Para os exercícios 1, 2 e 3 será utilizado o código fornecido no arquivo main.js.

1. Qual a saída do algoritmo?

A saída do algoritmo é a listagem dos primeiros 1000 números primos, organizados em páginas contendo 200 números cada (50 linhas por página e 4 colunas por linha).

2. Você julga que este código é limpo? Aponte quais erros o programador cometeu que prejudicaram a qualidade do código. Obs: não existe nenhum bug escondido no código.

O código não é limpo. Os principais problemas são:

1. Variáveis como M, RR, CC, ITIS não explicam sua finalidade.
2. Toda a lógica está em um único método estático.
3. Lógica confusa, mistura de while/do-while e variáveis reatribuídas (ex: J, K).
4. Falta de separação de responsabilidades.
5. Índices inconsistentes, uso de arrays com índice 1 (P[1] = 2).

3. Refatore o código do arquivo utilizando conceitos de Clean Code, de maneira que o código se torne mais limpo, legível e de fácil manutenção.

```
class PrimeNumberPrinter {

 static MAX_PRIMES = 1000;

 static ROWS_PER_PAGE = 50;

 static COLUMNS_PER_PAGE = 4;

 static ORD_MAX = 30;


 static generatePrimes() {

  const primes = new Array(this.MAX_PRIMES + 1).fill(0);

  const multipliers = new Array(this.ORD_MAX + 1).fill(0);

  primes[1] = 2;

  let currentOrd = 2;

  let nextSquare = 9;

  let primeCount = 1;
```

```
  for (let candidate = 3; primeCount < this.MAX_PRIMES; candidate += 2) {
    if (candidate === nextSquare) {
      currentOrd++;
      nextSquare = primes[currentOrd] ** 2;
      multipliers[currentOrd - 1] = candidate;
    }

    let isPrime = true;
    for (let n = 2; n < currentOrd && isPrime; n++) {
      while (multipliers[n] < candidate) {
        multipliers[n] += primes[n] * 2;
      }
      if (multipliers[n] === candidate) isPrime = false;
    }

    if (isPrime) {
      primeCount++;
      primes[primeCount] = candidate;
    }
  }

  return primes;
}

static formatPrimesPage(primes, pageOffset, rowsPerPage) {
  const pageLines = [];
  for (let row = 0; row < rowsPerPage; row++) {
    const lineValues = [];
```

```
    for (let col = 0; col < this.COLUMNS_PER_PAGE; col++) {
      const index = pageOffset + row + col * rowsPerPage;
      if (index <= this.MAX_PRIMES) lineValues.push(primes[index]);
    }
    pageLines.push(lineValues.join('|'));
  }
  return pageLines;
 }

 static printPrimes() {
  const primes = this.generatePrimes();
  let pageNumber = 1;
  let pageOffset = 1;

  while (pageOffset <= this.MAX_PRIMES) {
    console.log(` Page ${pageNumber}` );
    const page = this.formatPrimesPage(primes, pageOffset,
this.ROWS_PER_PAGE);
    page.forEach(line => console.log(line));
    pageNumber++;
    pageOffset += this.ROWS_PER_PAGE * this.COLUMNS_PER_PAGE;
  }
 }
}

PrimeNumberPrinter.printPrimes();
```

4. Explique como o conceito de middlewares no Express.js pode ser utilizado para evitar repetição de código.

Middlewares no Express.js são funções intermediárias que interceptam requisições e respostas, permitindo a reutilização de lógica comum em múltiplas rotas. Por exemplo, ao implementar autenticação, logging ou validação de dados, você pode encapsular essas operações em um middleware e aplicá-lo apenas às rotas necessárias. Isso elimina a duplicação de código e centraliza a lógica.

5. Tendo em vista duas abordagens de backend: uma utilizando um ORM (como Prisma e Sequelize) e outra utilizando apenas um query builder (como o Knex), quais as vantagens e desvantagens de cada abordagem?

ORMs (Prisma/Sequelize) abstraem o banco em objetos JavaScript, automatizando operações CRUD, relacionamentos e migrações, ideal para projetos que priorizam produtividade. Porém, podem gerar sobrecarga em consultas complexas e exigem adaptação à sua estrutura. Query Builders (Knex) operam próximo ao SQL, permitindo construir queries programaticamente com flexibilidade para otimizações manuais, porém demandam conhecimento de SQL e mais código repetitivo.

6. Faça uma query em SQL que traga em cada linha o nome de jogadores que se enfrentaram mais de duas vezes, onde em cada partida a soma dos pontos foi maior que 30 e a duração do jogo foi maior que 90 minutos. Não podem haver resultados repetidos.

```
SELECT DISTINCT
        CASE WHEN j1.id < j2.id THEN j1.nome ELSE j2.nome END AS jogador1,
 CASE WHEN j1.id < j2.id THEN j2.nome ELSE j1.nome END AS jogador2
FROM Jogador j1
JOIN Partidas p1 ON j1.id = p1.jogador1_id
JOIN Jogador j2 ON j2.id = p1.jogador2_id
JOIN Partidas p2 ON (j1.id = p2.jogador1_id AND j2.id = p2.jogador2_id) OR (j1.id = p2.jogador2_id AND j2.id = p2.jogador1_id)
WHERE (p1.pontos_jogador1 + p1.pontos_jogador2) > 30
AND p1.duracao > 90
GROUP BY jogador1, jogador2
HAVING COUNT(*) > 2;
```