Parte II - Aula 4
Laboratório de informática
Conceitos básicos de programação

Alex Abreu

Conteúdo

1 Listas

Além de números, valores Booleanos e expressões simbólicas, uma variável também pode estar atribuída a uma lista. Uma lista é uma coleção de valores ordenados entre colchetes e separados por vírgulas.

```
sage: lista=[1,2,3]
sage: lista
[1, 2, 3]
```

Acima atribuímos a variável `lista` a lista com 3 elementos 1, 2 e 3. Podemos recuperar os elementos de uma lista escrevendo `lista[i]`, onde `i` é a posição que queremos.

```
sage: lista[1]
2
```

Como você pode notar acima, o elemento da posição 1 da lista é o 2. Isso porque em programação o primeiro número é o 0.

```
sage: lista[0]
1
```

Podemos acessar o último elemento de uma lista do seguinte modo.

```
sage: lista[2]
3
sage: lista[-1]
3
```

Listas podem conter qualquer tipo de elemento, até outras listas.

```
sage: lista1=[1,3,True, x^2-2*x, [4,'texto']]
sage: lista1
[1, 3, True, x^2 - 2*x, [4, 'texto']]
sage: lista1[0]
1
sage: lista1[2]
True
sage: lista1[3]
x^2 - 2*x
sage: lista1[4]
[4, 'texto']
sage: lista1[4][1]
texto
```

Podemos somar duas listas usando o comando `+`. Este comando cria uma única lista cujas entradas são as entradas das duas listas somadas. Note que esta soma de listas não é comutativa.

```
sage: lista2=[1,2,3]
sage: lista3=[4,5,6]
sage: lista2+lista3
[1, 2, 3, 4, 5, 6]
sage: lista3+lista2
[4, 5, 6, 1, 2, 3]
```

Podemos usar os comandos `append` e `extend` para aumentar listas. O comando `append` aumenta a lista em uma entrada, enquanto que o `extend`. Note que estes comandos modificam a lista, em vez de criar uma lista nova.

```
sage: lista2.extend(lista3)
None
sage: lista2
[1, 2, 3, 4, 5, 6]
sage: lista2.append(34)
None
sage: lista2
[1, 2, 3, 4, 5, 6, 34]
```

Podemos deletar entradas de uma lista com o comando `del`. Abaixo deletamos a primeira entrada da `lista2`.

```
sage: del lista2[0]
sage: lista2
[2, 3, 4, 5, 6, 34]
```

Podemos modificar elementos de uma lista, simplesmente atribuindo outros valores.

```
sage: lista2
[2, 3, 4, 5, 6, 34]
sage: lista2[1]=14
sage: lista2
[2, 14, 4, 5, 6, 34]
```

Podemos criar listas usando o comando `range` ou `[a..b]`.

```
sage: range(15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
sage: [0..14]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
sage: range(4,17)
[4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
sage: [4..16]
[4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
sage: range(4,17,2)
[4, 6, 8, 10, 12, 14, 16]
sage: [4,7..19]
[4, 7, 10, 13, 16, 19]
sage: [4,7..20]
[4, 7, 10, 13, 16, 19]
sage: range(15,0,-1)
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
sage: [15,14..0]
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Podemos usar o comando `len` para obter o comprimento de uma lista.

```
sage: lista3=[1,2,3,[4,5],[6,7]]
sage: len(lista3)
5
```

Também conseguimos criar sublistas a partir de uma lista.

```
sage: lista4=range(10)
sage: lista4
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sage: lista4[3:]
[3, 4, 5, 6, 7, 8, 9]
sage: lista4[:5]
[0, 1, 2, 3, 4]
sage: lista4[3:7]
[3, 4, 5, 6]
sage: lista4[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
sage: lista4[7:2:-1]
[7, 6, 5, 4, 3]
```

1.1 Atividades

1. Crie uma lista começando com 19 e terminando em -3 com incrementos de -2. Qual o comprimento dessa lista?
2. Qual o último elemento da lista `range(0,10^7,47)`?

2 O laço `for`

Até agora aprendemos a resolver problemas matemáticos usando comandos do Sage. Mas, muitas vezes, precisamos resolver um problema onde a mesma operação é repetida várias vezes. Para isso usamos o laço `for`.

```
sage: lista=[2,4,5,7,33]
sage: for j in lista:
....:     j^2
4
16
```

```
25
49
1089
```

Acima calculamos os quadrados de todos os elementos da lista `lista`.

Algumas observações sobre a sintaxe do comando `for`. A primeira linha sempre será

```
for j in L:
```

onde `L` é uma lista e `j` é a variável que percorrerá os valores da lista. A frase se traduz literalmente em *para `j` em `L`*. Usualmente essa lista é uma lista da forma `range(a,b)`. Note que precisamos dos dois pontos no final da linha, ele indica que começaráo bloco de instruções.

Também é importante notar a identação da segunda linha, ou seja a segunda linha começa alguns caracteres depois da primeira. Em Python, essa identação é obrigatória.

Vamos ver outros exemplos. Começamos calculando 40!. Note que para calcular 100!, precisamos multiplicar $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdots 40$.

```
sage: fac=1                                                                    84
sage: for i in range(1,41):                                                    85
....:     fac=i*fac                                                            86
sage: fac                                                                      87
815915283247897734345611269596115894272000000000                              88
sage: factorial(40)                                                            89
815915283247897734345611269596115894272000000000                              90
```

No laço `for` acima realizamos a seguinte operação. Para cada `i` de 1 a 40, multiplicamos a variável `fac` por `i`, e guardamos o resultado na própria variável `fac`. Ao fim, o valor da variável `fac` é 40!. Você pode calcular o fatorial de qualquer número n, só substituindo `range(1,41)` por `range(1,n+1)`.

Vamos agora calcular a soma $S_n = 1 + 2 + \ldots + n$ dos n primeiros números.

```
sage: n=10                                                                     91
sage: soma=0                                                                   92
sage: for j in range(1,n+1):                                                   93
....:     soma=soma+j                                                          94
sage: soma                                                                     95
55                                                                             96
```

Se você lembra da fórmula da soma para progressões aritméticas, você sabe que

$$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}.$$

(Se você não lembra, pergunte ao seu professor como descobrir essa fórmula.) Podemos checar no Sage.

```
sage: n*(n+1)/2                                                                97
55                                                                             98
```

Mais ainda, podemos checar para todos os valores até n. Basta, no laço `for`, instruir o Sage a mostrar as somas parciais.

```
sage: n=15                                                                     99
sage: soma=0                                                                   100
sage: for j in range(1,n+1):                                                   101
....:     soma=soma+j                                                          102
....:     j, soma, j*(j+1)/2                                                   103
```

```
1,1,1
2,3,3
3,6,6
4,10,10
5,15,15
6,21,21
7,28,28
8,36,36
9,45,45
10,55,55
11,66,66
12,78,78
13,91,91
14,105,105
15,120,120
```

Em cada linha do resultado acima, podemos ver o valor de j, da soma $1 + 2 + \ldots + j$ e da expressão $\frac{j(j+1)}{2}$.

2.1 Atividades

1. Defina $Q_n = 1^2 + 2^3 + 3^2 + \ldots + n^2$.
   (a) Calcule Q_1, Q_2, Q_3, Q_4, Q_{239} e Q_{1397}.
   (b) Seja $f(x) = ax^3 + bx^2 + cx + d$. Encontre a, b, c e d tais que $f(1) = Q_1$, $f(2) = Q_2$, $f(3) = Q_3$ e $f(4) = Q_4$.
   (c) Verifique que $f(239) = Q_{239}$ e $f(1397) = Q_{1397}$.
   (d) Verifique que $f(n) = Q_n$ para todo $n \leq 100$.
   (e) Use o comando `factor` em $f(x)$.
2. Encontre a fórmula fechada para a soma dos cubos $C_n = 1^3 + 2^3 + \ldots + n^3$. Aqui a função f terá grau 4. Você consegue encontrar uma relação entre a soma dos cubos C_n a soma $S_n = 1 + 2 + \ldots + n$.
3. Encontre a fórmula para a soma dos n primeiros números ímpares $I_n = 1 + 3 + 5 + 7 + \ldots + (2n + 1)$.
4. A sequência de Fibonacci é definida por $F_1 = 1$, $F_2 = 1$ e $F_{n+1} = F_n + F_{n-1}$ para todo $n \geq 2$. Os primeiros termos são $1, 1, 2, 3, 5, 8, 13, 21, \ldots$. Encontre o milésimo termo da sequência de Fibonacci.

3 O laço `while`

Na última seção, vimos como repetir uma mesma operação n vezes. Mas muitas vezes, queremos repetir a mesma operação até que determinada condição seja satisfeita, ou seja, não sabemos a princípio quantas vezes devem ser repetidas a operação. Para isso usamos o laço `while` (cuja tradução é *enquanto*). Por exemplo, vamos recalcular $S_n = 1 + 2 + \ldots + n$ usando `while`.

```
sage: n=30                                                    104
sage: j=1                                                     105
sage: soma=0                                                  106
sage: while j<=n:                                             107
....:     soma=soma+j                                         108
....:     j=j+1                                               109
```

```
sage: soma
465
```

Algumas observações sobre a sintaxe do comando `for`. A primeira linha sempre será

```
while P:
```

onde P é uma expressão com valor booleano que depende de outras variáveis. Acima, os comandos

```
soma=soma+j
j=j+1
```

são repetidos enquanto o valor de `j` for menor ou igual que `n`. Note que com o comando `while` podemos criar um laço infinito. Tente compilar o código abaixo.

```
sage: j=0
sage: while j>=0:
....:     j^2
....:     j=j+1
```

O que acontece?

Até agora apenas vimos como usar o laço `while` como um substituto do laço `for`. Mas o laço `while` é um pouco mais versátil. Por exemplo, vamos tentar encontrar o primeiro número n tal que S_n é maior ou igual que 10^5.

```
sage: j=0
sage: soma=0
sage: while soma<10^5:
....:     j=j+1
....:     soma=soma+j
sage: j
447
sage: soma
100128
```

No código acima, enquanto a variável `soma` for menor que 10^5, o Sage continuará aumentando j em 1 e somando a variável `soma`.

3.1 Atividades

1. Considere a soma parcial $H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$ da série harmônica. Encontre o primeiro n tal que $H_n > 10$.

4 O Algoritmo de Euclides

O máximo divisor comum entre dois inteiros positivos a e b é o maior inteiro positivo d, tal que a e b sejam múltiplos de d, usualmente escrevemos $\text{mdc}(a, b)$. Nesta seção vamos aprender o algoritmo de Euclides que calcula o $\text{mdc}(a, b)$.

O algoritmo de Euclides foi descrito na obra *Os Elementos* de Euclides (por volta de 300 A.C.). Ele é baseado na seguinte observação, se $a = qb + r$ (aqui q é o quociente e r é o resto da divisão de a por b) então vale $\text{mdc}(a, b) = \text{mdc}(b, r)$. Mais ainda, se a é múltiplo de b, então $\text{mdc}(a, b) = b$.

Vamos então calcular o M.D.C entre 1450431 e 354312. Para isso, realizamos divisões sucessivas.

$$
\begin{aligned}
1450431 &= 4 \cdot 354312 + 33183 \\
354312 &= 10 \cdot 33183 + 22482 \\
33183 &= 1 \cdot 22482 + 10701 \\
22482 &= 2 \cdot 10701 + 1080 \\
10701 &= 9 \cdot 1080 + 981 \\
1080 &= 1 \cdot 981 + 99 \\
981 &= 9 \cdot 99 + 90 \\
99 &= 1 \cdot 90 + 9 \\
90 &= 10 \cdot 9 + 0
\end{aligned} \tag{1}
$$

Da primeira linha, descobrimos que $\text{mdc}(1450431, 354312) = \text{mdc}(354312, 33183)$, da segunda linhas, vemos que$\text{mdc}(354312, 33183) = \text{mdc}(33183, 22482)$ e assim por diante até $\text{mdc}(90, 9) = 9$, já que 90 é múltiplo de 9. Portando $\text{mdc}(1450431, 35431, 2) = 9$.

Podemos repetir as contas acima no Sage (lembre dos comandos `//` e `%` da primeira aula).

```
sage: a=1450431
sage: b=354312
sage: r1=a%b
sage: r1
33183
sage: r2=b%r1
sage: r2
22482
sage: r3=r1%r2
sage: r3
10701
sage: r4=r2%r3
sage: r4
1080
sage: r5=r3%r4
sage: r5
981
sage: r6=r4%r5
sage: r6
99
sage: r7=r5%r6
sage: r7
90
sage: r8=r6%r7
sage: r8
9
sage: r9=r7%r8
sage: r9
0
```

Como `r9` é igual a 0, é porque o M.D.C. é `r8`, que é 9. Vamos agora aprender a automatizar este processo. Em cada passo, temos três variáveis, o dividendo, o divisor e o resto. Vamos chamálos de `r1`, `r2` e `r3`. Quando passamos para o passo seguinte, substituímos `r1=r2`, `r2=r3` e `r3` vira o resto da divisão de `r2` por `r3`.

```
sage: a=1450431
sage: b=354312
sage: r1=a
sage: r2=b
```

```
sage: r3=a%b
sage: while r3>0:
....:     r1,r2,r3=r2,r3,r2%r3
....:     r1,r2,r3
354312,33183,22482
33183,22482,10701
22482,10701,1080
10701,1080,981
1080,981,99
981,99,90
99,90,9
90,9,0
```

Compare os resultados acima com os da Equação (1).

4.1 Atividades

1. Prove que se $a = qb + r$ então $\text{mdc}(a, b) = \text{mdc}(a, r)$.
2. Prove que se a é múltiplo de b então $\text{mdc}(a, b) = b$.
3. Por que o algoritmo de Euclides sempre termina?
4. Calcule $\text{mdc}(1099, 525)$ a mão, pelo algoritmo de Euclides.
5. Calcule $\text{mdc}(9506112, 4183179)$.
6. O Sage já tem um comando que calcula o M.D.C. de dois números. Este comando é `gcd`. Calcule `gcd(9506112,4183179)`.

5 Condicionais

Algumas vezes gostaríamos de realizar uma operação apenas se determinada condição for satisfeita. Para isso usamos o condicional `if`, `else` (cuja tradução é *se, caso contrário*). A síntaxe do condicional é da seguinte forma.

```
if condição:
    instruções
else:
    outras instruções
```

Por exemplo.

```
sage: n=40
sage: if n>30:
....:     u=n-1
....: else:
....:     u=n+1
sage: u
39
```

Trocando o valor de `n` para 20 (você não precisa reescrever o código, só substituir o valor de `n`).

```
sage: n=20
sage: if n>30:
....:     u=n-1
....: else:
....:     u=n+1
```

```
sage: u
21
```

No primeiro caso, como $40 > 30$ é verdadeiro, então o Sage realizou a instrução `u=n-1`. No segundo caso, como $20 > 30$ é falso, ele realizou a instrução `u=n+1`.

Claro que nos exemplos acima, se soubermos o valor de n nós mesmos poderíamos escolher a instrução a ser realizada. Mas quando termos que fazer esta escolha várias vezes dentro de um laço, temos que usar o condicional.

Vamos resolver o seguinte problema. Começamos com o número 1331 escrito numa folha de papel. A cada minuto, apagamos o número x da folha e escrevemos um novo número no lugar. Se x for par, escrevemos no lugar de x o número $x/2$. Se x for ímpar, escrevemos no lugar de x o número $x + 11$. Depois de 100 minutos, qual número estará escrito na folha?

Podemos escrever os casos iniciais abaixo.

$$1331 \to 1342 \to 671 \to 682 \to 341 \to 352 \to \cdots$$

Nosso algoritmo no Sage fica assim.

```
sage: a=1331
sage: for i in range(100):
....:     if a%2==0:
....:         a=a/2
....:     else:
....:         a=a+11
....:     a
sage: a
11
```

Em cada passo, subsitituimos a variável `a` por `a/2` ou `a+11` dependendo se `a` é ímpar ou não. Por fim, encontramos 11, que é o número que estará escrito na folha ao final de 100 minutos.

Como outro exemplo, podemos pensar no seguinte problema. Considere os quadrados perfeitos $1^2, 2^2, 3^2, \ldots, 100^2$. Na seção 2 aprendemos a calcular a soma dos n primeiros quadrados perfeitos. Mas, e se quisermos somar só aqueles que terminam em 1? Podemos fazer isso com o seguinte código.

```
sage: soma=0
sage: for j in range(1, 101):
....:     if j^2%10==1:
....:         soma=soma+j^2
sage: soma
66820
```

No código acima, para cada j na lista $1, 2, \ldots, 100$, checamos se j^2 deixa resto 1 por 10, ou seja, se seu último dígito é 1. Se sim, acrescentamos j^2 a soma, caso contrário, não fazemos nada. No fim vemos que o resultado é 66820.

5.1 Atividades

1. Encontre a soma de todos os números menores que 1000 que são múltiplos de 3 ou 5.
2. Encontre a soma de todos os números de Fibonacci pares menores que um milhão.

6 O método da bisseção

Nesta seção vamos aprender a encontrar raízes aproximadas de uma função f. Na verdade, já sabemos fazer isso com o comando `find_roots`, mas vamos aprender a criar nosso próprio método.

Esse método é chamado método da biseção. Seja $f\colon \mathbb{R} \to \mathbb{R}$ uma função real contínua. Suponha que existam a e b tais que $f(a) < 0$ e $f(b) > 0$. Pelo teorema do valor intermediário, existe $d \in (a, b)$ tal que $f(d) = 0$.

Vamos fazer um exemplo. Tome $f = x^2 + x - 20 * \log(x)$.

```
sage: f(x)=x^2+x-20*log(x)
sage: plot(f,1,10)
Graphics object consisting of 1 graphics primitive
```

Esboçando o gráfico de f, vemos que f tem uma raiz entre 4 e 6. De fato, $f(4) < 0$ e $f(6) > 0$.

```
sage: f(4.0)
-7.72588722239781
sage: f(6.0)
6.16481061543890
sage: plot(f,4,6)
Graphics object consisting of 1 graphics primitive
```

Vamos calcular $f(5)$.

```
sage: f(5.0)
-2.18875824868201
```

Como $f(5) < 0$, vemos que a raíz tem que estar depois do 5, então substituímos o 4 por 5. E olhamos para o intervalo $(5, 6)$.

```
sage: plot(f,5,6)
Graphics object consisting of 1 graphics primitive
```

De novo, vamos calcular $f(5.5)$.

```
sage: f(5.5)
1.65503815523149
```

como $f(5,5) > 0$, vemos que a raíz tem que estar antes de 5.5. Olhamos para o intervalo $(5, 5.5)$.

```
sage: plot(f,5,5.5)
Graphics object consisting of 1 graphics primitive
```

Podemos continuar assim, cada vez olhando para o ponto médio do intervalo. Até encontrarmos um intervalo pequeno o suficiente onde esteja a raiz.

Vamos agora entender como é o processo. Em cada passo possuímos dois números a e b com $f(a) < 0$ e $f(b) > 0$, tomamos $c = \frac{a+b}{2}$ e calculamos $f(c)$. Se $f(c) < 0$, então a raíz está no intervalo (c, b), ou seja, substituímos a por c. Se $f(c) > 0$, então a raíz está no intervalo (a, c), ou seja, substituímos b por c.

```
sage: f(x)=x^2+x-20*log(x)
sage: a=4.0
sage: b=6.0
sage: c=(a+b)/2
sage: while abs(f(c))>0.0000000001:
....:     if f(c)<0:
....:         a=c
....:     else:
....:         b=c
....:     c=(a+b)/2
sage: c, f(c)
(5.29541397799039, 9.81437153768638e-11)
```

```
sage: find_root(f(x),4,6)
5.29541397798
```

Nosso procedimento acima encontrou um ponto $c = 5.29541397799039$ tal que $f(c) \approx 9.8 \cdot 10^{-11}$ que é bem perto de 0. Comparado com o comando `find_root`, só esta diferente o último algarismo. Se você quiser, pode trocar o 0.0000000001 por 0 mesmo. Mas pode ser que demore muito o algoritmo.

6.1 Atividades

1. Refaça a atividade 1 da seção 2 da aula 3 usando o método da bisseção

7 Definindo comandos

Até agora aprendemos vários comandos no Sage, desde comandos para esboçar gráficos, a comandos para criar listas. Nesta seção, vamos aprender a definir nossos próprios comandos. Abaixo, criamos um comando simples.

```
sage: def soma_um(n):
....:     return n+1
```

Nosso comando tem como argumento um número (ou expressão simbólica) n e retorna a soma de n com um.

```
sage: soma_um(5)
6
sage: soma_um(45.32)
46.3200000000000
sage: soma_um(pi)
pi + 1
sage: soma_um(x^2+3*x)
x^2 + 3*x + 1
```

Note que se você tentar aplicar o comando `soma_um` a algo que não seja um número (ou expressão simbólica) obtemos um erro.

```
sage: soma_um([2,3])
```

Note que um comando pode ter mais de um argumento.

```
sage: def soma_produto(a,b):
....:     return a+b+a*b
sage: soma_produto(4,5)
29
sage: y=var('y')
sage: soma_produto(x,y)
x*y + x + y
sage: soma_produto(pi,4)
5*pi + 4
```

Nossos comandos podem ser mais complicados. Por exemplo, abaixou definimos um comando que calcula o n-ésimo número de Fibonacci.

```
sage: def fibo(n):
....:     a=1;
....:     b=1;
....:     for i in range(n-2):
....:         a,b=b,a+b
....:     return b
```

```
sage: fibo(1),fibo(2),fibo(3),fibo(4),fibo(5),fibo(6)
(1, 1, 2, 3, 5, 8)
sage: fibo(200)
280571172992510140037611932413038677189525
```

Em geral, para definirmos um comando, usamos a síntaxe:

```
def nome(argumentos):
    instruções
    return resultado
```

7.1 atividades

1. Defina o comando `mdc(a,b)` que calcula o M.D.C. de dois números interos positivos a e b.
2. Defina o comando `raiz_bissecao(f,a,b)` que encontra uma raiz aproximada da função f no intervalo (a, b).
3. Defina o comando `e_par(a)` que retorna `True` se a for par, e `False` se a for ímpar.
4. Defina comandos `soma(n)`, `soma_quadrados(n)` e `soma_cubos(n)` que retornam, respectivamente, a soma dos n primeiros números, dos n primeiros quadrados e dos n primeiros cubos. Verifique que `soma_cubos(n)==soma(n)^2`.
5. Defina o comando `maximo(L)` que retorna o maior número da lista `L`. Compare com `max(L)`
6. Seja $\star$ uma operação dada por $a \star b = ab + a + b$. Numa folha de papel estão escritos os números de 1 a 100. A cada minuto Joãozinho escolhe quaisquer dois números a e b escritos na folha, apaga esses números e escreve $a \star b$ na folha. Ao final de 99 minutos só existe um único número escrito na folha.
   (a) Prove que $\star$ é comutativa e associativa. Use o comando `soma_produto` definido acima para provar a associatividade.
   (b) Quantos e quais são os possíveis números escritos na folha ao final dos 99 minutos?

Criando listas	
Criar uma lista de 0 a b	`range(a+1)` ou `[0..a]`
Criar uma lista de a a b	`range(a,b+1)` ou `[a..b]`
Criar uma lista de a a b em incrementos de c	`range(a,b+1,c)` ou `[a, a+c..b]` ou `[a..b, step=c]`
Comandos para a lista `L`	
Acessando o i-ésimo elemento	`L[i]`
Acessando o último elemento	`L[-1]`
Deletar o i-ésimo elemento	`del L[i]`
Comprimento	`len(L)`
Concatenar com a lista `L2`	`L+L2`
Sublista começando no i-ésimo elemento	`L[i:]`
Sublista terminando no $j-1$-ésimo elemento	`L[:j]`
Sublista começando no i-ésimo elemento e terminando no $j-1$-ésimo elemento	`L[i:j]`
Sublista começando no $i+1$-ésimo elemento e terminando no j-ésimo elemento em ordem reversa	`L[j:i:-1]`
Laços e condicionais	
Realizar instrução para todo `j` numa lista `L`	`for j in L`
Realizar instrução enquanto condição `P` é verdadeira	`while P:`

Realizar instrução `A` se condição `P` for verdadeira, caso contrário realizar instrução `B`	`if P:`
	`  A`
	`else:`
	`  B`

Definindo comandos	
Definir o comando `comando` com argumentos `a_1`,..., `a_k`	`def comando(a_1,...,a_k):`

Tabela 1: Comandos para listas