

Prática de Programação em Sistemas de Informação

Conceitos básicos



Sumário

Conceitos básicos

Para Início de Conversa...	3
Objetivos	3
1. Introdução à Linguagem Python	4
Referências	16

Desenvolvimento do material

Alexandre Louzada

1ª Edição

Copyright © 2022, Afya.

Nenhuma parte deste material poderá ser reproduzida, transmitida e gravada, por qualquer meio eletrônico, mecânico, por fotocópia e outros, sem a prévia autorização, por escrito, da Afya.

Para Início de Conversa...

O Python é uma linguagem de programação que vem ganhando muita popularidade entre os desenvolvedores de software. Nesta unidade de aprendizagem, vamos entender o que justifica essa popularidade, suas características e aplicabilidades.

Em uma linguagem de programação, é comum trabalharmos com diferentes tipos de dados para tarefas como realizar cálculos, definir o nome de uma pessoa ou produto; por isso, programadores usam operadores para processar os dados, e variáveis para armazenar os dados durante a execução do programa. O uso de operadores e variáveis deve ocorrer de acordo com as características da linguagem de programação utilizada e algumas convenções adotadas pelos desenvolvedores.

De modo similar, os comandos de entrada e de saída de dados devem ser utilizados de acordo com as características e convenções associadas à linguagem de programação, que, como no caso da linguagem Python, pode envolver aspectos como a forma em que os dados serão exibidos.

Dessa forma, vamos estudar os tipos de dados, operadores, variáveis, comandos de entrada, saída e atribuição, e como aplicá-los na linguagem Python.

Objetivos

- Entender os fundamentos da programação estruturada, utilizando como base a linguagem Python.
- Construir diversos tipos de operações, aplicando-os no contexto de programas de computador.



1. Introdução à Linguagem Python

Python foi criado por Guido van Rossum, nascido em 1956, em Haarlem, na Holanda. Evidentemente, van Rossum não desenvolveu todos os componentes do Python, e a velocidade com que essa linguagem de programação se espalhou pelo mundo é resultado do trabalho contínuo de milhares de programadores, testadores e usuários da linguagem.

Sobre as circunstâncias em que o Python foi criado, van Rossum fez o seguinte comentário:

“ Em dezembro de 1989, eu estava procurando por um projeto de programação “passatempo” que me mantivesse ocupado durante a semana em torno do Natal. Meu escritório (...) estava fechado, mas eu tinha um computador em casa e não tinha muito mais em minhas mãos. Eu decidi escrever um interpretador para a nova linguagem de script que eu vinha pensando ultimamente: um descendente de ABC que atraísse os hackers do Unix / C. Eu escolhi o Python como um título de trabalho para o projeto, com um humor um pouco irreverente (e um grande fã do Flying Circus do Monty Python).”

Em 1999, Guido van Rossum definiu seus objetivos para o Python:

- Uma linguagem fácil e intuitiva, tão poderosa quanto a dos principais concorrentes.
- Código aberto, para que qualquer pessoa possa contribuir para o seu desenvolvimento.

- Código comprehensível.
- Permite a redução do tempo de desenvolvimento.

Cerca de 20 anos depois, fica claro que todas essas intenções foram cumpridas. Não é à toa que a linguagem ocupa uma posição de destaque entre as linguagens de programação mais populares no Índice da Comunidade de Programação TIOBE e no ranking do IEEE.



Curiosidade

O índice TIOBE é uma lista ordenada de linguagens de programação, classificada pela frequência de pesquisa na web, usando o nome da linguagem como a palavra-chave. Já o índice IEEE é uma revista editada pelo *Institute of Electrical and Electronics Engineers* e apresenta o ranking das melhores linguagens de programação (PYTHON SOFTWARE FOUNDATION, 1996).

O que torna o Python tão especial? Qual a razão de programadores experientes e novatos desejarem usá-lo? O que faz grandes empresas adotarem o Python para desenvolver seus principais produtos? Existem muitas razões (já listamos algumas delas), mas vamos citá-las novamente de uma maneira mais prática:

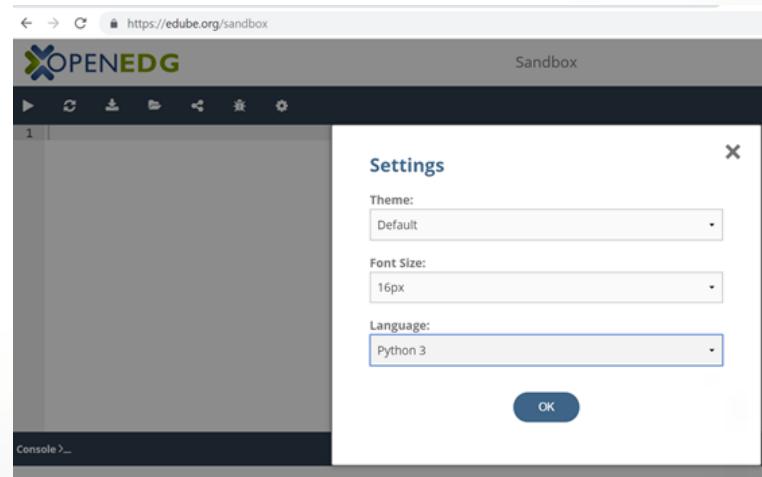
- **É fácil de aprender:** O tempo necessário para aprender Python é menor do que para muitas outras linguagens; isso significa que é possível iniciar a programação real mais rapidamente.
- **É fácil ensinar:** Sua curva de aprendizado é menor em comparação com outras linguagens; isto significa que o professor pode colocar mais ênfase em técnicas de programação gerais (independentes da linguagem).
- **É fácil de usar para escrever novos softwares:** Geralmente, é possível escrever código mais rápido ao usar o Python.
- **É fácil de entender:** Também é mais fácil entender o código de outra pessoa mais rápido se estiver escrito em Python.
- **É fácil de obter, instalar e implantar:** O Python é gratuito, aberto e multiplataforma (com o mesmo código, podemos gerar uma aplicação para diversos sistemas operacionais, como o Windows, Linux e Mac).

Importante

Python é uma linguagem de propósito geral, podendo ser utilizado para o desenvolvimento de aplicações para desktop, web, dispositivos móveis, sistemas embarcados, além de ser amplamente usado para análise de dados. É também uma linguagem interpretada, sendo assim, o código fonte, ao invés de ser compilado para gerar o executável (como na linguagem C), passa por um interpretador, que irá gerar a aplicação conforme o código.

É hora de começar a escrever um código Python real e funcional. Vai ser muito simples, por enquanto. Como vamos mostrar alguns conceitos e termos fundamentais, usaremos esse trecho de código como ponto de partida.

Inicie o Edube Sandbox, abrindo o navegador e digitando o endereço <https://edube.org/sandbox>. Em seguida, accese o último botão (em forma de engrenagem) e escolha a opção “Python 3”, que é a versão do Python que estamos utilizando.



Na Linha 1, digite o seguinte comando: **print ('Primeiro programa em Python!')**. Em seguida, é só acessar a primeira opção (executar) e, se tudo estiver de acordo, o resultado irá aparecer na área de console.



The screenshot shows a web-based Python sandbox environment. At the top, there's a header with the OPENEDG logo and the word "Sandbox". Below the header is a toolbar with standard file operations like Open, Save, and Print. The main area contains a code editor with one line of Python code: "print('Primeiro programa em Python!')". Below the code editor is a "Console" tab. Underneath the tab, the output of the program is displayed: "Primeiro programa em Python!". The entire interface is styled with a dark theme.

Como você pode ver, o primeiro programa consiste nas seguintes partes:

- a palavra print;
- um parêntese de abertura;
- uma aspa;
- um grupo de caracteres, formando a frase **Primeiro programa em Python!**;
- outra aspa;
- um parêntese de fechamento.

Cada um dos itens acima desempenha um papel muito importante no código. A palavra “print” é um nome de uma função do Python capaz de exibir um resultado que pode ser visualizado pelo usuário. Para que a função funcionasse de forma adequada, foi necessário digitar o que se desejava exibir (a frase) e alguns requisitos como o uso de aspa e parênteses.

Com o exemplo que vimos acima, foi possível comprovar algumas das afirmações sobre a linguagem Python, como a facilidade de compreensão e escrita não é mesmo?

Vamos, agora, nos familiarizar com alguns comandos básicos do Python.

Comentários

O Python possui capacidade de comentários para fins de documentação no código. Os comentários começam com #, e o Python renderiza o resto da linha como um comentário.





```
1 #Esse programa é um pequeno exemplo
2 print('Primeiro programa em Python!')
```

Console >_

Primeiro programa em Python!

Docstrings

O Python também estendeu a capacidade de documentação, chamada docstrings, que pode ser formada por uma ou várias linhas. Python usa aspas triplas no início e no final da docstring.

```
1 """
2 Esse programa é um pequeno exemplo
3 criado para mostrar como é simples
4 escrever um código para apresentar
5 uma mensagem na tela.
6 """
7 print('Primeiro programa em Python!')
```

Console >_

Primeiro programa em Python!

Criando Variáveis

Ao contrário de outras linguagens de programação, o Python não possui comando para declarar uma variável. Ela é criada no momento em que você atribui um valor a ela pela primeira vez.



```
1 x = 5  
2 y = "Maria"  
3 print(x)  
4 print(y)
```

Console >_

```
5  
Maria
```

As variáveis não precisam ser declaradas com nenhum tipo específico e podem até mudar o tipo depois de serem definidas.

! Importante

Em programação, este conceito é chamado de “duck typing” (tipagem do pato), baseado na expressão, em inglês, de James Whitcomb Riley, que dizia: “Quando eu vejo uma ave que caminha como um pato, nada como um pato e grasna como um pato, eu chamo esta ave de ‘pato’”.



```
1 x = 5  
2 y = "Maria"  
3 x = "Ana"  
4 print(x)  
5 print(y)
```

Console >_

```
Ana  
Maria
```

Uma variável pode ter um nome abreviado (como `emp`) ou um nome mais descritivo (`idade`, `salario-base`). Além disso, existem algumas regras que devem ser acatadas:

- Um nome de variável deve começar com uma letra ou o caractere de sublinhado.
- Um nome de variável não pode começar com um número.
- Um nome de variável só pode conter caracteres alfanuméricos e sublinhados.
- Python faz distinção entre maiúsculas e minúsculas.

Como vimos nos exemplos anteriores, o comando `print` é usado para exibir o resultado de variáveis. Para combinar texto e uma variável, o Python usa o caractere `+`. Veja o exemplo:

```
x = "Unigranrio"  
print("Universidade " + x)
```

Você também pode usar o caractere `+` para adicionar uma variável a outra variável.

```
x = "Universidade"  
y = "Unigranrio"  
z = x + y  
print(z)
```

! Importante

Para números, o caractere `+` funciona como um operador matemático, por isso, se você tentar combinar uma string e um número, o Python apresentará um erro.

Operadores

Um operador é um símbolo da linguagem de programação, que é capaz de operar nos valores e nas variáveis que os armazenam. Por exemplo, assim como na matemática, o sinal `+` é o operador que pode somar dois números, fornecendo o resultado da adição.

Nem todos os operadores Python são tão óbvios quanto este sinal, portanto, vamos exemplificar os operadores aritméticos disponíveis no Python com o auxílio da tabela a seguir:



Operador	Descrição	Exemplo	Resultado
+	Soma	print (3 + 2)	5
-	Subtração	print (7 - 5)	2
*	Multiplicação	print (4 * 4)	16
/	Divisão	print (10/5)	2.0
//	Divisão de inteiros	print (10//5)	2
**	Potenciação	print (3 ** 2)	9
%	Módulo (resto da divisão)	print (4 % 2)	0

O Python utiliza operadores relacionais para realizar comparações. O resultado de uma comparação é um valor lógico (ou booleano), verdadeiro ou falso (True ou False). Os exemplos a seguir comparam valores numéricos:

Operador	Descrição	Exemplo	Resultado
==	igual	print (3 == 2)	False
!=	diferente	print (7 != 5)	True
>	maior	print (4 > 4)	False
<	menor	print (10 < 5)	False
>=	maior ou igual	print (10 >= 5)	True
<=	menor ou igual	print (3 <= 2)	False

Como o resultado de uma comparação é um booleano, comparações podem ser encadeadas por meio de operadores lógicos.

Operador	Descrição	Exemplo	Resultado
and	e (conjunção)	print (2 == 2 and 4 > 3)	True
or	ou (disjunção)	print (2 != 2 or 4 > 4)	False
not	não (negação)	print (not (4 > 4))	True

Os operadores “bitwise” são usados para comparar números (binários).

Operador	Nome	Descrição
&	AND	Define cada bit como 1, se ambos os bits forem 1.
	OR	Define cada bit como 1, se um dos dois bits for 1.
^	XOR	Define cada bit como 1, se apenas um dos dois bits for 1.
~	NOT	Inverte todos os bits.
<<	Zero fill left shift	Retorna um valor “x” com os seus bits movidos “y” casas à esquerda, adicionando zeros aos bits “novos” da direita. É o mesmo que multiplicar x por $2^{**}y$.
>>	Signed right shift	Retorna um valor “x” com seus bits movidos “y” casas à direita. É o mesmo que $x // 2^{**}y$.

Os operadores de identidade são usados para comparar os objetos, não se forem iguais, mas, se forem realmente o mesmo objeto, com o mesmo local de memória.

Operador	Descrição	Exemplo
is	Retorna true se ambas as variáveis forem o mesmo objeto.	x is y
is not	Retorna true se ambas as variáveis não forem o mesmo objeto.	x is not y

Operadores de atribuição são usados para atribuir valores a variáveis. Por exemplo:

```
x=5  
print(x)
```

Sabemos que, ao executar o exemplo acima, será exibido o valor 5. Observe os outros operadores.

Operador	Exemplo	Equivalente a...
<code>+=</code>	<code>x += 3</code>	<code>x = x + 3</code>
<code>-=</code>	<code>x -= 3</code>	<code>x = x - 3</code>
<code>*=</code>	<code>x *= 3</code>	<code>x = x * 3</code>
<code>/=</code>	<code>x /= 3</code>	<code>x = x / 3</code>
<code>%=</code>	<code>x %= 3</code>	<code>x = x % 3</code>
<code>//=</code>	<code>x //= 3</code>	<code>x = x // 3</code>
<code>**=</code>	<code>x **= 3</code>	<code>x = x ** 3</code>
<code>&=</code>	<code>x &= 3</code>	<code>x = x & 3</code>
<code> =</code>	<code>x = 3</code>	<code>x = x 3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>>>=</code>	<code>x >>= 3</code>	<code>x = x >> 3</code>
<code><<=</code>	<code>x <<= 3</code>	<code>x = x << 3</code>

Você pode testar cada um dos operadores acima alterando o exemplo anterior, como no código a seguir.

```
x=5  
x += 3  
print(x) #exibe na tela o valor 8
```

Tipos de Dados

A linguagem Python permite trabalhar com dados numéricos inteiros, decimais (ponto flutuante) e complexos; caracteres e cadeias de caracteres (strings) e valores lógicos (verdadeiro e falso).

! Importante

Os números de ponto flutuante podem ser um número científico com um “e” para indicar a potência de 10. Por exemplo: 5e2.

Números complexos são escritos com um “j” como a parte imaginária. Por exemplo: 8j.

A documentação do Python não trata os tipos de dados elementares (primitivos) com a nomenclatura normalmente encontrada na documentação da maioria das linguagens (Tipo Primitivo). Na documentação oficial, os tipos primitivos são chamados de tipos built-ins, ou então, tipos construídos. Mais adiante, vamos considerar mais aspectos relacionados a esse tema.

Para verificar o tipo de qualquer objeto no Python, use a função type(), como no exemplo a seguir:

```
a = 4 # int  
b = 3.5 # float
```

```
c = 5e2 #float (o mesmo que 5 x 102)  
d = 8j # complex  
print(type(a))  
print(type(b))  
print(type(c))  
print(type(d))
```

Pode haver momentos em que um programador precise especificar um tipo em uma variável. Isso pode ser feito com um recurso chamado **casting**. O Python é uma linguagem orientada a objetos e, como tal, usa classes para definir tipos de dados, incluindo seus tipos primitivos. Casting em Python é feito usando as seguintes funções:

int ()	Constrói um número inteiro a partir de um literal inteiro, um literal float (arredondando para baixo para o número inteiro anterior), ou um literal string (fornecendo a string representa um número inteiro).
float ()	Constrói um número float a partir de um literal inteiro, um literal float ou um literal string (desde que a string represente um float ou um inteiro).
str ()	Constrói uma string a partir de uma ampla variedade de tipos de dados, incluindo strings, literais de inteiros e literais de float.

Exemplo:

```
x = int(1) # x passa a ser 1
```

```
y = int(2.8) # y passa a ser 2
z = int("3") # z passa a ser 3
x = float(1) # x passa a ser 1.0
y = float(2.8) # y passa a ser 2.8
z = float("3") # z passa a ser 3.0
w = float("4.2") # w passa a ser 4.2
x = str("s1") # x passa a ser 's1'
y = str(2) # y passa a ser '2'
z = str(3.0) # z passa a ser '3.0'
```

Vimos, até aqui, que as strings em Python são cercadas por aspas simples ou aspas duplas ('olá' é o mesmo que "olá") e que podem ser enviadas para a tela usando print(). Por exemplo: print ("olá"). Como muitas outras linguagens de programação populares, as strings no Python são matrizes de bytes representando caracteres unicode.

No entanto, o Python não tem um tipo de dados de caractere; um único caractere é simplesmente uma string com um comprimento de 1. Com o print, é possível usar colchetes para acessar elementos da string.

Exemplo:

```
a = "Python!"
print(a[1]) # exibe na tela y
```

```
b = "Programando em Python"
print(b[2:5]) # exibe na tela posição 2 para a posição 5 (não incluída):
ogr
```

Vamos destacar, agora, alguns métodos para trabalhar com strings:

- O método strip () remove qualquer espaço em branco do começo ou do fim.

```
a = "    Programando em Python    "
print(a.strip()) # exibe na tela "Programando em Python"
```

- O método len () retorna o tamanho de uma string.

```
a = "Programando em Python"
print(len(a)) # exibe na tela o valor 21
```

- O método lower () retorna a string em letras minúsculas.

```
a = "Programando em Python"
print(a.lower()) # exibe na tela "programando em python"
```

- O método upper () retorna a string em maiúsculas.

```
a = "Programando em Python"
print(a.upper()) # exibe na tela "PROGRAMANDO EM PYTHON"
```

- O método replace () substitui uma string por outra string.

```
a = "Python"  
print(a.replace("P", "C")) # exibe na tela "programando em Cython"
```

- O método `split()` divide a string em substrings para encontrar instâncias do separador.

```
a = "Programando,Python"  
print(a.split(",")) # exibe na tela ['Programando', 'Python']
```

Entrada de Dados

A função `input()` faz uma pausa no programa e espera uma entrada do usuário pelo terminal. A função `input()` faz uma pausa no programa e espera uma entrada de um dado pelo usuário. Para ler esse dado, a função `input()` espera que, após a digitação, o usuário aperte a tecla enter. Após isso, a função `input()` lê essa entrada como uma string; portanto, se a entrada esperada for um número, ela deve ser convertida usando o casting. Após isso, `input()` lê essa entrada como uma string; portanto, se a entrada esperada for um número, ela deve ser convertida usando o casting.

Exemplo 1:

```
print("Digite seu nome:")  
x = input()  
print("Muito bom te conhecer, " + x)
```

```
"""
```

Outra forma de fazer...

```
x = input("Digite seu nome:")  
print("Boa tarde, " + x)  
"""
```

Exemplo 2:

```
print('Programa para somar dois valores inteiros')  
valor1 = int(input('Entre com o primeiro valor: '))  
valor2 = int(input('Entre com segundo valor: '))  
soma = (valor1 + valor2)  
print('Resultado da soma:', soma)
```

Método `format()`

O método `format()` serve para criar uma string que contém campos entre chaves a serem substituídos pelos argumentos de format.

Exemplo 1:

```
frase = 'O filme {0} merece {1} estrelas'  
print(frase.format('Pantera Negra', 5))
```

No exemplo, podemos ver que os campos de substituição na string estão associados aos parâmetros de format por numeração dentro das chaves,

começando em zero para o primeiro parâmetro, 1 para o segundo parâmetro, e continuando assim sucessivamente.

Também podemos usar campos nomeados, sendo que estes devem vir depois dos parâmetros simples no método format, como no exemplo abaixo:

Exemplo 2:

```
texto = '{0} percorreu {quilometragem} quilômetros'
print('Programa para calcular quanto um motorista percorreu em uma
viagem')
nome = input('Entre com o nome do motorista:')
kmi = int(input('Digite a quilometragem inicial do veículo:'))
kmf = int(input('Digite a quilometragem final do veículo:'))
print(texto.format(nome, quilometragem = kmf - kmi ))
Estrutura base do format com strings, com o seguinte esquema:
:[preencher][alinhar][largura].[precisão]
```

Preencher	Alinhar	Largura	Precisão
Qualquer caractere	< esquerda > direita ^ centro	largura mínima do campo	largura máxima do campo

Exemplo:

```
texto = 'Programando com Python'
# alinha a direita com 20 espaços em branco
print("{0:>20}".format(texto))
# alinha a direita com 20 símbolos #
print("{0:#>20}".format(texto))
# imprime só as primeiras cinco letras
print("{0:.5}".format(texto))
Agora, vamos ver como format() trabalha com os números. Sua estrutura base é:
:[preencher][alinhar][sinal][largura].[precisão][tipo]
```

Exemplo:

```
print('Programa para calcular o salário de um funcionário')
nome = input('Entre com o nome do funcionário:')
base = float(input('Digite o salário base:'))
desconto = float(input('Digite o desconto:'))
salário = base - desconto
print('{0} teve média igual a: {1:4.2f}'.format(nome, salario))
```

Nesta unidade, vimos conceitos iniciais sobre a linguagem de programação Python. Assim como em outras linguagens, é possível usar

a linguagem Python para criar e utilizar linhas de comentários a fim de auxiliar na documentação do código. Em Python, também podemos construir variáveis, trabalhar com diferentes tipos de dados e utilizar operadores para processar os dados.

Vimos que a linguagem Python utiliza comandos de entrada e de saída que permitem que o usuário digite os dados e obtenha, na tela, informações relacionadas aos problemas que está resolvendo por intermédio do código que escreveu. Por meio desses conceitos, podemos criar uma sequência de instruções para realizar tarefas simples.

Referências

DEVMEDIA. **Unicode** - Conceitos básicos. Disponível em: <https://www.devmedia.com.br/unicode-conceitos-basicos/25169>. Acesso em: 01 mar. 2019.

PYTHON SOFTWARE FOUNDATION. **Foreword for “Programming Python”** (1st ed.). Disponível em: <https://www.python.org/doc/essays/foreword/>. Acesso em: 01 mar. 2019.