

Programação Orientada a Objetos para Dados

Programação Orientada a Objetos e o Universo de dados

Prof. Ms. Leonardo Rocha

- Unidade de Ensino: 1
- Competência da Unidade: Compreensão sobre o paradigma de programação de orientação a objetos.
- Resumo: Programação orientada a objetos aplicada à ciência de dados.
- Palavras-chave: POO, ciência de dados, herança.
- Título da Teleaula: Programação orientada objetos e o universo de dados.
- Teleaula nº: 1

Contextualização

Conhecendo a Programação Orientada a Objetos
Ferramentas, Bibliotecas e habilidades técnicas
Estrutura de repetição e condição em Python
Biblioteca Pandas

Habilidades e competências

Habilidade x Competência

A **Habilidade** é o *saber fazer*

A **Competência** é a coordenação desse *saber fazer* que produz algum tipo de benefício ou resultado.

Exemplo de habilidade:

Trabalho em equipe

Exemplo de competência:

Coordenar equipe de desenvolvimento garantindo as entregas em tempo hábil.

Habilidade x Competência

A **Habilidade** é o *saber fazer*

A **Competência** é a coordenação desse *saber fazer* que produz algum tipo de benefício ou resultado.

Exemplo de habilidade:

Trabalho em equipe

Exemplo de competência:

Coordenar equipe de desenvolvimento garantindo as entregas em tempo hábil.

Ciência de Dados

A ciência de dados exige habilidades e competências pois lida com dados, sua análise, sistematização e conversão em informação.

Analista de dados - Programação, fundamentos em estatística, manipulação, tratamento e visualização de dados.

Ciência de Dados

Engenheiro de dados - responsável para lidar com volume grande de dados. Deve extrair diferentes tipos de dados, tratá-los e armazená-los em BDs.

Programação, manipulação de dados e engenharia de software.

Ciência de Dados

Cientista de dados - desenvolve modelos preditivos com auxílio de algoritmos e estatística para identificar tendências através da identificação de padrões.

Programação, estatística, Machine Learning.

Habilidades de ciência de dados

HABILIDADES	ANALISTA DE DADOS	ENGENHEIRO DE DADOS	CIENTISTA DE DADOS	ENGENHEIRO DE MACHINE LEARNING
PROGRAMAÇÃO	✓	✓	✓	✓
ESTATÍSTICA	✓	✗	✓	✗
MANIPULAÇÃO DE DADOS	✓	✓	✓	✗
VISUALIZAÇÃO DE DADOS	✓	✗	✓	✗
MACHINE LEARNING	✗	✗	✓	✓
ENGENHARIA DE SOFTWARE	✗	✓	✗	✓

Ferramentas tecnológicas

Perfil em rede social profissional é fonte de busca. LinkedIn é uma rede social necessária.

Além disso, ter um repositório (portfólio) com os principais projetos é uma forma de mostrar o trabalho já desenvolvido, a experiência adquirida e a qualidade técnica do que se faz. Github, Gitlab, Bitbucket e outros são exemplos.



Programação para ciência de dados

Onde entra a programação?

A programação é fundamento base, assim como manipulação de dados, para atuação na área de ciência de dados. Note que a programação é base, inclusive, para a manipulação de dados. Portanto, programar é crucial para atuar nessa área.



Paradigmas de programação

Paradigma de programação, a grosso modo, é a forma de escrita (estrutura) de um determinado código, desde que ele a aceite. Existem linguagens que aceitam mais de um paradigma de programação. Um deles, a programação orientada a objetos apresenta como uma das principais características: a reutilização de código. Isso é possível, por exemplo, pela construção de classes.

Outro exemplo é o paradigma estruturado tem como principal característica seguir sequência.



POO

A programação orientada a objetos tem como pilar a:

- 1 - Abstração
- 2 - Encapsulamento
- 3 - Herança
- 4 - Polimorfismo



Abstração

Abstração significa definir e focar no que é necessário no desenvolvimento

Sistema de notas de alunos

Dados do aluno

Características físicas do aluno?



Encapsulamento

Detalhes interno do funcionamento dos métodos de uma classe permaneça oculto para os objetos, ou seja, é possível esconder propriedades e métodos da classe, impedindo mudanças nos atributos.

Herança

Herança é o termo que representa a relação entre classes diferentes no que tange à reutilização de código. Esse conceito representa a possibilidade de criar novas classes que podem herdar atributos e métodos daquela que passa a ser conhecida como classe pai. Esse movimento de herdar permite realizar alterações que atribuem novas características.

Polimorfismo

Diz respeito à capacidade da linguagem de programação de processar objetos de formas diferentes dependendo do seu tipo de dado ou classe. A vantagem é ter métodos com o mesmo nome mas que são implementados de formas diferentes. Exemplo: Classe conexão com método `abrirConexao()` será o mesmo para as classes filhas `ConexaoOracle` e `ConexaoPostgree` mas a implementação será diferente.

Python

Rank	Language	Type	Score
1	Python	🔒 📄 📄	100.0
2	Java	🔒 📄 📄	95.4
3	C	📄 📄 📄	94.7
4	C++	📄 📄 📄	92.4
5	JavaScript	🔒	88.1
6	C#	🔒 📄 📄	82.4
7	R	📄 📄	81.7
8	Go	🔒 📄	77.7
9	HTML	🔒	75.4
10	Swift	📄 📄	70.4

Fonte: <https://spectrum.ieee.org/top-programming-languages-2021>

Como pensar a POO para ciência de Dados?

Solução

Como veremos adiante, a Programação Orientada a Objetos é crucial para a utilização na ciência de dados. Isso porque é super importante utilizar de características da Orientação a Objetos para consulta e análise de dados. Por exemplo, o elemento e escrita de código para sua reutilização não só contribui para otimização do tempo mas também para aproveitar algum código que já tenha sido escrito e outras situações que exigem tal código.

Dúvidas

Bibliotecas

Bibliotecas

Uma biblioteca de software é uma coleção de subprogramas, ou seja, funções ou métodos que são compartilhados e que já estão prontas para o uso. Elas provêm serviços a programas independentes, permitindo o compartilhamento e alteração de código e dados de forma modular.

Pandas

Muito popular e uma das mais utilizadas, oferece ao programador uma forma de trabalhar com análise e estruturas de dados de forma simplificada.



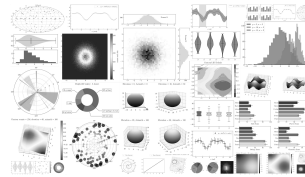
NumPy

Biblioteca que suporta processamento de multi-dimensionais arranjos e matrizes e conta com um grande conjunto de funções matemáticas e métodos para este fim.



Matplotlib

Excelente para ser utilizada no trabalho com visualização de diagramas e gráficos.



Características do Python

Características do Python

É uma linguagem com sintaxe simples (forma como o código é escrito).

A Indentação é levada a sério no Python pois é ela que determina a execução do código. Sem indentação, o código apresentará comportamentos diferentes daqueles projetados pelo programador.

Indentação

```

1 # Exemplo de como seria o código sem a indentação correta
2
3 # Importando a biblioteca random para gerar números aleatórios
4 import random
5
6 # Definindo a função para gerar um número aleatório
7 def gerar_numero():
8     return random.randint(0, 100)
9
10 # Gerando um número aleatório
11 num = gerar_numero()
12
13 # Exibindo o número gerado
14 print(f"O número gerado foi: {num}")
15
16 # Definindo a função para verificar se o número é par ou ímpar
17 def verificar_par_imp(num):
18     if num % 2 == 0:
19         return "Par"
20     else:
21         return "Ímpar"
22
23 # Verificando se o número é par ou ímpar
24 resultado = verificar_par_imp(num)
25
26 # Exibindo o resultado
27 print(f"O número {num} é {resultado}")
28
29 # Definindo a função para somar dois números
30 def somar(a, b):
31     return a + b
32
33 # Somando dois números
34 soma = somar(5, 10)
35
36 # Exibindo o resultado da soma
37 print(f"A soma de 5 e 10 é: {soma}")
38
39 # Definindo a função para calcular o fatorial de um número
40 def fatorial(n):
41     if n == 1:
42         return 1
43     else:
44         return n * fatorial(n - 1)
45
46 # Calculando o fatorial de um número
47 fatorial_resultado = fatorial(5)
48
49 # Exibindo o resultado do fatorial
50 print(f"O fatorial de 5 é: {fatorial_resultado}")
51
52 # Definindo a função para verificar se um número é primo
53 def eh_primo(n):
54     if n < 2:
55         return False
56     for i in range(2, n):
57         if n % i == 0:
58             return False
59     return True
60
61 # Verificando se um número é primo
62 primo_resultado = eh_primo(5)
63
64 # Exibindo o resultado da verificação
65 print(f"O número 5 é primo: {primo_resultado}")
66
67 # Definindo a função para calcular a média de uma lista de números
68 def calcular_media(lista):
69     return sum(lista) / len(lista)
70
71 # Calculando a média de uma lista de números
72 lista_numeros = [1, 2, 3, 4, 5]
73 media_resultado = calcular_media(lista_numeros)
74
75 # Exibindo o resultado da média
76 print(f"A média dos números da lista é: {media_resultado}")
77
78 # Definindo a função para verificar se uma string é palíndromo
79 def eh_palindromo(s):
80     return s == s[::-1]
81
82 # Verificando se uma string é palíndromo
83 palindromo_resultado = eh_palindromo("12321")
84
85 # Exibindo o resultado da verificação
86 print(f"A string '12321' é palíndromo: {palindromo_resultado}")
87
88 # Definindo a função para calcular a área de um círculo
89 def calcular_area(raio):
90     return 3.14 * raio ** 2
91
92 # Calculando a área de um círculo
93 raio = 5
94 area_resultado = calcular_area(raio)
95
96 # Exibindo o resultado da área
97 print(f"A área do círculo com raio 5 é: {area_resultado}")
98
99 # Definindo a função para verificar se um número é um número perfeito
100 def eh_numero_perfeito(n):
101     soma_divisores = 0
102     for i in range(1, n):
103         if n % i == 0:
104             soma_divisores += i
105     return soma_divisores == n
106
107 # Verificando se um número é um número perfeito
108 numero_perfeito_resultado = eh_numero_perfeito(6)
109
110 # Exibindo o resultado da verificação
111 print(f"O número 6 é um número perfeito: {numero_perfeito_resultado}")
112
113 # Definindo a função para calcular a soma dos números de 1 a n
114 def somar_1_a_n(n):
115     return n * (n + 1) // 2
116
117 # Calculando a soma dos números de 1 a n
118 n = 10
119 soma_resultado = somar_1_a_n(n)
120
121 # Exibindo o resultado da soma
122 print(f"A soma dos números de 1 a 10 é: {soma_resultado}")
123
124 # Definindo a função para verificar se um número é um número mágico
125 def eh_numero_magico(n):
126     soma_digitos = 0
127     while n > 0:
128         soma_digitos += n % 10
129         n //= 10
130     return soma_digitos == 1
131
132 # Verificando se um número é um número mágico
133 numero_magico_resultado = eh_numero_magico(1)
134
135 # Exibindo o resultado da verificação
136 print(f"O número 1 é um número mágico: {numero_magico_resultado}")
137
138 # Definindo a função para calcular a soma dos números pares de 1 a n
139 def somar_pares_1_a_n(n):
140     return n // 2 * (n // 2 + 1) * 2
141
142 # Calculando a soma dos números pares de 1 a n
143 n = 10
144 soma_pares_resultado = somar_pares_1_a_n(n)
145
146 # Exibindo o resultado da soma
147 print(f"A soma dos números pares de 1 a 10 é: {soma_pares_resultado}")
148
149 # Definindo a função para verificar se um número é um número feliz
150 def eh_numero_feliz(n):
151     soma_digitos_quadrados = 0
152     while n > 0:
153         soma_digitos_quadrados += (n % 10) ** 2
154         n //= 10
155     return soma_digitos_quadrados == 1
156
157 # Verificando se um número é um número feliz
158 numero_feliz_resultado = eh_numero_feliz(1)
159
160 # Exibindo o resultado da verificação
161 print(f"O número 1 é um número feliz: {numero_feliz_resultado}")
162
163 # Definindo a função para calcular a soma dos números ímpares de 1 a n
164 def somar_impares_1_a_n(n):
165     return n // 2 * (n // 2 + 1)
166
167 # Calculando a soma dos números ímpares de 1 a n
168 n = 10
169 soma_impares_resultado = somar_impares_1_a_n(n)
170
171 # Exibindo o resultado da soma
172 print(f"A soma dos números ímpares de 1 a 10 é: {soma_impares_resultado}")
173
174 # Definindo a função para verificar se um número é um número Armstrong
175 def eh_numero_armstrong(n):
176     soma_digitos_possencia = 0
177     num_str = str(n)
178     for i in num_str:
179         soma_digitos_possencia += int(i) ** len(num_str)
180     return soma_digitos_possencia == n
181
182 # Verificando se um número é um número Armstrong
183 numero_armstrong_resultado = eh_numero_armstrong(153)
184
185 # Exibindo o resultado da verificação
186 print(f"O número 153 é um número Armstrong: {numero_armstrong_resultado}")
187
188 # Definindo a função para calcular a soma dos números de 1 a n, mas apenas os que são divisíveis por 3
189 def somar_divisiveis_por_3_1_a_n(n):
190     soma = 0
191     for i in range(1, n + 1):
192         if i % 3 == 0:
193             soma += i
194     return soma
195
196 # Calculando a soma dos números de 1 a n, mas apenas os que são divisíveis por 3
197 n = 10
198 soma_divisiveis_resultado = somar_divisiveis_por_3_1_a_n(n)
199
200 # Exibindo o resultado da soma
201 print(f"A soma dos números de 1 a 10, apenas os divisíveis por 3, é: {soma_divisiveis_resultado}")
202
203 # Definindo a função para verificar se um número é um número de Fibonacci
204 def eh_numero_fibonacci(n):
205     a, b = 0, 1
206     while b < n:
207         a, b = b, a + b
208     return b == n
209
210 # Verificando se um número é um número de Fibonacci
211 numero_fibonacci_resultado = eh_numero_fibonacci(5)
212
213 # Exibindo o resultado da verificação
214 print(f"O número 5 é um número de Fibonacci: {numero_fibonacci_resultado}")
215
216 # Definindo a função para calcular a soma dos números de Fibonacci até n
217 def somar_fibonacci_até_n(n):
218     a, b = 0, 1
219     soma = 0
220     while b < n:
221         soma += b
222         a, b = b, a + b
223     return soma
224
225 # Calculando a soma dos números de Fibonacci até n
226 n = 10
227 soma_fibonacci_resultado = somar_fibonacci_até_n(n)
228
229 # Exibindo o resultado da soma
230 print(f"A soma dos números de Fibonacci até 10 é: {soma_fibonacci_resultado}")
231
232 # Definindo a função para verificar se um número é um número de Lucas
233 def eh_numero_lucas(n):
234     a, b = 1, 3
235     while b < n:
236         a, b = b, a + b
237     return b == n
238
239 # Verificando se um número é um número de Lucas
240 numero_lucas_resultado = eh_numero_lucas(3)
241
242 # Exibindo o resultado da verificação
243 print(f"O número 3 é um número de Lucas: {numero_lucas_resultado}")
244
245 # Definindo a função para calcular a soma dos números de Lucas até n
246 def somar_lucas_até_n(n):
247     a, b = 1, 3
248     soma = 0
249     while b < n:
250         soma += b
251         a, b = b, a + b
252     return soma
253
254 # Calculando a soma dos números de Lucas até n
255 n = 10
256 soma_lucas_resultado = somar_lucas_até_n(n)
257
258 # Exibindo o resultado da soma
259 print(f"A soma dos números de Lucas até 10 é: {soma_lucas_resultado}")
260
261 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas
262 def eh_numero_fibonacci_e_lucas(n):
263     return eh_numero_fibonacci(n) and eh_numero_lucas(n)
264
265 # Verificando se um número é um número de Fibonacci e de Lucas
266 numero_fibonacci_e_lucas_resultado = eh_numero_fibonacci_e_lucas(3)
267
268 # Exibindo o resultado da verificação
269 print(f"O número 3 é um número de Fibonacci e de Lucas: {numero_fibonacci_e_lucas_resultado}")
270
271 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas até n
272 def somar_fibonacci_e_lucas_até_n(n):
273     soma_fibonacci = somar_fibonacci_até_n(n)
274     soma_lucas = somar_lucas_até_n(n)
275     return soma_fibonacci + soma_lucas
276
277 # Calculando a soma dos números de Fibonacci e de Lucas até n
278 n = 10
279 soma_fibonacci_e_lucas_resultado = somar_fibonacci_e_lucas_até_n(n)
280
281 # Exibindo o resultado da soma
282 print(f"A soma dos números de Fibonacci e de Lucas até 10 é: {soma_fibonacci_e_lucas_resultado}")
283
284 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong
285 def eh_numero_fibonacci_e_lucas_e_armstrong(n):
286     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n)
287
288 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong
289 numero_fibonacci_e_lucas_e_armstrong_resultado = eh_numero_fibonacci_e_lucas_e_armstrong(1)
290
291 # Exibindo o resultado da verificação
292 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong: {numero_fibonacci_e_lucas_e_armstrong_resultado}")
293
294 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong até n
295 def somar_fibonacci_e_lucas_e_armstrong_até_n(n):
296     soma_fibonacci = somar_fibonacci_até_n(n)
297     soma_lucas = somar_lucas_até_n(n)
298     soma_armstrong = somar_armstrong_até_n(n)
299     return soma_fibonacci + soma_lucas + soma_armstrong
300
301 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong até n
302 n = 10
303 soma_fibonacci_e_lucas_e_armstrong_resultado = somar_fibonacci_e_lucas_e_armstrong_até_n(n)
304
305 # Exibindo o resultado da soma
306 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong até 10 é: {soma_fibonacci_e_lucas_e_armstrong_resultado}")
307
308 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz
309 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz(n):
310     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n)
311
312 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz
313 numero_fibonacci_e_lucas_e_armstrong_e_feliz_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz(1)
314
315 # Exibindo o resultado da verificação
316 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_resultado}")
317
318 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz até n
319 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_até_n(n):
320     soma_fibonacci = somar_fibonacci_até_n(n)
321     soma_lucas = somar_lucas_até_n(n)
322     soma_armstrong = somar_armstrong_até_n(n)
323     soma_feliz = somar_feliz_até_n(n)
324     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz
325
326 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz até n
327 n = 10
328 soma_fibonacci_e_lucas_e_armstrong_e_feliz_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_até_n(n)
329
330 # Exibindo o resultado da soma
331 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_resultado}")
332
333 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico
334 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico(n):
335     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n)
336
337 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico
338 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico(1)
339
340 # Exibindo o resultado da verificação
341 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_resultado}")
342
343 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico até n
344 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_até_n(n):
345     soma_fibonacci = somar_fibonacci_até_n(n)
346     soma_lucas = somar_lucas_até_n(n)
347     soma_armstrong = somar_armstrong_até_n(n)
348     soma_feliz = somar_feliz_até_n(n)
349     soma_magico = somar_magico_até_n(n)
350     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico
351
352 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico até n
353 n = 10
354 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_até_n(n)
355
356 # Exibindo o resultado da soma
357 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_resultado}")
358
359 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito
360 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito(n):
361     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n)
362
363 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito
364 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito(1)
365
366 # Exibindo o resultado da verificação
367 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_resultado}")
368
369 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito até n
370 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_até_n(n):
371     soma_fibonacci = somar_fibonacci_até_n(n)
372     soma_lucas = somar_lucas_até_n(n)
373     soma_armstrong = somar_armstrong_até_n(n)
374     soma_feliz = somar_feliz_até_n(n)
375     soma_magico = somar_magico_até_n(n)
376     soma_perfeito = somar_perfeito_até_n(n)
377     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito
378
379 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito até n
380 n = 10
381 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_até_n(n)
382
383 # Exibindo o resultado da soma
384 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_resultado}")
385
386 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz
387 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz(n):
388     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n)
389
390 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz
391 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz(1)
392
393 # Exibindo o resultado da verificação
394 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_resultado}")
395
396 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz até n
397 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_até_n(n):
398     soma_fibonacci = somar_fibonacci_até_n(n)
399     soma_lucas = somar_lucas_até_n(n)
400     soma_armstrong = somar_armstrong_até_n(n)
401     soma_feliz = somar_feliz_até_n(n)
402     soma_magico = somar_magico_até_n(n)
403     soma_perfeito = somar_perfeito_até_n(n)
404     soma_feliz_2 = somar_feliz_até_n(n)
405     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito + soma_feliz_2
406
407 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz até n
408 n = 10
409 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_até_n(n)
410
411 # Exibindo o resultado da soma
412 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_resultado}")
413
414 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico
415 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico(n):
416     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n)
417
418 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico
419 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico(1)
420
421 # Exibindo o resultado da verificação
422 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado}")
423
424 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico até n
425 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_até_n(n):
426     soma_fibonacci = somar_fibonacci_até_n(n)
427     soma_lucas = somar_lucas_até_n(n)
428     soma_armstrong = somar_armstrong_até_n(n)
429     soma_feliz = somar_feliz_até_n(n)
430     soma_magico = somar_magico_até_n(n)
431     soma_perfeito = somar_perfeito_até_n(n)
432     soma_feliz_2 = somar_feliz_até_n(n)
433     soma_magico_2 = somar_magico_até_n(n)
434     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito + soma_feliz_2 + soma_magico_2
435
436 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico até n
437 n = 10
438 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_até_n(n)
439
440 # Exibindo o resultado da soma
441 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado}")
442
443 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito
444 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito(n):
445     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n)
446
447 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito
448 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito(1)
449
450 # Exibindo o resultado da verificação
451 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado}")
452
453 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito até n
454 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_até_n(n):
455     soma_fibonacci = somar_fibonacci_até_n(n)
456     soma_lucas = somar_lucas_até_n(n)
457     soma_armstrong = somar_armstrong_até_n(n)
458     soma_feliz = somar_feliz_até_n(n)
459     soma_magico = somar_magico_até_n(n)
460     soma_perfeito = somar_perfeito_até_n(n)
461     soma_feliz_2 = somar_feliz_até_n(n)
462     soma_magico_2 = somar_magico_até_n(n)
463     soma_perfeito_2 = somar_perfeito_até_n(n)
464     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito + soma_feliz_2 + soma_magico_2 + soma_perfeito_2
465
466 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito até n
467 n = 10
468 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_até_n(n)
469
470 # Exibindo o resultado da soma
471 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado}")
472
473 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz
474 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz(n):
475     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n)
476
477 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz
478 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz(1)
479
480 # Exibindo o resultado da verificação
481 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_resultado}")
482
483 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz até n
484 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_até_n(n):
485     soma_fibonacci = somar_fibonacci_até_n(n)
486     soma_lucas = somar_lucas_até_n(n)
487     soma_armstrong = somar_armstrong_até_n(n)
488     soma_feliz = somar_feliz_até_n(n)
489     soma_magico = somar_magico_até_n(n)
490     soma_perfeito = somar_perfeito_até_n(n)
491     soma_feliz_2 = somar_feliz_até_n(n)
492     soma_magico_2 = somar_magico_até_n(n)
493     soma_perfeito_2 = somar_perfeito_até_n(n)
494     soma_feliz_3 = somar_feliz_até_n(n)
495     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito + soma_feliz_2 + soma_magico_2 + soma_perfeito_2 + soma_feliz_3
496
497 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz até n
498 n = 10
499 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_até_n(n)
500
501 # Exibindo o resultado da soma
502 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_resultado}")
503
504 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico
505 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico(n):
506     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n)
507
508 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico
509 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico(1)
510
511 # Exibindo o resultado da verificação
512 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado}")
513
514 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico até n
515 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_até_n(n):
516     soma_fibonacci = somar_fibonacci_até_n(n)
517     soma_lucas = somar_lucas_até_n(n)
518     soma_armstrong = somar_armstrong_até_n(n)
519     soma_feliz = somar_feliz_até_n(n)
520     soma_magico = somar_magico_até_n(n)
521     soma_perfeito = somar_perfeito_até_n(n)
522     soma_feliz_2 = somar_feliz_até_n(n)
523     soma_magico_2 = somar_magico_até_n(n)
524     soma_perfeito_2 = somar_perfeito_até_n(n)
525     soma_feliz_3 = somar_feliz_até_n(n)
526     soma_magico_3 = somar_magico_até_n(n)
527     soma_perfeito_3 = somar_perfeito_até_n(n)
528     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito + soma_feliz_2 + soma_magico_2 + soma_perfeito_2 + soma_feliz_3 + soma_magico_3 + soma_perfeito_3
529
530 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico até n
531 n = 10
532 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_até_n(n)
533
534 # Exibindo o resultado da soma
535 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_resultado}")
536
537 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito
538 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito(n):
539     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n)
540
541 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito
542 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito(1)
543
544 # Exibindo o resultado da verificação
545 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado}")
546
547 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito até n
548 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_até_n(n):
549     soma_fibonacci = somar_fibonacci_até_n(n)
550     soma_lucas = somar_lucas_até_n(n)
551     soma_armstrong = somar_armstrong_até_n(n)
552     soma_feliz = somar_feliz_até_n(n)
553     soma_magico = somar_magico_até_n(n)
554     soma_perfeito = somar_perfeito_até_n(n)
555     soma_feliz_2 = somar_feliz_até_n(n)
556     soma_magico_2 = somar_magico_até_n(n)
557     soma_perfeito_2 = somar_perfeito_até_n(n)
558     soma_feliz_3 = somar_feliz_até_n(n)
559     soma_magico_3 = somar_magico_até_n(n)
560     soma_perfeito_3 = somar_perfeito_até_n(n)
561     soma_feliz_4 = somar_feliz_até_n(n)
562     soma_magico_4 = somar_magico_até_n(n)
563     soma_perfeito_4 = somar_perfeito_até_n(n)
564     return soma_fibonacci + soma_lucas + soma_armstrong + soma_feliz + soma_magico + soma_perfeito + soma_feliz_2 + soma_magico_2 + soma_perfeito_2 + soma_feliz_3 + soma_magico_3 + soma_perfeito_3 + soma_feliz_4 + soma_magico_4 + soma_perfeito_4
565
566 # Calculando a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito até n
567 n = 10
568 soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado = somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_até_n(n)
569
570 # Exibindo o resultado da soma
571 print(f"A soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito até 10 é: {soma_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_resultado}")
572
573 # Definindo a função para verificar se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz
574 def eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz(n):
575     return eh_numero_fibonacci(n) and eh_numero_lucas(n) and eh_numero_armstrong(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n) and eh_numero_magico(n) and eh_numero_perfeito(n) and eh_numero_feliz(n)
576
577 # Verificando se um número é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz
578 numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_resultado = eh_numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz(1)
579
580 # Exibindo o resultado da verificação
581 print(f"O número 1 é um número de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz: {numero_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_resultado}")
582
583 # Definindo a função para calcular a soma dos números de Fibonacci e de Lucas e de Armstrong e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz e de mágico e de perfeito e de feliz até n
584 def somar_fibonacci_e_lucas_e_armstrong_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_e_magico_e_perfeito_e_feliz_até_n(n):
585     soma_fibonacci = somar_fibonacci_até_n(n)
586     soma_lucas = somar_lucas_até_n(n)
587     soma_armstrong = somar_armstrong_até_n(n)
588     soma_feliz = somar_feliz_até_n(n)
589     soma_magico = somar_magico_até_n(n)
590     soma_perfeito = somar_perfeito_até_n(n)
591     soma_feliz_2 = somar_feliz_até_n(n)
592     soma_magico_2 = somar_magico_até_n(n)
593     soma_perfeito_2 = somar_perfeito_até_n(n)
594     soma_feliz_3 = somar_feliz_até_n(n)
595     soma_magico_3 = somar_magico_até_n(n)
596     soma_perfeito_3 = somar_perfeito_até_n(n)
597     soma_feliz_4 = somar_feliz_até_n(n)
598     soma_magico_4 = somar_magico_até_n(n)
599     soma_perfeito_4 = somar_perfeito_até_n(n)
600     soma_feliz_5 = somar_feliz_até_n(n)
601     soma_magico_5 = somar_magico_até_n(n)
602     soma_perfeito_5 = somar_perfeito_até_n(n)
603     return soma_fibonacci + soma_lucas + soma_armstrong + soma_f
```

Prática com Idle

Listas
Tupla
Dicionário

Estruturas no Trinket

Condição
Repetição
Try e Except

Instalação do pandas

Para instalar o pandas, existe uma documentação que mostra os passos a serem seguidos. Na documentação é possível ver como realizar a instalação no Windows e no Linux.

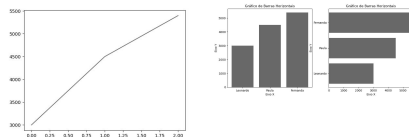
Fonte: https://pandas.pydata.org/docs/getting_started/install.html

Criando dataframe com dicionário

```
>>> import pandas as pd
>>> ds = pd.read_excel(r"~\Documentos\Profissional\Kroton\Disciplinas\Programacao Orientada\objetos\Parabados\exemplos\pandas\dados1.xlsx")
>>> ds
   cd_pedido  valor_venda  quantidade_de_itens
0           1      2500.0              5
1           2       100.0              8
2           3      9000.0              4
3           4       500.0              2
4           5     24500.0              2
5           6       400.0             15
6           7      6000.0             90
7           8     15000.0             10
8           9      9000.0             10
9          10      6000.0              5
10         11      100.0             34
>>> ds["valor_venda"].sum()
12000.0
>>> valor_vendas={"vendedor":["Leonardo", "Paula", "Fernanda"], "valor":[3000,4500,5400]}
>>> df = pd.DataFrame(valor_vendas)
>>> df
   vendedor  valor
0  Leonardo   3000
1   Paula    4500
2  Fernanda   5400
>>> |
```

Plotando gráficos

A plotagem de gráficos é feita pela biblioteca Matplotlib.



Documentação oficial Pandas

A documentação oficial está repleta de informações e exemplos para prática e compreensão da biblioteca Pandas e sua adoção na análise de dados. Vamos conhecer a documentação:

https://pandas.pydata.org/docs/getting_started/

Como criar código em Python para escrita sequência de Fibonacci?

Solução

```
#####  
#Exercício da SPI da primeira aula  
#Construir código para escrita da sequência Fibonacci #  
#Autor: Leonardo Rocha  
#####  
print("ALGORITMO PARA ESCRITA DE SEQUÊNCIA FIBONACCI")  
qtde=int(input("Informe quantos numeros Fibonacci você deseja: "))  
valor1 = 1  
valor2 = 1  
sequencia = [1,1]  
for count in range(2,qtde):  
    valor = sequencia[count -1] + sequencia[count - 2]  
    sequencia.append(valor)  
  
print(sequencia)  
#ou imprimindo com quebra de linha  
#(print(numero) for numero in sequencia)
```

Dúvidas

Recapitulando

Habilidades e Competências
Programação orientada a objetos aplicada à ciência de dados
Bibliotecas em Python
estrutura de código em Python