1. Faça um laço de repetição que comece a contar a partir do número 1 e que vá incrementando de 2 em 2 até chegar no número 9. Exemplo de saída que este sistema irá gerar quando executá-lo:

```
1 3 5 7 9
```

2. Elabore um programa que leia um número que o usuário digitar. Dependendo do número informado, das frases abaixo, o sistema imprimirá somente as que forem verdadeiras.
   - `O número é menor que 10.`
   - `O número é par.`
   - `O número está entre 8 e 16.`
   - `O número é 51 ou 80.`

   Por exemplo, se o usuário digitar o número “12”, o programa irá imprimir:

   ```
   O número é par.
   O número está entre 8 e 16.
   ```

   Se o usuário digitar o número “51”, então será impresso:

   ```
   O número é 51 ou 80.
   ```

   Ou, se o usuário digitar “101”, então o programa não imprime nada.

3. Na programação é comum relacionar dados usando um campo que identifica registros (**id**). Elabore uma modelagem de dados em que os registros estão relacionados através de um campo identificador (o **id),** atendendo as seguintes afirmações:
   - O bairro Betânia é da cidade Diamantina.
   - Os bairros Agostinho e Natal são da cidade Noronha.
   - A cidade Diamantina é do estado Goiás.
   - A cidade Noronha é do estado Paraná.

   Siga o exemplo desta outra modelagem já implementada:
   - Afirmações:
     - O produto Frango é da empresa Açougue.
     - O produto Pão é da empresa Padaria.
     - O produto Leite é da empresa Padaria.
   - Modelagem resultante (os *ids* foram gerados aleatoriamente):

   ```
   empresas = [{id:44, nome:'Açougue'},
               {id:71, nome:'Padaria'}]
   ```

```
produtos = [{id:83, nome:'Pão', empresa_id:71},
            {id:84, nome:'Leite', empresa_id:71},
            {id:85, nome:'Frango', empresa_id:44}]
```

4. Crie duas funções:
    - A primeira função receberá dois parâmetros e retornará como resultado uma concatenação de texto colocando uma vírgula entre os dois parâmetros ao uní-los.
    - A segunda função não receberá parâmetros; será feito a leitura de duas entradas que o usuário digitar; irá chamar a primeira função passando como argumento os dois textos lidos; por fim esta segunda função irá imprimir para o usuário informando qual foi o resultado que se obteve na chamada à primeira função.
    - Exemplo da primeira entrada: "Olá". Exemplo da segunda entrada: "Mundo". Exemplo da saída do sistema: "Olá,Mundo".

5. Escreva a função obter_colecao_mongodb(url_conexao, colecao) que irá se conectar no MogodDB utilizando alguma biblioteca do Python. Ela possui os seguintes parâmetros:
    - url_conexao: URI de conexão com banco de dados MongoDB, que também informa a base de dados (database). Por exemplo: a URI `mongodb://localhost/bancoexemplo', é a string de conexão para o banco "bancoexemplo" da minha máquina local ("localhost").
    - colecao: É o nome da coleção (collection) do MongoDB que estaremos acessando com esta função.

    Tempo estimado: 6 minutos. Dificuldade: quase média, a pessoa precisará lembrar como pegar o database ou da URI ou da própria biblioteca.

6. O DBA da empresa criou um script para fazer uma atualização no banco de dados MongoDB:

```
var sku = "" // a pessoa informa o sku aqui
var estoque = 0 // valor a ser informado do novo estoque

db.produto.update(
    {
        sku: sku
    },
    {
        $inc: estoque
    }
)
```

    O problema que esse script está em JavaScript do MongoDB. Logo, escreva para nós a função Python **ajustar_estoque()** com o seus devidos parâmetros para que realize a atualização na coleção produto conforme o script que o DBA passou para

nós.

Tempo estimado: 10 minutos. Dificuldade: média. A pessoa sabe ler o script do MongoDB? Consegue ver e traduzir o que precisamos para o Python?

7. Crie uma API Rest em Python, que responda à seguinte chamada:

   ```
   curl -X POST 'http://localhost:5000/soma' -H 'Content-type: application/json' -d '{"x": 1, "y"; 2}'
   ```

   A API /soma irá receber o valor **x** e somar com o valor **y** e retorná-lo em JSON no seguinte formato:

   ```
   {
       "resultado": <valor do resultado>
   }
   ```

   Para o exemplo, acima iremos retornar:

   ```
   {
       "resultado": 3
   }
   ```

   Os valores de entrada, **x** e **y** são obrigatórios e devem ser números.

   Tempo estimado: 6-8 minutos. Dificuldade: Fácil.

8. Crie um programa que seja uma API de um contador de números. Esta API irá manter o número em sua memória, e esta irá iniciar com o valor 0 (zero).
   Para o programa teremos as seguinte chamadas:

   - `POST /contador`

     ```
     {
         "numero": <numero>
     }
     ```

     Irá definir um novo número para o nosso contador. O método irá retornar um HTTP 201.

   - `GET /contador`

     Retorna o número que foi guardado na memória, em um JSON no formato:

     ```
     {
         "numero": <numero guardado>
     }
     ```

     E também um HTTP 200.

   - PUT /contador/incrementa

Irá incrementar o valor do número em 1 e retornar um HTTP 202.

- DELETE /contador

  Irá zerar o valor do contador. A API irá retornar um HTTP 202.

Seguem algumas chamadas que fizemos no terminal, para mostrar o comportamento de nosso programa, considere que iniciamos o programa agora e executamos os testes na ordem apresentada:

1) `curl -X GET 'http://localhost:5000/contador'`

   Retornou:
   `{ "numero": 0}`

2) `curl -X POST 'http://localhost:5000/contador' -d '{"numero": 123}'`

   Retornou um HTTP 201.

3) `curl -X GET 'http://localhost:5000/contador'`

   Retornou:
   `{ "numero": 123}`

4) `curl -X PUT 'http://localhost:5000/contador/incrementa'`

   Retornou um HTTP 202.

5) `curl -X GET 'http://localhost:5000/contador'`

   Retornou:
   `{ "numero": 124}`

6) `curl -X DELETE 'http://localhost:5000/contador'`

   Retornou um HTTP 202.

7) `curl -X GET 'http://localhost:5000/contador'`

   Retornou:
   `{ "numero": 0}`

Dificuldade: Média. Tempo: 9-17 minutos. Tempo para pensar no número na memória, e fazer cada método.

Link para envio do GitHub com as respostas da prova