UNIVERSIDADE
BEIRA INTERIOR

Operadores, exceções e ciclos

Ferramentas e Aplicações em Biotecnologia

Vasco Ferrinho Lopes
vasco.lopes@ubi.pt

UBI
Ano letivo 2023-2024

Revisão Operadores

Operadores muito importantes na programação, usados frequentemente nas expressões e testes condicionas onde estão envolvidas escolhas. Habitualmente, uma escolha envolve certas condições lógicas.

Operador	Descrição	Exemplo
==	igual a	a == b
!=	diferente de	a != b
>	maior que	a > b
>=	maior ou igual	a >= b
<	menor que	a < b
<=	menor ou igual	a <= b
or	disjunção	x<2 or x>4
and	conjunção	2<x and x<4
not	negação	not x>=3

Considerando como exemplo: **x=3; a=1; b=2**, qual é o valor lógico de cada um dos exemplos da tabela?

Revisão Operadores

```
a = 4

b=7

not a>7

not a>6 or b<8

not a>b == False

a < b+1 and b-a > 1

x = (a+b)/2
```

```
a<x and x<b

a < x < b

c= b-a-3

2*c < b or 2*c > a

b>a>c or a>b>c or a>c>b

not a<b or True

not (a<b or True)
```

Revisão Processamento Condicional

O caso mais simples é onde usamos só o ***if***. Na instrução da linha 1, se a *Condition* for verdadeira (*True*) então as *N* instruções que se seguem serão implementadas, e só nesse caso.

```
if Condition :
  Instruction_1
  Instruction_2
  ...
  Instruction_N
```

```
a = input("Give me a number: ")
a = float(a)

if a < 0 :
  print("Negative number.")
```

Revisão Processamento Condicional

O **segundo caso** expande o ***if*** definindo um bloco alternativo, através da instrução ***else***, que é executado caso a Condition do if seja False.

```
if Condition :
   Instruction_1
   ...
   Instruction_N
else:
   Instruction_1
   ...
   Instruction_M
```

```
a = input("Give me a number: ")
a = float(a)

if a < 0 :
   print("Negative number.")
else:
   print("Positive number.")
```

Revisão Processamento Condicional

O **terceiro caso** da instrução condicional que além de usar o ***if*** e opcionalmente o ***else*** no final, usa também um ou vários ***elif***.

```
if Condition1 :
  Block1(N1)
elif Condition2 :
  Block2(N2)
...
elif Conditionm :
  Blockm(Nm)
else:
  Blockm+1(Nm+1)
```

```
from random import randint

a = randint(0,100)
if a < 50 :
  print("Mediocre")
elif a < 75:
  print("Satisfaz")
else:
  print("Muito Bom")
```

Aqui *Block*(N) representa uma sequência de N instruções

Mecanismo de Proteção de Código

Existem situações em que o programa pode levar à geração de erros, com as mais variadas consequências. Consideramos aqui aquelas situações que não dependem do programador, mas dependem mais de factores externos, com é o caso do input de um utilizador. Vejamos o seguinte exemplo:

```
x = float(input("x = ? "))
y = float(input("y = ? "))
print("x/y =", x/y)
```

- O que acontece se o utilizador indicar **zero** (0) para o valor de y?
- O que acontece se o utilizador introduzir texto e não um número?

Mecanismo de Proteção de Código

A instrução try/except existe para que o programador consiga delimitar código crítico para poder dar resposta em situações de erro:

```
try:
  Block1(N1)
except:
  Block2(N2)
```

```
print('Program to Calculate x/y:')

try:
  x = float(input('  x = ? '))
  y = float(input('  y = ? '))
  print("  x/y =", x/y)
except Exception as e:
  print("ERROR:", e)

print('Programa Terminated.')
```

- O Block1(N1) é o que contém instruções críticas.
- O Block2(N2) é o código alternativo, caso uma situação anómala ocorra em Block1(N1), e só nesse caso.

Instruções de Repetição

- As instruções de repetição (ou “**ciclos**”) são um dos quatro “pilares” fundamen- tais da programação, a par dos outros três: “dados”, “operadores e “condicionais”.

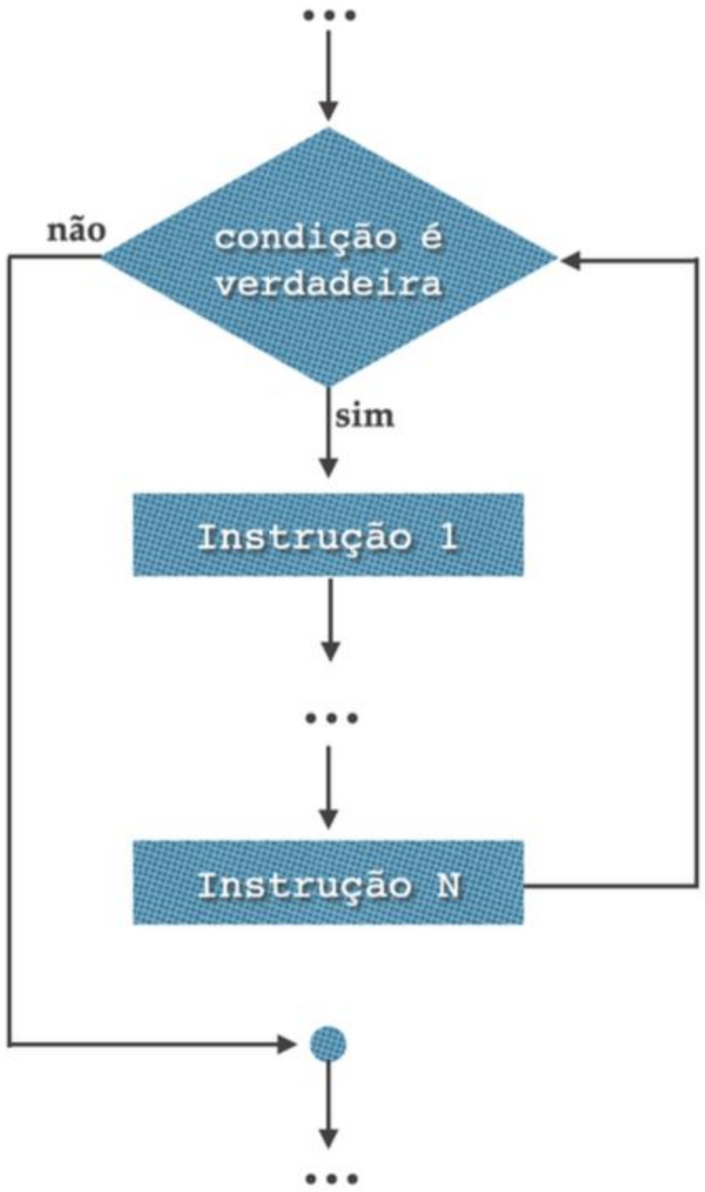
- **Ciclos** consistem num bloco de código que executa repetidamente, mediante a verificação de uma determinada condição lógica.

- De um modo geral, um ciclo compreende sempre uma instrução de controlo, que no fundo é uma instrução condicional, e um bloco de instruções associadas, ou englobadas.

- Em múltiplas situações, há a necessidade de repetir um processamento enquanto certa condição for verdadeira (ou até que seja verdadeira).

Instruções de Repetição

- Consiste num bloco de código que executa repetidamente, mediante a verificação de uma determinada condição lógica.



- Por exemplo, a) queremos um programa **que escreve 10 vezes a frase** “Programar é treinar a mente!”, ou b) um programa que **escreve** a **sequência de números de 1 a 100**, um número por linha.

Instruções de Repetição

- Em Python temos duas instruções de repetição: o ***while*** e o ***for***.

```
while Condition :
    Instruction_1
    ...
    Instruction_N
```

```
i = 0
while i < 10:
    print('Programar é treinar a mente!')
    i = i+1
```

- Podemos "**ler**" da seguinte forma: enquanto a condição lógica "*Condition*" for verdadeira, as ***N*** instruções do ciclo são executadas repetidamente.

- Nos ciclos ***while***, é **comum** que uma das instruções do ciclo contribua para que haja uma **convergência** para o **fim do ciclo**, isto é, que a "***Condition***" venha a ser ***False***.

Instruções de Repetição

- Em Python temos duas instruções de repetição: o ***while*** e o ***for***.

```
for VAR in STRUCT:
    Instruction_1
    ...
    Instruction_N
```

```
for x in [1,2,3]:
    print(x)

for x in {1,2,3}:
    print('Programming is cool!')
```

- O ciclo ***for*** é uma instrução de repetição que depende de uma variável de controlo pertencer aos valores possíveis de uma estrutura de dados, iterando sequencialmente sobre estes.

Instruções de Repetição

- O ciclo ***for*** é uma instrução de repetição que depende de uma variável de controlo pertencer aos valores possíveis de uma estrutura de dados, iterando sequencialmente sobre estes.

```
range(a, b, k) vai gerar:

   a, a+k, ..., a+nk

com:
   nk < b < (n+1)k
```

```
for x in range(4):
  print(x)

for x in range(100,107):
  print(x)

for x in range(1,10,2):
  print(x)
```

- A função ***range*** permite gerar uma sequência de valores inteiros, entre um limite inferior **a** e um limite superior **b**, com um incremento de **k** unidades. Esta função é frequentemente combinada com o ***for***.

Instruções de Repetição

- Quando **sabemos *à priori*** o **número exato de iterações** de um ciclo, é normalmente preferível utilizar um ciclo ***for*** em vez de um ciclo ***while***.

- Caso **não** se saiba **à priori** o **número exato de iterações** de um ciclo, tendemos a utilizar um **ciclo *while***, tendo em atenção especial a condição de paragem!

- Existem duas **condições importantes** quando trabalhamos com ciclos:
  - ***break*** - utilizado para para o ciclo quando executado
  - ***continue*** - utilizado para forçar a execução de um ciclo a começar a próxima iteração (ignora as instruções seguintes da iteração atual)

Exercício

- Escreva um programa que peça um número ao utilizador entre [0,100] e valide se o número introduzido está contido neste intervalo. Se não estiver, deve continuar a pedir um valor ao utilizador até estar.

- Escreva um programa que imprima os valores de 1 a 100, sendo que cada valor deve estar em uma linha diferente.

- Escreva um programa que dado um inteiro (N), escreva um triângulo com a seguinte estrutura:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
.............
1 2 3 ... ... N
```