Rafael Vieira Coelho

rafaelvc2@gmail.com

PARTE 1

4 - Condicionais

4.1 O operador módulo

O operador módulo trabalha com inteiros (e expressões que têm inteiros como resultado) e produz o resto da divisão do primeiro pelo segundo.

Em Python, o operador módulo é um símbolo de porcentagem (%).

A sintaxe é a mesma que a de outros operadores:

```
>>> quociente = 7 // 3
>>> print (quociente)
2

>>> resto = 7 % 3
>>> print (resto)
1
```

Então, 7 dividido por 3 é 2 e o resto é 1.

4.2 Expressões booleanas

Uma expressão booleana é uma expressão que é verdadeira (**True**) ou é falsa (**False**).

Em Python, uma expressão que é verdadeira tem o valor True, e uma expressão que é falsa tem o valor False.

O operador == compara dois valores e produz uma expressão booleana:

```
>>> 5 == 5
True

>>> 5 == 6
False
```

Operadores Relacionais

```
x == y          # testa se x é igual a y
x != y          # testa se x é diferente a y
x > y           # testa se x é maior que y
x < y           # testa se x é menor que y
x >= y          # testa se x é maior ou igual a y
x <= y          # testa se x é menor ou igual a y
in (not in)
```

```
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >= 21
False
```

python™

Ordem de Precedência dos Operadores

Nível	Categoria	Operadores
7(alto)	exponenciação	**
6	multiplicação	*,/,//,%
5	adição	+,-
4	relacional	==,!=,<=,>=,>,<
3	lógico	not
2	lógico	and
1(baixo)	lógico	or

python™

Operadores Lógicos

Existem três operadores lógicos: and, or, not (e, ou, não).

```
>>> y = 0
>>> y and 1
False

>>> 10 > 6 and 3 <= 8
True
```

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
```

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 22
>>> age_0 >= 21 and age_1 >= 21
True
```

python™

4.3 Execução condicional (Se-Então)

```
if x > 0:
    print ("x é positivo")
```

Assim como outras instruções compostas, a instrução if é constituída de um cabeçalho e de um bloco de instruções:

```
CABECALHO:
        PRIMEIRO COMANDO
        ...
        ULTIMO COMANDO
```

4.4 Execução alternativa (Senão)

```
if x % 2 == 0:
        print (x, "é par")
else:
        print (x, "é impar")
```

4.5 Condicionais encadeados

Às vezes existem mais de duas possibilidades e precisamos de mais que dois ramos.

Uma condicional encadeada é uma maneira de expressar uma operação dessas:

```
if x < y:
        print (x, "é menor que", y)
elif x > y:
        print (x, "é maior que", y)
else:
        print (x, "e", y, "são iguais")
```

4.5 Condicionais encadeados

elif é uma abreviação de “else if” (“senão se”). De novo, precisamente um ramo será executado.

Não existe limite para o número de instruções elif, mas se existir uma instrução else ela tem que vir por último:

```
if escolha == ’A’:
        funcaoA()
elif escolha == ’B’:
        funcaoB()
elif escolha == ’C’:
        funcaoC()
else:
        print ("Escolha inválida.")
```

python™

4.6 Condicionais aninhados

Um condicional também pode ser aninhado dentro de outra.

Poderíamos ter escrito o exemplo anterior como segue:

```
if x == y:
        print (x, "e", y, "são iguais")
else:
        if x < y:
                print (x, "é menor que", y)
        else:
                print (x, "é maior que", y)
```

Observação

Operadores lógicos frequentemente fornecem uma maneira de simplificar instruções condicionais aninhadas. Por exemplo, podemos reescrever o código a seguir usando uma única condicional:

```
if 0 < x:
        if x < 10:
                print ("x é um número positivo de um só algarismo.")
```

A instrução print é executada somente se a fizermos passar por ambos os condicionais, então, podemos usar um operador and:

```
if 0 < x and x < 10:
        print ("x é um número positivo de um só algarismo.")
```

Python provê uma sintaxe alternativa que é similar à notação matemática:

```
if 0 < x < 10:
        print ("x é um número positivo de um só algarismo.")
```

python™

4.7 A instrução return

- O comando return permite terminar a execução de uma função antes que ela alcance seu fim.

- Uma razão para usá-lo é se você detectar uma condição de erro:

```
import math

def imprimeLogaritmo(x):
        if x <= 0:
                print ("Somente números positivos, por favor.")
                return
        resultado = math.log(x)
        print ("O log de x é ", resultado)
```

Lembre-se que para usar uma função do módulo de matemática, math, você tem de importá-lo.

python™

4.7 A instrução return

- Podemos usar uma tupla para retornar mais de um valor na função.

```
import math
def imprimeLogaritmo(x):
        if x <= 0:
                return (False, 0)
        resultado = math.log(x)
        return (True, resultado)

def main():
    num = float(input("Qual o valor?"))
    (ok, result) = imprimeLogaritmo(num)
    if ok == True:
        print('O resultado é ', result)
    else:
        print('Somente números positivos, por favor.')
```

Lembre-se que para usar uma função do módulo de matemática, math, você tem de importá-lo.

python™

4.8 O comando while

Os computadores são muito utilizados para automatizar tarefas repetitivas para repetir tarefas idênticas ou similares sem cometer erros.

Para fazer a repetição, também chamada de iteração, em Python, podemos usar o comando while.

Um exemplo:

```
>>> num = 1
>>> while num <= 5:
....        print(num)
....        num += 1 # num = num + 1
```

```
1
2
3
4
5
```

python™

O comando break

Este comando é utilizado para parar e sair de uma repetição.

```
1  while True:
2      line = input('Acabou (sim/não)? ')
3      if line == 'sim':
4          break
5      print (line)
6
7  print ('Oba!')
```

```
Acabou (sim/não)?não
não
Acabou (sim/não)?não
não
Acabou (sim/não)?sim
Oba!
```

#Mesmo código sem utilizar o break (duas formas):

```
1  line = input('Acabou (sim/não)?')
2  while  line != 'sim':
3      print(line)
4      line = input('Acabou (sim/não)?')
5  print('Oba!')
```

```
1  nao_sair = True
2  while  nao_sair:
3      line = input('Acabou (sim/não)?')
4      nao_sair = (line != 'sim')
5      if nao_sair:
6          print(line)
7  print('Oba!')
8
```

4.9 O comando for

Faz a iteração em listas, tuplas e strings.

```
s = 0
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8]:
    s = s + i
    if s > 10:
        break

print "i=%i, s=%i" % (i, s)
```

python™

4.9 O comando for

Faz a iteração em listas, tuplas e strings.

```
r = []
for c in 'this is a string with blanks':
    if c == ' ': continue
    r.append(c)

print ''.join(r)
```

4.9 O comando for

```
cars = ['audi', 'bmw', 'subaru', 'toyota']
for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
```

```
Audi
BMW
Subaru
Toyota
```

python™

4.9 O comando for e Teste se uma lista está vazia (if)

```
requested_toppings = []
if requested_toppings:
    for requested_topping in requested_toppings:
        print("Adding " + requested_topping + ".")
        print("\nFinished making your pizza!”)
    else:
        print("Are you sure you want a plain pizza?")
```

```
Are you sure you want a plain pizza?
```

python™

Exercícios

1) Faça um Programa que peça dois números e imprima o maior deles (maior.py).

2) Faça um Programa que peça um valor e mostre na tela se o valor é positivo ou negativo (testa_valor.py).

3) Faça um Programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido (sexo.py).

4) Faça um Programa que peça os 3 lados de um triângulo. O programa deverá informar se os valores podem ser um triângulo. Indique, caso os lados formem um triângulo, se o mesmo é: equilátero, isósceles ou escaleno (triangulo.py).

Dica:
Três lados formam um triângulo quando a soma de quaisquer dois lados for maior que o terceiro.

Exercícios

5) Dada uma sequência de números inteiros diferentes de zero, terminada por um zero, calcular a sua soma. Por exemplo, para a sequência:

```
12    17    4    -6    8    0
```

o seu programa deve escrever o número `35`.

6) Dados números inteiros n e k, com k >= 0, calcular n elevado a k através de multiplicações sucessivas. Por exemplo, dados os números 3 e 4 o seu programa deve escrever o número 81 (3 * 3 * 3 * 3).

Exercícios

5) Desenvolva um gerador de tabuada, capaz de gerar a tabuada de qualquer número inteiro entre 1 a 10. O usuário deve informar de qual numero ele deseja ver a tabuada (tabuada.py). A saída deve ser conforme o exemplo abaixo:
Tabuada de 5:
5 X 1 = 5
5 X 2 = 10
...
5 X 10 = 50

6) Faça um programa que mostre os n termos da Série a seguir:
S = 1/1 + 2/3 + 3/5 + 4/7 + 5/9 + ... + n/m.

E imprima no final a soma da série (arquivo serie.py).

python™

Exercícios

7) Desenvolva um programa que faça a tabuada de um número qualquer inteiro que será digitado pelo usuário, mas a tabuada não deve necessariamente iniciar em 1 e terminar em 10. O valor inicial e final devem ser informados também pelo usuário, conforme exemplo abaixo (tabuada2.py):
Montar a tabuada de: 5
Começar por: 4
Terminar em: 7

Vou montar a tabuada de 5 começando em 4 e terminando em 7:
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
Obs: Você deve verificar se o usuário não digitou o final menor que o inicial.