Guião 1
Programação em Python

Ano Lectivo de 2016/2017

©Luís Seabra Lopes

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

Última actualização: 2016-10-12

I Objectivos

O presente guião centra-se na programação em Python, dando particular destaque ao processamento de listas e tuplos, programação ao estilo funcional, usando recursividade e expressões-*lambda*, e programação orientada a objectos. Este guião é usado nas disciplinas de *Inteligência Artificial*, da *Licenciatura em Engenharia Informática*, e *Introdução à Inteligência Artificial*, do *Mestrado Integrado em Engenharia de Computadores e Telemática*.

Este guião será parcialmente realizado nas primeiras 2 a 3 aulas práticas do semestre. Aconselha-se que os alunos completem o guião no âmbito do trabalho regular a realizar fora das aulas. Os exercícios que estejam no âmbito temático de uma dada aula devem ser completados antes da aula seguinte.

II Desenvolvimento de funções

Desenvolva funções para realizar as operações descritas em seguida. Embora seja possível construir soluções iterativas, reforce o seu domínio da programação recursiva, desenvolvendo soluções recursivas para os exercícios propostos. Ficará assim melhor preparado para implementação de algoritmos de inteligência artificial, muitos dos quais são recursivos.

Nota: Na ausência de indicação expressa em contrário, as funções que desenvolver não devem modificar os valores passados como parâmetros de entrada.

1 Funções para processamento de listas

1. Dada uma lista, retorna o seu comprimento.
2. Dada uma lista de números, retorna a respectiva soma.

3. Dada uma lista e um elemento, verifica se o elemento ocorre na lista. Retorna um valor booleano.

4. Dadas duas listas, retorna a sua concatenação.

   Nota: Embora seja possível programar a concatenação de forma recursiva, as listas de Python suportam operações de modificação que permitem implementar a concatenação sem usar qualquer tipo de processamento iterativo ou recursivo.

5. Dada uma lista, retorna a sua inversa.

6. Dada uma lista, verifica se forma uma capicua, ou seja, se, quer se leia da esquerda para a direita ou vice-versa, se obtêm a mesma sequência de elementos.

7. Dada uma lista de listas, retorna a concatenação dessas listas.

8. Dada uma lista, um elemento x e outro elemento y, retorna uma lista similar à lista de entrada, na qual x é substituido por y em todas as ocorrências de x.

9. Dadas duas listas ordenadas de números, calcular a união ordenada mantendo eventuais repetições.

10. Dado um conjunto, na forma de uma lista, retorna uma lista de listas que representa o conjunto de todos os subconjuntos do conjunto dado.

2 Funções para processamento de listas e tuplos

1. Dada uma lista de pares, produzir um par com as listas dos primeiros e segundos elementos desses pares.

   `separar([(a1, b1), ... (an, bn)]) = ([a1, ... an], [b1, ... bn])`

2. Dada uma lista l e um elemento x, retorna um par formado pela lista dos elementos de l diferentes de x e pelo número de ocorrências x em l.

   Exemplo:

   ```
   >>> remove_e_conta([1,6,2,5,5,2,5,2],2)
   ([1,6,5,5,5],3)
   ```

3. Dada uma lista, retorna o número de ocorrências de cada elemento, na forma de uma lista de pares (*elemento*,*contagem*).

3 Funções que retornam *None*

As operações a seguir descritas não podem ser realizadas para alguns valores de entrada. Nessas situações, as funções que vai desenvolver devem retornar *None*.

1. Dada uma lista, retornar o elemento que está à cabeça (ou seja, na posição 0).

2. Dada uma lista, retornar a sua cauda (ou seja, todos os elementos à excepção do primeiro).

3. Dado um par de listas com igual comprimento, produzir uma lista dos pares dos elementos homólogos.

4. Dada uma lista de números, retorna o menor elemento.

5. Dada uma lista de números, retorna um par formado pelo menor elemento e pela lista dos restantes elementos.

6. Dada uma lista de números, calcular o máximo e o mínimo, retornando-os num tuplo.

7. Dada uma lista de números, retorna um triplo formado pelos dois menores elementos e pela lista dos restantes elementos.

8. Dada uma lista ordenada de números, calcular se possível a respectiva média e mediana, retornando-as num tuplo.

4 Expressões-*lambda* e funções de ordem superior

Nesta secção, vai exercitar os seus conhecimentos sobre alguns dos mecanismos de programação funcional disponíveis na linguagem Python, nomeadamente expressões-*lambda*, funções de ordem superior, listas de compreensão (*list comprehensions*) e as funções pré-definidas *map*, *filter* e *reduce*. Desenvolva então funções para realizar as operações descritas em seguida.

1. (Implementar na forma de uma expressão-*lambda*:) Dado um número inteiro, retorna *True* caso o número seja ímpar, e *False* caso contrário.

2. (Implementar na forma de uma expressão-*lambda*:) Dado um número, retorna *True* caso o número seja positivo, e *False* caso contrário.

3. (Implementar na forma de uma expressão-*lambda*:) Dados dois números, x e y, retorna *True* caso $|x| < |y|$, e *False* caso contrário.

4. (Implementar na forma de uma expressão-*lambda*:) Dado um par (x, y), representando coordenadas cartesianas de um ponto no plano XY, retorna um par (r, θ), correspondente às coordenadas polares do mesmo ponto.

5. Dadas três funções de duas entradas, f, g e h, retorna uma função de três entradas, x, y e z, cujo resultado é dado por: $h(f(x, y), g(y, z)$.

6. Dada uma lista e uma função booleana unária, retorna *True* caso a função também retorne *True* para todos os elementos da lista, e *False* caso contrário. (Quantificador universal)

7. Dada uma lista e uma função booleana unária, retorna *True* caso a função retorne *True* para pelo menos um dos elementos da lista, e *False* caso contrário. (Quantificador existencial)

8. Dadas duas listas, retorna *True* se todos os elementos da primeira lista também ocorrem na segunda, e *False* caso contrário. (subconjunto)

9. Dada uma lista com pelo menos um elemento e uma relação de ordem (ou seja, uma função booleana binária usada para comparação elemento a elemento), retorna o menor elemento da lista de acordo com essa relação de ordem.

10. Dada uma lista com pelo menos um elemento e uma relação de ordem, retorna um par contendo o menor elemento da lista, de acordo com essa relação de ordem, e uma lista com os restantes elementos.

11. Dada uma lista com pelo menos dois elementos e uma relação de ordem, retorna um triplo contendo os dois menores elementos da lista, de acordo com essa relação de ordem, e uma lista com os restantes elementos.

12. Dada uma lista de pares (x,y), representando coordenadas cartesianas de pontos no plano XY, produzir uma lista de pares (r, θ), correspondentes às coordenadas polares dos mesmos pontos.

13. Dadas duas listas e uma relação de ordem, e partindo do pressuposto de que essas listas estão ordenadas segundo essa relação de ordem, retorna a junção ordenada das listas de entrada, de acordo com a mesma relação de ordem, mantendo eventuais repetições.

14. Dada uma lista de listas e uma função, aplica a função a cada um dos elementos das listas, retornando a concatenação das listas resultantes.

    conc_aplic ([[x11 ,..., x1n],...,[xm1 ,..., xmk]], f)= [f(x11),..., f(x1n),..., f(xm1),..., f(xmk)]

15. Dado um par de listas e uma função binária, retorna uma lista com os resultados da aplicação da função aos pares de elementos homólogos das duas listas.

    Ou seja: aplic_combin (([a1 ,..., an],[b1 ,..., bn]), F) devolve [F(a1,b1),..., F(an,bn)]

    Caso as listas não tenham o mesmo comprimento, retorna *None*.

16. Dada uma lista de listas $[[x_{11}, ..., x_{1n}], ..., [x_{m1}, ..., x_{mk}]]$, uma função e o seu elemento neutro, produzir a lista $[y_1, ..., y_n]$ em que y_i é a redução de $[x_{i1}, ..., x_{in}]$ através da função.

5 Ordenação de listas

1. Dada uma lista de números, calcular a lista ordenada pelos seguintes algoritmos:
   (a) Ordenação por selecção (*selection sort*);
   (b) Ordenação por borbulhamento (*bubble sort*);
   (c) Ordenação rápida (*quick sort*)

2. Similar ao número anterior, mas sem restrição no tipo dos elementos da lista de entrada. A função de ordenação recebe, num parâmetro adicional, a relação de ordem (uma função binária booleana para comparação elemento a elemento) segundo a qual a lista de entrada deve ser ordenada.

III Desenvolvimento de classes

Em seguida, são apresentados alguns problemas cuja resolução se torna mais fácil utilizando classes e métodos. Nestes problemas, utilize soluções recursivas ou iterativas segundo o que lhe parecer mais adequado.

1 Expressões artiméticas

As expressões artiméticas aqui consideradas contêm constantes, diversas (0 ou mais) ocorrências de uma mesma variável e operações de soma e multiplicação:
Exemplos:

$$73$$
$$x$$
$$3 + 4 * 7$$
$$2 * x + 1$$
$$2 + x * (x + 1)$$

Neste contexto, faça o seguinte:

1. Modele as expressões aritméticas através de um conjunto de classes e desenvolva os respectivos métodos de inticialização (__init__ ()) e conversão para cadeia de caractéres (__str__ ()).

   Sugestão: Desenvolva uma classe para cada um dos quatro tipos de expressões: constante, variável, soma e produto.

2. Para cada tipo de expressão, desenvolva um método que, dado um valor da variável, avalia a expressão e retorna o resultado.

3. Para cada tipo de expressão, desenvolva um método que a simplifica. Os casos de simplificação a considerar são os seguintes: elementos neutros da soma e da multiplicação; elemento aboservente da multiplicação. É retornada uma expressão equivalente mais simples.

4. Para cada tipo da expressão, desenvolva um método que implemente a derivação dessa expressão em ordem à variável, retornando a derivada já simplificada pelo método anterior.

2 Relações familiares e genealogia

Pretende-se desenvolver um programa que seja capaz de representar uma árvore genealógica e que possa responder a perguntas sobre as relações existentes.

1. Proponha classes para representar as relações familiares $Pai(x, y)$ e $Mae(x, y)$, juntamente os respectivos métodos de inicialização e conversão para cadeia de caractéres.

2. Proponha uma classe para representar uma família (ou genealogia) através de uma colecção de relações familiares, juntamente os respectivos métodos de inicialização e conversão para cadeia de caractéres.

3. Desenvolva um método que, dado um indivíduo, retorna um par formado pelos seus pais.

4. Desenvolva um método que, dado um indivíduo, retorna uma lista com os seus filhos.

5. Desenvolva um método que, dados dois indivíduos, retorna *True* caso o primeiro seja progenitor do segundo, e *False* caso contrário.

6. Desenvolva um método que, dados dois indivíduos, retorna *True* caso o primeiro seja antepassado do segundo, e *False* caso contrário.

3 Árvores binárias

Proponha uma representação para árvores binárias e programe métodos para realizar as operações indicadas a seguir:

1. Verificar se um elemento é membro de uma árvore binária.

2. Verificar se duas árvores binárias são isomórficas.

3. Verificar se uma árvore A1 é sub-árvore da árvore A2.

4. Gerar a lista de elementos numa dada árvore que verificam uma dada condição.

5. Gerar uma cópia de uma dada árvore.

6. Verificar se uma dada árvore é ordenada.
7. Inserir um valor numa árvore binária ordenada.
8. Determinar o caminho (lista de valores nos vários nós) que liga a raiz a uma dada folha.