Python para Aplicações em Eletrônica

Aula 04: Listas, funções e serial

Listas

Uso Básico

- Uso básico
  - **>> lista=[‘A’, ‘B’, ‘C’, 1, 2,’D’]**
  - **>> lista=[]**
- Inserção:
  - **>> lista.append(‘E’)**
- Acesso(**[]** e **[:]**):
- Deleção:
  - **>> del lista[2]**
- Método **len(str)**:
  - **>> print len(a)**

Uso Básico

```
 1 lista=['A',2,'Casas',3.45,7,8,9,'Bola']
 2 print lista
 3 lista.append('Ultimo')
 4 print lista
 5
 6 print lista[2]
 7 print lista[2:6]
 8 print lista[4:]
 9 del lista[2]
10 print lista
11 del lista[:]
12 print lista
```

```
['A', 2, 'Casas', 3.45, 7, 8, 9, 'Bola']
['A', 2, 'Casas', 3.45, 7, 8, 9, 'Bola', 'Ultimo']
Casas
['Casas', 3.45, 7, 8]
[7, 8, 9, 'Bola', 'Ultimo']
['A', 2, 3.45, 7, 8, 9, 'Bola', 'Ultimo']
[]
```

Operadores Básicos

- Os operadores básicos sobre *strings* são:
  - `+` e `*`: Operadores de concatenação
  - `[]` e `[:]`: Operadores de acesso
  - **`in`** e **`not in`**: Operador de pertinência.

```
1 l1=[1,2,3]
2 l2=[4,5,6]
3 print 'l1:',l1,' l2:',l2
4 print 'l1+l2:',l1+l2
5 print '3*l1',3*l1
6 print '2 in l1:', (2 in l1)
7 print '2 not in l2:', (2 not in l2)
```

```
l1: [1, 2, 3]  l2: [4, 5, 6]
l1+l2: [1, 2, 3, 4, 5, 6]
3*l1 [1, 2, 3, 1, 2, 3, 1, 2, 3]
2 in l1: True
2 not in l2: True
```

Funções Básicas

- O Python possui várias funções básicas que podem ser aplicadas a listas.
- As principais são:
  - **`cmp(lista1,lista2)`**: Compara os elementos das listas
  - **`len(lista)`**: Retorna o número de elementos da lista
  - **`max(lista)`** e **`min(lista)`**: Retorna o elemento máximo e mínimo da lista.
  - **`list(seq)`**: Converte uma tupla (ou *string*) em lista

Funções Básicas

```
1 l1=[1,2,3];l2=[4,5,6]
2 l3=[1.0,2,3.0]
3 print 'l1=l2? :',cmp(l1,l2)
4 print 'l1=l3? :',cmp(l1,l3)
5 s='A CASs123'
6 print 's:',s
7 print 'lista s:', list(s)
```

```
l1=l2? : -1
l1=l3? : 0
s: A CASs123
lista s: ['A', ' ', 'C', 'A', 'S', 's', '1', '2', '3']
```

Métodos Básicos

- A classe lista possui vários métodos próprios que podem ser utilizados.
- Os principais são:
  - **`.append(obj)`**, **`.insert(idx,obj)`** e **`.extend(seq)`**: Adcionam elementos a lista.
  - **`.count(obj)`**: Conta quantos elementos **`obj`** existem.
  - **`.pop([idx])`** e **`.remove(obj)`**: Removem elementos da lista.
  - **`.reverse()`** e **`.sort([func])`**: Reordena elementos da lista.

Métodos Básicos

```
 1 lista=[0,1,2,'00', 3]
 2 print lista
 3 lista.append(4)
 4 lista.extend([5,6,7,5,5,'AA',8])
 5 print lista
 6 lista.insert(2,'Aqui')
 7 print lista
 8 print 'Quantos 5::', lista.count(5)
 9 print '.pop()::',lista.pop()
10 print lista
11 print '.pop(2)::',lista.pop(2)
12 print lista
13 print '.remove(5)::',lista.remove(5)
14 print lista
15 lista.reverse()
16 print lista
17 lista.sort()
18 print lista
```

```
[0, 1, 2, '00', 3]
[0, 1, 2, '00', 3, 4, 5, 6, 7, 5, 5, 'AA', 8]
[0, 1, 'Aqui', 2, '00', 3, 4, 5, 6, 7, 5, 5, 'AA', 8]
Quantos 5:: 3
.pop():: 8
[0, 1, 'Aqui', 2, '00', 3, 4, 5, 6, 7, 5, 5, 'AA']
.pop(2):: Aqui
[0, 1, 2, '00', 3, 4, 5, 6, 7, 5, 5, 'AA']
.remove(5):: None
[0, 1, 2, '00', 3, 4, 6, 7, 5, 5, 'AA']
['AA', 5, 5, 7, 6, 4, 3, '00', 2, 1, 0]
[0, 1, 2, 3, 4, 5, 5, 6, 7, '00', 'AA']
```

Definição de Funções

Uso Básico

- Uso básico
  - **>> def nome_da_func([args]):**

    #aqui é definida a sua função

    [return valor]
  - **>> a = nome_da_funcao([args])**

```
1 # Funcao Basica
2 def areaRect(b,h):
3     res=b*h
4     return res
5
6 print 'A area eh:',areaRect(10,5),' m^2'
```

```
A area eh: 50  m^2
```

Alterando argumento

- Em Python, o argumento é passado na forma de referência (“ponteiro”).
- Sob certas condições, é possível manter as alterações.
- De maneira geral, deve-se se tomar cuidado com referências de variáveis.
- Ao fazer:
  - **`>> a=3;b=a`**
- Estamos atribuindo a **`b`** a referência de **`a`**. Se no futuro o valor de **`a`** mudar, muda também o valor de **`b`**.

Alterando argumento

```
 1 # Funcao Basica
 2 def myFunc(af,bf,l1f,l2f):
 3     af+=2
 4     bf=5
 5     l1f.extend([4,5,6])
 6     l2f =[4,5,6]
 7     print '[f]a:',af,' b:',bf
 8     print '[f]l1:',l1f,' l2:',l2f
 9
10 a = 7
11 b = 7
12 l1 = [1,2,3]
13 l2 = [1,2,3]
14 l3 = l1
15 l4=[];l4.extend(l1)
16 myFunc(a,b,l1,l2)
17 print '[m]a:',a,' b:',b
18 print '[m]l1:',l1,' l2:',l2
19 print 'l3:',l3
20 print 'l4:',l4
```

```
[f]a: 9  b: 5
[f]l1: [1, 2, 3, 4, 5, 6]  l2: [4, 5, 6]
[m]a: 7  b: 7
[m]l1: [1, 2, 3, 4, 5, 6]  l2: [1, 2, 3]
l3: [1, 2, 3, 4, 5, 6]
l4: [1, 2, 3]
```

Argumentos com **keys**

- É possível usar palavras-chaves (**keys**) para passar argumentos.

```
1 #!/usr/bin/python
2 def volRect(b,h):
3     return b*b*h
4
5 print 'O volume eh', volRect(h=4,b=10)
```

```
O volume eh 400
```

Variáveis Globais e Locais

- Para usar variáveis globais dentro de uma função, é preciso usar o comando **`global`**.
- Do contrário o python define uma variável local de mesmo nome.

```
 1 #!/usr/bin/python
 2 v1=8; v2=10 # Variaveis globais
 3 def volRect(b,h):
 4     global v2
 5     v1=100
 6     v2=200
 7     print 'v1:{} v2:{}'.format(v1,v2)
 8     return b*b*h
 9
10 volRect(h=4,b=10)
11 print 'v1:{} v2:{}'.format(v1,v2)
```

```
v1:100 v2:200
v1:8 v2:200
```

Variáveis Globais e Locais

- É possível usar a variável global sem definição, desde que não se tente alterar sua referência.
- Se mudar a referência, é criada uma variável local.

```
 1 #!/usr/bin/python
 2 v1=8; v2=10 # Variaveis globais
 3 def func1():
 4     print 'v1:{} v2:{}'.format(v1,v2)
 5
 6 def func2():
 7     print 'v1:{} v2:{}'.format(v1,v2)
 8     v1=7
 9
10 func1()
11 func2()
```

```
v1:8 v2:10
Traceback (most recent call last):
  File "ex08b.py", line 11, in <module>
    func2()
  File "ex08b.py", line 7, in func2
    print 'v1:{} v2:{}'.format(v1,v2)
UnboundLocalError: local variable 'v1' referenced before assignment
```

Comunicação Serial

Uso Básico

- A forma mais básica para iniciar uma comunicação serial é usando a biblioteca **`serial`**.
  - **`>> import serial`**
  - **`>> conn = serial.Serial('dispositivo',taxa, [timeout=])`**
  - **`>> if (conn.in_waiting>0):`**
  - **`s = conn.readline()`**
- **`s`** recebe uma string com o texto lindo na serial (lê até encontrar um '**`\n`**')

Biblioteca **`serial.tools.list_ports`**

- Permite listar o devices conectados ao computador.
- É muito útil para permitir ao usuário escolher com que porta ele quer se comunicar.
  - **`>> import serial.tools.list_ports`**
  - **`>> portas`** `= serial.tools.list_ports.comports()`

Código mais completo

```
 1 import serial
 2 import serial.tools.list_ports
 3 import time
 4
 5 ports = list(serial.tools.list_ports.comports())
 6 print 'Foram detectadas as seguintes portas:'
 7 i=0
 8 for p in ports:
 9     print p.device, ' -> ',i
10
11 idx = int(raw_input('Digite o numero da porta desejada:'))
12
13 comport = serial.Serial(ports[idx].device, 9600,timeout=0.10)
14 comport.readline() ## Read and eliminates previous data
15
16 ## Loop principal do programa
17 T=0.1 # periodo de 100ms
18 t_or=time.time() # Tempo inicial usado para gerar timestamps
```

Código mais completo

```
19 while (True):
20     ti=time.time()
21 ## Gets serial information
22     if (comport.in_waiting>0):
23         ## Read information in the serial port
24         txt_serial=comport.readline()
25         print txt_serial
26         txt_serial=txt_serial.rstrip(' \t\r\n\0') ## remove o caracteres
27         ## Converts information in numbers
28         dados = map(float, txt_serial.split(' '))
29         if len(dados)>=3:
30             ## Saves information
31             file = open('dados.txt','a')
32             file.write('{:.3f}\t{:.2f}\t{:.2f}\t{:.2f}\n'.format(
33                 (time.time()-t_or),dados[0],dados[1],dados[2]))
34             file.close()
35     ## Waits the next sample
36     time.sleep(T-(time.time()-ti))
```

Código mais completo

```
-1.81 1.35 -4.68

-1.96 1.48 -3.93

-2.00 1.61 -4.00

-1.92 1.76 -3.86

-1.72 1.93 -3.53

-1.41 2.10 -4.02

-1.01 2.28 -3.34

-0.52 2.47 -2.54
```

1	1.403	1.00	3.00	0.00
2	1.504	0.99	3.20	0.00
3	1.603	1.95	3.40	0.00
4	1.703	2.82	3.59	0.00
5	1.803	4.59	3.78	0.00
6	1.903	5.21	3.96	0.00
7	2.004	4.66	4.13	0.00
8	2.104	5.93	4.29	0.00
9	2.204	6.00	4.43	0.00
10	2.304	4.87	4.57	0.00
11	2.404	5.55	4.68	0.00
12	2.504	5.04	4.78	0.00
13	2.604	4.38	4.86	0.00
14	2.704	2.58	4.93	0.00
15	2.805	2.67	4.97	0.00
16	2.905	0.71	4.99	0.00
17	3.005	0.71	5.00	0.00
18	3.105	-0.28	4.98	0.00
19	3.205	-1.21	4.95	0.00
20	3.305	-2.06	4.89	0.00
21	3.405	-3.78	4.82	0.00
22	3.505	-4.36	4.73	0.00
23	3.606	-3.76	4.62	0.00
24	3.706	-3.97	4.49	0.00
25	3.806	-3.98	4.35	0.00
26	3.906	-3.79	4.20	0.00
27	4.006	-4.42	4.03	0.00
28	4.106	-3.86	3.85	0.00
29	4.206	-3.16	3.67	0.00
30	4.306	-1.32	3.48	0.00
31	4.407	-1.40	3.28	0.00
32	4.507	-0.42	3.08	0.00
33	4.607	1.58	2.88	0.00
34	4.707	1.56	2.68	0.00
35	4.807	3.47	2.49	0.00
36	4.907	4.28	2.30	0.00

Tópicos especiais

Código do Arduino

- Código do arduino para envio de dados na serial

```
#include <math.h>

void setup(){
  Serial.begin(9600);
}

int t=0;
float d1,d2,d3;
void loop(){
    d1=3*sin(0.2*t)+1+random(0,1);
    d2=2*sin(0.1*t)+3+random(0,0.5);
    d3=5*sin(0.2*t)+random(0,2);
    t=t+1;
    Serial.print(d1);
    Serial.print(" ");
    Serial.print(d2);
    Serial.print(" ");
    Serial.println(d3);

    delay(10);
}
```

Exercícios

Exercícios

1) Criar um código que simule diferentes dados no arduino (valores numéricos) e envie para o programa em Python. Seu programa deve ler os dados e armazená-los.

- *Salve os dados em arquivo*

Exercícios

2) Crie um código para ler dados gravados em arquivo .txt, fazer a analise dos dados e plotá-los em gráficos.

- *Utilize* ***`timestamps`*** *como os instantes de plot.*

Exercícios

3) Junte as duas funcionalidades anteriores em um único código. Defina um número de amostras e salve os dados apenas após coletar todas. Então, plote os dados obtidos.

- *Utilize* ***`timestamps`*** *como os instantes de plot.*