MAC2166 - Introdução a Computação - 2019S1 **Avaliação PRec**

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9

Utilize caneta azul ou preta e preencha completamente a quadrícula.
Exemplo: ■. **Não use ☒.**

Turma: (somente um número; consulte a pessoa responsável se não souber)

4	5	6	7	8	9	10	11	12	13	14	20

⟵ Marque as quadrículas ao lado para formar o seu número USP e escreva seu nome completo em letra legível na linha pontilhada abaixo. **Se seu número possui menos que 8 dígitos complete com zeros à esquerda.**

Nome:

. . . . . .

Esta prova tem duração de 120 minutos. Não desmonte a prova.

Q1 [2 pontos] Simule o código abaixo e selecione a opção correspondente à saída impressa do programa. Ignore as quebras de linhas geradas pelo comando `print`.

```
def f (L, e, d):
    if e > d:
        return True
    c = (e+d)//2
    print (L[c])
    x = f (L, e, c-1)
    y = f (L, c+1, d)
    return True

def g (L, e, d):
    if e > d:
        return True
    c = (e+d)//2
    x = g (L, e, c-1)
    y = g (L, c+1, d)
    print (L[c])
    return True

def main ():
    f ([35, 42, 14], 0, 2)
    g ([35, 42, 14], 0, 2)

main ()
```

Rascunho

- ☐ 35 42 14 35 42 14
- ☐ 14 35 42 42 35 14
- ☐ 35 14 42 42 35 14
- ☐ 42 35 14 35 14 42
- ☐ 42 42
- ☐ 14 35 42 14 35 42
- ☐ 42 14 35 35 14 42
- ☐ 35 42 14 42 14 35
- ☐ 14 42 35 42 35 42
- ☐ 42 35 14 42 35 14
- ☐ 35 42 42 35
- ☐ 14 35 42 35 42 14
- ☐ 35 42 14 14 35 42
- ☐ 42 35 14 14 35 42
- ☐ 42 35 35 42
- ☐ 35 14 42 42 14 35
- ☐ 42 14 35 42 14 35
- ☐ 42 35
- ☐ 14 42 35 42 35 14
- ☐ 35 14 42 35 14 42

Q2 [2 pontos]

Preencha as lacunas L1 até L3, de forma a obter uma função recursiva que calcula o número de ocorrências de um dado valor `a` em uma lista `L`, considerando seus `n` primeiros elementos, ou seja, quantas vezes esse valor `a` aparece entre os `n` primeiros elementos da lista. **Exemplo:** Para `L=[3,3,4,3,2,2,3,3]`, `n=8` e `a=3` retorna `5`.

A moda de um conjunto de dados é o valor que detém o maior número de observações, ou seja, o valor mais frequente (ou seja, o que "está na moda"...). No caso de empate qualquer um dos valores de frequência máxima pode ser considerado como sendo uma moda. Preencha as lacunas L4 até L8, de forma a obter uma função recursiva que calcula uma moda de uma lista de inteiros com `n` elementos.

Exemplo: Para `L=[3,3,4,3,2,2,3,3]` devolve moda `3`. Obs: No caso de empate, a seleção de qualquer uma das modas é satisfatória.

```
def Frequencia(a,L,n):
    L1
    L2
    L3
    return f
}

def Moda(L,n):
    L4
    L5
    L6
    L7
    if f1 > f2:
        return m
    L8
}

def main():
    L = [3,3,4,3,2,2,3,3]
    print(Moda(L, len(L)))
}
```

Rascunho

A cada opção errada que for selecionada, desconta-se nota do exercício.

L1:

- ☐ `if n == 0: return 0`
- ☐ `if n == 1: return -1`
- ☐ `if n == 0: return -1`
- ☐ `if n == 1: return 1`
- ☐ `if n <= 1: return 0`
- ☐ `if n == 1: return 0`
- ☐ `if n == 0: return 1`

L2:

- ☐ `f = Frequencia(a, L, n+1)`
- ☐ `f = Frequencia(a, L, n-1)`
- ☐ `f = Frequencia(a, L, n//2)`
- ☐ `f = Frequencia(n-1, L, a)`
- ☐ `f = Frequencia(a, L, n)`
- ☐ `f = Frequencia(a, L, n-2)`
- ☐ `f = Frequencia(n, L, a)`

L3:

- ☐ `if L[n-1] != a: f+=1`
- ☐ `if L[n] == a: f+=1`
- ☐ `if L[n] == a: f-=1`
- ☐ `if L[n] != a: f-=1`
- ☐ `if L[n-1] == a: f+=1`
- ☐ `if L[n-1] != a: f-=1`

L4:

- ☐ `if n == 2: return 2`
- ☐ `if n == 1: return L[0]`
- ☐ `if n == 1: return L[1]`
- ☐ `if n == 1: return L[n]`
- ☐ `if n == 0: return 0`
- ☐ `if n == 1: return 1`

L5:

- ☐ `m = Moda(L, n+1)`
- ☐ `m = Moda(L, n//2)`
- ☐ `m = Moda(L, n-2)`
- ☐ `m = Moda(L, n)`
- ☐ `m = Moda(L+1, n)`
- ☐ `m = Moda(L, n-1)`

L6:

- ☐ `f1 = Frequencia(m, L, n-1)`
- ☐ `f1 = Frequencia(m, L, n)`
- ☐ `f1 = Frequencia(0, L, n)`
- ☐ `f1 = Frequencia(L[0], L, n)`
- ☐ `f1 = Frequencia(m, L, n+1)`
- ☐ `f1 = Frequencia(L[n-1], L, n)`

L7:

- ☐ `f2 = Frequencia(L[n], L, n-1)`
- ☐ `f2 = Frequencia(L[n-1], L, n)`
- ☐ `f2 = Frequencia(m, L, n)`
- ☐ `f2 = Frequencia(m, L, n-1)`
- ☐ `f2 = Frequencia(L[0], L, n)`
- ☐ `f2 = Frequencia(L[n], L, n)`

L8:

- ☐ `else: return L[0]`
- ☐ `else: return n`
- ☐ `else: return L[n-1]`
- ☐ `else: return m`
- ☐ `else: return n-1`
- ☐ `else: return L[n]`

Q3 [3 pontos] Preencha a lacuna L1 da função `numcomb` abaixo, de forma a obter uma solução recursiva que calcula o número de combinações de `n` elementos tomados `k` a `k`:

$$C_k^n = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Exemplo:

Para a chamada `numcomb(3, 2)` a função devolve `3`.
Para a chamada `numcomb(5, 3)` a função devolve `10`.
Para a chamada `numcomb(7, 4)` a função devolve `35`.
Para a chamada `numcomb(7, 5)` a função devolve `21`.

Utilizando a função anterior, preencha as lacunas L2 e L3, de forma a obter uma função recursiva que imprime o triângulo de Pascal de grau `m`, isto é, com `m` linhas.

Exemplo: Para `m=7` a função deve imprimir:

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
1   6  15  20  15   6   1
```

```
def numcomb(n, k):
    if n == 0 or k == 0 or n == k:
        return 1
    L1
    return r

def TrianguloPascal(m):
    if m == 1:
        print(1)
    else:
        L2
        for k in range(m):
            L3
        print()
```

Rascunho

Selecione uma solução para L1:

L1:
- ☐ `a = numcomb(n-1, k); r = (a*n)//(n-k+1)`
- ☐ `a = numcomb(n-1, k); r = (k*a)//(n-k)`
- ☐ `a = numcomb(n-1, k); r = ((n-k)*a)//(k)`
- ☐ `a = numcomb(n-1, k); r = (a*n)//(n-k)`
- ☐ `a = numcomb(n-1, k); r = (a*n)//k`
- ☐ `a = numcomb(n-1, k); r = (a*n)//(n-k-1)`

Selecione uma segunda solução alternativa para L1:

L1:
- ☐ `a = numcomb(n, k-1); r = (a*(n-k-1))//k`
- ☐ `a = numcomb(n, k-1); r = (a*(n-k))//(k-1)`
- ☐ `a = numcomb(n, k-1); r = (a*(n-k+1))//(k+1)`
- ☐ `a = numcomb(n, k-1); r = (a*(n-k))//n`
- ☐ `a = numcomb(n, k-1); r = (a*(n-k+1))//k`
- ☐ `a = numcomb(n, k-1); r = (a*(n-k))//k`

Selecione uma terceira solução alternativa para L1:

L1:
- ☐ `a = numcomb(n-1, k-1); r = a - numcomb(n-1, k)`
- ☐ `a = numcomb(n-1, k-1); r = a * numcomb(n-1, k)`
- ☐ `a = numcomb(n-1, k-1); r = -a + numcomb(n-1, k)`
- ☐ `a = numcomb(n-1, k-1); r = -a * numcomb(n-1, k)`
- ☐ `a = numcomb(n-1, k-1); r = a // numcomb(n-1, k)`
- ☐ `a = numcomb(n-1, k-1); r = a + numcomb(n-1, k)`

Selecione uma quarta solução alternativa para L1:

L1:
- ☐ `a = numcomb(n-1, k-1); r = (n*a)//(k+1)`
- ☐ `a = numcomb(n-1, k-1); r = (k*a)//n`
- ☐ `a = numcomb(n-1, k-1); r = (n*a)//(k-1)`
- ☐ `a = numcomb(n-1, k-1); r = (n*a)//k`
- ☐ `a = numcomb(n-1, k-1); r = ((k+1)*a)//n`
- ☐ `a = numcomb(n-1, k-1); r = ((k-1)*a)//n`

L2:
- ☐ `TrianguloPascal(m//2)`
- ☐ `TrianguloPascal(m-1)`
- ☐ `for k in range(m+1): TrianguloPascal(k)`
- ☐ `for k in range(m-1,0,-1): TrianguloPascal(k)`
- ☐ `TrianguloPascal(m-2)`
- ☐ `for k in range(m): TrianguloPascal(k)`

L3:
- ☐ `print(numcomb(m-1, k+1), end = ' ')`
- ☐ `print(numcomb(m-1, k), end = ' ')`
- ☐ `print(numcomb(k+1, m-1), end = ' ')`
- ☐ `print(numcomb(k, m), end = ' ')`
- ☐ `print(numcomb(m, k), end = ' ')`
- ☐ `print(numcomb(k, m-1), end = ' ')`

Q4 [3 pontos] Nesta questão você deve elaborar um programa que, dado um número maior ou igual a zero, diga qual o dígito que apresenta um número maior de repetições consecutivas. Por exemplo para o número 12233340000 você deveria indicar que o número 0 aparece 4 vezes. Assinale a alternativa que contém os blocos corretos na ORDEM correta.

DICA 1: As variáveis do programa são APENAS: `dig`:um digito; `digmax`:dígito de frequência máxima; `cont`:um contador; `max`:-um valor máximo; `num`:um número; `udig`:dígito anterior.

DICA 2: Não tente usar todas as combinações, tente codificar o programa na área de rascunho e depois escolha os trechos adequados. O RASCUNHO NÃO SERÁ CONSIDERADO NA NOTA.

DICA 3: Na solução sugerida nós tratamos o primeiro dígito fora do loop para simplificar o código interno; os dígitos são extraídos da direita para a esquerda; o número zero precisa de tratamento especial.

```
#A
main()
#B
def main():
    num = int(input())
#C
    max = 0
#D
    max = 0
    ult = 0
    digmax = 0
#E
    max = 1
    ult = ''
    digmax = 0
#F
    dig = num % 10
    num = num // 10
    digmax = dig
    cont = 1
#G
    num = num /10
    dig = num % 10
    cont = 1
#H
    dig = num //10
    num  = num % 10
    cont = 0
    maxdig = 0
#I
            cont = 1
#J
            dig = 0
#K
            udig = 1
#L
    if (max == 1):
#M
    if (udig == 0):
#N
    if (digmax == 0):
#O
        if (dig == udig):
            cont += 1
#P
        if (dig > udig):
            cont = cont + 1
#Q
    else:
#R
        else:
            if (cont > max):
                max = cont
                digmax = udig
#S
            if (cont > max):
                max = cont
                digmax = udig
#T
    while (num > 0):
        udig = dig
        dig = num %10
        num = num // 10
#U
    while (not(dig ==0)):
        udig = dig
        dig = num %10
        num = num // 10
#V
    while (num > 0):
        num = num // 10
        dig = num %10
        udig = dig
#W
        print('Maior tem apena um digito (',digmax,')')
#X
         print('Maior tem ', max, ' digitos (',digmax,')')
#Y
    else:
#Z
    if (num == 0):
        cont = 1
        print('Maior tem apenas um digito (0)')
        return
```

- ☐ `B,Z,H, V, O, R, I, L,W,Y,X,A`
- ☐ `B,C,U,G, T, O, S, J, L,W,X,A`
- ☐ `B,C,Z,G, M,N, S, J, L,W,X,A`
- ☐ `B,C,H, T, O, R, J, KL,W,Y,X`
- ☐ `B,C,Z,G, U,M,N,J, S, J, L,W,X,A`
- ☐ `B,C,H, T, O, S, I, L,W,Y,X`
- ☐ `B,C,Z,G, U, O, S, J, L,W,X`
- ☐ `B,C,Z,F, T, O, R, I, L,W,Y,X,A`
- ☐ `B,C,Z,H, V, O, R, I, L,W,Y,X,A`
- ☐ `B,C,Z,G, U, O, S, J, L,W,X,A`
- ☐ `B,C,Z,F, U, O, R, I, L,W,Y,X,A`
- ☐ `B,Z,H, V, O, S, J,K, L,W,Y,X,A`

Rascunho