



TÉCNICO
LISBOA

Fundamentos da Programação

Solução do exame

09 de Janeiro de 2020

11:30–13:30

1. (1.0) Para cada uma das seguintes afirmações, diga se é verdadeira ou falsa. Cada resposta certa vale 0.5 valores e *cada resposta errada desconta 0.2 valores*.

(a) Em Python, o valor devolvido por uma instrução de atribuição corresponde ao valor da expressão que se encontra no lado esquerdo do sinal de =.

Resposta:

Falsa

(b) A programação funcional não usa explicitamente os ciclos `while` e `for`.

Resposta:

Verdadeira

2. Usando palavras suas e, no máximo, em três linhas explique os seguintes conceitos. Explicações dadas através de exemplos serão classificadas com zero valores.

(a) (0.5) Tipo de dados.

Resposta:

Um conjunto de elementos (as constantes do tipo), juntamente com um conjunto de operações que se aplicam a esses elementos.

(b) (0.5) Visibilidade de um nome.

Resposta:

O conjunto de instruções dentro das quais o nome pode ser utilizado.

(c) (0.5) Objeto.

Resposta:

Uma entidade com estado interno (um conjunto de nomes), o qual só pode ser acedido por um conjunto de operações definidas dentro do objeto (os métodos).

3. (1.0) Escreva a função `soma_divisores` que recebe um número inteiro positivo n , e tem como valor a soma de todos os divisores de n . No caso de n ser 0 deverá devolver 0. Não é necessário validar a correção do argumento. Por exemplo,

```
>>> soma_divisores(20)
42
>>> soma_divisores(13)
14
```

Resposta:

```
def soma_divisores(n):
    res = 0
    for i in range(1, n+1):
        if n%i == 0:
            res = res + i
    return res
```

4. (1.0) Escreva a função `codifica` que recebe um número inteiro positivo e que devolve uma codificação para esse número do seguinte modo: (a) cada algarismo par é substituído pelo algarismo par anterior, entendendo-se que o algarismo par anterior a 0 é o 8; (b) cada algarismo ímpar é substituído pelo algarismo ímpar seguinte, entendendo-se que o algarismo ímpar seguinte a 9 é o 1; (c) o número obtido é invertido. Não pode recorrer a cadeias de caracteres nem a listas. Não é necessário verificar a correcção do argumento. Por exemplo,

```
>>> codifica(2095)
7180
```

Resposta:

```
def codifica(num):
    res = 0
    while num != 0:
        digito = num % 10
        num = num // 10
        if digito % 2 == 0:
            if digito == 0:
                digito = 8
            else:
                digito = digito - 2
        else:
            digito = (digito + 2) % 10
        res = res * 10 + digito
    return res
```

5. (1.0) Escreva a função, `junta_ordenados`, que recebe dois tuplos contendo inteiros, o primeiro ordenado por ordem crescente e o segundo ordenado por ordem decrescente e devolve um tuplo ordenado por ordem crescente com os elementos dos dois tuplos. Não pode usar algoritmos de ordenação. Não é necessário validar os argumentos. Por exemplo,

```
>>> junta_ordenados((5, 6, 23, 45), (90, 67, 56, 34, 24, 6, 1))
(1, 5, 6, 6, 23, 24, 34, 45, 56, 67, 90)
```

Resposta:

```
def junta_ordenados(t1, t2):

    def inverta(t):
        res = ()
        for i in range(len(t)-1, -1, -1):
            res = res + (t[i], )
        return res

    i1 = 0
    i2 = len(t2) - 1
    res = ()
    while i1 < len(t1) and i2 >= 0:
        if t1[i1] < t2[i2]:
            res = res + (t1[i1], )
            i1 = i1 + 1
        else:
            res = res + (t2[i2], )
            i2 = i2 - 1
    res = res + t1[i1:] + inverta(t2[0:i2+1])
    return res
```

6. Considere a seguinte gramática em notação BNF, na qual o símbolo inicial é $\langle \text{prim} \rangle$:

$\langle \text{prim} \rangle ::= \langle \text{letra} \rangle \langle \text{numeros} \rangle \langle \text{letras} \rangle$

$\langle \text{letras} \rangle ::= \langle \text{letra} \rangle |$
 $\langle \text{letra} \rangle \langle \text{letras} \rangle$

$\langle \text{numeros} \rangle ::= \langle \text{num} \rangle |$
 $\langle \text{num} \rangle \langle \text{numeros} \rangle$

$\langle \text{letra} \rangle ::= A | B | C | D$

$\langle \text{num} \rangle ::= 1 | 2 | 3 | 4$

(a) (0.5) Indique os símbolos terminais e os símbolos não terminais da gramática.

Símbolos terminais:

Resposta:

A, B, C, D, 1, 2, 3, 4

Símbolos não terminais:

Resposta:

$\langle \text{prim} \rangle$, $\langle \text{letras} \rangle$, $\langle \text{numeros} \rangle$, $\langle \text{letra} \rangle$, $\langle \text{num} \rangle$

(b) (0.5) Indique, justificando no caso de não pertencer, quais das seguintes expressões pertencem ou não pertencem ao conjunto de frases da linguagem definida pela gramática:

i. 234B

Resposta:

Não pertence pois falta uma letra antes do número e pelo menos uma letra depois do número.

ii. A1A

Resposta:

Pertence.

iii. ABCD

Resposta:

Não pertence pois falta pelo menos um número entre o A e o B.

iv. letrasnumeros

Resposta:

Não pertence pois `letrasnumeros` não é um símbolo terminal.

v. D4444444444441234AAAAAAAA

Resposta:

Pertence.

- (c) (1.5) Escreva o predicado `reconhece`, que recebe como argumento uma cadeia de caracteres e devolve *verdadeiro* apenas se o seu argumento corresponde a uma frase da linguagem definida pela gramática. Não é necessário validar a validade do argumento. Por exemplo,

```
>>> reconhece('B1ABC')
True
>>> reconhece('1234C')
False
```

Resposta:

```
def reconhece(frase):
    if frase[0] not in 'ABCD':
        return False
    i = 1
    while i <= len(frase)-1 and frase[i] in '1234':
        i = i + 1
    if i > 1 and i != len(frase): # foi encontrada pelo menos um
                                   # numero e ainda resta qualquer
                                   # coisa para analisar
        while i <= len(frase)-1 and frase[i] in 'ABCD':
            i = i + 1
        return i == len(frase)
    else:
        return False
```

7. (1.0) Escreva a função, `soma_cumulativa`, que recebe uma lista de números e que devolve uma lista que contém a soma cumulativa da lista recebida, ou seja, o elemento na posição `i` da lista devolvida contém a soma de todos os elementos da lista original nas posições de 0 a `i`. Não é necessário validar os dados de entrada. Por exemplo,

```
>>> soma_cumulativa([1, 2, -7, 8, 10])
[1, 3, -4, 4, 14]
```

Resposta:

```
def soma_cumulativa(lst):
    res = []
    soma = 0
    for i in range (len(lst)):
        soma = soma + lst[i]
        res = res + [soma]
    return res
```

8. (1.5) Escreva a função `conta_palavras` que recebe uma cadeia de caracteres correspondente a um texto e que produz um dicionário em que as chaves são todas as palavras que este contém e o valor associado corresponde ao número de vezes que essa palavra aparece no texto. Assuma que entre as palavras podem existir vários espaços em branco, bem como no início e no fim da cadeia de caracteres. Pode assumir que todos os caracteres são minúsculos e que não existem símbolos de pontuação. Não é necessário validar o argumento. Por exemplo,

```
>>> cc = 'a aranha arranha a ra a ra arranha a aranha ' \
        + 'nem a aranha arranha a ra nem a ra arranha a aranha'
>>> conta_palavras(cc)
{'aranha': 4, 'arranha': 4, 'ra': 4, 'a': 8, 'nem': 2}
```

Resposta:

```
def conta_palavras(cc):
    def coloca_dic(d, pal):
        if pal in d:
            d[pal] = d[pal] + 1
        else:
            d[pal] = 1
    # ignora os eventuais brancos no início
    i = 0
    while cc[i] == ' ':
        i = i + 1
    res, palavra, ant = {}, '', ''
    for i in range(i, len(cc)):
        if cc[i] != ' ' or ant != ' ':
            if cc[i] == ' ' and ant != ' ':
                coloca_dic(res, palavra)
                palavra = ''
            else:
                palavra = palavra + cc[i]
        ant = cc[i]
    # a última palavra, se não seguida de brancos, é
    # colocada no dicionário
    if cc[len(cc) - 1] != ' ':
        coloca_dic(res, palavra)
    return res
```

9. (1.0) Escreva a função `val_serie` que calcula o valor aproximado da série:

$$\sum_{i=0}^{\infty} \frac{x^i}{i!}$$

para um dado valor de x e com uma precisão inferior a 0.001. O seu programa deve ter em atenção que o i -ésimo termo da série pode ser obtido do termo na posição $i - 1$, multiplicando-o por x/i . Por esta razão, não pode usar as funções potência nem fatorial. Não é necessário validar o argumento. Por exemplo,

```
>>> val_serie(2)
7.3887125220458545
>>> val_serie(0)
1
```

Resposta:

```
def val_serie(x):
    soma = 0
    termo = 1
    pos_termo = 0
    while termo >= 0.001:
        soma = soma + termo
        pos_termo = pos_termo + 1
        termo = termo * x / pos_termo
    return soma
```

10. Escreva a função `soma_pares` que recebe uma lista contendo números inteiros (não é necessário verificar a validade do argumento) e que devolve a soma dos números pares da lista.

(a) (1.0) Usando um processo iterativo.

Resposta:

```
def soma_pares(lst):
    soma = 0
    for el in lst:
        if el % 2 == 0:
            soma = soma + el
    return soma
```

(b) (1.0) Usando recursão com operações adiadas. NOTA: não pode usar atribuição nem ciclos.

Resposta:

```
def soma_pares(lst):
    if lst == []:
        return 0
    elif lst[0] % 2 == 0:
        return lst[0] + soma_pares(lst[1:])
    else:
        return soma_pares(lst[1:])
```

(c) (1.0) Usando recursão de cauda. NOTA: não pode usar atribuição nem ciclos.

Resposta:

```
def soma_pares(lst):

    def soma_pares_aux(lst, res):
        if lst == []:
            return res
        elif lst[0] % 2 == 0:
            return soma_pares_aux(lst[1:], res+lst[0])
        else:
            return soma_pares_aux(lst[1:], res)

    return soma_pares_aux(lst, 0)
```

(d) (1.0) Usando um ou mais dos funcionais sobre listas (`filtra`, `transforma`, `acumula`). O corpo da sua função apenas pode ter uma instrução, a instrução `return`.

Resposta:

```
def soma_pares(lst):
    return acumula(lambda x, y: x + y, \
                  filtra(lambda x: x % 2 == 0, lst))
```

11. Suponha que desejava criar o tipo *vetor*. Um vetor num referencial cartesiano pode ser representado pelas coordenadas da sua extremidade (x, y) , estando a sua origem no ponto $(0, 0)$. Podemos considerar as seguintes operações básicas para vetores:

- *Construtor:*

- $vetor : \mathbb{R}^2 \mapsto vetor$
 $vetor(x, y)$ tem como valor o vetor cuja extremidade é o ponto (x, y) .

- *Seletores:*

- $abscissa : vetor \mapsto \mathbb{R}$
 $abscissa(v)$ tem como valor a abscissa da extremidade do vetor v .
- $ordenada : vetor \mapsto \mathbb{R}$
 $ordenada(v)$ tem como valor a ordenada da extremidade do vetor v .

- *Reconhecedores:*

- $eh_vetor : universal \mapsto lógico$
 $eh_vetor(arg)$ tem valor *verdadeiro* apenas se arg é um vetor.
- $eh_vetor_nulo : vetor \mapsto lógico$
 $eh_vetor_nulo(v)$ tem valor *verdadeiro* apenas se v é o vetor $(0, 0)$.

- *Teste:*

- $vetores_iguais : vetor \times vetor \mapsto lógico$
 $vetores_iguais(v_1, v_2)$ tem valor *verdadeiro* apenas se os vetores v_1 e v_2 são iguais.

(a) (0.5) Defina uma representação para vetores utilizando dicionários.

Resposta:

$$\mathfrak{R}[(x, y)] = \{ 'x' : x, 'y' : y \}.$$

(b) (1.5) Escreva as operações básicas, de acordo com a representação escolhida. Apenas o construtor deve validar o argumento.

Resposta:

```
def vetor(x, y):
    if isinstance(x, (int, float)) and isinstance(y, (int, float)):
        return { 'x' : x, 'y' : y }
    else:
        raise ValueError('vetor: argumentos incorrectos')

def abscissa(v):
    return v['x']

def ordenada(v):
    return v['y']

def eh_vetor(arg):
    return isinstance(arg, dict) and len(arg) == 2 and \
           'x' in arg and isinstance(arg['x'], (int, float)) and \
           'y' in arg and isinstance(arg['y'], (int, float))
```

```
def eh_vetor_nulo(v):
    return abs(v['x'] < 0.001) and abs(v['y'] < 0.001)

def vetores_iguais(v1, v2):
    return abs(v1['x'] - v2['x']) < 0.001 and \
           abs(v1['y'] - v2['y']) < 0.001
```

- (c) (0.5) Escreva uma função de alto nível para calcular a soma de dois vetores. A soma dos vetores (a, b) e (c, d) é dada pelo vetor $(a + c, b + d)$.

Resposta:

```
def soma_vetores(v1, v2):
    return vetor(abcissa(v1) + abcissa(v2), \
                ordenada(v1) + ordenada(v2))
```

12. (2.0) Defina a classe `estacionamento` que simula o comportamento de um parque de estacionamento. O parque tem uma dada capacidade e uma tarifa que depende do tempo em que um automóvel esteve estacionado, de acordo com a seguinte tabela:

primeira meia hora	€0,50
primeira hora	€1,00
todas as horas ou frações seguintes	€0,75

Por exemplo, se um automóvel esteve estacionado 3 horas e 20 minutos irá pagar €3,25 (€1,00 pela primeira hora e €3 * 0,75 pelas 2 horas e 20 minutos seguintes). Assuma que sempre que entra um automóvel a sua matrícula é associada à hora em que entrou (horas e minutos). Não se considera a data de entrada pois todos os dias à meia noite o parque fica vazio. Sempre que sai um automóvel é indicada a hora em que saiu e calcula-se a quantia a pagar. As instâncias desta classe são criadas indicando a capacidade do parque. As operações disponíveis são as seguintes:

- $entra : str \times \mathbb{N}_0 \times \mathbb{N}_0 \mapsto \mathbb{N}_0$
 $entra(matr, h, m)$, regista que o automóvel com matrícula $matr$ entrou no parque às h horas e m minutos ($0 \leq h \leq 23$ e $0 \leq m \leq 59$). Esta função verifica a legalidade das horas e minutos de entrada. Devolve o número de automóveis no parque. Se o parque estiver cheio gera uma mensagem. Se um automóvel com a mesma matrícula já estiver no parque, gera uma mensagem e o automóvel não entra.
- $sai : str \times \mathbb{N}_0 \times \mathbb{N}_0 \mapsto \mathbb{R}_0$
 $sai(matr, h, m)$, regista que o automóvel com matrícula $matr$ saiu do parque às h horas e m minutos ($0 \leq h \leq 23$ e $0 \leq m \leq 59$). Esta função verifica a legalidade das horas e minutos de saída, comparando-a com as horas e minutos de entrada. Devolve o valor a pagar. Se o parque estiver vazio gera uma mensagem. Se o automóvel com a matrícula não estiver no parque, gera uma mensagem.
- $lugares_livres : \{\} \mapsto \mathbb{N}_0$
 $lugares_livres()$, devolve o número de lugares livres do parque.

Por exemplo,


```
>>> e = estacionamento(2)
>>> e.entra('AH-51-83', 5, 12)
1
>>> e.entra('AH-51-83', 12, 30)
'entra: o carro está no parque'
>>> e.sai('AH-51-83', 5, 10)
'sai: horas erradas'
>>> e.entra('15-42-DA', 9, 27)
0
>>> e.entra('15-SR-60', 9, 31)
'entra: parque cheio'
>>> e.lugares_livres()
0
>>> e.sai('15-42-DA', 9, 27)
'valor a pagar: Euro 0.5'
e.entra('15-SR-60', 9, 31)
0
>>> e.sai('AH-51-83', 17, 35)
'valor a pagar: Euro 10.0'
```

Resposta:

```
class estacionamento:

    def __init__(self, capacidade):
        '''
        self.cp é um inteiro que representa a capacidade do parque;
        self.parque é un dicionário cujas chaves são as matrículas
        dos automóveis estacionados e cujos valores são um tuplo
        contendo a hora e minutos em que o automóvel entrou no parque
        '''
        if isinstance(capacidade, int):
            self.cp = capacidade
            self.parque = {}
        else:
            raise ValueError ('estacionamento: capacidade não é inteiro')

    def dados_corretos(self, matr, h, m):
        return isinstance(matr, str) and isinstance(h, int) and \
            isinstance(m, int) and 0 <= h <= 23 and 0 <= m <= 59

    def lugares_livres(self):
        return self.cp - len(self.parque)

    def entra(self, matr, h, m):

        if self.dados_corretos(matr, h, m):
            if self.cp - len(self.parque) > 0:
                if matr in self.parque:
                    return 'entra: o carro está no parque'
                else:
                    self.parque[matr] = (h, m)
                    return self.cp - len(self.parque)
            else:
```

```
        return 'entra: parque cheio'
    else:
        return 'entra: dados errados'

def sai(self, matr, h, m):

    def calcula_preco(h_e, m_e, h_a, m_a):

        def ceil(n):
            if n - int(n) == 0.0:
                return int(n)
            else:
                return int(n) + 1

        if self.dados_corretos(matr, h, m):
            if len(self.parque) > 0:
                min_estac = (h_a * 60 + m_a) - (h_e * 60 + m_e)
                if min_estac < 0:
                    return 'sai: horas erradas'
                elif min_estac <= 30:
                    return 0.5
                elif min_estac <= 60:
                    return 1.0
                else:
                    return 1.0 + ceil((min_estac / 60) - 1) * 0.75
            else:
                return 'sai: o parque está vazio'
        else:
            return 'sai: dados errados'

    if matr in self.parque:
        calc_pag = calcula_preco(self.parque[matr][0], \
                                self.parque[matr][1], \
                                h, \
                                m)
        if isinstance(calc_pag, float):
            del(self.parque[matr])
            return 'valor a pagar: Euro ' + str(calc_pag)
        else:
            return calc_pag
    else:
        return 'sai: matricula inexistente'
```