# Prova 3
# MAC-115 – Introdução à Computação
### 1º semestre de 2017 - IAG

**nome** (em letras de forma **LEGÍVEIS**):

**assinatura**:

**professor**:

**No. USP**:

**Instruções.**

(1) Não destaque as folhas deste caderno.
(2) A prova pode ser feita a lápis.
(3) A legibilidade também faz parte da nota !
(4) A prova consta de 3 questões. Verifique antes de começar a prova se o seu caderno de questões está completo.
(5) Não é permitido o uso de folhas avulsas para rascunho.
(6) Não é necessário apagar rascunhos no caderno de questão mas especifique qual é a resposta e qual é o rascunho.
(7) Só é permitido usar os recursos dados nas aulas até o dia desta prova e deve-se seguir todas as restrições dadas também.
(8) A prova é sem consulta.

**Não escrever nesta parte da folha**

| Questão | Nota | Observação |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |
| Total | | |

## Boa Sorte !

**Questão 1 (valor=3.0).** Escreva um programa que leia uma sequência de caracteres digitada pelo usuário e crie um dicionário principal, associando a cada letra que ocorre na frase um dicionário secundário de letras sucessoras, associadas por sua vez ao número de ocorrências daquele par. Por exemplo, na frase "teste" o dicionário principal será {'t':{'te':2}, 'e':{'es':1}, 's':{'st':1}}, enquanto na frase

"semessaaranhanemasanhaarranhaocarronemosarroarranhaaespanha",

o dicionário `principal` conterá as letras `'a', 'c', 'e', 'h', 'm', 'n', 'o', 'p', 'r', 's'` e seus respectivos dicionários, sendo por exemplo o dicionário

`principal['a'] = {'an':6, 'ar':5, 'as':1, 'ao':1, 'aa':3, 'ae':1}`

(as 6 ocorrências de `'an'` estão sublinhadas no exemplo).

Obs: Se você não souber resolver a questão com dicionários mas souber com listas, a questão valerá 2.0 pontos.

**Questão 2 (valor=4.0).** Considere a função definida abaixo:

```
def ordena(v):
    n = len(v)
    print(v)
    for i in range(int(n/2)):
        x = i
        y = i
        for j in range(i,n-i):
            if v[j]<v[x]:
                x = j
            if v[j]>v[y]:
                y = j
        print("x =",x,"y =",y)
        # troca os valores entre v[i] e v[x]
        v[i], v[x] = v[x], v[i]
        if y!=i: # <--- essa é a linha 15
            v[y], v[n-i-1] = v[n-i-1], v[y]
        else:
            v[x], v[n-i-1] = v[n-i-1], v[x]
        print(v)
```

(1) Simule a execução de `ordena(NUSP)`, onde `NUSP` é a lista formada pelos dígitos do seu número USP (por exemplo, para o número USP 0123456 teríamos `NUSP = [0,1,2,3,4,5,6]`). Você pode organizar o rascunho da sua simulação como preferir, mas deve indicar claramente as saídas dos comandos `print` (só estas serão corrigidas).

(2) Explique *sucintamente* a finalidade das variáveis `x` e `y` na implementação, bem como o papel das variáveis `i` e `j` na estratégia de ordenação implementada. Como essa estratégia se relaciona com o algoritmo de *ordenação por seleção* estudado?

(3) As linhas 15 a 18 do código definem trocas diferentes quando `y!=i` e quando `y==i`. Mostre uma lista pequena `v` que justifica a necessidade desse `if/else`, simulando o que aconteceria se no lugar das linhas 15 a 18 o código trouxesse apenas a troca `v[y], v[n-i-1] = v[n-i-1], v[y]` (correspondente à linha 16). Dica: o problema está associado à condição que corresponde ao `else`.

**Questão 3 (valor=3.0)).** Chamamos de *unidade complexa* a qualquer número com representação Cartesiana $z = a + ib$ tal que $|z| = \sqrt{a^2 + b^2} = 1$ (ou seja, $z$ pertence ao círculo unitário). A *relação de Euler* permite a representação polar desses números através da expressão

$$z = e^{i\omega} = \cos(\omega) + i\text{sen}(\omega),$$

de onde se deduz que $a = \cos(\omega)$ e $b = \text{sen}(\omega)$.

**Atenção: nessa questão você não pode usar a biblioteca de funções matemáticas (import math)**

(1) Escreva uma função `polar2cartesiana()` que recebe $\omega \in [-\pi, +\pi]$ e devolve um valor complexo `complex(a,b)` associado à representação Cartesiana de $z = e^{i\omega}$. Para calcular o $\cos(\omega)$, utilize a série de Taylor

$$\cos\omega = 1 - \frac{\omega^2}{2!} + \frac{\omega^4}{4!} - \frac{\omega^6}{6!} + \ldots + (-1)^k \frac{\omega^{2k}}{(2k)!} + \ldots$$

enquanto os termos gerados satisfizerem $|\texttt{termo}| > 10^{-8}$. Você *deve* calcular os termos novos sempre a partir do termo anterior (não use nem `**` nem `fatorial`). Para calcular $\text{sen}(\omega)$, utilize a propriedade $\text{sen}(\omega) = \pm\sqrt{1 - \cos^2(\omega)}$ (o sinal da raiz deve ser o mesmo do $\omega$).

(2) Uma `raiz n-ésima da unidade` é qualquer número $z \in \mathbb{C}$ tal que $z^n = 1$ (em particular, tem-se $|z| = 1$, de onde $z$ é obrigatoriamente uma unidade complexa). O método de Newton permite obter raízes n-ésimas através do iterador[1]

$$z_{k+1} = z_k - \frac{z_k^n - 1}{n z_k^{n-1}}, \qquad k \geq 0.$$

Escreva uma função `raizDaUnidade` que recebe $n$ e devolve uma raiz n-ésima da unidade, aplicando o método de Newton acima a partir de uma unidade complexa $z_0$ sorteada aleatoriamente e parando quando $|z_{k+1} - z_k| < 10^{-8}$. Use $\omega =$`6.2832*(random.random()-0.5)` para inicializar $z_0 = e^{i\omega}$, usando `import random` e a função do item (1), mesmo que não a tenha feito. Você pode usar o operador `**` nesse item para calcular $z_k^n$ e $z_k^{n-1}$.

[1] Esse iterador é idêntico ao do EP3 para $f(z) = z^n - 1$.