atividade_1

April 17, 2025

1. Listagem de Elementos de um Conjunto

[4]:
```
V = {'a', 'e', 'i', 'o', 'u'}
print(V)
```

```
{'e', 'u', 'a', 'o', 'i'}
```

Em Python, conjuntos são representados por elementos entre chaves {}, separados por vírgulas. Conjuntos são coleções não ordenadas e não indexadas de elementos únicos. A ordem de impressão pode não corresponder à ordem de inserção.

2. Verificação de Elementos em um Conjunto

[3]:
```
V = {'a', 'e', 'i', 'o', 'u'}
print('a' in V)  # Retorna True se 'a' está no conjunto
```

```
True
```

A verificação de pertencimento em conjuntos usa o operador in, que é altamente eficiente em Python (O(1) em média), pois os conjuntos são implementados como tabelas hash.

3. Criação de um Conjunto com Propriedades Específicas

[7]:
```
B = {x for x in range(11, 20) if x % 2 == 0}
print(B)
```

```
{16, 18, 12, 14}
```

Podemos criar conjuntos usando compreensão de conjuntos (set comprehension). A compreensão é útil quando há um padrão ou quando o conjunto é grande.

4. Comparação de Conjuntos

[9]:
```
V = {'a', 'e', 'i', 'o', 'u'}
C = {'i', 'o', 'u'}
print(C.issubset(V))  # Retorna True se todos elementos de C estão em V
# também podemos fazer dessa forma:
print(C <= V)
```

```
False
False
```

Para verificar se um conjunto é subconjunto de outro, podemos usar o método issubset() ou o operador <=. Esta operação verifica se todos os elementos do primeiro conjunto estão contidos no segundo.

5. Descrição de Conjuntos por Compreensão

[10]:
```
D = {x for x in range(1, 11) if x % 3 == 0}
print(D)
```

```
{9, 3, 6}
```

A compreensão de conjuntos (set comprehension) permite criar conjuntos de forma concisa. Neste caso, geramos números de 1 a 10 e filtramos apenas os divisíveis por 3. Note que range(1, 11) inclui 1 mas exclui 11.

6. União de Conjuntos

[11]:
```
A = {1, 2, 3}
B = {3, 4, 5}
uniao = A.union(B)
print(uniao)
```

```
{1, 2, 3, 4, 5}
```

A união de conjuntos combina todos os elementos distintos de ambos conjuntos. Em Python, podemos usar o método union() ou o operador |. A união é uma operação comutativa (A B = B A).

7. Interseção de Conjuntos

[12]:
```
A = {1, 2, 3}
B = {3, 4, 5}
intersecao = A.intersection(B)
print(intersecao)
```

```
{3}
```

A interseção retorna apenas os elementos comuns a ambos conjuntos. Podemos usar o método intersection() ou o operador &. Assim como a união, é uma operação comutativa.

8. Diferença entre Conjuntos

[14]:
```
A = {1, 2, 3}
B = {3, 4, 5}
diferenca = A.difference(B)
print(diferenca)
```

```
{1, 2}
```

A diferença de conjuntos (A - B) retorna elementos que estão em A mas não em B. Importante: não é comutativa (A - B B - A). A diferença simétrica (elementos em A ou B, mas não em ambos) pode ser obtida com A.symmetric_difference(B) ou A ^ B.

9. Diferença Simétrica entre Conjuntos

[15]:
```python
A = {1, 2, 3}
B = {3, 4, 5}
diff_simetrica = A.symmetric_difference(B)  # ou: diff_simetrica = A ^ B
print(diff_simetrica)
```

```
{1, 2, 4, 5}
```

A diferença simétrica retorna elementos que estão em apenas um dos conjuntos (união sem a interseção). É equivalente a (A - B)   (B - A) ou (A   B) - (A   B).

10. Subconjuntos e Superconjuntos

[16]:
```python
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}
print(A.issubset(B))  # ou: print(A <= B) → True
print(B.issuperset(A))  # ou: print(B >= A) → True
```

```
True
True
```

Um conjunto A é subconjunto de B se todos seus elementos estão em B. B é superconjunto de A se contém todos elementos de A. O método issubset() e operador <= são equivalentes.

11. Números Pares Maiores que 10

[17]:
```python
B = {x for x in range(12, 100, 2)}
print(B)
```

```
{12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50,
52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90,
92, 94, 96, 98}
```

Usamos range(início, fim, passo) para gerar números pares eficientemente. O início é 12 pois é o primeiro par maior que 10.

12. Números Primos Menores que 20

[18]:
```python
def is_prime(n):
    if n < 2: return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0: return False
    return True

P = {x for x in range(2, 20) if is_prime(x)}
print(P)
```

```
{2, 3, 5, 7, 11, 13, 17, 19}
```

Criamos uma função auxiliar para verificar primalidade e usamos compreensão de conjunto para filtrar os primos.

13. Números Ímpares Divisíveis por 3 até 30

[24]:
```
I = {x for x in range(3, 31) if x % 3 == 0}
print(I)
```

```
{3, 6, 9, 12, 15, 18, 21, 24, 27, 30}
```

Geramos números ímpares de 3 a 31 e filtramos os divisíveis por 3.

14. Quadrados Perfeitos Menores que 100

[25]:
```
Q = {x**2 for x in range(1, int(100**0.5)+1)}
print(Q)
```

```
{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
```

Calculamos o maior inteiro cujo quadrado é menor que 100 ($\sqrt{100} = 10$) e geramos quadrados de 1 a 9.

15. Múltiplos de 5 entre 10 e 50

[28]:
```
M = {x for x in range(15, 50, 5)}
print(M)
```

```
{35, 40, 45, 15, 20, 25, 30}
```

Usamos range com passo 5 para gerar diretamente os múltiplos. O início é 15 pois é o primeiro múltiplo de 5 maior que 10.

Slide 19 - Operação Produto Cartesiano

[29]:
```
# Definindo os conjuntos A e B
A = {1, 2}
B = {3, 4}

# Calculando o produto cartesiano A × B
AxB = {(a, b) for a in A for b in B}

# Calculando o produto cartesiano B × A
BxA = {(b, a) for b in B for a in A}

# Calculando o produto cartesiano A × A (A²)
AxA = {(a1, a2) for a1 in A for a2 in A}

# Imprimindo os resultados
print("A × B =", sorted(AxB))  # sorted apenas para ordenar a exibição
print("B × A =", sorted(BxA))
print("A × A (A²) =", sorted(AxA))
```

```
A × B = [(1, 3), (1, 4), (2, 3), (2, 4)]
B × A = [(3, 1), (3, 2), (4, 1), (4, 2)]
A × A (A²) = [(1, 1), (1, 2), (2, 1), (2, 2)]
```

Slide 27 - Relação Reflexiva

[42]:

```python
# Relação   no conjunto Z (números inteiros)
def reflexiva_menor_igual(a):
    return a <= a

print("Relação   em Z:")
print("5   5:", reflexiva_menor_igual(5))
print("-3   -3:", reflexiva_menor_igual(-3))
print("0   0:", reflexiva_menor_igual(0))

# Relação de inclusão   em uma coleção C de conjuntos
def reflexiva_inclusao(conjunto, colecao):
    return conjunto.issubset(conjunto)

C = [{1, 2}, {3, 4}]
print("\nRelação   em coleção de conjuntos:")
print("{1,2}   {1,2}:", reflexiva_inclusao({1, 2}, C))



# Relação   (perpendicularidade) em retas no plano
class Reta1:
    def __init__(self, a, b, c):  # Formato: ax + by + c = 0
        self.a = a
        self.b = b
        self.c = c

    def coeficiente_angular(self):
        if self.b == 0:
            return float('inf')  # Reta vertical
        return -self.a / self.b

def reflexiva_perpendicular(r1, r2):
    """
    Verifica se duas retas são perpendiculares
    Retorna:
        True - se são perpendiculares
        False - se não são perpendiculares ou se são a mesma reta
    """
    if r1 == r2:
        return False  # Uma reta não é perpendicular a si mesma

    m1 = r1.coeficiente_angular()
    m2 = r2.coeficiente_angular()

    # Caso uma reta seja vertical e a outra horizontal
    if (m1 == float('inf') and m2 == 0) or (m1 == 0 and m2 == float('inf')):
        return True
```

```
    # Caso geral: produto dos coeficientes angulares deve ser -1
    return abs(m1 * m2 + 1) < 1e-9  # Tolerância numérica

print("\nRelação   em retas:")
print("r   r:", reflexiva_perpendicular(Reta1(1, 1, 0), Reta1(1, 1, 0)))




# Relação || (paralelismo) em retas no plano
class Reta:
    def __init__(self, m, b):
        self.m = m  # Coeficiente angular
        self.b = b  # Coeficiente linear

def reflexiva_paralelismo(r1, r2):
    if r1 == r2:
        return True
    return r1.m == r2.m

print("\nRelação || em retas:")
print("r || r:", reflexiva_paralelismo(Reta(2, 3), Reta(2, 3)))


# Relação | (divisibilidade) em N
def reflexiva_divisibilidade(a, b):
    if a == b:
        return a % b == 0
    return False

print("\nRelação | em N:")
print("5 | 5:", reflexiva_divisibilidade(5, 5))
print("12 | 12:", reflexiva_divisibilidade(12, 12))
```

```
Relação   em Z:
5   5: True
-3   -3: True
0   0: True

Relação   em coleção de conjuntos:
{1,2}   {1,2}: True

Relação   em retas:
r   r: False

Relação || em retas:
r || r: True
```

```
Relação | em N:
5 | 5: True
12 | 12: True
```

19. Relação Simétrica:

[32]:
```
def is_simetrica(relacao):
    for (a, b) in relacao:
        if (b, a) not in relacao:
            return False
    return True

R1 = {(1,1), (1,2), (2,3), (1,3), (4,4)}
R2 = {(1,1), (1,2), (2,1), (2,2), (3,3), (4,4)}
R3 = {(1,3), (2,1)}
R4 = set()
R5 = {(1,1), (1,2), (2,1), (1,3), (3,1), (2,2), (2,3), (3,2), (3,3)}

print("R1 é simétrica?", is_simetrica(R1))
print("R2 é simétrica?", is_simetrica(R2))
print("R3 é simétrica?", is_simetrica(R3))
print("R4 é simétrica?", is_simetrica(R4))
print("R5 é simétrica?", is_simetrica(R5))
```

```
R1 é simétrica? False
R2 é simétrica? True
R3 é simétrica? False
R4 é simétrica? True
R5 é simétrica? True
```

Slide 29 - Transitividade

[34]:
```
def is_transitiva(relacao):
    for (a, b) in relacao:
        for (c, d) in relacao:
            if b == c and (a, d) not in relacao:
                return False
    return True

R1 = {(1,1), (1,2), (2,3), (1,3), (4,4)}
R2 = {(1,1), (1,2), (2,1), (2,2), (3,3), (4,4)}
R3 = {(1,3), (2,1)}
R4 = set()
A = {1, 2, 3}
R5 = {(a, b) for a in A for b in A}  # A × A

print("R1 é transitiva?", is_transitiva(R1))
print("R2 é transitiva?", is_transitiva(R2))
```

```
print("R3 é transitiva?", is_transitiva(R3))
print("R4 é transitiva?", is_transitiva(R4))
print("R5 é transitiva?", is_transitiva(R5))
```

```
R1 é transitiva? True
R2 é transitiva? True
R3 é transitiva? False
R4 é transitiva? True
R5 é transitiva? True
```

21. Relações de Equivalência:

[33]:
```
def is_reflexiva(relacao, conjunto):
    return all((a, a) in relacao for a in conjunto)

def is_simetrica(relacao):
    return all((b, a) in relacao for (a, b) in relacao)

def is_transitiva(relacao):
    return all((a, c) in relacao for (a, b) in relacao for (c, d) in relacao if␣
 ↪b == c)

def is_equivalencia(relacao, conjunto):
    return (is_reflexiva(relacao, conjunto) and
            is_simetrica(relacao) and
            is_transitiva(relacao))

# Caso 1: R = {(1,1), (2,2), (3,3), (1,2), (2,1)} em {1,2,3}
R1 = {(1,1), (2,2), (3,3), (1,2), (2,1)}
S1 = {1, 2, 3}
print("Relação 1 é de equivalência?", is_equivalencia(R1, S1))

# Caso 2: Relação "x + y é par" em N (implementada para um subconjunto finito)
def soma_par(a, b):
    return (a + b) % 2 == 0

S2 = {1, 2, 3, 4, 5}
R2 = {(a, b) for a in S2 for b in S2 if soma_par(a, b)}
print("Relação 2 é de equivalência?", is_equivalencia(R2, S2))

# Caso 3: Relação "x = y^(2n)" em {0,1}
def quadrado_par(a, b):
    if a == b: return True
    if 0 in (a, b) and 1 in (a, b): return False
    return True

S3 = {0, 1}
R3 = {(a, b) for a in S3 for b in S3 if quadrado_par(a, b)}
```

```python
print("Relação 3 é de equivalência?", is_equivalencia(R3, S3))
```

```
Relação 1 é de equivalência? True
Relação 2 é de equivalência? True
Relação 3 é de equivalência? True
```