
Cartilha de Python

Um pequeno guia para programar na
Linguagem



Por Edkallenn Lima, servidor público e
professor

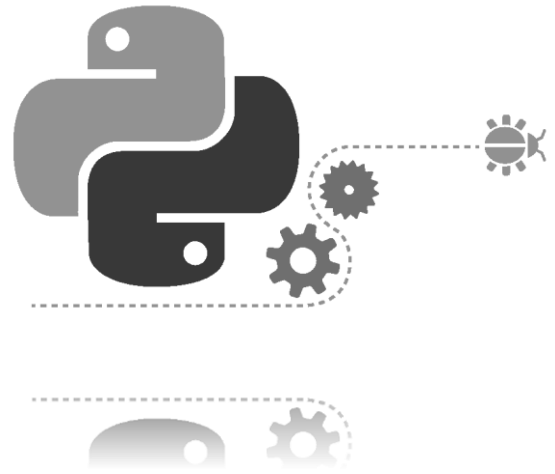
Python

- Criada e conduzida por Guido Von Rossum
- Mantida atualmente pela Python Software Foundation, em um processo comunitário (opensource)
 - Mudanças fundamentais são discutidas em comunidade
 - Aprovação final das mudanças é revisada pelo criador
- O nome Python vem do grupo de humor inglês Monty Python



Python

- É uma linguagem de propósito geral
- Ela pode ser usada para escrever código para qualquer tarefa
- Python é interpretada e Orientada a Objetos
- Há duas versões que são desenvolvidas concomitantemente:
 - Python 2 e Python 3
 - Usaremos aqui Python 3
 - As versões não são compatíveis





Objetivos de Python

Algumas características da linguagem.

- **Clareza e simplicidade do código**
- **Portabilidade (bytecode)**
- **Multipropósito**
- **Multiparadigma**
- **Linguagem de tipagem dinâmica (tipos de dados de alto nível)**
- **Interoperabilidade com outras linguagens (C, C++, Java, etc)**

– Características

Clareza e simplicidade



Features:

- Eliminação de delimitadores de código (não precisa de ; ao final de instruções)
- Recuo sintático (indentação obrigatória)
- Tipagem dinâmica
- Gerenciamento de memória automático (garbage collector)
- Autocontida

– Como é um programa em Python?

```
1  # -*- coding: utf-8 -*-
2
3  import sys
4  print('Bem vindo ao Python, versão', sys.version)
5  print ('Digite um número inteiro:')
6  n = int(input())
7  for i in range(n):
8      print ('Mensagem ', i)
9      print ('Obrigado, e até logo!')
```

– Características

Portabilidade



Features:

- Modelo de execução baseado em máquina virtual
- Para executar em uma plataforma basta ter um interpretador Python disponível.
 - Windows, Mac, Linux e Unix
 - Jython → Python para Java
 - IronPython → Python para o .NET antigo (antes do .Net Core)
 - PyObjc → Python para Cocoa
 - Versão mais comum: CPython (escrita em C)

– Características

Multipropósito



Features:

- Foi criada inicialmente para ser uma linguagem de Script de Shell no [sistema operacional Amoeba](#)
- Hoje pode ser usada em diferentes domínios de aplicação:
 - Desktop (Tk/Tcl, wxPython, Jython, IronPython)
 - Web (Django, Flask, Pylons, Grok, TurboGears, Web2Py etc)
 - Web services
 - Data Science

– Características

Multiparadigma



Features:

- Python suporta construções, principalmente nestes paradigmas:
 - Imperativa (estruturada)
 - Funções, estruturas de controle, módulos
 - Orientada a objetos (imperativa)
 - Classes, objetos, herança e polimorfismo
 - Funcional
 - List comprehensions, manipulações e funções

- Características

Tipagem dinâmica...



Features:

- Tudo é um objeto
- Variáveis atuam como referências (ponteiros)
- Tudo ocorre em tempo de execução
- Objetos possuem estrutura dinâmica
 - Atributos são acrescentados em tempo de execução
 - Classes podem ser modificadas em tempo de execução

– Características

... e forte



Features:

- Tipagem forte significa que o interpretador do Python avalia as expressões (evaluate) e não faz coerções automáticas entre tipos não compatíveis (conversões de valores)

– Características

Interoperabilidade



Features:

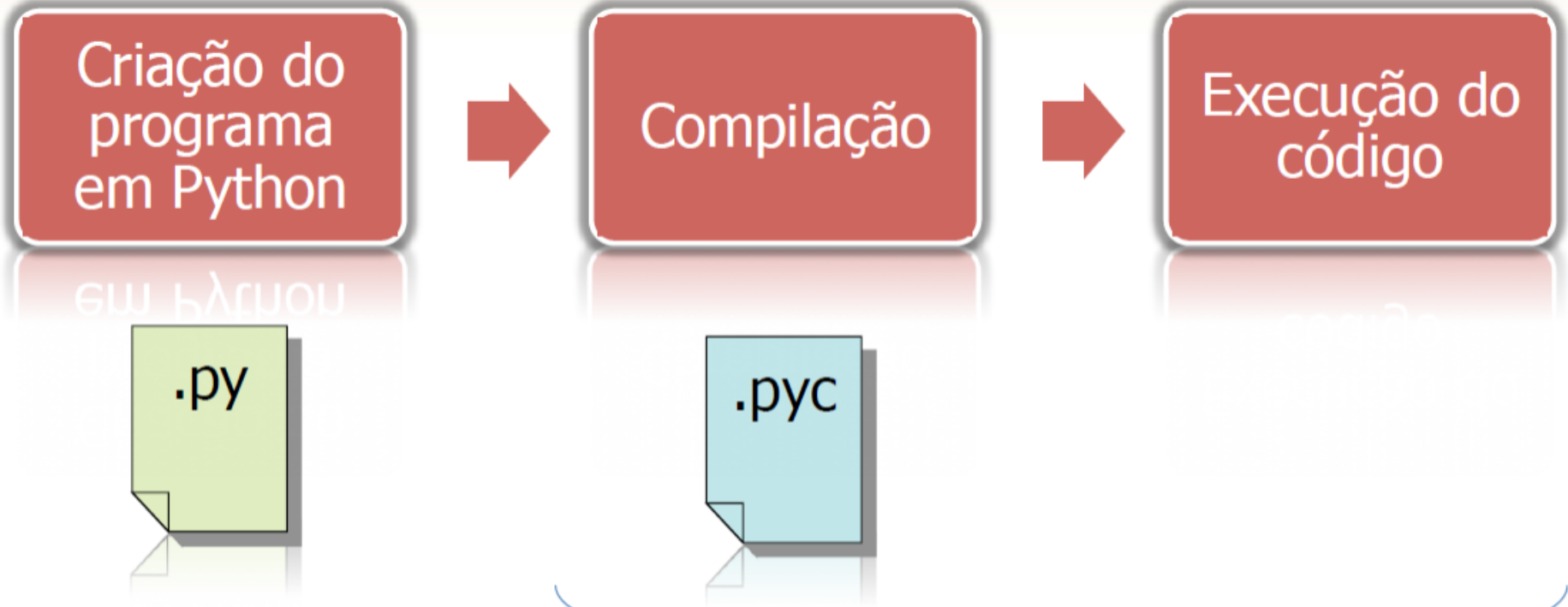
- Módulos do Python podem ser implementados em código nativo
 - Portanto, escritos em linguagens com C, C++ etc.
- Programas em Python podem realizar chamadas ao sistema operacional, iniciar processos, etc.
- Entre outras coisas



PORQUE APRENDER PYTHON?

- Fácil de aprender a praticar
- Fortemente usado no mercado: Google, Facebook(Instagram), Microsoft, Dropbox, Globo.com, etc.
- Utilizando em várias áreas - web, data science, devops, automação, IA e muito mais.

Execução de um programa em Python



Realizados automaticamente pelo
interpretador do python

Interpretador interativo

- Uma das grandes características de Python é o fato da linguagem ter um “terminal interativo” onde comandos podem ser digitados e testados com uma resposta na hora
 - Excelente para testar código e rotinas
 - Além de facilitar o aprendizado
-

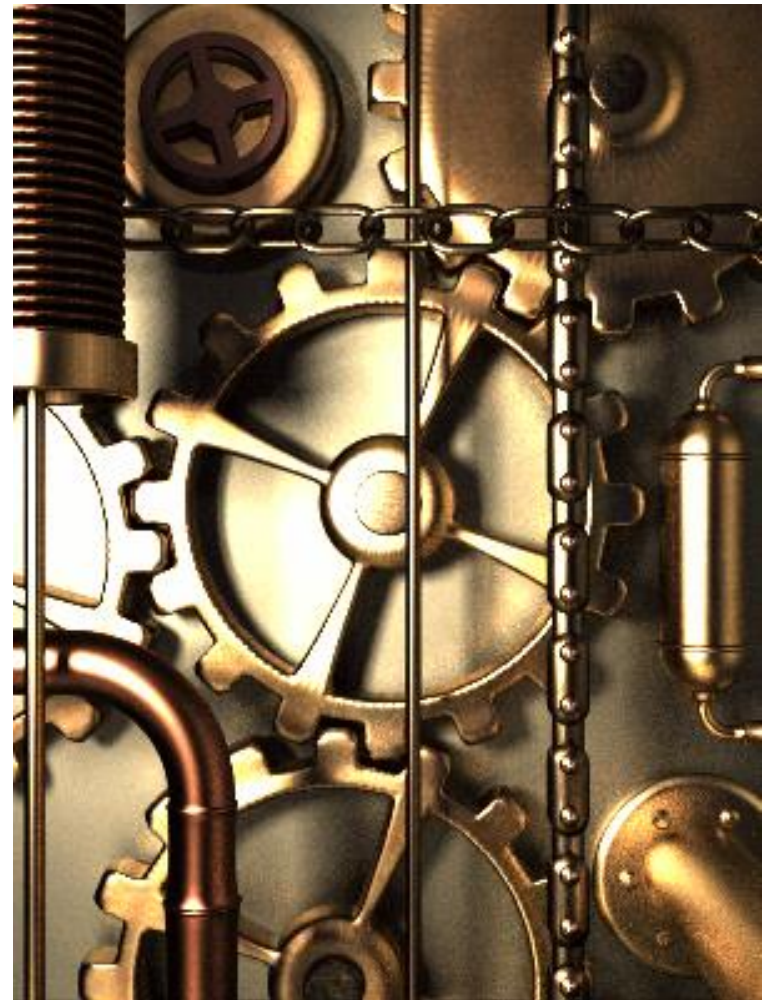
A **melhor forma** de **aprender a programar** é usando o **interpretador** em modo **interativo**.

Dessa forma você pode **digitar comandos linha por linha** e **observar a cada passo como o computador interpreta e executa esses comandos**.

Para fazer isso em Python, há duas maneiras:

1-executar o interpretador em modo texto (chamado "Python (command line)" no Windows, ou simplesmente python no Linux)

2-usar o IDLE, que é um ambiente baseado em janelas.



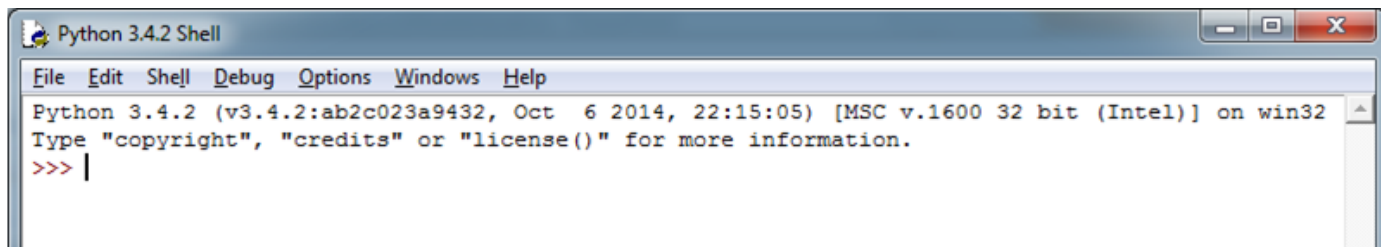
- Vamos usar o IDLE (ou o interpretador Python – command line) para realizar os primeiros passos em Python neste curso



```
ed1rac@ed1rac-VirtualBox: /  
ed1rac@ed1rac-VirtualBox:/$ python  
Python 2.7.3 (default, Apr 10 2013, 05:46:21)  
[GCC 4.6.3] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```



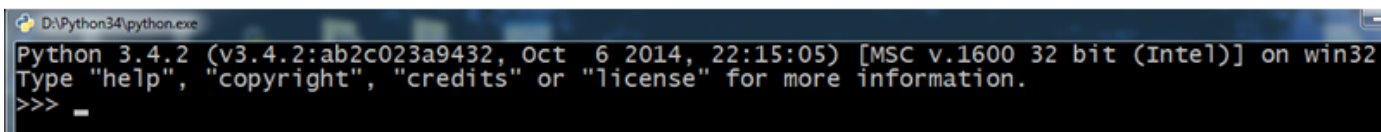
Ao executar o interpretador que você escolheu, você verá uma mensagem com informações de *copyright* mais ou menos como essa:

A screenshot of a Windows application window titled "Python 3.4.2 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area displays the following information: "Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32", followed by "Type 'copyright', 'credits' or 'license()' for more information." and a red prompt ">>>" with a cursor.

```
Python 3.4.2 Shell

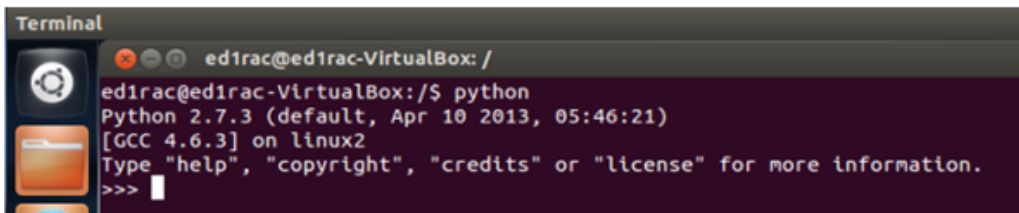
File Edit Shell Debug Options Windows Help

Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

A screenshot of a Windows command prompt window with the title bar "D:\Python34\python.exe". The command prompt shows the same Python version and copyright information as the first screenshot, with a red prompt ">>>" and a cursor.

```
D:\Python34\python.exe

Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

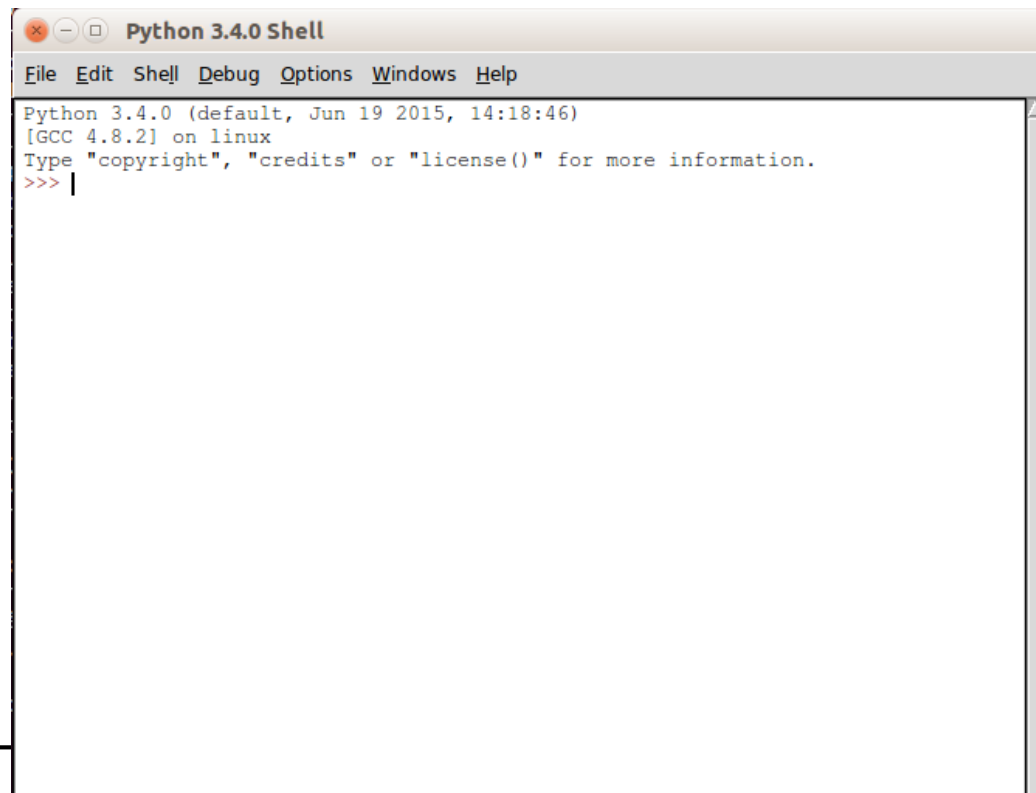
A screenshot of a Linux terminal window titled "Terminal". The prompt is "ed1rac@ed1rac-VirtualBox: /". The user has entered the command "python", and the terminal displays the following information: "Python 2.7.3 (default, Apr 10 2013, 05:46:21)", "[GCC 4.6.3] on linux2", and "Type 'help', 'copyright', 'credits' or 'license' for more information." followed by a red prompt ">>>" and a cursor.

```
Terminal

ed1rac@ed1rac-VirtualBox: /

ed1rac@ed1rac-VirtualBox:/$ python
Python 2.7.3 (default, Apr 10 2013, 05:46:21)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

IDLE no Linux



The screenshot shows a terminal window titled "Python 3.4.0 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area displays the following output:

```
Python 3.4.0 (default, Jun 19 2015, 14:18:46)
[GCC 4.8.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> |
```



Conceitos Básicos

- O símbolo ">>>" exibido pelo interpretador é o que os americanos chamam de "**prompt**", que alguns traduzem por "**aviso**"
 - Em computação, o **prompt** informa o usuário que **o sistema está pronto** para receber um novo comando).
 - Para sair do interpretador você pode **fechar a janela do IDLE**, ou **teclar [CTRL]+[D]** (no IDLE ou no interpretador em UNIX) ou **[CTRL]+[Z]** e **então [ENTER]** (no interpretador DOS).
-

Expressões básicas

Expressões aritméticas

- +, -, *, /
- // %
- **

• Expressões de comparação

- >, <, >=, <=, ==, !=

• Expressões lógicas

- and, or, not



Dica

É interessante usar sempre o interpretador interativo para testar os comandos!



Literais

- Inteiros (long da linguagem C)
 - ◆ 0, 123, 4444, 231
- Longos (inteiros de precisão arbitrária - bignum)
 - ◆ 0L, 231L, 4444L
 - ◆ (o L pode ser minúsculo, mas é bom evitar)
- Ponto flutuante
 - ◆ 1.23, 444.4, 123.4e+56
- Inteiros podem ser em hexa ou octal
 - ◆ 0xFF, 0xbabaca, 0123, 02222, 010



Literais

→ Números complexos

◆ • $100 + 2j$

→ • Strings (delimitados com aspas ou apóstrofes)

◆ • 'tulio', "toffolo"

→ • Booleans

◆ • True ou False (mas o Python considera como verdadeiro, além de True, qualquer valor diferente de 0, None, ou estruturas de dados vazias)

→ • Referência a nenhum objeto (equivalente a NULL)

◆ • None

Uma calculadora?

```
>>> 7/2
3.5
>>> 7*2+14-3/2.25*5
21.333333333333336
>>> 2.4*2
4.8
>>> 1 + 2 * 3
7
>>> |
```

```
>>> (1 + 2) * 3
9
>>> (9 - ( 1 + 2 ) ) / 3.0
2.0
>>> (9 - ( 1 + 2 ) / 3.0
)
8.0
>>> (9 - 1 + 2) / 3.0
3.3333333333333335
>>> |
```

```
>>> largura = 20
>>> altura = 5*9
>>> largura * altura
900
>>> |
```

Não esqueça de executar os comandos!

Muito mais que uma calculadora

```
>>> x = y = z = 4 * 2
>>> x
8
>>> y
8
>>> z
8
>>>
```

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7 / 2
3.5
>>> |
```

```
>>> 1j * 1j
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3 + 1j*3
(3+3j)
>>> (3 + 1j)*3
(9+3j)
>>> (1 + 2j)/(1+1j)
(1.5+0.5j)
```

Execute os comandos!

Calculadora de complexos

```
>>> a = 1.5 + 0.5j  
>>> a.real  
1.5  
>>> a.imag  
0.5
```

Execute os comandos!

—

... e mais

```
>>> a = 1.5 + 0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

```
>>> taxa = 12.5 / 100
>>> preco = 100.50
>>> preco * taxa
12.5625
>>> preco + _
113.0625
>>> round(_,2)
113.06
```

Execute os comandos!

Muito mais...

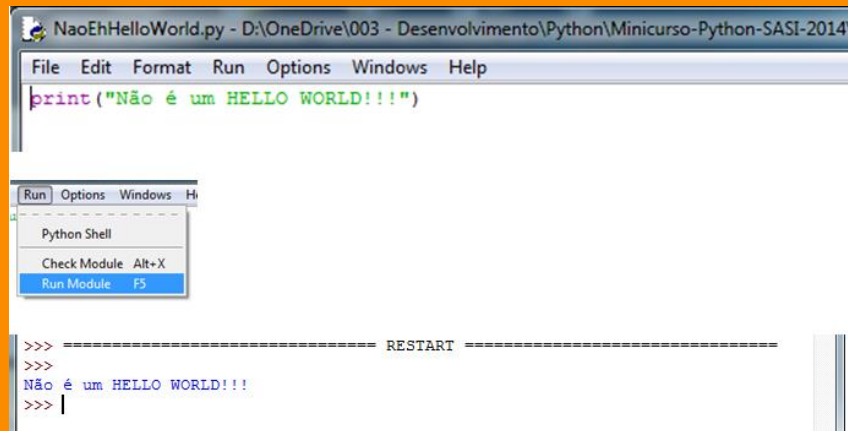
```
>>> 2**3
8
>>> 4**2
16
>>> 10**10
10000000000
```

```
>>>
>>> 2**0.5
1.4142135623730951
>>> 3**(1/2)
1.7320508075688772
>>> 2**(1/2)
1.4142135623730951
>>> 12**(1/4)
1.8612097182041991
>>> 8**(1/3)
2.0
>>> 2**(1/10)
1.0717734625362931
```

Execute os comandos!

Comando de saída de dados

```
>>> print('Olá mundo')  
Olá mundo  
>>> |
```



Não esqueça de executar todos os comandos!

Iniciando...

Esqueça **declarações de tipos** de variáveis
(**por enquanto...**)

Esqueça **begin e end**

Esqueça **{ e }** e demais **delimitadores** de
bloco

Se você já era **organizado** não sofrerá!

A **indentação** é obrigatória!



Dica

Se escrevermos algo que o **interpretador não reconhece**, aparecerá na tela uma mensagem de erro.

É um **mau hábito ignorar essas mensagens**, mesmo que elas pareçam difíceis de entender num primeiro momento.



Blocos

Em Python, os blocos de código são delimitados pelo uso de endentação, que deve ser constante no bloco de código,

É considerada uma boa prática manter a consistência no projeto todo e evitar a mistura de tabulações e espaços

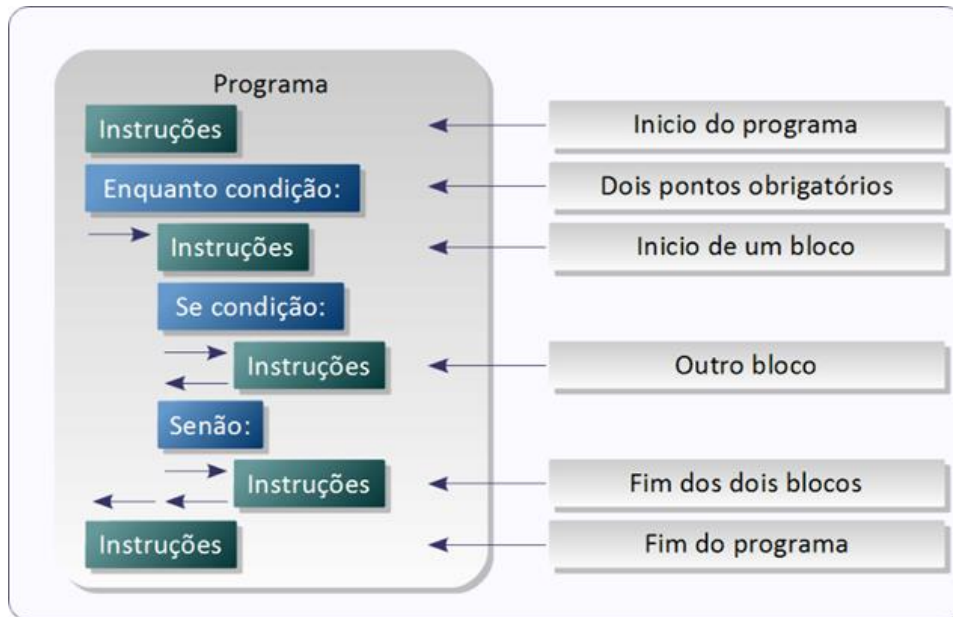
Use **Endentação** (recuo de código) para **melhorar a legibilidade** de seus programas

Ver verbete Endentação na Wikipedia (<http://pt.wikipedia.org/wiki/Endenta%C3%A7%C3%A3o>)

Em Python, a endentação é OBRIGATÓRIA para delimitar blocos.


A recomendação oficial de estilo de codificação é usar quatro espaços para endentação

Blocos



A linha anterior ao bloco sempre termina com dois pontos (:) e representa uma estrutura de controle da linguagem ou uma declaração de uma nova estrutura (uma função, por exemplo).

Exemplo



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> #Para i na lista 234, 654, 378, 798:
>>> for i in [234, 654, 378, 798]:
    #Se o resto dividindo por 3 for igual a zero:
    if i % 3 == 0:
        #imprime...
        print(i, '/3 = ', i/3)

234 /3 = 78.0
654 /3 = 218.0
378 /3 = 126.0
798 /3 = 266.0
>>> |
```



Exemplo

Python é uma linguagem orientada a objetos

As estruturas de dados têm atributos (os dados em si) e métodos (rotinas associadas aos dados)

Os atributos quanto os métodos são acessados usando o operador ponto (.)

Para mostrar o atributo de um objeto:

`print(objeto.atributo)`

Para executar um método:

`objeto.método(argumentos)`

Mesmo um método sem argumentos precisa de parênteses:

`objeto.método()`

O ponto também é usado para acessar estruturas de módulos que foram importados pelo programa

Exemplo

É muito comum em um programa que certos conjuntos de instruções sejam executados de forma condicional, em casos como validar entradas de dados, por exemplo:

Sintaxe:

if <condição>:

<bloco de código>

elif <condição>:

<bloco de código>

elif <condição>:

<bloco de código>

else:

<bloco de código>

Na qual:

- <condição>: sentença que possa ser avaliada como verdadeira ou falsa.
- <bloco de código>: sequência de linhas de comando.
- As cláusulas elif e else são opcionais e podem existir vários elifs para o mesmo if, porém apenas um else ao final.
- Parênteses só são necessários para evitar ambiguidades.

Exemplo

```
testelFTemperatura.py - D:\OneDrive\003 - Desenvolvimento\Python\
File Edit Format Run Options Windows Help
temp = int(input('Entre com a temperatura: '))
if temp < 0:
    print('Congelando...')
elif 0 <= temp <= 20:
    print('Frio!')
elif 21 <= temp <= 25:
    print('Normal')
elif 26 <= temp <= 35:
    print('Quente!')
else:
    print('Muito Quente')
```

```
>>> ===== RESTART =====
>>>
Entre com a temperatura: 27
Quente!
>>> |
```



Dissecando...



C# Java
C/C++ Python
PHP



- No exemplo anterior “Entre com a temperatura: ” é a mensagem indicando que o programa espera pela digitação
- 27 é a entrada digitada e “Quente” é a resposta (saída) do programa
- Se o bloco de código for composto de apenas uma linha, ele pode ser escrito após os dois pontos:

```
if temp < 0: print 'Congelando...'
```

- Python também suporta a expressão:

```
<variável> = <valor 1> if <condição> else <valor 2>
```



Dissecando...



C# Java
C/C++ Python
PHP



- Exercício:
- 1 - Fazer um programa para ler duas notas e calcular a média das duas exibindo na tela: “Aprovado” se a média for maior que 7, “Prova Final” se a média estiver entre 4 e 7 e “Reprovado” se a média for menor que 4
- 2 – Fazer um programa que peça dois números e imprima o maior deles. Exibir mensagem se eles forem iguais.
- 3 – Faça um programa que leia três números e exiba o menor deles
- 4 - Faça um Programa que peça um número inteiro e determine se ele é par ou impar.

Exemplo 1

```
exercicio1MediaAprovadoReprovado.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/P...
File Edit Format Run Options Windows Help
nota1 = float(input('Entre com a primeira nota: '))
nota2 = float(input('Entre com a segunda nota: '))
media = (nota1 + nota2)/2
if media > 7:
    print('Sua media é: ', media, ' e vc foi aprovado!!')
elif 4 < media < 7:
    print('Sua media é: ', media, ' e vc vai fazer FINAL. Estude mais!!')
else:
    print('Você foi reprovado. Sua media, ', media, ' foi insuficiente!!')
```

```
>>> ===== RESTART =====
>>>
Entre com a primeira nota: 3
Entre com a segunda nota: 2
Você foi reprovado. Sua media, 2.5 foi insuficiente!!
>>> ===== RESTART =====
>>>
Entre com a primeira nota: 7
Entre com a segunda nota: 7.5
Sua media é: 7.25 e vc foi aprovado!!
>>> ===== RESTART =====
>>>
Entre com a primeira nota: 7
Entre com a segunda nota: 5.5
Sua media é: 6.25 e vc vai fazer FINAL. Estude mais!
>>>
```

Exemplo 2

```
exercicio2MaiorMenorDoisNumeros.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenv
File Edit Format Run Options Windows Help
num1 = input('Digite um número: ')
num2 = input('Digite outro número: ')
if num1 == num2:
    print('Os números são iguais!!')
elif num1 > num2:
    print('O primeiro é maior que o segundo!!')
else:
    print('O segundo é maior que o primeiro!!')
```

```
>>> ===== RESTART =====
>>>
Digite um número: 10
Digite outro número: 10
Os números são iguais!!
>>> ===== RESTART =====
>>>
Digite um número: 10
Digite outro número: 12
O segundo é maior que o primeiro!!
>>> ===== RESTART =====
>>>
Digite um número: 12
Digite outro número: 10
O primeiro é maior que o segundo!!
>>>
```


Exemplo 3

```
exemplo3Menor3Numeros.py - C:/Users/edkallenn.esl/OneDrive/003
File Edit Format Run Options Windows Help
num1 = input('Informe o primeiro número: ')
num2 = input('Informe o primeiro número: ')
num3 = input('Informe o primeiro número: ')
menor = num1
if num2 < menor:
    menor = num2
if num3 < menor:
    menor = num3
print('O menor é: ', menor)
```

```
>>> ===== RESTART =====
>>>
Informe o primeiro número: 1
Informe o primeiro número: 2
Informe o primeiro número: 3
O menor é: 1
>>> ===== RESTART =====
>>>
Informe o primeiro número: 20
Informe o primeiro número: 10
Informe o primeiro número: 30
O menor é: 10
>>> ===== RESTART =====
>>>
Informe o primeiro número: 30
Informe o primeiro número: 20
Informe o primeiro número: 10
O menor é: 10
>>>
```

Exemplo 4

```
exemplo4ParImpar.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Min
File Edit Format Run Options Windows Help
print('Programa que diz se numero é par ou ímpar!')
num1 = int(input('Digite um número: '))
if num1%2==0:
    print('O número ', num1, ' é par!')
else:
    print('O número ', num1, ' é ímpar!')
```

```
>>> ===== RESTART =====
>>>
Programa que diz se numero é par ou ímpar!
Digite um número: 12
O número 12 é par!
>>> ===== RESTART =====
>>>
Programa que diz se numero é par ou ímpar!
Digite um número: 11
O número 11 é ímpar!
>>>
```



Laços



C# Java
C/C++ Python
PHP



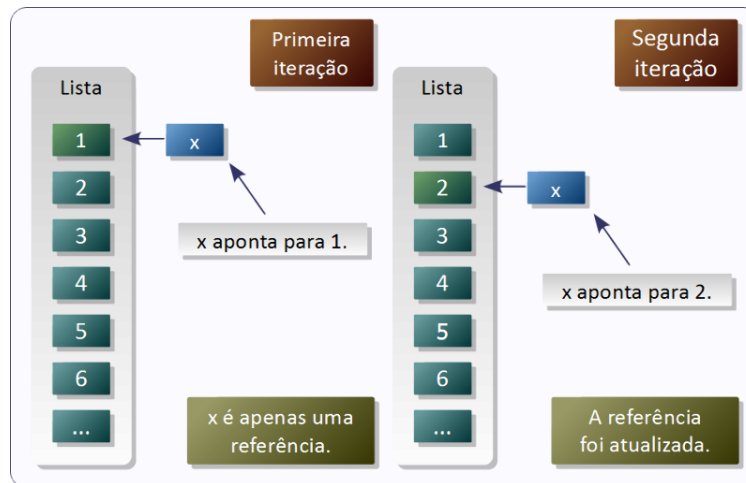
- Laços (*loops*) são **estruturas de repetição**, geralmente usados para processar coleções de dados
- **For**
- É a estrutura de repetição mais usada no Python.
- A instrução aceita não só sequências estáticas, mas também sequências geradas por **iteradores**.
- **Iteradores** são estruturas que permitem iterações, ou seja, acesso aos itens de uma coleção de elementos, de forma sequencial.



Laços



- **Iteradores** são estruturas que permitem iterações, ou seja, acesso aos itens de uma coleção de elementos, de forma sequencial.
- Exemplo de iteração:





Laços



- Durante a execução de um laço *for*, a referência aponta para um elemento da sequência.
- A cada iteração, a referência é atualizada, para que o bloco de código do *for* processe o elemento correspondente.
- A clausula *break* interrompe o laço e *continue* passa para a próxima iteração.
- O código dentro do *else* é executado ao final do laço, a não ser que o laço tenha sido interrompido por *break*.

C# Java
C/C++ Python
PHP





Laços



pythonTM

C# Java
C/C++ Python
PHP



- Sintaxe:

for <referência> in <sequência>:

<bloco de código>

continue

break

else:

<bloco de código>

Não
esquecer
dos dois
pontos!!



Laços



- Exemplo:

```
>>>
>>> #Soma de 0 a 99
>>> soma = 0
>>> for x in range(1,100):
        soma = soma + x

>>> print(soma)
4950
>>>
```



Laços



C# Java
C/C++ Python
PHP



- range?
- Digite no interpretador:

```
>>>
>>> range
<class 'range'>
>>>
>>>
```

- Quando você digita o nome de uma função (ou uma classe, no caso) sem fornecer dados, Python limita-se a dizer a que se refere o nome
- Para ajuda, no interpretador, digite **help()**
- Em seguida digite range



Laços

C# Java
C/C++ Python
PHP



```
help> range
Help on class range in module builtins:
```

```
class range(object)
| range(stop) -> range object
| range(start, stop[, step]) -> range object
|
| Return a virtual sequence of numbers from start to stop by step.
|
| Methods defined here:
|
| __contains__(self, key, /)
|     Return key in self.
|
| __eq__(self, value, /)
|     Return self==value.
|
| __ge__(self, value, /)
|     Return self>=value.
|
| __getattr__(self, name, /)
|     Return getattr(self, name).
|
| __getitem__(self, key, /)
|     Return self[key].
|
| __gt__(self, value, /)
|     Return self>value.
|
| __hash__(self, /)
|     Return hash(self).
|
| __iter__(self, /)
|     Implement iter(self).
|
| __le__(self, value, /)
|     Return self<=value.
|
| __len__(self, /)
|     Return len(self).
```

onTM



Laços



- range? É uma **classe iteradora**!

```
help> range
Help on class range in module builtins:

class range(object)
| range(stop) -> range object
| range(start, stop[, step]) -> range object
|
| Return a virtual sequence of numbers from start to stop by step.
```

- Para testar, faça:

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

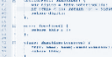
1
2
3
4
5
6
7
8
9
>>> |
```



Laços



C# Java
C/C++ Python
PHP



- Ou...

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

```
1
3
5
7
9
```

- Assim, podemos.

```
>>> for x in range(0,10+1,2):
      print(x)
```

```
0
2
4
6
8
10
```



Laços



- O iterador `range(m, n, p)` é muito útil em laços
- Pois retorna uma lista de inteiros, começando em `m` e menores que `n`, em passos de comprimento `p`, que podem ser usados como sequência para o laço.

C# Java
C/C++ Python
PHP





Laços



C# Java
C/C++ Python
PHP



- While
- Executa um bloco de código atendendo a uma condição.
- Sintaxe:

```
while <condição>:
    <bloco de código>
    continue
    break
else:
    <bloco de código>
```

- O bloco de código dentro do laço *while* é repetido enquanto a condição do laço estiver sendo avaliada como verdadeira.
- As cláusulas **break**, **continue** e **else** são tratadas da mesma forma que no laço **for**.



Laços



- Exemplo:

```
>>>
>>> #soma de 0 a 99
>>> soma = 0
>>> x = 1
>>> while x < 100:
>>>     soma = soma + x
>>>     x = x + 1
>>> print(soma)
4950
>>>
```

- O laço *while* é adequado quando não há como determinar quantas iterações vão ocorrer e não há uma sequência a seguir.



Laços



C# Java
C/C++ Python
PHP



- Exercício
- 1 – Exibir os primeiros 100 números pares na tela
- 2 – Exibir os 20 primeiros múltiplos de um numero x inserido pelo usuário
- 3 – Faça um programa que Calcule o somatório de todos os números de 1 até um número n inserido pelo usuário
- 4 - Faça um Programa que calcule o fatorial de um número



Tipos de dados



- Variáveis no interpretador Python são criadas através da **atribuição** e **destruídas pelo coletor de lixo** (garbage collector), quando não existem mais referências a elas.
- Os nomes das variáveis devem começar com uma letra (sem acentuação) ou sublinhado (`_`) e seguido por letras (sem acentuação), dígitos ou sublinhados (`_`), sendo que maiúsculas e minúsculas são consideradas diferentes.

C# Java
C/C++ Python
PHP





C# Java
C/C++ Python
PHP



Tipos de dados



- Existem vários tipos simples de dados pré-definidos no Python, tais como:
 - Números (inteiros, reais, complexos, ...)
 - Texto
- Além disso, existem tipos que funcionam como coleções. Os principais são:
 - Lista
 - Tupla
 - Dicionário



Tipos de dados



C# Java
C/C++ Python
PHP



- Os tipos no Python podem ser:
 - Mutáveis: permitem que os conteúdos das variáveis sejam alterados.
 - Imutáveis: não permitem que os conteúdos das variáveis sejam alterados.
- Em Python, os nomes de variáveis são **referências**, que podem ser alteradas em **tempo de execução**.
- Os tipos e rotinas mais comuns estão implementados na forma de **builtins**, ou seja, eles **estão sempre disponíveis** em tempo de execução, sem a necessidade de importar nenhuma biblioteca.



Tipos de Dados



- Lembrando que a tipagem de Python é dinâmica!!

```
>>> a = 1
>>> type(a)
<class 'int'>
>>> a = 'Python é lindo, bicho!'
>>> type(a)
<class 'str'>
>>> a = 3.141592
>>> type(a)
<class 'float'>
>>> a = ['Einstein', 'Newton', 'Leibiniz', 'Paul Dirac']
>>> type(a)
<class 'list'>
>>> a = 4 + 5j
>>> type(a)
<class 'complex'>
```



Tipos de Dados



C# Java
C/C++ Python
PHP



- Como tudo é objeto, lembre-se do método “dir”
- Ele é seu AMIGO!

```
>>> a = 4 + 5j
>>> type(a)
<class 'complex'>
>>> dir(a)
['_abs_', '_add_', '_bool_', '_class_', '_delattr_', '_dir_', '_divm
od_', '_doc_', '_eq_', '_float_', '_floordiv_', '_format_', '_ge_',
'_getattr_', '_getnewargs_', '_gt_', '_hash_', '_init_', '_int_',
'_le_', '_lt_', '_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_po
s_', '_pow_', '_radd_', '_rdivmod_', '_reduce_', '_reduce_ex_', '_rep
r_', '_rfloordiv_', '_rmod_', '_rmul_', '_rpow_', '_rsub_', '_rtruedi
v_', '_setattr_', '_sizeof_', '_str_', '_sub_', '_subclasshook_', '_t
ruediv_', '_conjugate_', '_imag_', '_real_']
>>>
>>>
```

Tipos de Dados

- Assim ficamos sabendo, por exemplo, que as listas, em Python, tem um método `sort()`

```
>>> a = [('Einstein', 'Newton', 'Leibiniz', 'Paul Dirac')]
>>> type(a)
<class 'list'>
>>> dir(a)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```



Tipos de Dados



C# Java
C/C++ Python
PHP



- **Números**
- Python oferece alguns tipos numéricos na forma de *builtins*:
 - Inteiro (*int*): $i = 1$
 - Real de ponto flutuante (*float*): $f = 3.14$
 - Complexo (*complex*): $c = 3 + 4j$
- Além dos números inteiros convencionais, existem também os inteiros longos, que tem dimensão arbitrária e são limitados pela memória disponível.
- As conversões entre inteiro e longo são realizadas de forma automática.
- A função *builtin* `int()` pode ser usada para converter outros tipos para inteiro, incluindo mudanças de base.

Tipos de Dados

- Exemplo:

```
TiposNumeros.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python
File Edit Format Run Options Windows Help
# Convertendo de real para inteiro
print ('int(3.14) =', int(3.14))
# Convertendo de inteiro para real
print ('float(5) =', float(5))
# Calculo entre inteiro e real resulta em real
print ('5.0 / 2 + 3 = ', 5.0 / 2 + 3)
# Inteiros em outra base
print ("int('20', 8) =", int('20', 8)) # base 8
print ("int('20', 16) =", int('20', 16)) # base 16
# Operações com números complexos
c = 3 + 4j
print ('c =', c)
print ('Parte real:', c.real)
print ('Parte imaginária:', c.imag)
print ('Conjugado:', c.conjugate())
```



Tipos de Dados



C# Java
C/C++ Python
PHP



- Saída:

```
>>> ===== RESTART =
>>>
int(3.14) = 3
float(5) = 5.0
5.0 / 2 + 3 = 5.5
int('20', 8) = 16
int('20', 16) = 32
c = (3+4j)
Parte real: 3.0
Parte imaginária: 4.0
Conjugado: (3-4j)
>>>
```

- Os números reais também podem ser representados em notação científica, por exemplo: 1.2e22.

Em Python TUDO é um objeto

- 5, por exemplo, é uma instância de int.
- Veja:

```
>>> 5 + 3
8
>>> type(5)
<class 'int'>
>>> dir(5)
['_abs_', '_add_', '_and_', '_bool_', '_ceil_', '_class_', '_delattr_', '_dir_', '_divmod_', '_doc_', '_eq_', '_float_', '_floor_', '_floordiv_', '_format_', '_ge_', '_getattr_', '_getnewargs_', '_gt_', '_hash_', '_index_', '_init_', '_int_', '_invert_', '_le_', '_lshift_', '_lt_', '_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_or_', '_pos_', '_pow_', '_radd_', '_rand_', '_rdivmod_', '_reduce_', '_reduce_ex_', '_repr_', '_rfloordiv_', '_r_lshift_', '_rmod_', '_rmul_', '_ror_', '_round_', '_rpow_', '_rrshift_', '_rshift_', '_rsub_', '_rtruediv_', '_rxor_', '_setattr_', '_sizeof_', '_str_', '_sub_', '_subclasshook_', '_truediv_', '_trunc_', '_xor_', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
>>>
```



Operadores



- O Python tem uma série de operadores definidos para manipular **números**, através de cálculos aritméticos, operações **lógicas** (que testam se uma determina **condição** é verdadeira ou falsa) ou processamento **bit-a-bit** (em que os números são tratados na forma binária).

C# Java
C/C++ Python
PHP





Operadores



C# Java
C/C++ Python
PHP



● Aritméticos

- Soma (+)
- Diferença (-)
- Multiplicação (*)
- Divisão (/): entre dois inteiros funciona igual à divisão inteira. Em outros casos, o resultado é real.
- Divisão inteira (/): o resultado é truncado para o inteiro imediatamente inferior, mesmo quando aplicado em números reais, porém neste caso o resultado será real também.
- Módulo (%): retorna o resto da divisão.
- Potência (**): pode ser usada para calcular a raiz, através de expoentes fracionários (exemplo: `100 ** 0.5`).
- Positivo (+)
- Negativo (-)



Operadores



C# Java
C/C++ Python
PHP



- Exemplos

```
>>> 5/2
2.5
>>> 5//2
2
>>> 5**2
25
>>> 5*2
10
>>> 5+2
7
>>> 5-2
3
>>> 5%2
1
>>> +5
5
>>> -5
-5
>>>
```



Operadores

C# Java
C/C++ Python
PHP



- **Lógicos**
- Menor (<)
- Maior (>)
- Menor ou igual (<=)
- Maior ou igual (>=)
- Igual (==)
- Diferente (!=)



pythonTM



Operadores



C# Java
C/C++ Python
PHP



- Exemplos:

```
>>>
>>> 5>2
True
>>> 5<2
False
>>> 5<=2
False
>>> 5>=2
True
>>> 5==2
False
>>> 5==5
True
>>> 5!=2
True
>>> .
```



Operadores



C# Java
C/C++ Python
PHP



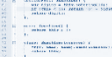
- Operações bit-a-bit:
- Deslocamento para esquerda (<<)
- Deslocamento para direita (>>)
- E bit-a-bit (&)
- Ou bit-a-bit (|)
- Ou exclusivo bit-a-bit (^)
- Inversão (~)



Operadores



C# Java
C/C++ Python
PHP



- Durante as operações, os números serão convertidos de forma adequada (exemplo: $(1.5+4j) + 3$ resulta em $4.5+4j$).
- Caso os tipos sejam incompatíveis, uma exceção é gerada!
- Além dos operadores, também existem algumas funções *builtin* para lidar com tipos numéricos:
 - `abs()`, que retorna o valor absoluto do número,
 - `oct()`, que converte para octal,
 - `hex()`, que converte para hexadecimal,
 - `pow()`, que eleva um número por outro e
 - `round()`, que retorna um número real com o arredondamento especificado.



Operadores



pythonTM

C# Java
C/C++ Python
PHP



- Exemplos:

```
>>> x = 2
>>> y = 3
>>> abs(x)
2
>>> oct(x)
'0o2'
>>> hex(x)
'0x2'
>>> pow(x,y)
8
>>> x=2.5
>>> round(x)
2
>>> x=-2
>>> abs(x)
2
```



Texto (Strings)



C# Java
C/C++ Python
PHP



- A string no Python é um tipo especializado para armazenar texto.
- Como são imutáveis, não é possível adicionar, remover ou mesmo modificar algum caractere de uma *string*. Para realizar essas operações, o Python precisa criar um nova *string*.
- Exemplos:
 - `s = 'Led Zeppelin'`
 - `u = 'Björk'`
- A string-padrão segue o padrão Unicode, suportando acentos e caracteres especiais



Texto (Strings)



C# Java
C/C++ Python
PHP



- A string no Python é um tipo especializado para armazenar texto.
- Como são imutáveis, não é possível adicionar, remover ou mesmo modificar algum caractere de uma *string*. Para realizar essas operações, o Python precisa criar um nova *string*.
- Exemplos:
 - `s = 'Led Zeppelin'`
 - `u = 'Björk'`
- A string-padrão segue o padrão Unicode, suportando acentos e caracteres especiais



Texto (Strings)



- A inicialização de strings pode ser:
 - Com aspas simples ou duplas.
 - Em várias linhas consecutivas, desde que seja entre três aspas simples ou duplas.
 - Sem expansão de caracteres (exemplo: `s = r'\n'`, em que `s` conterá os caracteres `\` e `n`).

C# Java
C/C++ Python
PHP



Texto (Strings)



- Operações com strings:

```
OperacoesStrings.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/1
File Edit Format Run Options Windows Help

s = 'Camelo'

# Concatenação
print ('O ' + s + ' foi embora correndo!')

# Interpolação
print ('tamanho de %s => %d' % (s, len(s)))

# String tratada como sequência
for ch in s: print (ch)

# Strings são objetos
if s.startswith('C'): print (s.upper())

# o que acontecerá?
print (3 * s)
# 3 * s é consistente com s + s + s
|
```

```
>>>
O Camelo foi embora correndo!
tamanho de Camelo => 6
C
a
m
e
l
o
CAMELO
CameloCameloCamelo
>>>
>>>
```



Texto (Strings)



C# Java
C/C++ Python
PHP



- Operador % é usado para fazer interpolação de strings. A interpolação é mais eficiente no uso de memória do que a concatenação convencional.
- Símbolos usados na interpolação:
 - **%s**: *string*.
 - **%d**: inteiro.
 - **%o**: octal.
 - **%x**: hexadecimal.
 - **%f**: real.
 - **%e**: real exponencial.
 - **%%**: sinal de percentagem.
- Os símbolos podem ser usados para apresentar números em diversos formatos.

Texto (Strings)



- Exemplos:

```
InterpolacaoSimbolosString.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python-SASI-2014/Fontes/
File Edit Format Run Options Windows Help

# Zeros a esquerda
print ('Agora são %02d:%02d.' % (16, 30))

# Real (número após o ponto controla as casas decimais)
print ('Percentagem: %.1f%%, Exponencial: %.2e' % (5.333, 0.00314))

# Octal e hexadecimal
print ('Decimal: %d, Octal: %o, Hexadecimal: %x' % (10, 10, 10))

>>>
Agora são 16:30.
Percentagem: 5.3%, Exponencial:3.14e-03
Decimal: 10, Octal: 12, Hexadecimal: a
>>>
>>>
```

Texto (Strings)



- Além do operador “%”, também é possível fazer interpolação usando o método de string e a função chamada **format()**

```
MetodoFormatStrings.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python-SASI-2
File Edit Format Run Options Windows Help
musicos = [('Page', 'guitarrista', 'Led Zeppelin'),
            ('Fripp', 'guitarrista', 'King Crimson')]

# Parâmetros identificados pela ordem
msg = '{0} é {1} do {2}'

for nome, funcao, banda in musicos:
    print(msg.format(nome, funcao, banda))

# Parâmetros identificados pelo nome
msg = '{saudacao}, são {hora:02d}:{minuto:02d}'

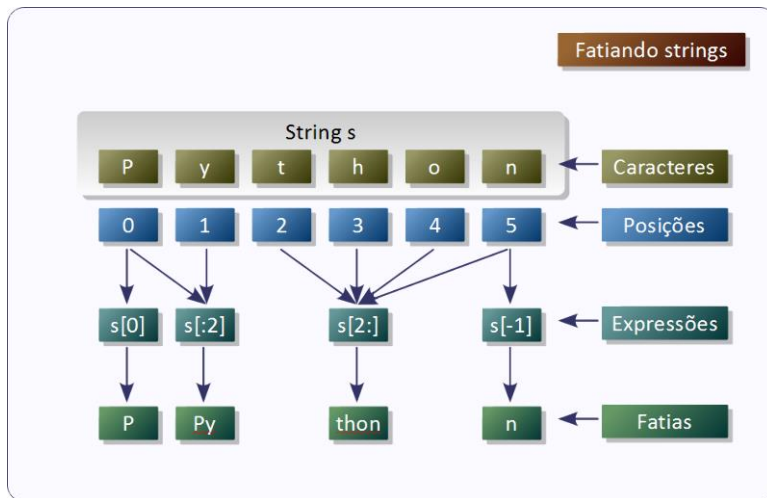
print(msg.format(saudacao='Bom dia', hora=7, minuto=30))

# Função builtin format()
print('Pi =', format(3.14159, '.3e'))
```

```
>>>
Page é guitarrista do Led Zeppelin
Fripp é guitarrista do King Crimson
Bom dia, são 07:30
Pi = 3.142e+00
>>>
```


Texto (Strings)

- A função **format()** pode ser usada para formatar apenas um dado de cada vez.
- Fatias (*slices*) de *strings* podem ser obtidas colocando índices entre colchetes após a *string*.





Texto (Strings)



- Os índices no Python:
 - Começam em zero.
 - Contam a partir do fim se forem negativos.
 - Podem ser definidos como trechos, na forma `[inicio:fim + 1:intervalo]`. Se não for definido o início, será considerado como zero. Se não for definido o `fim + 1`, será considerado o tamanho do objeto. O intervalo (entre os caracteres), se não for definido, será 1.
- É possível inverter *strings* usando um intervalo negativo:

```
>>>
>>> print('Python'[::-1])
nohtyP
>>>
```

Texto (Strings)



- Várias funções para tratar com texto estão implementadas no módulo *string*.

```
*ModuloString.py - C:/Users/edkallenn.es/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python-S
File Edit Format Run Options Windows Help
# importando o módulo string
import string

# O alfabeto
a = string.ascii_letters

# Rodando o alfabeto um caractere para a esquerda
b = a[1:] + a[0]

# A função maketrans() cria uma tabela de tradução
# entre os caracteres das duas strings que ela
# recebeu como parâmetro.
# Os caracteres ausentes nas tabelas serão
# copiados para a saída.
tab = str.maketrans(a, b)

# A mensagem...
msg = '''Esse texto será traduzido..
Vai ficar bem estranho.
'''

# A função translate() usa a tabela de tradução
# criada pela maketrans() para traduzir uma string
print (msg.translate(tab))
```

```
>>>
Fttf ufyup tfsá usbevAjep..
Wbj gjdbs cfn ftusboip.
```

```
>>>
>>>
```

Texto (Strings)



- O módulo também implementa um tipo chamado *Template*, que é um modelo de *string* que pode ser preenchido através de um dicionário.
- Os identificadores são iniciados por cifrão (\$) e podem ser cercados por chaves, para evitar confusões.

```
*StringTemplate.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python-SASI-2014/F
File Edit Format Run Options Windows Help
# importando o módulo string
import string

# Cria uma string template
st = string.Template('$aviso aconteceu em $quando')

# Preenche o modelo com um dicionário
s = st.substitute({'aviso': 'Falta de eletricidade',
                  'quando': '03 de Abril de 2002'})

# Mostra:
# Falta de eletricidade aconteceu em 03 de Abril de 2002
print (s)

>>>
Falta de eletricidade aconteceu em 03 de Abril de 2002
>>>
```



Listas



C# Java
C/C++ Python
PHP



- Listas são coleções heterogêneas de objetos, que podem ser de qualquer tipo, inclusive outras listas.
- As listas **no Python são mutáveis**, podendo ser alteradas a qualquer momento.
- Listas podem ser fatiadas da mesma forma que as *strings*, mas como as listas são mutáveis, é possível fazer atribuições a itens da lista.
- Sintaxe:
 - **`lista = [a, b, ..., z]`**

Listas (operações)



```

ListasOperacoes.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso
File Edit Format Run Options Windows Help
# Uma nova lista: Brit Progs dos anos 70
progs = ['Yes', 'Genesis', 'Pink Floyd', 'ELP']

# Varrendo a lista inteira
for prog in progs:
    print (prog)

# Trocando o último elemento
progs[-1] = 'King Crimson'

# Incluindo
progs.append('Camel')

# Removendo
progs.remove('Pink Floyd')

# Ordena a lista
progs.sort()

# Inverte a lista
progs.reverse()

# Imprime numerado
for i, prog in enumerate(progs):
    print (i + 1, '=>', prog)

# Imprime do segundo item em diante
print (progs[1:])
    
```

```

>>>
Yes
Genesis
Pink Floyd
ELP
1 => Yes
2 => King Crimson
3 => Genesis
4 => Camel
['King Crimson', 'Genesis', 'Camel']
>>>
>>>
    
```



Listas



- O iterador `enumerate()` retorna uma tupla de dois elementos a cada iteração: um número sequencial e um item da sequência correspondente.
- A lista possui o método `pop()` que facilita a implementação de filas e pilhas

C# Java
C/C++ Python
PHP





Listas



C# Java
C/C++ Python
PHP



- Exemplo:

```

ListasEnumerate.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python-SASI-2014/F
File Edit Format Run Options Windows Help
lista = ['A', 'B', 'C']
print ('lista:', lista)

# A lista vazia é avaliada como falsa
while lista:
    # Em filas, o primeiro item é o primeiro a sair
    # pop(0) remove e retorna o primeiro item
    print ('Saiu', lista.pop(0), ', faltam', len(lista))

# Mais itens na lista
lista += ['D', 'E', 'F']
print ('lista:', lista)

while lista:
    # Em pilhas, o primeiro item é o último a sair
    # pop() remove e retorna o último item
    print ('Saiu', lista.pop(), ', faltam', len(lista))
    
```




Listas



- Saída:

```
>>>
lista: ['A', 'B', 'C']
Saiu A , faltam 2
Saiu B , faltam 1
Saiu C , faltam 0
lista: ['D', 'E', 'F']
Saiu F , faltam 2
Saiu E , faltam 1
Saiu D , faltam 0
>>>
```



Listas



- As operações de ordenação (*sort*) e inversão (*reverse*) são realizadas na própria lista, sendo assim, não geram novas listas.





Tuplas



C# Java
C/C++ Python
PHP



- Semelhantes as listas, porém **são imutáveis**: não se pode acrescentar, apagar ou fazer atribuições aos itens.
- Sintaxe:

tupla = (a, b, ..., z)

- Os parênteses são opcionais.
- Particularidade: tupla com apenas um elemento é representada como:
- `t1 = (1,)`



Tuplas



C# Java
C/C++ Python
PHP



- Os elementos de uma tupla podem ser referenciados da mesma forma que os elementos de uma lista:

primeiro_elemento = tupla[0]

- Listas podem ser convertidas em tuplas:

tupla = tuple(lista)

- E tuplas podem ser convertidas em listas:

lista = list(tupla)

- Embora a tupla possa conter elementos mutáveis, esses elementos não podem sofrer atribuição, pois isto modificaria a referência ao objeto.

- Exemplo (usando o modo iterativo)

```
>>>
>>> t = ([1, 2], 4)
>>> t[0].append(3)
>>> t
([1, 2, 3], 4)
>>> t[0] = [1, 2, 3]
Traceback (most recent call last):
  File "<pyshell#329>", line 1, in <module>
    t[0] = [1, 2, 3]
TypeError: 'tuple' object does not support item assignment
>>>
```



Tuplas



- Listas e tuplas podem ter seus elementos desempacotados facilmente por meio de atribuição:

```
>>> l = [1,2,3]
>>> a,b,c = l
>>> print(a, '+', b, '+', c, '=', a+b+c)
1 + 2 + 3 = 6
>>>
```



Outras Sequências



C# Java
C/C++ Python
PHP



- O Python provê entre os *builtins* também:
 - *set*: sequência mutável unívoca (sem repetições) não ordenada.
 - *frozenset*: sequência imutável unívoca não ordenada.
- Os dois tipos implementam operações de conjuntos, tais como: união, interseção e diferença.



Outras Sequências



- O Python provê entre os *builtins* também:
 - *set*: sequência mutável unívoca (sem repetições) não ordenada.
 - *frozenset*: sequência imutável unívoca não ordenada.
- Os dois tipos implementam operações de conjuntos, tais como: união, interseção e diferença.

C# Java
C/C++ Python
PHP



Outras Sequências



pythonTM



```

OutrasSequencias.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Pytho
File Edit Format Run Options Windows Help
# Conjuntos de dados
s1 = set(range(3))
s2 = set(range(10, 7, -1))
s3 = set(range(2, 10, 2))

# Exibe os dados
print ('s1:', s1, '\ns2:', s2, '\ns3:', s3)

# União
s1s2 = s1.union(s2)
print ('União de s1 e s2:', s1s2)

# Diferença
print ('Diferença com s3:', s1s2.difference(s3))

# Interseção
print ('Interseção com s3:', s1s2.intersection(s3))

# Testa se um set inclui outro
if s1.issuperset([1, 2]):
    print ('s1 inclui 1 e 2')

# Testa se não existe elementos em comum
if s1.isdisjoint(s2):
    print ('s1 e s2 não tem elementos em comum')
    
```



Outras Sequências



C# Java
C/C++ Python
PHP



- Saída:

```
>>>
s1: {0, 1, 2}
s2: {8, 9, 10}
s3: {8, 2, 4, 6}
União de s1 e s2: {0, 1, 2, 8, 9, 10}
Diferença com s3: {0, 1, 9, 10}
Interseção com s3: {8, 2}
s1 inclui 1 e 2
s1 e s2 não tem elementos em comum
>>>
```

- Quando uma lista é convertida para set, as repetições são descartadas.



Dicionários



C# Java
C/C++ Python
PHP



- Um dicionário é uma lista de associações compostas por uma chave única e estruturas correspondentes.
- Dicionários são mutáveis, tais como as listas.
- A chave precisa ser de um tipo imutável, geralmente são usadas *strings*, mas também podem ser tuplas ou tipos numéricos.
- Já os itens dos dicionários podem ser tanto mutáveis quanto imutáveis.
- O dicionário do Python não fornece garantia de que as chaves estarão ordenadas.



Dicionários



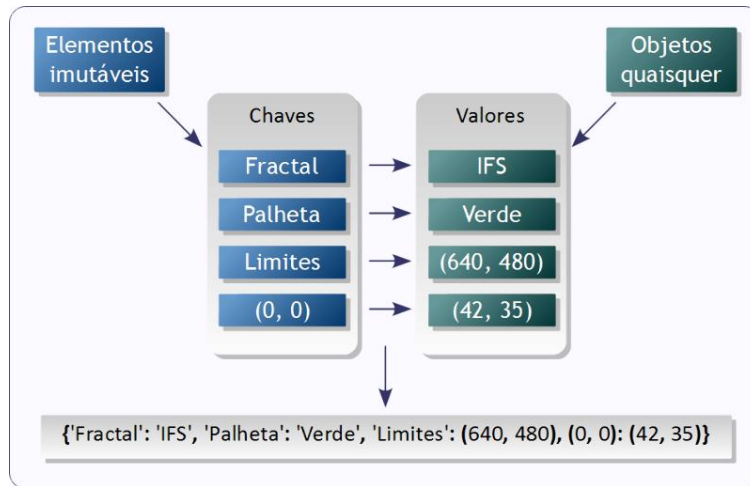
C# Java
C/C++ Python
PHP



- Sintaxe:

`dicionario = {'a': a, 'b': b, ..., 'z': z}`

- Estrutura:

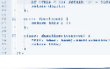




Dicionários



C# Java
C/C++ Python
PHP



- Exemplo de dicionário:

```
dic = {'nome': 'Shirley Manson', 'banda': 'Garbage'}
```

- Acessando elementos:

```
print (dic['nome'] )
```

- Adicionando elementos:

```
dic['album'] = 'Version 2.0'
```

- Apagando um elemento do dicionário:

```
del dic['album']
```

- Obtendo os itens, chaves e valores:

```
print('itens', dic.items())
print('chaves', dic.keys())
print('valores', dic.values())
```



Dicionários



C# Java
C/C++ Python
PHP



```
>>>
>>> dic = {'nome': 'Shirley Manson', 'banda': 'Garbage'}

>>> print (dic['nome'] )

Shirley Manson
>>> dic['album'] = 'Version 2.0'

>>> del dic['album']

>>> print('itens', dic.items())
itens dict_items([('banda', 'Garbage'), ('nome', 'Shirley Manson')])
>>> print('chaves', dic.keys())
chaves dict_keys(['banda', 'nome'])
>>> print('valores', dic.values())
valores dict_values(['Garbage', 'Shirley Manson'])
```

- Exemplos com dicionários

```
ExemploDicionarios.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso
File Edit Format Run Options Windows Help

# Progs e seus albums
progs = {'Yes': ['Close To The Edge', 'Fragile'],
        'Genesis': ['Foxtrot', 'The Nursery Crime'],
        'ELP': ['Brain Salad Surgery']}

# Mais progs
progs['King Crimson'] = ['Red', 'Discipline']

# items() retorna uma lista de
# tuplas com a chave e o valor
for prog, albums in progs.items():
    print(prog, '=>', albums)
print(len(progs), 'bandas')

# Se tiver 'ELP', deleta
if 'ELP' in progs:
    del progs['ELP']

print('Agora', len(progs), 'bandas')
```

```
>>>
Genesis => ['Foxtrot', 'The Nursery Crime']
Yes => ['Close To The Edge', 'Fragile']
King Crimson => ['Red', 'Discipline']
ELP => ['Brain Salad Surgery']
4 bandas
Agora 3 bandas
>>>
```

Matriz Esparsa

```
MatrizEsparsa.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Py
File Edit Format Run Options Windows Help

# Matriz esparsa implementada
# com dicionário

# Matriz esparsa é uma estrutura
# que só armazena os valores que
# existem na matriz

dim = 6, 12
mat = {}

# Tuplas são imutáveis
# Cada tupla representa
# uma posição na matriz
mat[3, 7] = 3
mat[4, 6] = 5
mat[6, 3] = 7
mat[5, 4] = 6
mat[2, 9] = 4
mat[1, 0] = 9

for lin in range(dim[0]):
    for col in range(dim[1]):
        # Método get(chave, valor)
        # retorna o valor da chave
        # no dicionário ou se a chave
        # não existir, retorna o
        # segundo argumento
        print (mat.get((lin, col), 0), end=' ')
    print()
```


Matriz Esparsa (saída)

```
>>>
0 0 0 0 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 4 0 0
0 0 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 0 5 0 0 0 0 0
0 0 0 0 6 0 0 0 0 0 0 0
>>>
>>>
```

Gerando a matriz esparsa

- dhfgl

```
File Edit Format Run Options Windows Help
# Matriz em forma de string
matriz = '''0 0 0 0 0 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 4 0 0
0 0 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 0 5 0 0 0 0 0
0 0 0 0 6 0 0 0 0 0 0 0'''

mat = {}

# Quebra a matriz em linhas
for lin, linha in enumerate(matriz.splitlines()):

    # Quebra a linha em colunas
    for col, coluna in enumerate(linha.split()):

        coluna = int(coluna)
        # Coloca a coluna no resultado,
        # se for diferente de zero
        if coluna:
            mat[lin, col] = coluna

print(mat)
# Some um nas dimensões pois a contagem começa em zero
print('Tamanho da matriz completa:', (lin + 1) * (col + 1))
print('Tamanho da matriz esparsa:', len(mat))
```



Gerando a matriz esparsa

C# Java
C/C++ Python
PHP



- Saída:

```
>>>
{(5, 4): 6, (3, 7): 3, (1, 0): 9, (4, 6): 5, (2, 9): 4}
Tamanho da matriz completa: 72
Tamanho da matriz esparsa: 5
>>>
>>>
```



Verdadeiro, falso, nulo

C# Java
C/C++ Python
PHP



- Em Python, o tipo booleano (*bool*) é uma especialização do tipo inteiro (*int*). O verdadeiro é chamado *True* e é igual a 1, enquanto o falso é chamado *False* e é igual a zero.
- Os seguintes valores são considerados falsos:
 - *False* (falso).
 - *None* (nulo).
 - 0 (zero).
 - "" (string vazia).
 - [] (lista vazia).
 - () (tupla vazia).
 - {} (dicionário vazio).
 - Outras estruturas com o tamanho igual a zero.
- São considerados verdadeiros todos os outros objetos fora dessa lista.
- O objeto *None*, que é do tipo *NoneType*, do Python representa o nulo e é avaliado como falso pelo interpretador.



Operadores booleanos

C# Java
C/C++ Python
PHP



- Com operadores lógicos é possível construir condições mais complexas para controlar desvios condicionais e laços.
- Os operadores booleanos no Python são: *and*, *or*, *not*, *is* e *in*.
 - *and*: retorna um valor verdadeiro se e somente se receber duas expressões que forem verdadeiras.
 - *or*: retorna um valor falso se e somente se receber duas expressões que forem falsas.
 - *not*: retorna falso se receber uma expressão verdadeira e vice-versa.
 - *is*: retorna verdadeiro se receber duas referências ao mesmo objeto e falso em caso contrário.
 - *in*: retorna verdadeiro se receber um item e uma lista e o item ocorrer uma ou mais vezes na lista e falso em caso contrário.



Operadores booleanos

C# Java
C/C++ Python
PHP



- O cálculo do valor resultante na operação *and* ocorre da seguinte forma: se a primeira expressão for verdadeira, o resultado será a segunda expressão, senão será a primeira.
- Já para o operador *or*, se a primeira expressão for falsa, o resultado será a segunda expressão, senão será a primeira.
- Para os outros operadores, o retorno será do tipo *bool* (*True* ou *False*).
- Além dos operadores booleanos, existem as funções *all()*, que retorna verdadeiro quando todos os itens forem verdadeiros na sequência usada como parâmetro, e *any()*, que retorna verdadeiro se algum item o for.



Funções

C# Java
C/C++ Python
PHP



- Funções são blocos de código identificados por um nome, que podem receber parâmetros pré-determinados.
- No Python, as funções:
 - Podem retornar ou não objetos.
 - Aceitam *Doc Strings*.
 - Aceitam parâmetros opcionais (com *defaults*). Se não for passado o parâmetro será igual ao *default* definido na função.
 - Aceitam que os parâmetros sejam passados com nome. Neste caso, a ordem em que os parâmetros foram passados não importa.
 - Tem *namespace* próprio (escopo local), e por isso podem ofuscar definições de escopo global.
 - Podem ter suas propriedades alteradas (geralmente por decoradores).



Funções

C# Java
C/C++ Python
PHP



- *Doc Strings* são *strings* que estão associadas a uma estrutura do Python. Nas funções, as *Doc Strings* são colocadas dentro do corpo da função, geralmente no começo.
- O objetivo das *Doc Strings* é servir de documentação para aquela estrutura.



Funções

C# Java
C/C++ Python
PHP



- Sintaxe:

```
def func(parametro1, parametro2=padrao):
```

```
    """
```

Doc String

```
    """
```

<bloco de código>

```
    return valor
```

Funções

- Os parâmetros com *default* devem ficar após os que não tem *default*.
- Exemplo (fatorial com recursão):

```
FatorialRecursivo.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Mini
File Edit Format Run Options Windows Help
# Fatorial implementado de forma recursiva

def fatorial(num):
    if num <= 1:
        return 1
    else:
        return (num * fatorial(num - 1))

# Testando fatorial()
print (fatorial(5))
```

Funções

- Exemplo (fatorial sem recursão):

```
FatorialNaoRecursivo.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenv
File Edit Format Run Options Windows Help
def fatorial(n):
    n = n if n > 1 else 1
    j = 1
    for i in range(1, n + 1):
        j = j * i
    return j

# Testando...
for i in range(1, 6):
    print(i, '->', fatorial(i))
```

Funções

- Exemplo – Fibonacci Recursivo

```
FibonacciRecursivo.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Mir
File Edit Format Run Options Windows Help
def fib(n):
    """Fibonacci:
    fib(n) = fib(n - 1) + fib(n - 2) se n > 1
    fib(n) = 1 se n <= 1
    """
    if n > 1:
        return fib(n - 1) + fib(n - 2)
    else:
        return 1

# Mostrar Fibonacci de 1 a 5
for i in [1, 2, 3, 4, 5]:
    print(i, '=>', fib(i))
```

Funções

- Exemplo – Fibonacci Não Recursivo

```
FibonacciNaoRecursivo.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Min
File Edit Format Run Options Windows Help
def fib(n):
    """Fibonacci:
    fib(n) = fib(n - 1) + fib(n - 2) se n > 1
    fib(n) = 1 se n <= 1
    """

    # Dois primeiros valores
    l = [1, 1]

    # Calculando os outros
    for i in range(2, n + 1):
        l.append(l[i - 1] + l[i - 2])

    return l[n]

# Mostrar Fibonacci de 1 a 5
for i in [1, 2, 3, 4, 5]:
    print(i, '=>', fib(i))
|
```

Funções

- Exemplo – Conversão RGB



```

ConversaoRGB.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Python/Minicurso-Python-SA
File Edit Format Run Options Windows Help

def rgb_html(r=0, g=0, b=0):
    """Converte R, G, B em #RRGGBB"""

    return '#%02x%02x%02x' % (r, g, b)

def html_rgb(color='#000000'):
    """Converte #RRGGBB em R, G, B"""

    if color.startswith('#'): color = color[1:]

    r = int(color[:2], 16)
    g = int(color[2:4], 16)
    b = int(color[4:], 16)

    return r, g, b # Uma sequência

print (rgb_html(200, 200, 255))
print (rgb_html(b=200, g=200, r=255)) # O que houve?
print (html_rgb('#c8c8ff'))
    
```



Funções

C# Java
C/C++ Python
PHP



- Observações:

- Os argumentos com padrão devem vir por último, depois dos argumentos sem padrão.
- O valor do padrão para um parâmetro é calculado quando a função é definida.
- Os argumentos passados sem identificador são recebidos pela função na forma de uma lista.
- Os argumentos passados com identificador são recebidos pela função na forma de um dicionário.
- Os parâmetros passados com identificador na chamada da função devem vir no fim da lista de parâmetros.

Funções

- Exemplo de como receber todos parâmetros

```
ReceberTodosOsParametros.py - C:/Users/edkallenn.esl/OneDrive/003 - Desenvolvimento/Pyth
File Edit Format Run Options Windows Help
# *args - argumentos sem nome (lista)
# **kwargs - argumentos com nome (dicionário)

def func(*args, **kwargs):
    print (args)
    print (kwargs)

func('peso', 10, unidade='k')
```




Funções

C# Java
C/C++ Python
PHP



- No exemplo, kargs receberá os argumentos nomeados e args receberá os outros.
- O interpretador tem definidas algumas funções *builtin*, incluindo `sorted()`, que ordena sequências, e `cmp()`, que faz comparações entre dois argumentos e retorna -1 se o primeiro elemento for maior , 0 (zero) se forem iguais ou 1 se o último for maior. Essa função é usada pela rotina de ordenação, um comportamento que pode ser modificado.



Funções – tipagem dinâmica

C# Java
C/C++ Python
PHP



- Variáveis (e parâmetros) não tem tipos declarados e podem ser associados a objetos de qualquer tipo
- Também conhecida como “duck typing” (tipagem pato) nas comunidades Python, Ruby e Smalltalk

```
>>> def dobro(n):
        '''devolve duas vezes n'''
        return n + n

>>> dobro(7)
14
>>> dobro('Spam')
'SpamSpam'
>>> dobro([10, 20, 30])
[10, 20, 30, 10, 20, 30]
```

Tipagem Forte

- Python não faz conversão automática de tipos

- Exceções, por praticidade

- int → long → float

- str → unicode

```
>>> '9' + 10
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    '9' + 10
TypeError: Can't convert 'int' object to str implicitly
>>> "9" - 10
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    "9" - 10
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> "9" + (-10)
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    "9" + (-10)
TypeError: Can't convert 'int' object to str implicitly
>>>
```



C# Java
C/C++ Python
PHP



Orientações básicas para escrever programas

-*- coding: utf-8 -*-

- Coloque na primeira linha para indicar a codificação do arquivo
- Comentários
 - Comentários começam com o caracter # (fora de um literal string) e vão até o final da linha
- Espaços em branco e tabulações são ignorados, exceto no começo da linha (quando marcam o “recoo sintático”)
 - Lembre-se de manter o recoo sintático consistente
 - Decida logo entre utilizar tabulações ou espaços !!!
 - Ou seja feliz e use TAB (rsrsrsrs)

Perguntas e Discussão

