PROVA DE RECUPERAÇÃO

MAC0119 INTRODUÇÃO À PROGRAMAÇÃO DE COMPUTADORES (ICB)

2º SEMESTRE DE 2025

Nome completo: _____

NUSP: _____

Instruções:

(*1*) Esta prova é individual.
(*2*) A prova consiste de **4** questões (contando a Questão 0 nesta página). Note que é possível tirar mais de 10 nesta prova `:-)`
(*3*) Para ter nota integral em uma questão, a escrita de sua solução deve estar boa.
(*4*) As respostas devem estar nos locais indicados.
(*5*) Não é permitido o uso de aparelhos eletrônicos de qualquer natureza.
(*6*) Não destaque as folhas deste caderno.
(*7*) Não use folhas avulsas para rascunho. Não é necessário apagar seus rascunhos.
(*8*) Não é permitido consultar colegas.
(*9*) O único material que você pode consultar é seu *cheat sheet* (uma folha A4, frente e verso).

Assinatura:

Sua assinatura acima atesta a autenticidade e originalidade de seu trabalho e que você compromete-se a seguir o código de ética da USP em todas as suas atividades, incluindo esta prova.

Boa sorte!

Q	0	1	2	3	**Total**
Nota					

Q0. **[0.5 pontos]** Leia o conteúdo desta página e preencha os itens requisitados. Assine acima, e atente ao significado de sua assinatura.

Data: 2026/2/2, 8:37pm

Q1. [**3.5 pontos**] Considere o seguinte código.

```
def len_ORF(s):
    for i in range(3, len(s), 3): # assumes s[0 : 3] == 'ATG'
        if s[i : i + 3] in ['TAA', 'TAG', 'TGA']:
            return i
    return -1

def ORFs(s):
    orfs = []
    # consider only the 0 frame
    for i in range(0, len(s), 3):
        if s[i : i + 3] == 'ATG':
            j = len_ORF(s[i:])
            if (j > 0):
                orfs.append(s[i : i + j])
    return orfs

def all_ORFs(s):
    # call ORFs() for frames 0, 1 and 2
    orfs = ORFs(s)
    orfs.extend(ORFs(s[1:]))
    orfs.extend(ORFs(s[2:]))
    return orfs
```

Segue um exemplo que ilustra o comportamento de `all_ORFs()`:

```
>>> x = all_ORFs('CATGCCCATGGGGAAATTTTGACC')
>>> print(x)
['ATGCCCATGGGGAAATTT', 'ATGGGGAAATTT']
>>>
```

Em cada item abaixo, é dada uma chamada de função. Você deve dizer em cada item o valor devolvido pela chamada dada.

(i) `all_ORFs('CTTCCGAATCCTGTA')`

Resposta:

(ii) `all_ORFs('AACCATGATCGTGGA')`

Resposta:

(*iii*) `all_ORFs('TCGACTAATGTAATC')`

Resposta:

(*iv*) `all_ORFs('TTGTGATGCATGACA')`

Resposta:

(*v*) `all_ORFs('TTATGAATGGTAATGTGTTGAGCA')`

Resposta:

(*vi*) `all_ORFs('GTCATGGTATGCATGTTAACGTGA')`

Resposta:

(*vii*) `all_ORFs('AATGTATGCCCATATGCTCCGCATGCTAAACG')`

Resposta:

Q2. **[4 pontos]**

(i) Considere as seguintes funções:

```
# All possible shufflings of s and t
def shuffle(s, t):
    shuffleR(s, t, '')

# p followed by all possible shufflings of s and t
def shuffleR(s, t, p):
    if s == '':
        print(p + t)
        return
    if t == '':
        print(p + s)
        return
    shuffleR(s[1:], t, p + s[0])
    shuffleR(s, t[1:], p + t[0])
```

(a) Qual é a saída do seguinte código?

```
shuffle('A', 'ab')
```

Resposta:

(b) Qual é a saída do seguinte código?

```
shuffle('AB', 'ab')
```

Resposta:

(c) Qual é a saída do seguinte código?

```
shuffle('AB', 'abc')
```

Resposta:

Resposta (continuação):

(*ii*) Considere as seguintes funções:

```
def flatten1(l):
    ll = []
    for x in l:
        if type(x) == type([]): # if x is a list
            ll.extend(x)
        else:
            ll.append(x)
    return ll

def flatten2(l):
    ll = []
    for x in l:
        if type(x) == type([]): # if x is a list
            ll.extend(flatten2(x))
        else:
            ll.append(x)
    return ll
```

Antes de continuar, você pode achar interessante estudar o seguinte exemplo:

```
>>> x = [0, 1]
>>> x.append(2)
>>> print(x)
[0, 1, 2]
>>> x.extend([3, 4])
>>> print(x)
[0, 1, 2, 3, 4]
>>> x.append([5, 6])
>>> print(x)
[0, 1, 2, 3, 4, [5, 6]]
>>>
```

(*a*) Qual é a saída do seguinte código?

```
print(flatten1([0, [1], [2, 3]]))
```

Resposta:

(*b*) Qual é a saída do seguinte código?

```
print(flatten1([0, [1], [2, 3, [4, 5]]]))
```

Resposta:

(*c*) Qual é a saída do seguinte código?

```
print(flatten2([0, [1], [2, 3, [4, 5]]]))
```

Resposta:

(*d*) Qual é a saída do seguinte código?

```
print(flatten2([0, [[[1, 2], [3, 4]], [[], [5, [6]]]]]))
```

Resposta:

Q3. [**4 pontos**]

(i) Considere o seguinte trecho de código.

```
codons_list = {'F': ['TTT', 'TTC'],
               'L': ['TTA', 'TTG', 'CTT', 'CTC', 'CTA', 'CTG'],
               'I': ['ATT', 'ATC', 'ATA'],
               'M': ['ATG'],
               'V': ['GTT', 'GTC', 'GTA', 'GTG'],
               'S': ['TCT', 'TCC', 'TCA', 'TCG', 'AGT', 'AGC'],
               'P': ['CCT', 'CCC', 'CCA', 'CCG'],
               'T': ['ACT', 'ACC', 'ACA', 'ACG'],
               'A': ['GCT', 'GCC', 'GCA', 'GCG'],
               'Y': ['TAT', 'TAC'],
               '|': ['TAA', 'TAG', 'TGA'],
               'H': ['CAT', 'CAC'],
               'Q': ['CAA', 'CAG'],
               'N': ['AAT', 'AAC'],
               'K': ['AAA', 'AAG'],
               'D': ['GAT', 'GAC'],
               'E': ['GAA', 'GAG'],
               'C': ['TGT', 'TGC'],
               'W': ['TGG'],
               'R': ['CGT', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'],
               'G': ['GGT', 'GGC', 'GGA', 'GGG']}

# Sequencias de códons que codificam p
def possible_genes(p):
    possible_genesR(p, '')

# gprefix seguido de sequencias de códons que codificam p
def possible_genesR(p, gprefix):
    if p == '':
        print(gprefix)
        return
    for c in codons_list[p[0]]:
        possible_genesR(p[1:], gprefix + c)
```

(a) Qual é a saída produzida pela seguinte chamada de `possible_genes()`?

```
possible_genes('NCW')
```

Resposta:

(*b*) Qual é a saída produzida pela seguinte chamada de `possible_genes()`?

```
possible_genes('YQE')
```

Resposta:

(*ii*) Suponha agora que, dada uma sequência de bases `DNA`, queremos saber a sequência de aminácidos que `DNA` codifica. Por exemplo, a sequência de bases `TATCAAGAA`, codifica a sequência de aminoácidos `YQE`. Podemos resolver este problema com a função abaixo:

```
def coding_strand_to_aa(DNA, codon_to_aa):
    p = ''
    for i in range(0, len(DNA), 3):
        p += codon_to_aa[DNA[i : i + 3]]
    return p
```

Entretanto, a função acima usa o dicionário `codon_to_aa`, que diz a qual aminácido corresponde cada códon. Por exemplo, `codon_to_aa['TAT']` vale `'Y'`, como pode ser visto no dicionário `codons_list` acima.

Você deve agora escrever uma função de cabeçalho

```
def invert_codons_list(d):
```

tal que, ao executarmos

```
codon_to_aa = invert_codons_list(codons_list)
```

obtemos o dicionário `codon_to_aa` como precisamos em `coding_strand_to_aa()` acima. Com sua função `invert_codons_list()`, poderemos, por exemplo, fazer o seguinte:

```
>>> codon_to_aa = invert_codons_list(codons_list)
>>> coding_strand_to_aa('TATCAAGAA', codon_to_aa)
'YQE'
>>>
```

Escreva aqui sua função `invert_codons_list()` por completo.

Resposta:

* * *

Rascunho:

Rascunho:

Rascunho: