Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Programação I / Programação Funcional
Prof. Oberlan Romão

Aula Prática 10
Listas, Strings e HOF

1) Usando recursão, existem basicamente duas formas de percorrermos uma lista:

- usando um parâmetro (variável) opcional para controlar o número de vezes que a função será chamada recursivamente (normalmente indo de 0 até o tamanho da lista);
- usando fatiamento

Por exemplo:

```python
def funcao1(L, i = 0, R = []):
    if i == len(L):
        return R
    else:
        print(f"L[0] = {L[0]} L = {L} R = {R}")
        if L[i] in R:
            return funcao1(L, i + 1, R)
        else:
            return funcao1(L, i + 1, R + [L[i]])

def funcao2(L, R=[]):
    if len(L) == 0:
        return R
    else:
        print(f"L[0] = {L[0]} L = {L} R = {R}")
        if L[0] in R:
            return funcao2(L[1:], R)
        else:
            return funcao2(L[1:], R + [L[0]])
```

a) Qual o objetivo dessas funções? Abra o arquivo `ex1.py` e execute o código para ver alguns exemplos de utilização das funções. Note o que é impresso em cada chamada recursiva das funções. Essa impressão é a mesma nas duas funções? Por quê? O resultado final é o mesmo?

b) Qual o resultado da chamada `funcao1([1, 2, 3], 1)`? Está correto? Por quê?

2) (`ex2.py`) Implemente duas funções recursivas que recebem uma lista L de inteiros e retornam a soma dos elementos de L. A primeira, `soma1`, deve utilizar a variável `i` para controlar o número de vezes que a função será chamada recursivamente. Já a função `soma2` deve usar fatiamento.

3) (ex3.py) O código abaixo ilustra uma forma de gerar os divisores de um número n. Utilizando essa função, implemente outra função, chamada `listaPrimos`, que gere uma lista com os n primeiros números primos. Por exemplo, a execução de `listaPrimos(5)` deve gerar como resultado a lista `[2, 3, 5, 7, 11]`.

```
def divisores(n, L = [], i = 1):
    if i <= n:
        if n % i == 0:
            return divisores(n, L + [i], i + 1)
        else:
            return divisores(n, L, i + 1)
    else:
        return L
```

Qual objetivo de `print([x**2 for x in L])`?

4) Uma forma de gerarmos listas bem comum em Python é usando compreensão de listas. Sua sintaxe é bem simples: `[expressao for variavel in iteravel clausula_if]`. Veja alguns exemplos:

```
N = [0, 1, 2, 3, 4, 5]
S = [x**2 for x in N]
print(S) # [0, 1, 4, 9, 16, 25]

S = [x**2 for x in range(6)] #Usando a função range
print(S) # [0, 1, 4, 9, 16, 25]

L = [abs(x) for x in [-2, -1, 0, 1, 2, 3]]
print(L) # [2, 1, 0, 1, 2, 3]

S = ["laranja", "banana", "uva", "kiwi"]
L = [len(x) for x in S]
print(L) # [7, 6, 3, 4]

L = [x for x in "ABC"]
print(L) # ["A", "B", "C"]

L = [x*3 for x in "ABC"]
print(L) # ["AAA", "BBB", "CCC"]

L = [x for x in [-2, -1, 0, 1, 2, 3] if x >= 0]
print(L) # [0, 1, 2, 3]

S = ["laranja", "banana", "uva", "kiwi"]
L = [x for x in S if len(x) > 5]
print(L) # ["laranja", "banana"]
```

a) Abra o Shell do Python (terminal) e execute esses comandos para ver o resultado.

b) (ex4.py) Usando compreensão de listas, gere uma lista com números primos menores que 30.

5) Um poderoso recurso de Python (e linguagens do paradigma funcional) é permitir a passagem de funções como parâmetro. Tais funções são chamadas de funções de alta ordem (*High Order Functions*). Python oferece, entre outras, duas funções integradas que são de alta ordem: map e filter. Veja alguns exemplos de utilização de cada uma dessas funções:

```
lista1 = [1, 4, 9, 16 , 25]
lista2 = list(map(math.sqrt, lista1))
print(lista2) # [1.0, 2.0, 3.0, 4.0, 5.0]

lista3 = list(map(lambda x: x**2, lista1))
print(lista3) # [1, 16, 81, 256, 625]

L = [4, -9, 25, -36, -49]
print(list(map(lambda x: math.sqrt(abs(x)), L))) # [1.0, 2.0, 3.0, 4.0, 5.0]

print(list(filter(lambda y: y % 2 == 1, L))) # [-9, 25, -49]
print(list(filter(lambda y: y > 0, L))) # [4, 25]
```

Note que para cada elemento da lista passada como parâmetro é aplicada a função também passada como parâmetro. Como você pode imaginar, podemos implementar nosso próprio `map` e/ou `filter`. Veja um exemplo, considerando a função `map`:

```
def myMap(f, x, V = []):
    if len(x) == 0:
        return V
    else:
        a = f(x[0])
        return myMap(f, x[1:], V + [a])

def funcao1(x):
    return x**2

funcao2 = lambda x: x**3

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y1 = myMap(funcao1, x)
y2 = myMap(funcao2, x)
y3 = myMap(lambda a:a**4, x)
print(y1) # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
print(y2) # [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
print(y3) # [1, 16, 81, 256, 625, 1296, 2401, 4096, 6561, 10000]
```

a) A função `myMap` foi implementada recursivamente usando fatiamento, implemente-a usando um parâmetro (variável) opcional para controlar o número de vezes que a função será chamada recursivamente.

b) Quais modificações seriam necessárias para implementar a função `myFilter`?

6) (Esse você não precisa entregar) Abra o arquivo `ex6.py`. Ele é o mesmo arquivo `base.py` do EP2, mas com alguns detalhes/comentários a mais. Caso não tenha começado a implementar o EP2, ele irá te ajudar. Vamos começar a modificá-lo. Primeiramente, mude o nome do arquivo para a sua matrícula. Ao abrir o arquivo você verá vários trechos comentados. Em cada `(<num>)` você deve implementar a função sugerida ou chamar uma determinada função. Note que essa estrutura é bem similar ao fluxograma da Figura 2 do EP2. Agora é com você, divirta-se :)