

Os elementos da programação

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá



Conteúdo

Introdução

A linguagem de programação Python

Tipos e operações com números

Operações com cadeia de caracteres

Operações com valores booleanos

Operações relacionais

Erros

Criação de funções

Criação de tipos de dados

Introdução

Uma linguagem de programação é mais do que um meio para instruir o computador a fazer tarefas. Uma linguagem também serve como um meio para organizar e expressar as nossas ideias sobre processos.

Todo linguagem de programação fornece:

- Tipos de dados e operações primitivas
 - São as entidades fundamentais da linguagem
- Meios de combinação
 - Como elementos compostos são construídos a partir de outros elementos
- Meios de abstração
 - Como os elementos compostos podem ser nomeados e manipulados como uma unidade

A linguagem de programação Python

A linguagem de programação Python

- Python é uma linguagem de programação de propósito geral simples e ao mesmo tempo poderosa
- Usado por grandes empresas como Google e Yahoo!
- Também é usado por cientistas para cálculos numéricos (com a biblioteca NumPy entre outras)

A linguagem de programação Python

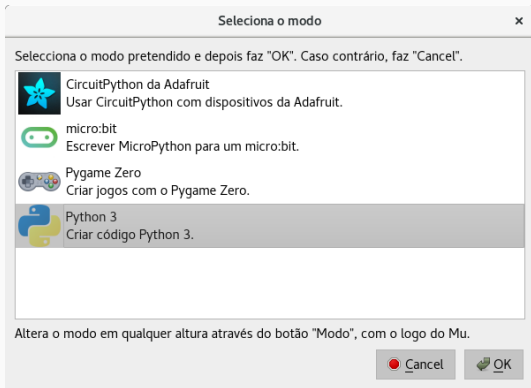
- Vantagens em relação ao Pascal, C e C++
 - Sintaxe mais simples
 - Interpretada (é mais fácil interagir e testar os programas)
 - Tipagem dinâmica
 - Gerência automática de memória
 - Biblioteca padrão extensa

A linguagem de programação Python

- Desvantagens em relação ao Pascal, C e C++
 - Os programas são geralmente mais lentos e consomem mais memória
 - Os erros de tipo são detectados apenas durante a execução do programa

Um editor simples de código Python para iniciantes

<https://codewith.mu/>



Selecione o modo Python 3 e clique em OK



The screenshot displays the Mu Python IDE interface. At the top, the window title is "Mu 1.0.0 - ex1.py". Below the title bar is a toolbar with 13 icons and their corresponding labels: Modo, Novo, Carregar, Guardar, Parar, Diagnóstico, REPL, Gráfico, Aumentar, Diminuir, Aspecto, Verificar, Ajuda, and Sair. The main area is divided into two sections. The upper section is the code editor, showing a Python function definition:

```
1 def dobro(x):  
2     return x + x
```

The lower section is the interactive shell, with the prompt "Em execução: ex1.py" and the following interaction:

```
>>> dobro(20)  
40  
>>>
```

Two red text labels are overlaid on the image: "Área de definições" is positioned over the code editor, and "Área de interação" is positioned over the interactive shell. At the bottom right of the IDE window, there is a "Python" label and a gear icon.

- Área de **interação**

- Digite expressões (pequenos trechos de código), pressione enter, o Python irá avaliar a expressão e exibir o resultado

```
>>> 3 + 4 * 2  
11
```

- Área de **definições**

- Para fazer novas definições crie um novo arquivo (“Novo”)
- Digite as definições e salve o arquivo (“Guardar”)
- Execute o arquivo (“Correr”)
- Teste as novas definições na janela de interações
- Pare a execução (“Parar”)

Tipos e operações com números

Tipos e operações com números

Podem ser

- Inteiros (`int`)
- Ponto flutuante (`float`) - representação aproximada de números reais
- Complexos, frações e decimais (não estudaremos estes tipos)

- Operações aritméticas

```
>>> 3 + 2      # soma
```

```
5
```

```
>>> 4 - 8      # subtração
```

```
-4
```

```
>>> 3 * 6      # multiplicação
```

```
18
```


Tipos e operações com números

- Operações aritméticas

```
>>> 7 / 3 # divisão
```

```
2.3333333333333335
```

```
>>> 7 // 3 # divisão pelo piso
```

```
          # descarta a parte fracionária
```

```
2
```

```
>>> 7 % 3 # resto da divisão pelo piso
```

```
1
```

```
>>> 9.0 // 2.5
```

```
3.0
```

```
>>> 9.0 % 2.5
```

```
1.5
```

Tipos e operações com números

- Operações aritméticas

```
>>> pow(2, 3) # exponenciação
8
>>> 2 ** 3    # exponenciação
8
>>> - 4       # menos unário
-4
>>> abs(-5)   # valor absoluto
5
```

Tipos e operações com números

- Podemos compor expressões assim como fazemos na matemática

```
>>> 3 + 4 * 2
```

```
11
```

```
>>> 2 + 4 / 2 ** 3
```

```
2.5
```

Tipos e operações com números

- O Python utiliza a mesma precedência que estamos acostumados na matemática. Use o acrônimo PEMDAS para lembrar
 - **P**arênteses
 - **E**xponenciação
 - **M**ultiplicação e **D**ivisão
 - **A**dição e **S**ubtração
- Operadores com a mesma precedência são avaliados da direita para a esquerda, exceto a exponenciação

- Exemplo

```
>>> 3 + 4 * 2
```

```
11
```

```
>>> (3 + 4) * 2
```

```
14
```

```
>>> 2 + 4 / 2 ** 3
```

```
2.5
```

```
>>> ((2 + 4) / 2) ** 3
```

```
9
```

Tipos e operações com números

- Qual é o resultado das seguintes expressões?

15 // 7

15 % 7

12 // 27

12 % 27

3 * 4 - 5 / 8 // 3

8 / 4 / 2

2 - 4 ** 3 / 9 % 5

- Conversões de números

```
>>> int(3.4)    # Transforma um valor para int
```

```
3
```

```
>>> int(3.5)
```

```
3
```

```
>>> int(3.6)
```

```
3
```

Tipos e operações pré-definidas

- Conversões de números

```
>>> round(3.4) # Faz arredondamento de um número
```

```
3
```

```
>>> round(3.5)
```

```
4
```

```
>>> round(3.6)
```

```
4
```

```
>>> float(12) # Transforma um valor para float
```

```
12.0
```


Operações com cadeia de caracteres

Operações com cadeia de caracteres

- Uma cadeia de caracteres (*string*) é usada geralmente para armazenar informações simbólicas, como por exemplo palavras e textos
- Uma cadeia de caracteres é escrito em Python entre aspas simples ou dupla

```
>>> 'casa'
```

```
'casa'
```

```
>>> "gota d'agua"
```

```
"gota d'agua"
```

Operações com cadeia de caracteres

- Assim como existem operações pré-definidas para números, também existem operações pré-definidas para cadeia de caracteres

```
>>> 'casa' + ' da ' + 'sogra'
```

```
'casa da sogra'
```

```
>>> 'abc' * 3
```

```
'abcabcabc'
```

```
>>> 'y' + 'a' * 10 + 'h'
```

```
'yaaaaaaaaaah'
```

```
>>> len('física')
```

```
6
```

Operações com cadeia de caracteres

- Conversão de números para cadeia de caracteres

```
>>> str(123)
```

```
'123'
```

```
>>> str(12.3)
```

```
'12.3'
```

- Conversão de cadeia de caracteres para números

```
>>> int('123')
```

```
123
```

```
>>> float('12.3')
```

```
12.3
```

Operações com valores booleanos

Operações com valores booleanos

- Existem dois valores booleanos: True e False
- Operações or

```
>>> False or False
```

```
False
```

```
>>> False or True
```

```
True
```

```
>>> True or False
```

```
True
```

```
>>> True or True
```

```
True
```

Operações com valores booleanos

- Operação and

```
>>> False and False
```

```
False
```

```
>>> False and True
```

```
False
```

```
>>> True and False
```

```
False
```

```
>>> True and True
```

```
True
```

Operações com valores booleanos

- Operação not

```
>>> not False
```

```
True
```

```
>>> not True
```

```
False
```


Operações relacionais

Operações relacionais

- Operações relacionais

```
>>> 3 > 1 + 2      # maior
```

```
False
```

```
>>> 1 + 2 >= 2 + 1 # maior ou igual
```

```
True
```

```
>>> 4 - 1 < 4      # menor
```

```
True
```

```
>>> 4 <= 4         # menor ou igual
```

```
True
```

```
>>> 2 - 1 == 3     # igual
```

```
False
```

```
>>> 4 * 2 != 8     # diferente
```

```
False
```

Erros

Os erros em programação podem ser classificados em três tipos:

- Erros sintáticos
- Erros em tempo de execução
- Erros lógicos

Erros sintáticos

Quando o texto do programa não está de acordo com as regras.
Os erros sintáticos são informados antes da execução do programa.

```
>>> 2a
```

```
File "<stdin>", line 1
```

```
2a
```

```
^
```

SyntaxError: invalid syntax

```
>>> 4 * 5)
```

```
File "<stdin>", line 1
```

```
4 * 5)
```

```
^
```

SyntaxError: invalid syntax

Erros em tempo de execução

São chamados assim porque ocorrem durante a execução do programa. Em geral ocorrem porque alguma expressão não faz sentido, como por exemplo, soma um número com uma cadeia de caracteres.

```
>>> 2 + "3"
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> int("abc")
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: 'abc'
```

O programa executa até o fim mas o resultado gerado não é o esperado.

```
# Vamos calcular a raiz quadrada de 2!  
>>> 2 ** 1 / 2  
1
```

Criação de funções

Criação de funções

Uma das formas de compor novas operações e a criação de funções. A forma inicial de criação de funções é:

```
def nome_da_funcao(parametro1, parametro2, ...):  
    return expressao
```

Exemplo de criação de funções

Defina uma função que calcula o total de segundos de um determinado tempo dado em horas, minutos e segundos.

```
def segundos(horas, minutos, segundos):  
    return 3600 * horas + 60 * minutos + segundos
```

Exemplos de uso

```
>>> segundos(1, 10, 12)  
4212
```

Exemplo de criação de funções

Defina uma função que calcule o valor da hipotenusa dados os valores dos catetos de um triângulo retângulo.

```
def hipotenusa(x, y):  
    return (x ** 2 + y ** 2) ** 0.5
```

Exemplos de uso

```
>>> hipotenusa(3.0, 4.0)  
5.0
```

Exemplo de criação de funções

Defina uma função que crie um texto justificado a direita a partir do texto e do limite de caracteres.

```
def justifica_direita(texto, limite):  
    return ' ' * (limite - len(texto)) + texto
```

Exemplos de uso

```
>>> justifica_direita('casa', 10)  
'      casa'
```

Criar novas funções pode parecer difícil, mas nos veremos uma “receita de projeto” que nos guiará na criação de novas funções.

Criação de tipos de dados

Criação de tipos de dados

Uma das formas de criar novos tipos de dados e a criação de tuplas nomeadas. A forma inicial de criação de novos tipos é:

```
from typing import NamedTuple
```

```
class NomeDoTipo(NamedTuple):  
    campo1: tipo1  
    campo2: tipo2  
    ...
```

Exemplo de criação de tipos de dados

Crie um novo tipo de dado que represente um ponto no plano cartesiano.

```
from typing import NamedTuple
```

```
class Ponto(NamedTuple):  
    x: float  
    y: float
```

Exemplos de uso

```
>>> Ponto(2.0, 3.0)  
Ponto(x=2.0, y=3.0)  
>>> Ponto(2.0, 3.0) == Ponto(3.0, 2.0)  
False
```


Exemplo de uso de novos tipos de dados

Defina uma função que calcule a distância entre dois pontos no plano cartesiano.

```
def distancia(p1, p2):  
    return ((p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2) ** 0.5
```

Exemplos de uso

```
>>> distancia(Ponto(2.0, 7.0), Ponto(6.0, 4.0))  
5.0
```

- Livro Pense em Python 2ª edição. Allen B. Downey
 - Capítulo 1 - A jornada do programa, Seções 1.1, 1.2, 1.4 - 1.7
 - Capítulo 3 - Funções, Seções 3.1 - 3.3.
 - Capítulo 6 - Funções com resultados, Seções 6.1 - 6.4.