Programação Script

Comandos de Repetição

Aula 04

Prof. Felipe A. Louza

Roteiro

Roteiro

Comandos Repetitivos

Até agora vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.

- Exemplo:

```
1 # leitura dos dados
2 x = int(input())
3 if (x % 2 == 0): #decide se é par
4   print(x, "é par")
5 else
6   print(x, "é ímpar")
```

Comandos Repetitivos

Entretanto, frequentemente é necessário executar um bloco de comandos várias vezes para se obter o resultado esperado.

- A execução repetida de um bloco de comandos é chamada de iteração.
- Exemplo:

```
1 print(2**0)
2 print(2**1)
3 print(2**2)
4 print(2**3)
```

```
1 i = 0             #i==0
2 print(2**i)
3 i += 1            #i==1
4 print(2**i)
5 i += 1            #i==2
6 print(2**i)
7 i += 1            #i==3
8 print(2**i)
```

Comandos Repetitivos

Vamos fazer um programa que imprime de 2^0 até 2^{10}:

```
1  i = 0          #i==0
2  print(2**i)
3  i += 1         #i==1
4  print(2**i)
5  i += 1         #i==2
6  print(2**i)
7  i += 1         #i==3
8  ...
9  ...
10 print(2**i)    #i==10
```

Comandos Repetitivos

Como fazer um programa que imprime de 2^0 até 2^n (o valor de n é dado pelo usuário):

```
1  n = int(input('Digite um número:'))
2  i = 0          #i==0
3  if(n >= 0):
4    print(2**i)
5    i += 1       #i==1
6  if(n >= 1):
7    print(2**i)
8    i += 1       #i==2
9  ...
10 ...
11 if(n >= 10):
12   print(2**i)
```

- Note que o programa é válido apenas para $n \leq 10$.

Roteiro

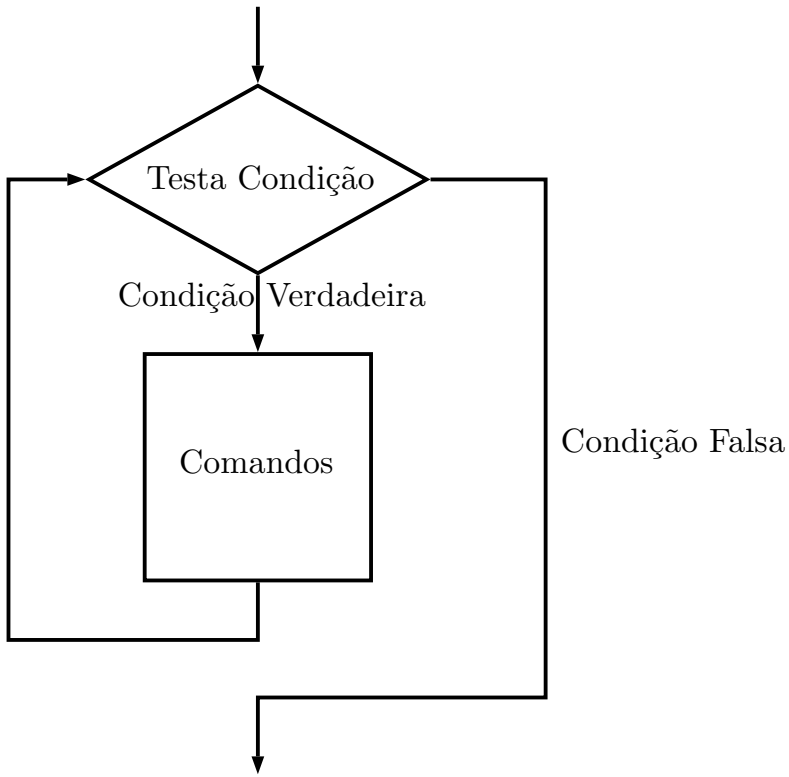
O comando `while`

O comando `while` executa um bloco de comando(s) enquanto a condição é verdadeira (`True`).

```
while (condição):
  comando1
  comando2
  ...
  comandoN
```

O comando `while`

- **Passo 1**: Testa a condição.
  - Se a condição for `True` vai para o **Passo 2**.
- **Passo 2.1**: Executa os comandos.
- **Passo 2.2**: Volta para o **Passo 1**.



O comando `while`

Programa que imprime de 2^0 até 2^n (o valor de n é dado pelo usuário):

```
1 n = int(input('Digite um número:'))
2 #iteração
3 i = 0          #i==0
4 while(i <= n):
5   print(2**i)
6   i += 1       #i==1, 2, ..., n
```

O comando `while`

1. O que acontece se a condição for falsa na primeira vez?

```
1  a = 1
2  while(a != a):
3      a=a+1
```

2. O que acontece se a condição for sempre verdadeira?

```
1  a = 1
2  while(a == a):
3      a=a+1
```

O comando `while`

```
1 count = 0
2 while (count < 10):
3   # Ponto A
4   print ("Olá...", count)
5   count = count + 1
6   # Ponto B
7 # Ponto C
```

- Analise o código acima e assinale a(s) alternativa(s) correta(s):

  a) `count < 10` é sempre `True` no ponto C.
  b) `count < 10` é sempre `False` no ponto B.
  c) `count < 10` é sempre `True` no ponto A.
  d) `count < 10` é sempre `False` no ponto C.
  e) `count < 10` é sempre `True` no ponto B.

Roteiro

O comando `while-else`

Ao final do `while` podemos utilizar a instrução `else`.

```
1  while (condição):
2    comando1
3    comando2
4    ...
5    comandoN
6  else:
7    comando1
8    comando2
9    ...
10   comandoM
```

O comando `while-else`

Programa que imprime de 2^0 até 2^n (o valor de n é dado pelo usuário):

```
1 n = int(input('Digite um número:'))
2 #iteração
3 i = 0          #i==0
4 while(i <= n):
5   print(2**i)
6   i += 1       #i==1, 2, ..., n
7 else:
8   print("FIM.")
```

- Nem sempre faz sentido ter o `else`.

Roteiro

Exemplo 1

Faça um programa que lê um número inteiro n e que calcula o valor de

$$\sum_{i=1}^{n} i$$

Exemplo 1

```
1 n = int(input("Digite o valor de n: "))
2
3 soma = 0
4 i = 1
5 while (i<=n):
6   soma = soma+i          # i = 1, 2, 3, 4, 5
7   i = i + 1
8
9 print("O valor da soma é {}".format(soma))
```

Chamamos soma de "variável acumuladora".

Exemplo 1

Podemos verificar o resultado comparando com:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

```
1 #soma da PA:
2 resultado = n*(n+1)/2
3
4 #validação
5 if(soma!=(n*(n+1))/2):
6   print("Houve algum erro!!")
```

Exemplo 2

Faça um programa que lê números do teclado até que o usuário digite 0.

Exemplo 2

```
1 teste = True
2 while (teste):
3   x = int(input("Digite um número: "))
4   if (x == 0):
5     teste = False
```

Chamamos `teste` de "indicador de passagem".

Exemplo 3

Faça um programa que calcula a soma de todos os números digitados pelo usuário.

- O seu programa deve perguntar para o usuário qual o tamanho da sequência de números a ser somada.

Exemplo 3

```
1  n = int(input("Digite a quantidade de numeros: "))
2
3  soma = 0
4  i = 0
5
6  while (i<n):
7    valor = int(input("Digite um valor a ser somado: "))
8    soma = soma+valor
9    i = i + 1
10
11 print("A soma dos valores digitados é {}".format(soma))
```

Exemplo 4

Faça um programa que lê *n* números inteiros do teclado, e no final informa qual foi o maior número lido.

- O programa deve encerrar quando o usuário digita 0.

Exemplo 4

```
1 valor = 1
2 max = 0
3
4 while (valor > 0):
5   valor = int(input("Digite um número: "))
6   if(valor > max):
7     max = valor
8
9 print("O maior valor digitado foi {}".format(max))
```

Exemplo 5

Faça um programa que lê um número n do teclado, e calcula $n!$

Exemplo 5

```
1 n = int(input('Digite n:'))
2
3 fat = 1 #corresponde a 0!
4 i=1
5 while (i<=n):
6     fat = fat*i
7     i = i + 1
8
9 print("Fatorial de {} é {}".format(n,fat))
```

Estamos calculando $n! = 1 * 2 * 3 * \cdots * (n-1) * n$.

Roteiro

Laços e o comando `break`

O comando `break` faz com que a execução de um laço seja terminada, passando a execução para o próximo comando depois do final do laço.

```
1 i = 0
2 while (i<10):
3   if(i >= 5):
4     break
5   print(i)
6   i = i + 1
7
8 print("Terminou o laço")
```

- O que será impresso?

Laços e o comando break

A condição do laço e o comando break tem o mesma função.

```
1 i = 0
2 while (i<=10):
3   print(i)
4   i = i + 1
```

```
1 i = 0
2 while True:
3   if(i > 10):
4     break
5   print(i)
6   i = i + 1
```

Laços e o comando break

Exemplo:

- Faça um programa que lê um número n e verifica se ele é um número primo:

```
1  n = int(input("Digite um número:"))
2  eprimo = True
3
4  i = 2
5  while (i < n):
6    if (n%i == 0):
7      eprimo = False
8      break
9    i = i + 1
10
11 if(eprimo):
12     print("É primo!!");
13 else:
14     print("Não é primo!!");
```

Poderíamos verificar apenas até $n/2$.

Laços e o comando continue

O comando continue faz com que a execução de um laço seja alterada para final do laço.

```
1 i = 0
2 while (i <= 10):
3   if i==5:
4     i = i + 1
5     continue
6   print(i)
7   i = i + 1
8
9 print('Terminou o laço')
```

- O que será impresso?

Laços e o comando `continue`

Exemplo:

- Imprimindo área de um círculo, apenas se raio for par ($1 \leq r \leq 10$).

```
1 i = 1
2 while (i <= 10):
3   if( r % 2  != 0): #se o número for ímpar pulamos
4     i = i + 1
5     continue
6   area = 3.1415*r**2
7   print("%.2f" %area)
8   i = i + 1
```

- Mas note que poderíamos escrever algo mais simples:

```
1 i = 2
2 while (i <= 10):
3   area = 3.1415*r**2
4   print("%.2f" %area)
5   i = i + 2
```

Roteiro

Extra: `math.sqrt()`

A linguagem **Python** permite o uso de funções matemáticas.

- A função `math.sqrt(x)` calcula $\sqrt{x}$.
- Porém, muitas funções não são nativas da linguagem e ficam localizadas em módulos externos.

```
>>> math.sqrt(16)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    math.sqrt(16)
NameError: name 'math' is not defined
```

Extra: `math.sqrt()`

Para se usar a função `math.sqrt()` (raiz quadrada), por exemplo, é necessário importar o módulo `math`:

```
1 >>> import math
2 >>> math.sqrt(16)
3 4
```

Outras funções do módulo `math`

Mais funções do módulo `math`:

`math.cos(x)`	retorna o cosene de x
`math.sin(x)`	retorna o seno de x
`math.tan(x)`	retorna o tangente de x
`math.sqrt(x)`	retorna a raiz quadrada de x
`math.pow(x, y)`	retorna the value de x to the power de y
`math.fabs(x)`	retorna o valor absoluto de x
`math.factorial(x)`	retorna o fatorial de x
`math.isfinite(x)`	verifica se x é um número finito
`math.log(x, b)`	retorna o logaritmo de x na base b
`math.log10(x)`	retorna o logaritmo de x na base 10
`math.log2(x)`	retorna o logaritmo de x na base 2
`math.gcd(x,y)`	retorna o máximo divisor comum de x e y

https://www.w3schools.com/python/module_math.asp

Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.