TÉCNICO LISBOA

Fundamentos da Programação

Solução do Primeiro Teste

26 de Outubro de 2013

09:00–10:30

1. **(1.0)** O que é um processo computacional? Qual a relação entre um programa e um processo computacional?

   Resposta:

   Um processo computacional é um ente imaterial que existe dentro de um computador durante a execução de um programa, e cuja evolução ao longo do tempo é ditada pelo programa.

2. (a) **(0.7)** Diga o que é um algoritmo.

   Resposta:

   Um algoritmo é uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito com uma quantidade de esforço finita.

   (b) **(0.8)** Quais são as características de um algoritmo?

   Resposta:

   i. *Um algoritmo é rigoroso*. Cada instrução do algoritmo deve especificar exacta e rigorosamente o que deve ser feito, não havendo lugar para ambiguidade.

   ii. *Um algoritmo é eficaz*. Cada instrução do algoritmo deve ser suficientemente básica e bem compreendida de modo a poder ser executada num intervalo de tempo finito, com uma quantidade de esforço finita.

   iii. *Um algoritmo deve terminar*. O algoritmo deve levar a uma situação em que o objectivo tenha sido atingido e não existam mais instruções para ser executadas.

3. Considere a linguagem cujas frases começam pelo símbolo `a`, o qual é seguido por um número par de ocorrências da sequência dos símbolos `bcd`, após o que terminam com o símbolo `e`. Por exemplo `abcdbcde` e `abcdbcdbcdbcde` são frases da linguagem, `ae` e `abcde` não o são.

   (a) **(1.0)** Escreva uma gramática em notação BNF para a linguagem apresentada.

   Resposta:

   ⟨frase⟩ ::= `a` ⟨meio⟩$^+$ `e`

   ⟨meio⟩ ::= `bcdbcd`

   (b) **(0.5)** Diga quais são os símbolos terminais e não terminais da sua linguagem.

   Resposta:

   Símbolos terminais: `a`, `b`, `c`, `d`, `e`.

   Símbolos não terminais: ⟨frase⟩, ⟨meio⟩

4. Suponha que num programa em Phython se efectuam as atribuições:

```
a = (1, 2, 3, (4, 5))
b = ['a', 'b', 'c']
```

Para cada uma das seguintes instruções, diga o que é feito pelo Python. Se alguma das instruções gerar um erro (sintáctico ou semântico) explique a razão do erro. Assuma que todas as alíneas correspondem a instruções que são executadas imediatamente após as atribuições acima.

(a) **(0.5)**

```
b[1], b[2] = a[2], a[1]
```

Resposta:

A lista `b` passa a ser `['a', 3, 2]`.

(b) **(0.5)**

```
a[1] = 10
```

Resposta:

Origina um erro pois os tuplos são imutáveis.

(c) **(0.5)**

```
del(b[0])
```

Resposta:

A lista `b` passa a ser `['b', 'c']`.

(d) **(0.5)**

```
if b[2] = 5:
   a = 12
```

Resposta:

Origina um erro sintáctico pois `b[2] = 5` não é uma condição.

5. O Python apresenta duas alternativas para a instrução de atribuição, a atribuição simples e a atribuição múltipla.

(a) **(1.0)** Usando a notação BNF, defina a sintaxe da instrução de atribuição *simples*.

Resposta:

⟨instrução de atribuição simples⟩ ::= ⟨nome⟩ = ⟨expressão⟩

Nesta definição, ⟨nome⟩ corresponde a qualquer nome em Python e ⟨expressão⟩ corresponde a uma expressão em Python.

(b) **(1.0)** Defina a semântica da instrução de atribuição *simples*.

Resposta:

Ao encontrar uma instrução da forma ⟨nome⟩ = ⟨expressão⟩, o Python começa por avaliar a ⟨expressão⟩ após o que associa ⟨nome⟩ ao valor da ⟨expressão⟩. A execução de uma instrução de atribuição não devolve nenhum valor, mas sim altera o valor de um nome ou cria um nome se este não existir.

6. **(1.0)** Escreva um programa em Python que lê um número inteiro, verifica que o valor lido é um inteiro, e se este for par, escreve metade do número lido e se for ímpar escreve o dobro do número lido.

Resposta:

```
v = eval(input('Escreva um numero inteiro\n> '))
if isinstance(v, int):
    if v % 2 == 0:
        print(v // 2)
    else:
        print(v * 2)
else:
    raise ValueError ('o numero nao e inteiro')
```

7. **(1.5)** Escreva um programa em Python que vai pedindo ao utilizador que forneça números inteiros (não é necessário verificar se o número fornecido é um inteiro). Quando o utilizador fornecer o número `0`, o seu programa escreve um real que corresponde à média dos números lidos. Por exemplo,

```
Escreva um numero inteiro
(0 para terminar)
> 6
Escreva um numero inteiro
(0 para terminar)
> 4
Escreva um numero inteiro
(0 para terminar)
> 0
Media: 5.0
```

Resposta:

```
soma = 0
quant = 0
print('Escreva um numero inteiro\n(0 para terminar)')
n = eval(input('> '))
while n != 0:
    soma = soma + n
    quant = quant + 1
    print('Escreva um numero inteiro\n(0 para terminar)')
    n = eval(input('> '))
if quant == 0:
    print('Nao foram introduzidos numeros')
else:
    print('Media:', soma / quant)
```

8. **(1.5)** Diga o que é escrito pela seguinte instrução:

```
for i in range(1):
    for j in range(3, 5):
        for k in range(5, 1, -2):
            if (i + j) % 2 == 0:
                print(i, j, k)
```

Resposta:

```
0 4 5
0 4 3
```

9. **(1.5)** Escreva uma função em Python com o nome `numero_algarismos_impares` que recebe um inteiro positivo, `n`, e devolve o número de algarismos ímpares de `n`. Por exemplo:

```
>>> numero_algarismos_impares(2233456)
3
>>> numero_algarismos_impares(2)
0
```

Resposta:

```
def numero_algarismos_impares(num):
    algarismos = 0
    while num != 0:
        alg = num % 10
        if alg % 2 != 0:
            algarismos = algarismos + 1
        num = num // 10
    return algarismos
```

10. **(1.5)** Escreva uma função chamada `parte` que recebe como argumentos uma lista, `lst`, e um elemento, `e`, e que devolve uma lista de dois elementos, contendo na primeira posição a lista com os elementos de `lst` menores que `e`, e na segunda posição a lista com os elementos de `lst` maiores ou iguais a `e`. Não é necessário validar os argumentos da sua função. Por exemplo,

```
>>> parte([2, 0, 12, 19, 5], 6)
[[2, 0, 5], [12, 19]]
>>> parte([7, 3, 4, 12], 3)
[[], [7, 3, 4, 12]]
```

Resposta:

```
def parte(lst, e):
    menores = []
    maiores = []
    for el in lst:
        if el < e:
            menores = menores + [el]
        else:
            maiores = maiores + [el]
    return [menores, maiores]
```

11. **(1.5)** Escreva uma função em Python com o nome `junta_ordenadas` que recebe como argumentos duas listas ordenadas por ordem crescente e devolve uma lista também ordenada com os elementos das duas listas. Não é necessário validar os argumentos da sua função. Por exemplo,

```
junta_ordenadas([2, 5, 90], [3, 5, 6, 12])
[2, 3, 5, 5, 6, 12, 90]
```

Resposta:

```
def junta_ordenadas(l1, l2):
    res = []
    i = 0
    j = 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            res = res + [l1[i]]
            i = i + 1
        else:
            res = res + [l2[j]]
            j = j + 1
    #uma das listas foi totalmete processada
    if i == len(l1): # trata dos elementos por juntar em l2
        res = res + l2[j:]
    if j == len(l2): # trata dos elementos por juntar em l1
        res = res + l1[i:]
    return res
```

12. **(1.5)** Escreva uma função em Python com o nome `remove_repetidos` que recebe uma lista e devolve a lista obtida da lista original em que todos os elementos repetidos foram removidos. Por exemplo,

```
>>> remove_repetidos([2, 4, 3, 2, 2, 2, 3])
[2, 4, 3]
>>> remove_repetidos([2, 5, 7])
[2, 5, 7]
```

Resposta:

```
def remove_repetidos(l):
    for i in range(len(l)-1, 1, -1):
        if l[i] in l[0:i-1]:
            del(l[i])
    return l
```

13. **(2.0)** Considere a linguagem do exercício 3: as frases começam pelo símbolo `a`, o qual é seguido por um número par de ocorrências da sequência dos símbolos `bcd`, após o que terminam com o símbolo `e`. Por exemplo `abcdbcde` e `abcdbcdbcdbcde` são frases da linguagem, `ae` e `abcde` não o são.

Escreva em Python uma função, chamada `reconhece`, que recebe uma cadeia de caracteres e tem o valor `True` se o seu agrumento é uma frase da linguagem e `False` se o seu argumento não é uma frase da linguagem. Por exemplo,

```
>>> reconhece('abbb')
False
>>> reconhece('abcdbcde')
True
>>> reconhece('abcdbcdbcde')
False
```

Resposta:

```
def reconhece(f):
    if len(f) < 8:
        return False
    if f[0] == 'a' and f[-1] == 'e':
        # numero caracteres entre 'a' e 'e'
        num_c_meio = len(f) - 2
        if num_c_meio % 6 == 0:
            # o numero destes caracteres é multiplo de 6
            for i in range(1, num_c_meio, 6):
                if f[i:i+6] != 'bcdbcd':
                    return False
            return True
        else:
            return False
    else:
        return False
```