SCC 124 - Introdução à Programação para Engenharias

Comandos

Professor: André C. P. L. F. de Carvalho, ICMC-USP
Pos-doutorando: Isvani Frias-Blanco
Monitor: Henrique Bonini de Britto Menezes

Aula de hoje

- Comandos
  - Comandos simples
  - Blocos de comandos
- Comandos de controle
  - Definem que comandos serão executados
    - De acordo com uma condição (condicionais)
    - De forma repetida (repetitivos)

Comandos

- Programas Python são compostos por funções
  - Funções são compostas por comandos
- Comandos em Python podem ser:
  - Comandos simples
    - Realizam uma ação específica
  - Comandos de controle
    - Afetam a maneira como outros comandos são executados

Comandos simples

- Maioria dos comandos de um programa em Python
- Incluem:
  - Operação de atribuição
  - Chamada de função
  - Operação de leitura / escrita
- Podem ser reunidos em blocos de comandos

Bloco de comandos

- Contém um ou mais comandos simples e/ou blocos de comandos
  - Com o mesmo recuo (indentação)
  - Quanto tem mais de um comando, eles são executados na sequência em que aparecem
  - Também chamado de comando composto

comando

comando 1
comando 2
...
comando n

Comandos de controle

- Controlam a execução de outros comandos
- Podem ser:
  - Condicionais (que condicionam)
    - Condicionam a execução de um bloco de comandos ao valor de uma expressão
      - Bloco pode ter um único comando
  - De repetição (iterativos, que repetem)
    - Permitem que um bloco de comandos seja executado mais de uma vez

Comandos de controle

- Comandos de controle condicionais
  - Avaliação de uma condição define se um bloco de comandos será executado
    - Execução condicional
  - A maneira mais fácil de fazer isso é por meio do comando *if*

Indica que em seguida vem um bloco de comando

```
if condicao:
    bloco comandos1
else:
    bloco comandos2
```


Comando if

```
>>> x = int(input('Entrar com um valor inteiro: '))
>>> if x < 0:
...         x = 0
...         print ('Valor era negativo, mudou para zero')
...     else:
...         print ('Valor era zero ou positivo')
...
```

- A função input faz parte da linguagem Python (built-in)
  - Imprime mensagem na tela e espera que o usuário entre com um valor

Comando if

- Alguns códigos precisam de estruturas de decisão mais complicadas
  - Por exemplo, quando existem mais que duas possibilidades
  - Uma alternativa é utilizar uma ou mais opções *elif* (abreviação de *else if*)
- Apenas uma opção *else* é permitida em um comando *if*

Comando if

```
>>> x = int(input("Digitar um valor inteiro: "))
>>> if x < 0:
...         x = 0
...         print ('Valor negativo, mudou para zero')
...     elif x == 0:
...         print ('Zero')
...     elif x == 1:
...         print ('Um')
...     else:
...         print ('Valor maior que um')
...
```

- O que ocorre Quando um *elif* for *True?*
  - Comandos *elif* e *else* abaixo dele não são avaliados

Exemplo

```
pontos = int(input("Digitar um valor inteiro: "))
if (pontos >= 90):
    nota = 'A'
elif (pontos >= 80):
    nota = 'B'
elif (pontos >= 70):
    nota = 'C'
elif (pontos >= 60):
    nota = 'D'
else:
    nota = 'F'
```


Comandos iterativos

- Permitem a execução repetida de um bloco de comandos
  - Que pode ter um únicomcomando
- Comandos iterativos da linguagem Python
  - Comando *while*
  - Comando *for*

Comando while

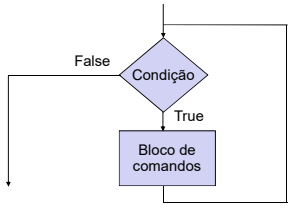
- Comando iterativo mais simples
- Executa um bloco de comandos repetidamente
  - Até uma condição se tornar falsa

```
while (expressão condicional):
    bloco de comandos
```


Comando while

- Teste condicional é executado antes de cada repetição
  - Cada repetição é um ciclo do *loop* (laço)
  - Se o primeiro teste retornar valor *False*, o corpo do *loop* nunca é executado

Comando while



Exemplo

```
n = 12
i = 1
while i <n:
    print (i)
    i=i+1
```


Exemplo

```
n = 12
i = 1
while i <n:
    print (i)
    i=i+1
```

Produz:

```
1
2
3
4
5
6
7
8
9
10
11
```


Comando while

- Utilizado quando repetição de um bloco depende de uma condição ser verdade
- Vários problemas de programação não se encaixam na estrutura do comando *while*
  - As vezes o teste seria mais natural em algum lugar no meio do *loop*
    - Para permitir que algumas operações sejam realizadas dentro do *loop*
      - *Loop* baseado em sentinela

Comando while

- *Loop* baseado em sentinela
  - Sai do loop quando o valor de uma variável for igual ao sentinela
    - Senão, continuar no *loop*
  - Como sair do *loop* quando um dado valor é lido (problema do *loop* e meio)?
    - Copiar parte do código para fora do *loop*
    - Colocar um comando *break* após a leitura

Exemplo (opção 1)

```
total = 0
numNotas = 0
nota = int (input( 'Digite nota, -1 para sair:  ')) # ler nota
while nota != -1:
    total += nota
    numNotas = numNotas + 1
    nota = int (input('digite nota, -1 para sair: ')) # ler nota
if numNotas != 0:
    media = float(total) / numNotas
    print ('Media da classe eh', media)
```


Comando while

- Utilização de comando *break* após a leitura
  - Em geral, estratégia mais eficiente

```
while (x > 0):
    Pede que usuário entre um valor e lê o valor
    if (valor == sentinela):
      break
.....
```


Exemplo (opção 2)

```
total = 0
numNotas = 0
nota = 1
while (nota != -1):
   nota = int (input ('Digitar nota (-1 para sair): '))
   if nota == -1:
      break
   total += nota
   numNotas += 1
if numNotas != 0:
   media = float(total)/numNotas
   print ('Media da classe eh', media)
```


Exemplo

```
nnomes = 0
while nnomes != 10:
   #print('Entra nome: ')
   #letra = input()
   letra = input('Entrar nome: ')
   if letra == '$':
      break
   nnomes = nnomes + 1
   print (' Ola ', letra)
if nnomes < 10:
   print ('Listou menos que 10 nomes')
```


Comando for

- Utilizado conhece quantas vezes um bloco de comandos deve ser executado
- Iteração é definida pelos itens de uma sequência
  - Segue a ordem em que os itens aparecem na sequência
  - Pode usar qualquer tipo de sequência
    - Ex.: string, lista, ...

Exemplo

```
>>> a = ['gato', 'janela', 'sejogou']
>>> for x in a:
...         print (x, len(x))
...
gato 4
janela 6
sejogou 7
```

Número de letras em um string

Comando for

- Não é seguro alterar sequência que está sendo usada na iteração
  - Se usar, pode apenas para sequências mutáveis, como listas
    - Se necessário, iteração deve modificar uma cópia da lista
      - Para não perder lista original
      - Indexação com fatias pode ser usada para isso

Exemplo

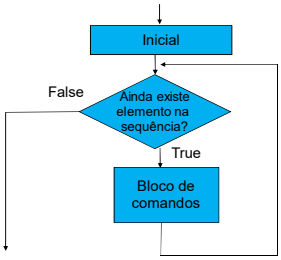
```
>>> a = [‘estudou', ‘passou', ‘comemorou']
>>> for x in a[:]: # Corre a lista inteira
...         if len(x) > 6:
                a.insert(0, x)
...
>>> a
```


Exemplo

```
>>> a = [‘estudou', ‘passou', ‘comemorou']
>>> for x in a[:]: # Corre a lista inteira
...         if len(x) > 6:
                a.insert(0, x)
...
>>> a
[‘comemorou', ‘estudou', ‘passou', ‘comemorou']
```

Comando *for* usa a lista original, não a lista modificada

Comando for



Exemplo

- O que faz o programa abaixo?

```
for t in range ( 10):
    print t
print (“fim da lista”)
```

Exemplo

- O que faz o programa abaixo?

```
for t in range ( 10):
    print t
print ("fim da lista")
```

Produz:

```
0
1
2
3
4
5
6
7
8
9
fim da lista!
```


Função range

- Facilita repetição usando uma sequência de números
- Retorna um objeto que parece uma lista, mas não é
  - Retorna os itens que permitem gerar a lista
  - Faz isso para salvar espaço
  - Ex.: >>> *range (10)* *ou*
    range (0, 10)

Função range

- Retorna um objeto iterável
  - Pode ser usado por funções e permite obter itens sucessivos
    - É o que faz o comando *for* quando usa a função *range*
  - Ex.: >>> *list (range (10))*
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  - Valor (limite) fornecido nunca faz parte da lista gerada

Função range

- Por default começa no 0 e aumenta de 1
- É possível também definir início e incremento da sequencia (ini, fim, inc)

```
>>> list (range(5, 10))
[5, 6, 7, 8, 9]
>>> list (range(0, 10, 3))
[0, 3, 6, 9]
>>> list (range(-10, -100, -30))
[-10, -40, -70]
```


Função range

- Sequência para iteração não precisa ser numérica
  - Combinar funções *range* e *len*

```
>>> a = ['Maria', 'tinha', 'uma', 'febre', 'alta']
>>> for i in range(len(a)):
...         print (i, a[i])
...
```


Função range

- Sequência para iteração não precisa ser numérica
  - Combinar funções *range* e *len*

```
>>> a = ['Maria', 'tinha', 'uma', 'febre', 'alta']
>>> for i in range(len(a)):
...         print (i, a[i])
...
0 Maria
1 tinha
2 uma
3 febre
4 alta
```

Saída de laços

- Comando *break*
  - Sai do laço menos externo
- Comando *continue*
  - Pula próximos comandos do bloco e passa para a próxima iteração do laço
- Vale para comandos *while* e *for*

Exemplo

```
for i in range ( 5):
    s = input('Entre um string: ')
    if s == 'sair':
        break
    print ('tamanho do string eh ', len(s))
print ('Terminou')
```

```
for i in range ( 5):
    s = input('Entre um string : ')
    if s == 'sair':
        break
    if len(s) < 3:
        continue
    print ('Entrada tem tamanho suficiente')
...
```


Comando pass

- Não faz nada
  - Usado quando um comando é sintaticamente requerido, mas nenhuma ação é necessária

```
>>> while True:
...         pass # Esperando Ctrl  C
```

```
>>> def funcaofaz ():
...         pass #  guardando espaco para escrever
```


Exercício

- Escreve o menor trecho de programa que imprime:
  - Verdadeiro se (3 <= x <= 4) ou (x > 7 e x < 30 e x é múltiplo de 4)
  - Falso se ((x >= 10 ou x < -3) ou x é múltiplo de 3) e condição anterior é falsa
  - Não pode usar operadores lógicos **and** nem **or**

Exercício

- Re-escreva os trechos de programas abaixo usando o comando *for*

```
x = 14
while  (x >= 3):
  print  (x)
  x =  x - 5
```

```
y = 70
while ( y <= 90):
  print  (y)
  y = y + 5
```


Conclusão

- Comandos
  - Comandos simples
  - Blocos de comandos
- Comandos de controle
  - Condicionais
    - if
  - Repetitivos
    - for
    - while

Perguntas