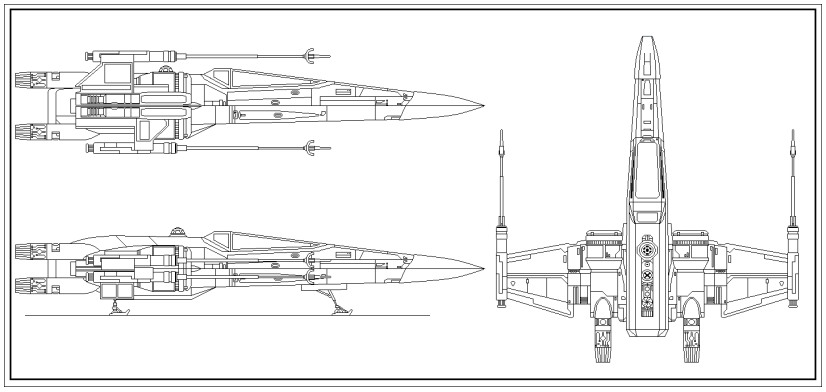
MC102 - Algoritmos e Programação de Computadores

Lab 06

Data da Primeira Chance: 4 de maio de 2023
Peso: 2

A Nova República precisa da sua ajuda, programador! A nova geração de caças X-Wing T-70 está sendo produzida, mas o computador de bordo da nave necessita de um programa para cálculo de vetores de hiperespaço, para ajudar o droide astromecânico a calcular o salto para a velocidade da luz. Os caças já estão sendo produzidos, mas sem este programa não é possível viajar para outros planetas.



Fonte https://www.deviantart.com/wingzero-01-custom/art/T-70-X-Wing-Space-Superiority-Fighter-2-0-Line-Art-580239780

Para este programa serão necessárias algumas operações com vetores, e como a física do hiperespaço é muito complexa, podem ser necessárias operações com vetores de tamanhos diferentes. Será fornecido um vetor inicial e um conjunto de cálculos para serem realizados até um vetor final:

- **soma_vetores**: soma elemento a elemento de dois vetores. Caso os vetores tenham tamanhos diferentes, deve-se adicionar 0s (zeros) ao final do vetor menor.
    - $V1 = [1, 2, 3]$ e $V2 = [2, 3]$
    - $Resultado = [3, 5, 3]$
- **subtrai_vetores**: subtração elemento a elemento de dois vetores ($V1 - V2$). Caso os vetores tenham tamanhos diferentes, deve-se adicionar 0s (zeros) ao final do vetor menor.
    - $V1 = [1, 2]$ e $V2 = [2, 3, -1]$
    - $Resultado = [-1, -1, 1]$

- **multiplica_vetores**: multiplicação elemento a elemento de dois vetores. Caso os vetores tenham tamanhos diferentes, deve-se adicionar 1s (uns) ao final do vetor menor.
    - $V1 = [1, 2]$ e $V2 = [2, 3, -1]$
    - $Resultado = [2, 6, -1]$
- **divide_vetores**: divisão inteira (*//*) elemento a elemento de dois vetores. Caso os vetores tenham tamanhos diferentes, deve-se adicionar 0s (zeros) se o primeiro vetor for menor, ou 1s (uns) se o segundo vetor for menor.
    - $V1 = [4, 7]$ e $V2 = [2, 3, -1]$
    - $Resultado = [2, 2, 0]$
- **multiplicacao_escalar**: multiplica cada elemento por um escalar.
    - $V = [1, 2, 3]$ e $E = 4$
    - $Resultado = [4, 8, 12]$
- **n_duplicacao**: duplica o vetor $N \geq 0$ vezes. Se for 0 (zero) retorna uma lista vazia ([]).
    - $V = [1, -2]$ e $N = 3$
    - $Resultado = [1, -2, 1, -2, 1, -2]$
- **soma_elementos**: soma todos os elementos do vetor.
    - $V = [1, -2,3,10]$
    - $Resultado = 12$
- **produto_interno**: multiplica elemento a elemento e depois soma o resultado. Caso os vetores tenham tamanhos diferentes, deve-se adicionar 1s (uns) ao final do vetor menor.
    - $V1 = [1, 2, 2]$ e $V2 = [2, 3]$
    - $Resultado = \langle V1, V2 \rangle = 1 * 2 + 2 * 3 + 2 * 1 = 10$
- **multiplica_todos**: multiplica cada elemento do primeiro vetor por todos os elementos do segundo vetor e soma o resultado.
    - $V1 = [1, 2]$ e $V2 = [2, 3]$
    - $Resultado = [1 * 2 + 1 * 3, 2 * 2 + 2 * 3] = [5, 10]$
- **correlacao_cruzada**: esta operação é muito utilizada em redes neurais e em processamento de imagens. Ela consiste em utilizar uma máscara (um vetor menor), que caminha pelo vetor calculando um produto interno. A operação de correlação cruzada segue a fórmula a seguir.

$$CC[i] = \sum_{j=0}^{k} V[i+j]M[j]$$

  Neste caso, k é o tamanho da máscara, n é o tamanho do vetor, e a operação se repete para todo i de 0 até $n - k + 1$. Note que o vetor resultante tem um tamanho menor que a entrada.
    - $V = [3, -2, 3, -10, 8]$ e $M = [-1, 2, -1]$
    - $Resultado = [3 * (-1) + (-2) * 2 + 3 * (-1),$
      $(-2) * (-1) + 3 * 2 + (-10) * (-1),$
      $3 * (-1) + (-10) * 2 + 8 * (-1)] = [-10, 18, -31]$

Como simplificação, neste laboratório usaremos somente inteiros na operação de divisão; mais informações na seção de Regras e Avaliação.

Abaixo estão todas as funções que precisarão ser implementadas:

Funções	**Parâmetros**	**Saída**
`soma_vetores`	`vetor1:list[int]` `vetor2:list[int]`	`list[int]`
`subtrai_vetores`	`vetor1:list[int]` `vetor2:list[int]`	`list[int]`
`multiplica_vetores`	`vetor1:list[int]` `vetor2:list[int]`	`list[int]`
`divide_vetores`	`vetor1:list[int]` `vetor2:list[int]`	`list[int]`
`multiplicacao_escalar`	`vetor:list[int]` `escalar:int`	`list[int]`
`n_duplicacao`	`vetor:list[int] n:int`	`list[int]`
`soma_elementos`	`vetor:list[int]`	`int`
`produto_interno`	`vetor1:list[int]` `vetor2:list[int]`	`int`
`multiplica_todos`	`vetor1:list[int]` `vetor2:list[int]`	`list[int]`
`correlacao_cruzada`	`vetor:list[int]` `mascara:list[int]`	`list[int]`

Importante: cuidado com as operações que retornam números inteiros **soma_elementos** e **produto_interno**. O programa receberá uma lista de comandos para serem executados, ou seja, as funções acima, e elas sempre recebem pelo menos um vetor como parâmetro. Desta forma, é preciso transformar o resultado destas duas funções de um **inteiro** para um **vetor com uma única dimensão**. Assim, o próximo comando que o programa receber já poderá operar normalmente com o resultado já que ele será um vetor, por exemplo:

vetor_corrente = [soma_elementos(vetor_corrente)]

Importante 2: um código tem que ser bem escrito, bem documentado e bem formatado, de forma que seja fácil para outros desenvolvedores entenderem a lógica implementada. Além disso, testes automatizados são importantes para garantir que o código seja o mais correto possível. Desta forma, serão utilizadas ferramentas para garantir que nosso programa esteja bem formatado e correto para fazer os cálculos da nave. Por sorte, outro programador da Nova

República já implementou testes automatizados para você rodar, para garantir que tudo esteja funcionando.

Entrada

A entrada do programa consiste em uma lista de números **inteiros** separados por vírgula, representando o vetor inicial, e uma sequência de comandos a serem executados a partir deste vetor. Cada operação será representada por uma **string** na primeira linha, e o parâmetro (vetor2, escalar, n) será passado na linha seguinte. O seu programa deve receber os comandos e realizar as operações até receber a string **fim**.

Na leitura da entrada, os vetores fornecidos sempre terão tamanho pelo menos 1, mas as funções devem ser implementadas para tratar vetores vazios (**[]**) na entrada. Para a operação **divide_vetores**, você pode supor que não serão fornecidos vetores com o valor 0 (zero). Para a operação de **correlacao_cruzada**, você pode supor que a **máscara** sempre terá um tamanho menor ou igual ao vetor corrente.

Saída

Seu programa deve imprimir o vetor corrente após cada operação executada. Somente não deve ser impresso o vetor após o recebimento do comando **fim**.

Exemplos

Exemplo 1:

Entrada

```
2,-3,4
soma_vetores
-1,2,3,4
subtrai_vetores
-1,2,3
multiplica_vetores
1,0,-2
divide_vetores
-1,-1,3,3,1
fim
```

Saída

```
[1, -1, 7, 4]
[2, -3, 4, 4]
[2, 0, -8, 4]
[-2, 0, -3, 1, 0]
```

Exemplo 2:

Entrada

```
1,-1
n_duplicacao
3
soma_elementos
produto_interno
3,-3
fim
```

Saída

```
[1, -1, 1, -1, 1, -1]
[0]
[-3]
```

Exemplo 3:

Entrada

```
1,2,-3
multiplica_todos
-1,2
correlacao_cruzada
1,-1
fim
```

Saída

```
[1, 2, -3]
[-1, 5]
```

Ferramentas de Qualidade de Código

Como critérios de qualidade, serão utilizadas as ferramentas **Flake8** (formatação de código), **Mypy** (checagem de tipos) e **Pytest** (testes automatizados). Para instalá-los localmente, é necessário utilizar o **pip** através de um dos seguintes comandos, dependendo de como o Python está instalado na sua máquina:

```
pip install flake8 mypy pytest
// ou
pip3 install flake8 mypy pytest
```

```
// ou
python -m pip install flake8 mypy pytest
// ou
python3 -m pip install flake8 mypy pytest
// ou
python3.10 -m pip install flake8 mypy pytest
```

Para rodar o Flake8, passe o arquivo de código:

```
flake8 lab06.py
// ou
python -m flake8 lab06.py
// ou
python3 -m flake8 lab06.py
// ou
python3.10 -m flake8 lab06.py
```

No caso do Mypy, será utilizada a flag --strict para que sempre seja feita a checagem de tipos das funções:

```
mypy --strict lab06.py
// ou
python -m mypy --strict lab06.py
// ou
python3 -m mypy --strict lab06.py
// ou
python3.10 -m mypy --strict lab06.py
```

No Pytest, é necessário passar como parâmetro um dos códigos de teste fornecidos, ou não passar **nenhum arquivo** para rodar todos os testes:

```
pytest test_correlacao_cruzada.py
// ou
python -m pytest test_correlacao_cruzada.py
// ou
python3 -m pytest test_correlacao_cruzada.py
// ou
python3.10 -m pytest test_correlacao_cruzada.py
// ou para rodar todos os testes
python3.10 -m pytest
```

Sobre o Pytest, o código lab06.py precisa estar no mesmo repositório dos códigos "test_*.py". Além disso, se você estiver tendo problemas de input no momento de rodar os testes, pode ser necessário definir uma função ***main*** no seu código principal.

Regras e Avaliação

Neste laboratório, seu código-fonte será analisado e deve conter as funções mencionadas acima: **soma_vetores**, **subtrai_vetores**, **multiplica_vetores**, **divide_vetores**, **multiplicacao_escalar**, **n_duplicacao**, **soma_elementos**, **produto_interno**, **multiplica_todos** e **correlacao_cruzada**. Além destas funções, outras também podem ser definidas se necessário, para ajudar a organização e a não duplicação de código.

Além disso, você não pode usar bibliotecas (isto é, o comando *import*), exceto para a parte de tipos da biblioteca *typing*, se necessário. Também, está proibida a utilização de funções para calcular a soma do vetor, como o *sum* do Python.

Outro fato importante é que sempre trabalharemos com **inteiros**. Números em **ponto flutuante** podem diferir se as operações forem feitas em diferentes ordens, isso pela maneira como as informações são armazenadas na memória (pesquise se tiver curiosidade!). Então, neste laboratório será utilizada a **divisão inteira** (*//*).

Todos os casos de testes e códigos de testes unitários estão disponíveis no seguinte link: testes Lab06. Os arquivos com a extensão ".in" contém as entradas dos testes e nos arquivos com final ".out" as saídas correspondentes. Os arquivos de textos podem ser abertos com qualquer editor de texto. Arquivos "test_*.py" serão rodados no CodePost com o Pytest para verificar as funções implementadas via testes automatizados, e o código também será usado como entrada para o Flake8 e o Mypy (com a flag --strict) para verificação de formatação e de tipagem.

Seu código será avaliado não apenas pelos testes do CodePost, mas também pela qualidade. Dentre os critérios subjetivos de qualidade de código, analisaremos neste laboratório: o uso e reuso apropriado de funções; o tamanho das funções; se as funções estão autocontidas; a escolha de bons nomes de funções e variáveis; a ausência de diversos trechos de código repetidos desnecessariamente; documentação de código. Note, porém, que essa não é uma lista exaustiva, pois outros critérios podem ser analisados dependendo do código apresentado visando mostrar ao aluno como o código poderia ser melhor.

Submissão

Você deverá submeter no CodePost, na tarefa Lab 06, um arquivo com o nome `lab06.py`. Todas as funções **obrigatórias** do laboratório deverão estar presentes neste arquivo, mas também poderão ser enviados outros arquivos que contenham funções auxiliares se desejar. Após a correção da primeira entrega, será aberta uma tarefa Lab 06 - Segunda Chance, com prazo de entrega apropriado.