# **CEN0336 - Introdução a Programação de Computadores Aplicada a Ciências Biológicas**

# **Prova II**

Nome: Biatriz Adrielli Pedroza da Silva

No USP: 12528154

## **Instruções Gerais**

Vocês terão desde o dia 21 de novembro as 22:01 e até o dia 5 de dezembro as 23:50 para resolver as atividades da prova.

São 5 perguntas/atividades, cada uma precisa de uma resposta neste arquivo de word, que tem que ser carregado no sistema e-disciplinas até o dia e horário limite. Além disso algumas perguntas precisam que vocês desenvolvam scripts e carreguem num repositório github específico para a prova (indicado na pergunta 1). No dia 5 de dezembro as 23:51 o professor descarregará todos os repositórios para avaliar seus scripts. Os critérios de avaliação dos scripts são:

Eles têm que rodar sem erros de sintaxe (15%)

Eles têm que ser logicamente corretos (20%)

Eles têm que ter comentários explicando o que é feito em cada passo (20%)

Eles têm que ter controle da entrada dos usuários (10%)

Eles têm que ter controle de erros usando blocos try/except (20%)

Eles têm que implementar o uso de funções (15%)

## **Perguntas/Atividades**

1. Crie um repositório no github para esta prova. Escreva o endereço completo do seu repositório github como resposta desta pergunta.

Lembre-se que o repositório tem que ter o nome segundo o padrão:

P2\_NNNNNNN-CEN0336

Onde NNNNNNN é seu número USP.

A resposta tem que ter o endereço completo, iniciando com https://"

O professor irá clonar seu repositório, com as respostas para os outros pontos da prova.

**Resposta:**

https://github.com/AdriBia/P2\_12528154-CEN0336

1. Crie um script de python com o nome script\_getORF.py que realize as seguintes operações:
2. Ler um arquivo multifasta da linha de comandos, usando o modulo sys.
3. Para cada registro fasta identificar o Open Reading Frame (Fase de Leitura Aberta: <https://pt.wikipedia.org/wiki/Fase_de_leitura_aberta> ou <https://en.wikipedia.org/wiki/Open_reading_frame>) mais comprido dentre as 6 fases de leitura possíveis. Os ORFs neste caso não têm introns. Repare que uma fase de leitura aberta tem que iniciar com um codon de início da tradução e terminar com um códon de terminação da tradução (<https://pt.wikipedia.org/wiki/C%C3%B3digo_gen%C3%A9tico>), é seu comprimento tem que ser múltiplo de três (3)
4. Daquele ORF mais comprido identificado, gerar o peptídeo codificado.
5. Imprimir o ORF mais comprido no arquivo ORF.fna. O formato da sequência tem que ser fasta, e seu identificador, tem que ser o identificador original, adicionando o string: “\_frameX\_START\_END”, onde X é a Fase de leitura (1,2,3,4,5 ou 6), e START e END são as coordenadas na sequencia
6. Imprimir o peptídeo correspondente ao ORF mais comprido no arquivo ORF.faa. O formato da sequência tem que ser fasta, e seu identificador, tem que ser o identificador original, adicionando o string: “\_frameX\_START\_END”, onde X é a Fase de leitura (1,2,3,4,5 ou 6), e START e END são as coordenadas na sequencia
7. Não esquece que tem que processar todos os registros do arquivo fasta original, i.e., tem que ter o mesmo número de entradas nos arquivos ORF.faa, ORF.fna que no arquivo original.
8. Carregue seu script, e os arquivos ORF.faa e ORF.fna, assim como o arquivo multifasta que usou para testes no seu repositório.
9. Calcule o md5sum do seu script e coloque esse número como resposta desta pergunta.

Resposta:

Repositório:

https://github.com/AdriBia/P2\_12528154-CEN0336/blob/main/script\_getORF.py

Md5sum: 7905cc4fbbd652ba979701fa27cb3d5c

1. Explique a diferença entre os dois sets:

set1=set('accggttaata')

set2={'accggttaata'}

No primeiro set, set1 = set('accggttaata'), a função set() cria um conjunto a partir de uma string. Isso resulta em um set contendo apenas os caracteres únicos da string, ou seja, as letras individuais sem duplicatas definindo o set1 como o conjunto {'a', 'c', 'g', 't'}.

O segundo set, set2={‘accggttata’} cria um set com um único elemento, uma String ‘accggttaata.

Texto

Descrição gerada automaticamente

1. Descreva com suas próprias palavras o que o seguinte pseudocódigo faz. Faz uma avaliação desse pseudocódigo e relate quaisquer problemas que possa ter.

Defina TOTAL com zero

Defina CONTADOR NOTAS como zero

While CONTADOR NOTAS é menor ou igual a dez

      Entre (input) a seguinte nota

      Some a nota ao TOTAL

Defina a média da disciplina como o TOTAL dividido por dez (TOTAL/10)

Imprima na tela a média da disciplina

Implemente esse pseudocódigo num script de python e carregue ele no seu repositório da prova com o nome notas.py. Calcule o md5sum do seu script e coloque esse número também como resposta desta pergunta.

O Pseudocódigo calcula a média de uma disciplina a partir de 10 notas.

Defina TOTAL com zero #define TOTAL como uma variável que recebe o valor zero

Defina CONTADOR NOTAS como zero #define CONTADOR NOTAS como uma variável que também recebe o valor zero

While CONTADOR NOTAS é menor ou igual a dez #define que uma ação ação será realizada enquanto a Variável CONTADOR NOTAS for menor que 10

      Entre (input) a seguinte nota ##pede ao usuário a nota a ser calculada

      Some a nota ao TOTAL #soma a nota inserida à variável total

Defina a média da disciplina como o TOTAL dividido por dez (TOTAL/10) #define a média das notas

Imprima na tela a média da disciplina #imprime o resultado

O código não contém uma verificação do formato das notas, não sendo especificado intervalo, se recebe notas com valores decimais ou negativas, da mesma forma não possui uma especificação para lidar com entradas inválidas como letras ou outros caracteres, afetando o loop de contagem de notas.

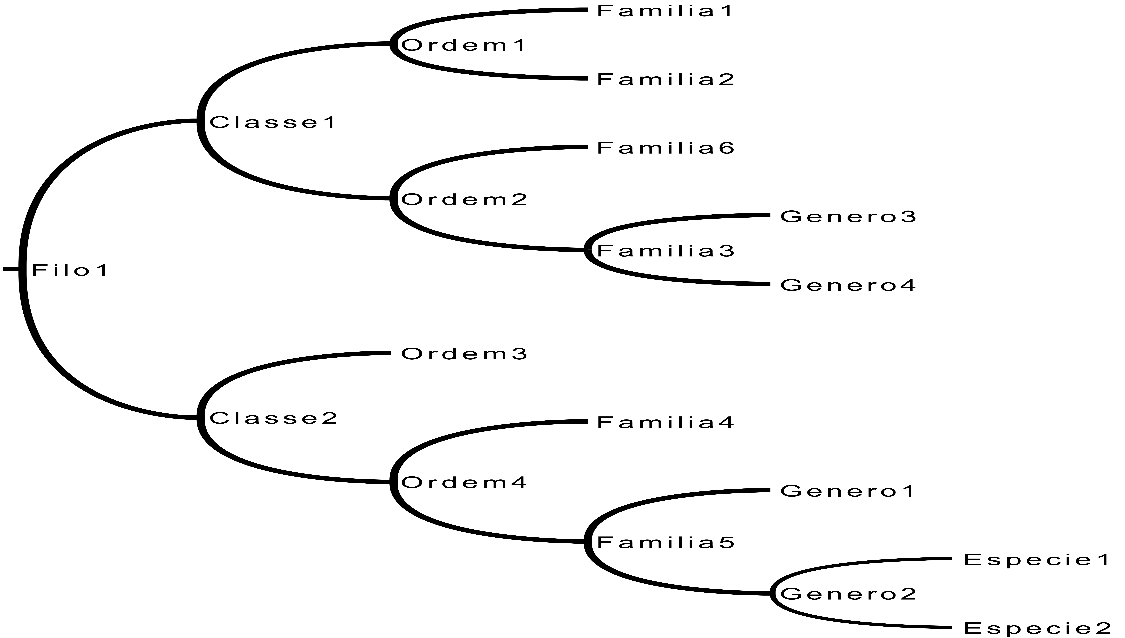
O cálculo da média também toma como base o número de 10 notas sempre, ao invés de calcular com base no número de notas inseridas.

Repositório:  
<https://github.com/AdriBia/P2_12528154-CEN0336/blob/main/notas.py>

Md5sum: afbde492a38c8ec8fe68d9fe6fdf1c17

1. Uma árvore filogenética é uma representação gráfica ou computacional que descreve as relações evolutivas entre diferentes espécies, genes ou outros elementos biológicos. Ela é amplamente usada em biologia para inferir a ancestralidade comum e entender como diferentes grupos se relacionam ao longo do tempo. Em bioinformática, árvores filogenéticas podem ser representadas no formato Newick, uma notação textual que descreve a hierarquia das ramificações, ou em estruturas computacionais como dicionários.

Considere a seguinte árvore filogenética:



A mesma arvore filogenética pode ser presentada como um dicionário de dicionários, assim:

arvore = {

"Filo1": {

"Classe1": {

"Ordem1": {

"Familia1": {},

"Familia2": {}

},

"Ordem2": {

"Familia3": {

"Genero3": {},

"Genero4": {}

}

}

},

"Classe2": {

"Ordem3": {},

"Ordem4": {

"Familia4": {},

"Familia5": {

"Genero1": {},

"Genero2": {

"Especie1": {},

"Especie2": {}

}

}

}

}

}

}

Cada nó da árvore pode ter outros nós como filhos (subárvores). O objetivo desenvolver um script que calcule o número total de nós na árvore (incluindo os ramos internos e folhas). Escreva uma função recursiva chamada contar\_nos que percorre a árvore e retorna o total de nós.

**Repositório:**

**https://github.com/AdriBia/P2\_12528154-CEN0336/blob/main/contar\_nos.py**

