Processamento da Informação

Fabrício Olivetti de França

02 de Fevereiro de 2019

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Topics

1. Tipos Padrões
2. Variáveis Auxiliares
3. Condicionais

Tipos Padrões

Tipos Padrões

A linguagem Python fornece diversos tipos internos da linguagem.

Além disso, para cada tipo (ou classe de tipos), também temos funções e operadores já implementados.

Tipos Numéricos

Os tipos numéricos compreendem os valores inteiros (`int`) e reais (`float`), também chamados de ponto flutuante.

Tipo Inteiro

O tipo `int` no Python contém os valores inteiros de precisão arbitrária (limitados a memória do sistema).

Tipo Inteiro

Ou seja, teoricamente podemos representar qualquer número inteiro, na prática o maior número que pode ser representado depende de quanto temos de memória em nosso computador.

Tipo Real

O tipo `float` no Python contém valores reais definidos pela representação binária de ponto flutuante de 64bits de precisão.

Tipo Real

Nesse caso, nossos números são limitados a uma faixa de precisão (valores muito pequenos são considerados zero), além de possuir os valores especiais `inf, -inf` para representar infinito.

Operadores Numéricos

Operador	Resultado
x + y	soma
x - y	subtrai
x * y	multiplica
x // y	quociente
x % y	resto
x / y	divisão
-x	negação
x**y	potencia
abs(x)	valor absoluto

Tipos Boleanos

Os tipos booleanos contém apenas dois valores: `True, False` representando, respectivamente, Verdadeiro e Falso.

Tipos Boleanos

Operador	Resultado
x or y	se x for falso, então y, senão x
x and y	se x for falso, então x, senão y
not x	se x for falso, então True, senão False

Tipos Texto

O tipo `str` representam textos (ou *strings*) e contém uma sequência de símbolos que o representam de tamanho arbitrário.

Tipos Texto

Um tipo texto é definido por uma sequência entre aspas simples ou duplas:

```
"Isso é um texto"
'Isso também é um texto'
```

Operadores de Texto

Operador	Resultado
x + y	concatena dois textos
x * n	sendo n inteiro, repete x por n vezes

Comparações

Para comparar valores de algum dos tipos descritos anteriormente, podemos usar os operadores relacionais, eles retornam um valor booleano:

Operador	Resultado
x < y	estritamente menor que
x <= y	menor ou igual que
x > y	estritamente maior que
x >= y	maior ou igual que
x == y	igual
x != y	diferente

Exercício

Crie uma função que calcula a média entre dois números:

```
def media(x, y):
    ????
```

Resposta

```
def media(x, y):
    return x + y / 2
```

Está correto?

Desambiguidade

Lembrem-se, um algoritmo deve ser desambíguo, mas o que ocorre primeiro, a soma ou a divisão?

Ordem de Precedência

Para definir a ordem das operações, cada linguagem de programação define uma ordem de prioridade para elas. No Python temos do menor para o maior:

Ordem	Operadores
1	`or`
2	`and`
3	`not`
4	`<, <=, >, >=, ==, !=`
5	`+, -`
6	`*, /, //, %`
7	`**`

Ordem de Precedência

Logo, `x + y / 2` dividiria `y` por `2` para então somar com `x`.

Podemos definir ordem de operações com parênteses.

Resposta Correta

```
def media(x, y):
    return (x + y) / 2
```

Exercício

Crie uma função que retorna se um número é par ou não. Para isso, crie antes uma função que indica se um valor **x** é divisível por um número **n**.

Exercício

```
def divisivel(x, n):
    return (x % n) == 0
```

Exercício

```
def eh_par(x):
    return divisivel(x, 2)
```

Exercício

Como você implementaria a função `eh_impar`?

Exercício

```
def eh_impar(x):
    return not eh_par(x)
```

ou

```
def eh_impar(x):
    return not divisivel(x,2)
```

Variáveis Auxiliares

Funções mais elaboradas

Exercício: crie uma função `raiz_mais` que recebe três argumentos `a, b, c` e retorna a raiz da equação de segundo grau como:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Funções matemáticas

Para utilizar funções matemáticas você deve utilizar a biblioteca **math**:

```
import math


def raiz(x):
    return math.sqrt(x)
```

Funções mais elaboradas

```
def raiz_mais(a, b, c):
    return (-b + math.sqrt(b**2 - 4*a*c)) // (2*a)
```

Funções mais elaboradas

Embora ainda seja possível entender o código, ele começa a ficar um pouco complicado demais.

Variáveis Auxiliares

Para melhorar a leitura do código, quebramos a função em várias linhas utilizando **variáveis auxiliares**.

Uma **variável auxiliar** é uma expressão nomeada que pode ser utilizada a partir do momento de sua definição.

Variáveis Auxiliares

```
def raiz_mais(a, b, c):
    delta = b**2 - 4*a*c
    return (-b + math.sqrt(delta)) // (2*a)
```

Variáveis auxiliares

Uma variável auxiliar recebe o conteúdo da expressão avaliada. Na linha:

```
delta = b**2 - 4*a*c
```

A variável `delta` receberá o valor da expressão ao substituir `a, b, c` pelos valores fornecidos.

Variáveis auxiliares

Podemos substituir ou atualizar a definição de uma variável a qualquer momento:

```
def raiz_mais(a, b, c):
    delta = b**2 - 4*a*c
    raiz_delta = math.sqrt(delta)
    return (-b + raiz_delta) // (2*a)
```

Passo a passo

A execução passo a passo de `raiz_mais(3, 1, -2)` ficaria:

Python Tutor

Exercício

Modifique a função abaixo e introduza novas variáveis para facilitar a leitura:

```
def idade_em_segundos(idade):
    return idade*12*30*24*60*60
```

Para isso crie as variáveis auxiliares `meses, dias, horas, minutos`.

Exercício

```
def idade_em_segundos(idade):
    meses = idade*12
    dias  = meses*30
    horas = dias*24
    minutos = horas*60
    return minutos*60
```

Condicionais

Desvios Condicionais

Em certas situações precisamos fazer escolhas do tipo de operação a ser feito:

- Se delta for negativo, emito uma mensagem de erro.
- Se um jogador apostou em Pedra e outro em Papel, o segundo ganhou.
- Se a lâmpada estiver acesa, apaga.

Desvios Condicionais

Para isso utilizamos a intrução `if..else` que controla qual linha do código deverá ser executada:

```
if condicao:
    bloco 1
else:
    bloco 2
```

Desvios Condicionais

No código anterior, caso a condição seja verdadeira, o bloco de instruções 1 será executado, caso contrário, o bloco 2 será executado.

Lâmpada e Botão

Revisando o código da aula passada:

```
def botao(lamp):
    if lamp == "On":
        return "Off"
    else:
        return "On"
```

Condições

As condições da instrução `if` devem ser expressões que avaliam para um valor do tipo `Bool`:

```
lamp == "On" = True / False
x > 0        = True / False
b1 and b2     = True / False
```

XOR

A função booleana **XOR** possui a seguinte tabela verdade:

x1	x2	x1 XOR x2
F	F	F
F	V	V
V	F	V
V	V	F

XOR

Implemente a função `xor(x1, x2)` utilizando os operadores `not`, `and`, `or`.

XOR

```
def xor(x1, x2):
    if (x1 and x2) or (not x1 and not x2):
        return False
    else:
        return True
```

XOR

Algumas soluções podem ser mais simples do que aparentam...

```
def xor(x1, x2):
    return x1 != x2
```

Próxima aula

Exercitaremos o uso de condicionais e aprenderemos sobre **condicionais compostas**, quando precisamos testar múltiplas condições.