = [lista_ordenda[0]]* → Neste bloco, o código percorre a lista ordenada e só aceita um item se ele for diferente do anterior, aqui ocorre a retirada dos excessos.

a) Execute o código acima, usando como exemplo a palavra AMAR.

- A função *permutacoes* gera 24 anagramas(com os repetidos), conforme vemos na Figura 18;
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui dois anagramas iguais), filtra os anagramas(lista os $\frac{24}{2} = 12$ anagramas).

```
Digite uma palavra qualquer: AMAR
  1. ['AAMR', 'AAMR']
  2. ['AARM', 'AARM']
  3. ['AMAR', 'AMAR']
  4. ['AMRA', 'AMRA']
  5. ['ARAM', 'ARAM']
  6. ['ARMA', 'ARMA']
  7. ['MAAR', 'MAAR']
  8. ['MARA', 'MARA']
  9. ['MRAA', 'MRAA']
 10. ['RAAM', 'RAAM']
 11. ['RAMA', 'RAMA']
 12. ['RMAA', 'RMAA']

Número de anagramas:  24

Número de anagramas repetidos em cada caixa:  2


Anagramas da palavra AMAR:
  1. AAMR
  2. AARM
  3. AMAR
  4. AMRA
  5. ARAM
  6. ARMA
  7. MAAR
  8. MARA
  9. MRAA
 10. RAAM
 11. RAMA
 12. RMAA
Quantidade de anagramas da palavra AMAR: 12
```

Figura 21: Saída do Código 2.12 usando a palavra AMAR.
Fonte: Autoria própria(2026).

b) O resultado obtido é, de fato, a quantidade de anagramas da palavra AMAR?

O resultado obtido é a quantidade de anagramas da palavra AMAR $\rightarrow$ $\frac{4 \times 3 \times 2 \times 1}{2 \times 1} = \frac{4!}{2!} = \frac{24}{2} = 12$, conforme visto no item e) da Atividade 12.

c) Execute o código usando, agora, como exemplo a palavra AAZA.

- A função *permutacoes* gera 24 anagramas(com os repetidos), conforme vemos na Figura 20;
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui seis anagramas iguais), filtra os anagramas(lista os $\frac{24}{6} = 4$ anagramas).

```
Digite uma palavra qualquer: AAZA
  1. ['AAAZ', 'AAAZ', 'AAAZ', 'AAAZ', 'AAAZ', 'AAAZ']
  2. ['AAZA', 'AAZA', 'AAZA', 'AAZA', 'AAZA', 'AAZA']
  3. ['AZAA', 'AZAA', 'AZAA', 'AZAA', 'AZAA', 'AZAA']
  4. ['ZAAA', 'ZAAA', 'ZAAA', 'ZAAA', 'ZAAA', 'ZAAA']

Número de anagramas:  24

Número de anagramas repetidos em cada caixa:  6


Anagramas da palavra AAZA:
  1. AAAZ
  2. AAZA
  3. AZAA
  4. ZAAA
Quantidade de anagramas da palavra AAZA: 4
```

Figura 22: Saída do Código 2.12 usando a palavra AAZA.
Fonte: Autoria própria(2026).

d) O resultado obtido é, de fato, a quantidade de anagramas da palavra AAZA?

O resultado obtido é a quantidade de anagramas da palavra AAZA $\rightarrow \frac{4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = \frac{4!}{3!} = \frac{24}{6} = 4$, conforme visto no item g) da Atividade 12.

e) Execute o Código 2.12, usando a palavra SOSSEGO. Quantos anagramas essa palavra possui?

- A função *permutacoes* gera 5040 anagramas(com os repetidos);
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui seis anagramas iguais), filtra os anagramas(lista os $\frac{5040}{12} = 420$ anagramas).

Pelo que vimos, nos itens anteriores, o resultado obtido é a quantidade de anagramas da palavra SOSSEGO $\rightarrow \frac{7!}{3! \times 2!} = \frac{5040}{12} = 420$. Importante notar, que cada "caixa" possui doze anagramas iguais(como os "S" são iguais, contamos cada anagrama $3! = 6$ vezes e, como os "O" são iguais, contamos cada anagrama $2! = 2$ vezes), ou seja, para cada "anagrama" da palavra SOSSEGO produzido, doze são repetidos.

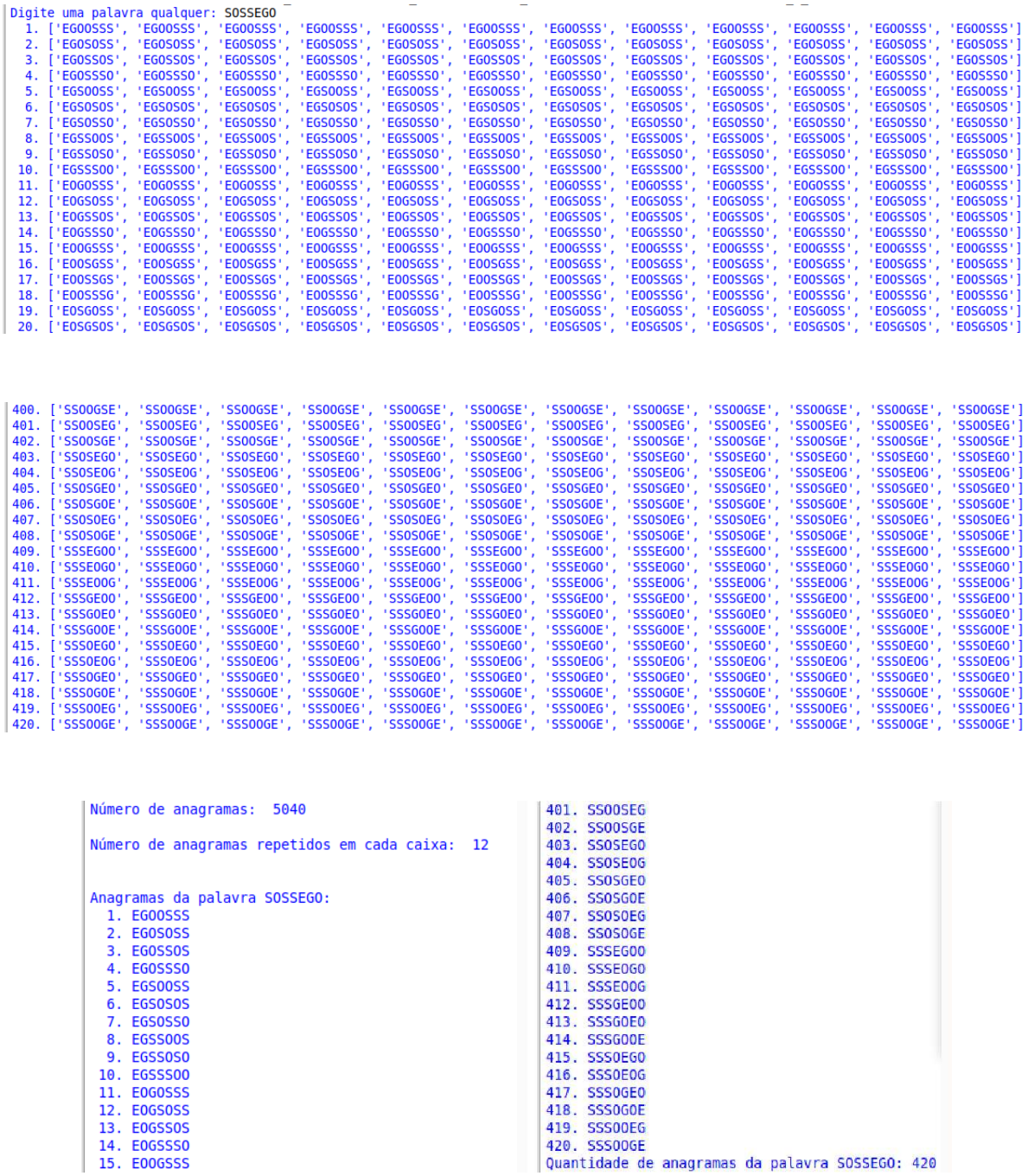

Figura 23: Resumo da Saída do Código 2.12 usando a palavra SOSSEGO.
Fonte: Autoria própria(2026).

f) Determine o número de anagramas das palavras MANIA, CORRER e PASSISTA, usando o Princípio Fundamental da Contagem e o Princípio k para 1.

Para cada palavra, temos:

- **Objetos básicos:** As letras que compõem cada palavra, considerando-as distintas, por exemplo: para a palavra MANIA, temos $letras = [M,A,N,I,A]$.
- **Configurações:** Cada configuração é um anagrama da palavra considerada.
- **Restrições:** Temos uma única restrição, cada letra só pode ser usada uma única vez.
- **Decisões:** A quantidade de decisões é igual à quantidade de elementos do conjunto formado pelos objetos básicos: $1^{\underline{a}}$) Escolher a primeira letra do anagrama; $2^{\underline{a}}$) Escolher a segunda letra do anagrama; $3^{\underline{a}}$) Escolher a terceira letra do anagrama; $4^{\underline{a}}$) Escolher a quarta letra do anagrama; e, assim por diante.
- **MANIA**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $5 \times 4 \times 3 \times 2 \times 1 = 5! = 120$ anagramas;
  - Como há duas letras "A", contamos cada anagrama $2! = 2$ vezes;
  - Logo, pelo Princípio k para 1 (nesse caso $k = 2!$), a palavra MANIA possui $\frac{120}{2!} = \frac{120}{2} = 60$ anagramas.
- **CORRER**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 6! = 720$ anagramas;
  - Como há três letras "R", contamos cada anagrama $3! = 6$ vezes;
  - Logo, pelo Princípio k para 1(nesse caso $k = 3!$), a palavra CORRER possui $\frac{720}{3!} = \frac{720}{6} = 120$ anagramas.

- **PASSISTA**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 8! = 40320$ anagramas;
  - Como há três letras "S", contamos cada anagrama $3! = 6$ vezes e, há, também duas letras "A", contamos cada anagrama $2! = 2$ vezes, logo, estamos contando cada anagrama $2! \times 3! = 2 \times 6 = 12$ vezes;
  - Logo, pelo Princípio k para 1(nesse caso $k = 2! \times 3!$), a palavra CORRER possui $\frac{40320}{2! \times 3!} = \frac{40320}{2 \times 6} = \frac{40320}{12} = 3360$ anagramas.

***Atividade* 14:** Vamos ao seguinte problema: "Considere 5 pessoas A, B, C, D e E. Quantas filas com 3 dessas 5 pessoas podemos formar?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: São as 5 pessoas: $\{A, B, C, D, E\}$.
- Configurações: Cada configuração é uma fila com três dessas cinco pessoas. Matematicamente, é uma tripla ordenada $(pessoa\_1, pessoa\_2, pessoa\_3)$, onde $pessoa\_1$ é a pessoa no começo da fila, $pessoa\_2$ é a pessoa no meio da fila e $pessoa\_3$ é a última pessoa da fila.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma lugar na fila para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a pessoa do início da fila($pessoa\_1$); 2ª) Escolher a pessoa do meio da fila($pessoa\_2$); 3ª) Escolher a pessoa do final da fila($pessoa\_3$).
Temos, então:
1ª decisão: 5 maneiras;
2ª decisão: 4 maneiras;
3ª decisão: 3 maneiras.

c) Execute o seguinte código 2.13.

I. **Linha 1 à Linha 21:** *def formar_filas(pessoas)*Define uma função que forma filas.

II. **Linha 10:** *for pessoa_1 in pessoas:* → Representa a 1ª Decisão. O computador "escolhe" uma pessoa para ser a primeira da fila e "fixa" ela.

IV. **Linha 11:** *for pessoa_2 in pessoas:* → Representa a 2ª Decisão. O computador "escolhe" uma pessoa para ser a segunda da fila e "fixa" ela.

V. **Linha 12:** *if pessoa_2 != pessoa_1 :* → Esta linha é a materialização da restrição de que cada pessoa ocupa um único lugar na fila. Ela diz: "Eu estou prestes a gerar a tripla $(A,A,?)$, mas a restrição do problema diz que ela não serve. Descarte-a."

VI. **Linha 13:** *for pessoa_3 in pessoas:* → Representa a 3ª Decisão. Para cada pessoa que está "fixada" na 1ª e 2ª decisões, o computador testa todas as outras pessoas possíveis para ser a última da fila.

VII. **Linha 14:** *if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:* → Esta linha é a materialização da restrição de que que cada pessoa ocupa um único lugar na fila. Ela diz: "Eu estou prestes a gerar a tripla (A,B,A), por exemplo, mas a restrição do problema diz que ela não serve. Descarte-a."E, "Eu estou prestes a gerar a tripla (A,B,B), mas a restrição do problema diz que ela não serve. Descarte-a."

VIII. **Linhas 15 e 16:** *contador_filas += 1* → Conta a fila formada; $fila\_formada$ = *pessoa_1,pessoa_2,pessoa_3* → Materializa uma configuração (a tripla ordenada).

IX. **Linha 17:** *resultado.append(fila_formada)* → Realiza a inserção da tripla formada na lista *resultado*, que conterá todas as filas formadas.

X. **Resultado da execução do código:** O programa imprimirá as filas formadas.

E ao final: *Total de filas diferentes: 60.* Para a lista *pessoas* = [*A*,*B*,*C*,*D*,*E*].

Na Figura 24 temos a saída após a execução do código.

```
============================================================
PASSO 1: Forma as filas.
============================================================
Fila   1: ['A', 'B', 'C']
Fila   2: ['A', 'B', 'D']
Fila   3: ['A', 'B', 'E']
Fila   4: ['A', 'C', 'B']
Fila   5: ['A', 'C', 'D']
Fila   6: ['A', 'C', 'E']
Fila   7: ['A', 'D', 'B']
Fila   8: ['A', 'D', 'C']
Fila   9: ['A', 'D', 'E']
Fila  10: ['A', 'E', 'B']
Fila  11: ['A', 'E', 'C']
Fila  12: ['A', 'E', 'D']
Fila  13: ['B', 'A', 'C']
Fila  14: ['B', 'A', 'D']
Fila  15: ['B', 'A', 'E']
Fila  16: ['B', 'C', 'A']
Fila  17: ['B', 'C', 'D']
Fila  18: ['B', 'C', 'E']
Fila  19: ['B', 'D', 'A']
Fila  20: ['B', 'D', 'C']
Fila  21: ['B', 'D', 'E']
Fila  22: ['B', 'E', 'A']
Fila  23: ['B', 'E', 'C']
Fila  24: ['B', 'E', 'D']
Fila  25: ['C', 'A', 'B']
Fila  26: ['C', 'A', 'D']
Fila  27: ['C', 'A', 'E']
Fila  28: ['C', 'B', 'A']
Fila  29: ['C', 'B', 'D']
Fila  30: ['C', 'B', 'E']
Fila  31: ['C', 'D', 'A']
Fila  32: ['C', 'D', 'B']
Fila  33: ['C', 'D', 'E']
Fila  34: ['C', 'E', 'A']
Fila  35: ['C', 'E', 'B']
Fila  36: ['C', 'E', 'D']
Fila  37: ['D', 'A', 'B']
Fila  38: ['D', 'A', 'C']
Fila  39: ['D', 'A', 'E']
Fila  40: ['D', 'B', 'A']
Fila  41: ['D', 'B', 'C']
Fila  42: ['D', 'B', 'E']
Fila  43: ['D', 'C', 'A']
Fila  44: ['D', 'C', 'B']
Fila  45: ['D', 'C', 'E']
Fila  46: ['D', 'E', 'A']
Fila  47: ['D', 'E', 'B']
Fila  48: ['D', 'E', 'C']
Fila  49: ['E', 'A', 'B']
Fila  50: ['E', 'A', 'C']
Fila  51: ['E', 'A', 'D']
Fila  52: ['E', 'B', 'A']
Fila  53: ['E', 'B', 'C']
Fila  54: ['E', 'B', 'D']
Fila  55: ['E', 'C', 'A']
Fila  56: ['E', 'C', 'B']
Fila  57: ['E', 'C', 'D']
Fila  58: ['E', 'D', 'A']
Fila  59: ['E', 'D', 'B']
Fila  60: ['E', 'D', 'C']

Total de filas diferentes: 60
```

Figura 24: Saída do Código 2.13.
Fonte: Autoria própria(2026).

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

As configurações satisfazem as condições do problema.
Obtemos 60 configurações.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma pessoa para ser a primeira da fila e, feito isso, há quatro maneiras de escolher uma pessoa para ser a segunda da fila e, feito isso, há três maneiras de escolher uma pessoa para ser a última da fila, temos que, podemos formar $5 \times 4 \times 3 = 60$ filas com três pessoas. Resultado também encontrado ao executar o Código 2.13.

f) Esse problema é parecido com algum problema que já foi resolvido? Qual?

Esse problema é idêntico ao problema da Atividade 7. É um problema onde as configurações são arranjos simples de cinco pessoas tomadas três a três.

g) E quantas filas com 4 pessoas podemos formar?

Agora, teremos quatro decisões a tomar.
Então, pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma pessoa para ser a primeira da fila e, feito isso, há quatro maneiras de escolher uma pessoa para ser a segunda da fila e, feito isso, há três maneiras de escolher uma pessoa para ser a terceira da fila e, feito isso, há duas maneiras de escolher uma pessoa para ser a última da fila, temos que, podemos formar $5 \times 4 \times 3 \times 2 = 120$ filas com quatro pessoas. Ou seja, temos 120 arranjos de cinco pessoas tomadas quatro a quatro.

***Atividade* 15:** Agora, vamos a esse problema: "Considere 5 pessoas A, B, C, D e E. Quantos grupos com 3 dessas 5 pessoas podemos formar?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: São as 5 pessoas: $\{A, B, C, D, E\}$.
- Configurações: Cada configuração é um grupo com três dessas cinco pessoas. Matematicamente, é um conjunto $\{pessoa\_1, pessoa\_2, pessoa\_3\}$, formado por três das cinco pessoas consideradas.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos, então uma restrição implícita no problema, uma lugar no grupo para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a primeira pessoa do grupo(*pessoa*_1); 2ª) Escolher a segunda pessoa do grupo(*pessoa*_2); 3ª) Escolher a terceira pessoa do grupo(*pessoa*_3).
Temos, então:
1ª decisão: 5 maneiras;
2ª decisão: 4 maneiras;
3ª decisão: 3 maneiras.

c) A solução desse problema é igual à solução do problema da Atividade 14? Ou seja, a quantidade de filas é igual à quantidade de grupos? Justifique.

Na maioria das vezes, a resposta dada pelos alunos a essa pergunta é SIM. Vejamos:
Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma pessoa para ser o primeira do grupo e, feito isso, há quatro maneiras de escolher uma pessoa para ser a segunda do grupo e, feito isso, há três maneiras de escolher uma pessoa para ser a terceira do grupo, temos que, podemos formar $5 \times 4 \times 3 = 60$ grupos com três pessoas.
Mas, essa resposta está equivocada, a configuração que desejamos formar é um conjunto(grupo) com 3 pessoas, isto é, se os objetos de cada configuração devem ser escolhidos como um grupo, então a escolha de uma configuração deve ser feita de maneira *simultânea*. Então, não apenas um objeto, mas um grupo inteiro de objetos é escolhido ao mesmo tempo. Isso quer dizer que, por exemplo, ao montar o grupo $\{A,B,C\}$ tanto faz se, organizando cada uma pessoa sucessivamente, obtemos, por exemplo, A, e depois B, e depois C; ou se obtemos primeiro, por exemplo, C, e depois B, e depois A. Isto é, todas as filas: (A,B,C),(A,C,B),(B,A,C),(B,C,A),(C,A,B), (C,B,A) representam o mesmo grupo $\{A,B,C\}$.

Uma ação que funciona, para convencer os alunos, é o professor escolher três alunos da sala e dizer que aquele é o grupo escolhido. Depois disso, trocando os integrantes do grupo de lugar entre eles, perguntar se o grupo é o mesmo. Logo, a quantidade de filas com três das cinco pessoas NÃO é igual a quantidade de grupos com três das cinco pessoas, estamos contando grupos demais.

d) Se a resposta do item anterior for NÃO, corrija a resposta, obtendo, assim, a resposta correta.

Vamos usar a mesma estratégia que utilizamos no problema da Atividade 12. Comecemos colocando os grupos iguais em uma mesma "caixa":
[[A,B,C],[A,C,B],[B,A,C],[B,C,A],[C,A,B],[C,B,A]]
[[A,B,D],[A,D,B],[B,A,D],[B,D,A],[D,A,B],[D,B,A]]
[[A,B,E],[A,E,B],[B,A,E],[B,E,A],[E,A,B],[E,B,A]]
[[A,C,D],[A,D,C],[C,A,D],[C,D,A],[D,A,C],[D,C,A]]
[[A,C,E],[A,E,C],[C,A,E],[C,E,A],[E,A,C],[E,C,A]]
[[A,D,E],[A,E,D],[D,A,E],[D,E,A],[E,A,D],[E,D,A]]
[[B,C,D],[B,D,C],[C,B,D],[C,D,B],[D,B,C],[D,C,B]]
[[B,C,E],[B,E,C],[C,B,E],[C,E,B],[E,B,C],[E,C,B]]
[[B,D,E],[B,E,D],[D,B,E],[D,E,B],[E,B,D],[E,D,B]]
[[C,D,E],[C,E,D],[D,C,E],[D,E,C],[E,C,D],[E,D,C]]
Observemos que cada "caixa" possui seis arranjos(filas), ou seja, para cada grupo produzido com três pessoas dentre cinco pessoas consideradas, corresponde seis filas com três pessoas dentre cinco pessoas consideradas. Com isso, podemos concluir, usando o Princípio k para 1(onde $k = 6$, nesse caso), que a quantidade de grupos com três pessoas dentre as pessoas A,B,C,D,E é igual a $\frac{60}{6} = 10$.
Importante notar que as seis filas colocadas em cada "caixa" são as permutações simples das 3 pessoas que compõem um grupo.
Grupos com três pessoas dentre A,B,C,D,E:

$$[\{A,B,C\}, \{A,B,D\}, \{A,B,E\}, \{A,C,D\}, \{A,C,E\},$$

$$\{A,D,E\}, \{B,C,D\}, \{B,C,E\}, \{B,D,E\}, \{C,D,E\}]$$

.

O código da Atividade 16 tem como objetivo corrigir o excesso encontrado na Atividade 15, isto é, calcular a quantidade de grupos com três pessoas dentre as cinco pessoas A,B,C,D,E dadas.

***Atividade* 16:** Considere o código 2.14 em Python que resolve um determinado problema de contagem:

a) Execute o código acima.

Analisando o código:
A ideia por trás desse Código é a mesma que utilizamos no item d) da Atividade 15.

- **Linhas 1 a 21:** *def formar_filas(pessoas):* $\rightarrow$ O código gera todos as filas formadas por três pessoas dentre cinco pessoas dadas;
- **Linhas 23 a 40:** *def ordenar_e_imprimir_filas(filas):* $\rightarrow$ O código ordena cada uma das filas formadas por três pessoas dentre cinco pessoas dadas;
- **Linhas 42 a 74:** *def remover_duplicatas_sort(lista):* $\rightarrow$ Neste bloco, o código agrupa as filas ordenadas e iguais em uma mesma "caixa" e, depois, remove as duplicatas.

b) O que esse código fornece como resultado?

Ao executar o código, teremos:

- A função *formar_filas* gera as 60 filas formadas por três pessoas, dadas cinco pessoas;
- A função *ordenar_e_imprimir_filas* ordena as filas.
- A função *remover_duplicatas_sort* agrupa as filas iguais em "caixas"(vemos que cada "caixa" possui seis filas iguais), filtra as filas(lista os $\frac{60}{6} = 10$ grupos com três pessoas, dadas cinco pessoas).

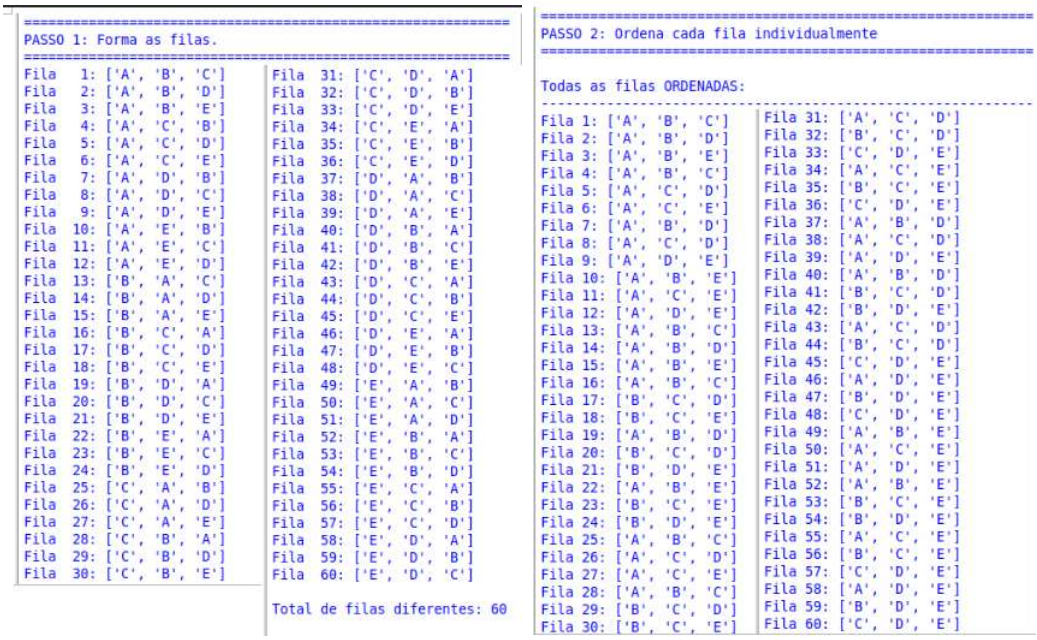

Figura 25: Saída do Código 2.14.
Fonte: Autoria própria(2026).

c) Compare-o com o Código 2.13. Qual é a diferença?

> A diferença está na presença das funções *ordenar_e_imprimir_filas* e *remover_duplicatas_sort* presentes no Código 2.14 que, a partir das filas formadas pelo Código 2.13, fornece os grupos com três pessoas escolhidas dentre cinco pessoas.

d) Esse código fornece a solução do problema da Atividade 15?

Sim. As configurações satisfazem as condições do problema.
Obtemos 10 configurações (10 grupos formados por três pessoas escolhidas dentre cinco pessoas).

e) Usando a ideia do código acima, dadas cinco pessoas quantos grupos com quatro pessoas podemos formar? E, dadas dez pessoas quantos grupos com quatro pessoas podemos formar?

- **Dadas cinco pessoas quantos grupos com quatro pessoas podemos formar?**
  - **Dadas cinco pessoas quantos filas com quatro pessoas podemos formar?**

    Os objetos básicos: São as 5 pessoas: $\{P1, P2, P3, P4, P5\}$.
    Cada configuração é uma fila com quatro dessas cinco pessoas.
    Temos uma restrição implícita no problema, uma lugar na fila para cada pessoa.
    Precisamos tomar 4 decisões: $1^{\underline{a}}$) Escolher a pessoa do início da fila; $2^{\underline{a}}$) Escolher a segunda pessoa da fila; $3^{\underline{a}}$) Escolher a terceira pessoa da fila; $4^{\underline{a}}$) Escolher a pessoa do final da fila.
    Temos, então:
    $1^{\underline{a}}$ decisão: 5 maneiras, pois antes de iniciarmos a formação da fila, temos 5 pessoas disponíveis;
    $2^{\underline{a}}$ decisão: 4 maneiras, pois após a $1^{\underline{a}}$ decisão ter sido realizada, para qualquer pessoa escolhida na $1^{\underline{a}}$ decisão , temos 4 pessoas disponíveis;
    $3^{\underline{a}}$ decisão: 3 maneiras, pois após a $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões terem sido feitas, para quaisquer pessoas escolhidas nas $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões, temos 3 pessoas disponíveis;
    $4^{\underline{a}}$ decisão: 2 maneiras, pois após a $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões terem sido feitas, para quaisquer pessoas escolhidas nas $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões, temos 2 pessoas disponíveis.
    Pelo Princípio Fundamental da Contagem, como há 5 maneiras de escolher uma pessoa para ser $1^{\underline{a}}$ pessoa da fila e, feito isso, há 4 maneiras de escolher uma pessoa para ser a $2^{\underline{a}}$ pessoa da fila e, feito isso, há 3 maneiras de escolher uma pessoa para ser a $3^{\underline{a}}$ pessoa da fila e, feito isso, há 2 maneiras de escolher uma pessoa para ser a $4^{\underline{a}}$ pessoa da fila, temos que, podemos

formar $5 \times 4 \times 3 \times 2 = 120$ filas.

– **Quantas filas "iguais" serão colocadas em cada caixa?**

Para resolver esse problema notemos, por exemplo, que 24 permutações simples das quatro pessoas $P1$,$P2$,$P3$,$P4$:

$$[P1, P2, P3, P4], [P1, P2, P4, P3], [P1, P3, P2, P4], [P1, P3, P4, P2],$$

$$[P1, P4, P2, P3], [P1, P4, P3, P2], [P2, P1, P3, P4], [P2, P1, P4, P3],$$

$$[P2, P3, P1, P4], [P2, P3, P4, P1], [P2, P4, P1, P3], [P2, P4, P3, P1],$$

$$[P3, P1, P2, P4], [P3, P1, P4, P2], [P3, P2, P1, P4], [P3, P2, P4, P1],$$

$$[P3, P4, P1, P2], [P3, P4, P2, P1], [P4, P1, P2, P3], [P4, P1, P3, P2],$$

$$[P4, P2, P1, P3], [P4, P2, P3, P1], [P4, P3, P1, P2], [P4, P3, P2, P1],$$

correspondem ao único grupo $\{P1$,$P2$,$P3$,$P4\}$.

Então, em cada caixa teremos 24 filas iguais, ou seja, a cada grupo formado por 4 pessoas dentre as 5 pessoas dadas, corresponde 24 permutações simples das mesmas 4 pessoas$(4! = 24)$.

Logo, pelo Princípio k para 1(nesse caso, $k = 4! = 24$), podemos formar $\frac{120}{24} = 5$ grupos com 4 pessoas escolhidas dentre 5 pessoas dadas.

- **Dadas dez pessoas quantos grupos com quatro pessoas podemos formar?**

  – **Dadas dez pessoas quantos filas com quatro pessoas podemos formar?**

  Os objetos básicos: São as 10 pessoas:

  $\{P1, P2, P3, P4, P5, P6, P7$,$P8, P9$,$P10\}$.

Cada configuração é uma fila com quatro dessas dez pessoas.

Temos uma restrição implícita no problema, uma lugar na fila para cada pessoa.

Precisamos tomar 4 decisões: 1ª) Escolher a pessoa do início da fila; 2ª) Escolher a segunda pessoa da fila; 3ª) Escolher a terceira pessoa da fila; 4ª) Escolher a pessoa do final da fila.

Temos, então:

1ª decisão: 10 maneiras, pois antes de iniciarmos a formação da fila, temos 5 pessoas disponíveis;

2ª decisão: 9 maneiras, pois após a 1ª decisão ter sido realizada, para qualquer pessoa escolhida na 1ª decisão , temos 9 pessoas disponíveis;

3ª decisão: 8 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª e 2ª decisões, temos 8 pessoas disponíveis;

4ª decisão: 7 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª, 2ª e 3ª decisões, temos 7 pessoas disponíveis.

Pelo Princípio Fundamental da Contagem, como há 10 maneiras de escolher uma pessoa para ser 1ª pessoa da fila e, feito isso, há 9 maneiras de escolher uma pessoa para ser a 2ª pessoa da fila e, feito isso, há 8 maneiras de escolher uma pessoa para ser a 3ª pessoa da fila e, feito isso, há 7 maneiras de escolher uma pessoa para ser a 4ª pessoa da fila, temos que, podemos formar $10 \times 9 \times 8 \times 7 = 5040$ filas.

– **Quantas filas "iguais" serão colocadas em cada caixa?**

Para resolver esse problema notemos, por exemplo, que 24 permutações simples das quatro pessoas $P1$,$P2$,$P3$,$P4$:

$$[P1, P2, P3, P4], [P1, P2, P4, P3], [P1, P3, P2, P4], [P1, P3, P4, P2],$$

$$[P1, P4, P2, P3], [P1, P4, P3, P2], [P2, P1, P3, P4], [P2, P1, P4, P3],$$

$$[P2, P3, P1, P4], [P2, P3, P4, P1], [P2, P4, P1, P3], [P2, P4, P3, P1],$$

$$[P3, P1, P2, P4], [P3, P1, P4, P2], [P3, P2, P1, P4], [P3, P2, P4, P1],$$

$$[P3, P4, P1, P2], [P3, P4, P2, P1], [P4, P1, P2, P3], [P4, P1, P3, P2],$$

$$[P4, P2, P1, P3], [P4, P2, P3, P1], [P4, P3, P1, P2], [P4, P3, P2, P1],$$

correspondem ao único grupo $\{P1, P2, P3, P4\}$.

Então, em cada caixa teremos 24 filas iguais, ou seja, a cada grupo formado por 4 pessoas dentre as 5 pessoas dadas, corresponde 24 permutações simples das mesmas 4 pessoas($4! = 24$).

Logo, pelo Princípio k para 1(nesse caso, $k = 4! = 24$), podemos formar $\frac{5040}{24} = 210$ grupos com 4 pessoas escolhidas dentre 10 pessoas dadas.

Referências

BRASIL. **Base Nacional Comum Curricular (BNCC)**. Brasília, 2018. Disponível em: https://basenacionalcomum.mec.gov.br.Acesso em: 16 jul. 2025.

CERIOLI, Márcia R.; VIANA, Petrucio. **Minicurso de Combinatória de Contagem**. II Colóquio de Matemática da Região Sul. Universidade Estadual de Londrina, PR, Abril 2012.

FERREIRA, Sidney da Silva. **A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python**. 2026. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal Fluminense, Niterói.

LINS, Romulo Campos. O Modelo dos Campos Semânticos: Estabelecimentos e Notas de Teorizações. *In:* ANGELO, Claudia Laus *et al.* (Ed.). **Modelo dos Campos Semânticos e Educação Matemática: 20 anos de História**. São Paulo: Midiograf, 2012.

LOCKWOOD, Elise; GIBSON, Bryan R. Combinatorial Tasks and Outcome Listing: Examining Productive Listing among Undergraduate Students. **Educational Studies in Mathematics**, v. 91, p. 247–270, Fevereiro 2016.

LOCKWOOD, Elise Nicole. **Student Approaches to Combinatorial Enumeration**: The Role of Set-Oriented Thinking. Jan. 2011. Tese de Doutorado – Portland State University, United States of America. Disponível em https://pdxscholar.library.pdx.edu/open_access_etds.

MORGADO, Augusto César *et al.* **Análise Combinatória e Probabilidade**. Rio de Janeiro: SBM, 2006.

SPREAFICO, Elen Viviani Pereira; SILVA, Kenia Cristina Pereira. **Livro Aberto de Matemática**: Análise Combinatória. Rio de Janeiro: IMPA-OS, 2021. Disponível em https://umlivroaberto.impa.br/producao/analise-combinatoria/. Acesso em: 16 maio 2026.