Python

Alex Calegari Fracaroli
Igor Silva Epitácio Pereira
Rodrigo Caldeira de Melo

Sumário

1. Introdução e História

- Surgiu em 1989;
- Criada por Guido van Rossum;
- Monty Python and the Flying Circus;
- Licença compatível com Software Livre;
- Linguagem de altíssimo nível (VHLL);
- Tipagem Dinâmica;
- Multiparadigma (OO, funcional e procedural);
- Compilada + Interpretada;
- Aumentar a produtividade do programador;

2. Versões e Instalação

- A implementação oficial do Python é mantida pela PSF (Python Software Foundation) e escrita em C, e por isso, é também conhecida como CPython. A versão estável mais recente está disponível para download no endereço:

  http://www.python.org/download/

- Para a plataforma Windows, basta executar o instalador. Para outras plataformas, como em sistemas Linux, geralmente o Python já faz parte do sistema, porém em alguns casos pode ser necessário compilar e instalar o interpretador a partir dos arquivos fonte.

  A versão utilizada no trabalho foi a 2.7.5+

3. Por que utilizar Python?

- A linguagem Python é uma linguagem de alto nível, interpretada, orientada a objetos com uma semântica dinâmica. Suas estruturas de alto nível, combinadas com sua tipagem de amarração dinâmica a faz muito atrativa para desenvolvimento de largos aplicativos assim como para uso como linguagem de script ou de colagem.

- A sintaxe simples do Python encoraja a reutilização de código simplificando a manutenção e a normalização de dados em módulos e pacotes distintos.

Por que utilizar Python?

Linguagem Simples, Legível, Clara e Elegante!

Um Exemplo: Escrever um simples *"nome, seja bem vindo(a)! "*

- Em **C**:

```
#include <stdio.h>

int main(){
        char nome[200];
        printf("Digite seu nome: ");
        scanf("%s", nome);
        printf("\n %s, Seja bem vindo(a)\n", nome);
        return 0;
}
```

Por que utilizar Python?

- Em **JAVA**:

```
public class Hello{
        public static void main(String args[]) {
                java.util.Scanner s = new java.util.Scanner(System.in);
                System.out.print("Digite seu nome:");
                String nome = s.nextLine();
                System.out.println("\n" + nome + ", Seja bem vindo(a) :)\n");
        }
}
```

Por que utilizar Python?

- Em **Python**:

```
nome = raw_input('Digite seu nome: ')
print ("\n%s, Seja bem vindo(a) :)\n" % nome);
```

Python oferece maior facilidade para entender.

A linguagem vai direto ao ponto. Simples de usar. Foco no problema, sem perder tempo na sintaxe.

4. Como Compilar e Interpretar

- O código fonte é traduzido pelo Python para bytecode, que é um formato binário com instruções para o interpretador. O bytecode é multiplataforma e pode ser distribuído e executado sem fonte original.

- Por padrão, o interpretador compila o código e armazena o bytecode em disco, para que a próxima vez que o executar, não precise compilar novamente o programa, reduzindo o tempo de carga na execução.

- Quando um programa ou um módulo é evocado, o interpretador realiza a análise do código, converte para símbolos, compila (se não houver bytecode atualizado em disco) e executa na máquina virtual Python.

Como Compilar e Interpretar

- O bytecode é armazenado em arquivos com extensão “.pyc” (bytecode normal) ou “.pyo” (bytecode otimizado). O bytecode também pode ser empacotado junto com o interpretador em um executável, para facilitar a distribuição da aplicação, eliminando a necessidade de instalar Python em cada computador.

Como Compilar e Interpretar

Modo Interativo

- O interpretador Python pode ser usado de forma interativa, na qual as linhas de código são digitadas em um terminal. Para evocar o modo interativo basta executar o interpretador:

```
a2009101791@labgrad02 ~ $ python
Python 2.7.5+ (default, Feb 27 2014, 19:37:08)
[GCC 4.8.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- O modo interativo é uma característica diferencial da linguagem, pois é possível testar e modificar trechos de código antes da inclusão do código em programas, fazer extração e conversão de dados ou mesmo analisar o estado dos objetos que estão em memória, entre outras possibilidades.

5. Sintaxe

- **Indentação**:
  - Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, freqüentemente usando palavras e não pontuações como em outras linguagens.
  - Para a separação de blocos de código, a linguagem usa espaços em branco e indentação ao invés de delimitadores visuais como chaves (C, Java) ou palavras (BASIC, Fortran,Pascal).
  - Diferente de linguagens com delimitadores visuais de blocos, em Python a indentação é obrigatória. O aumento da indentação indica o início de um novo bloco, que termina da diminuição da indentação.

Sintaxe

- **Indentação**
  - ➢ O código abaixo está correto para os dois exemplos, mas o analisador léxico verificará se a indentação está coerente.

Indentação incorreta

```
def valor1():
while True:
try:
c = int(raw_input('Primeiro Valor: '))
return c
except ValueError:
print 'Inválido!'
```

Indentação correta

```
def valor1():
    while True:
        try:
            c = int(raw_input('Primeiro Valor: '))
            return c
        except ValueError:
            print 'Inválido!'
```

Sintaxe

- **Comentários**
  - O caractere # marca o inicio de comentário. Qualquer texto depois do # será ignorado até o fim da linha , com exceção dos comentários funcionais.
  - Para comentário em bloco, usa-se três aspas simples ao início fim do bloco.

Sintaxe

- **Comentário funcional**
  - ➢ É possível usar codificação diferente de ASCII() em arquivos de código Python. A melhor maneira de fazê-lo é através de um comentário adicional logo após a linha #!:

    # -*- coding: encoding -*-

  - ➢ Definir o interpretador que será utilizado para rodar o programa em sistemas UNIX, através de um comentário começando com "#!" no inicio do arquivo, que indica o caminho para o interpretador (geralmente a linha de comentário será algo como "***#!/usr/bin/env python"***).

Sintaxe

- Palavras reservadas da linguagem

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Sintaxe

- **Operadores**

Operador	Nome	Explicação	Exemplos
+	Adição	Executa a soma de dois valores	3 + 5 dá 8. 'a' + 'b' dá 'ab'
-	Subtração	Torna um número negativo ou executa subtração de um número menos outro	-5.2 torna um número negativo. 50 - 24 dá 26
*	Multiplicação	Executa a multiplicação de dois números ou retorna a string repetida tantas vezes	2 * 3 dá 6. 'La' * 3 dá 'LaLaLa'.
**	Potência	Retorna x elevado a y	3 ** 4 dá 81 (3 * 3 * 3 * 3)
/	Divisão	Divide x por y	4/3 dá 1 (divisão de inteiros dá um inteiro) 4.0/3 ou 4/3.0 dá 1.3333333333333333
%	Módulo	Retorna o resto da divisão	8/3 dá 2. -25.5%2.25 dá 1.5
<<	Deslocamento de bits a esquerda	Desloca tantos bits a esquerda	2 << 2 dá 8. - 2 é representado em bits por 10 Deslocar dois bits a esquerda dá 1000 o qual é representado pelo decimal 8
>>	Deslocamento de bits a direita	Desloca tantos bits a direita	11 >> 1 dá 5 - 11 é representado em bits por 1011. Deslocar um bit a direita dá 101 o qual é representado pelo decimal 5

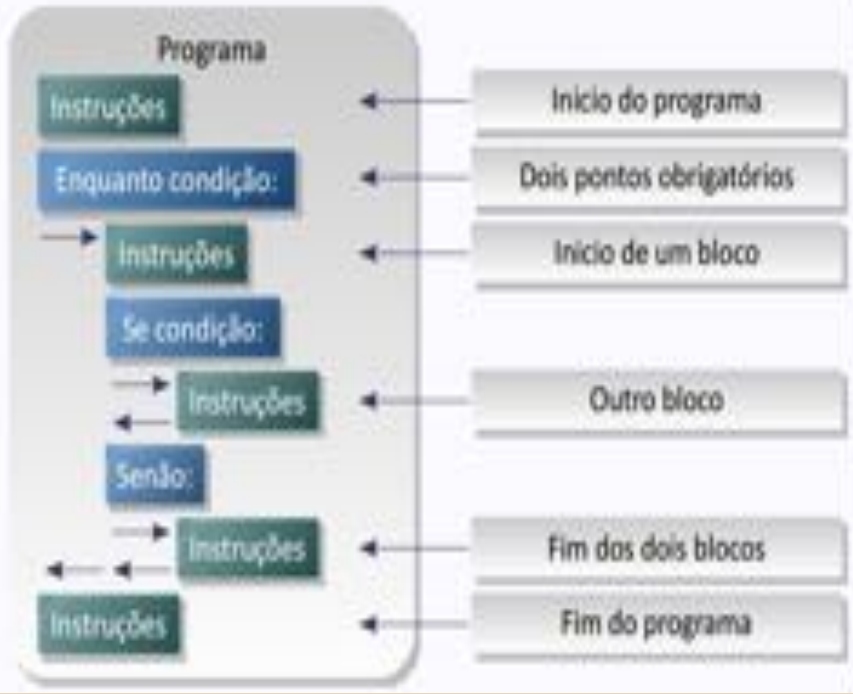
Sintaxe

- **Operadores**

&	Operador bit a bit AND	Bits configurados nos dois operadores são configurados no resultado	5 & 3 dá 1
|	Operador bit a bit OR	Bits configurados em um ou outro operador são configurados no resultado	5 | 3 dá 7
^	Operador bit a bit XOR	Bits configurados em um ou outro operador, mas não em ambos, são configurados no resultado	5 ^ 3 dá 6
~	Operador bit a bit NOT	Bits configurado no operador não são configurados no resultado e vice-versa	~5 dá -6
<	Menor que	Retorna se x é menor que y. Toda comparação retorna 1 para verdadeiro e 0 para falso. Isto é equivalente as variáveis especiais `True` e `False` respectivamente.	5 < 3 dá 0 (False) e 3 < 5 dá 1 (True). Comparações podem ser encadeadas: 3 < 5 < 7 dá True
>	Maior que	Retorna se x é maior que y	5 > 3 retorna True. Se os dois operandos são números, eles são primeiro convertidos para um tipo comum
<=	Menor ou igual a	Retorna se x é menor ou igual a y	x = 3; y = 6; x <= y retorna True
>=	Maior ou igual a	Retorna se x é maior ou igual a y	x = 4; y = 3; x >= 3 retorna True
==	Igual a	Compara se os objetos são iguais	x = 2; y = 2; x == y retorna True. x = 'str'; y = 'str'; x == y retorna False. x = 'str'; y = 'str'; x == y retorna True
!=	Diferente de	Compara se os objetos são diferentes	x = 2; y = 3; x != y retorna True

Sintaxe

- **Blocos**
  - O controle de bloco é feito por indentação.
  - É considerada uma boa prática para manter a consistência no projeto e evitar a mistura de tabulações e espaços;
  - A linha anterior ao bloco sempre termina com dois pontos.



Sintaxe

- Strings devem estar dentro de aspas.
- Variáveis controlam dados na memória e possuem tipos.
- Diretivas são comandos da linguagem.
- Indentação separa blocos de comandos.

Sintaxe

- = atribuição (a=10).
- == comparação (i % 3==0).
- : abrem blocos de instruções.

Exemplo:

```
# 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
```

Saída:

```
234 / 3 = 78
654 / 3 = 218
378 / 3 = 126
798 / 3 = 266
```

Sintaxe

- **Objeto**
  - Python suporta a maioria das técnicas da programação orientada a objeto.
  - Um programa OO em Python pode ser descrito como um conjunto de classes (pré-definidas e definidas) que possuem atributos e métodos, e que são instanciadas em objetos, durante a execução do programa.
  - Sendo assim as estruturas de dados possuem atributos (os dados em si) e métodos (rotinas associadas aos dados). Tanto os atributos quanto os métodos são acessados usando ponto (.)

Sintaxe

- **Objeto**
  - Para mostrar um atributo:
    - ***print objeto.atributo***
  - Para executar um método:
    - ***objeto.metodo(argumentos)***
  - O ponto também é usado para acessar estruturas de módulos que foram importados pelo programa.

6. Controle de Fluxo

- If

```
>>> x = int(raw_input("Favor digitar um inteiro: "))
Favor digitar um inteiro: 42
>>> if x < 0:
...     x = 0
...     print 'Negativo alterado para zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Unidade'
... else:
...     print 'Mais'
```

7. Estruturas de Repetição

- **For**
  - ➢ O for de Python itera sobre os itens de uma sequência (uma lista ou uma string), na ordem em que parecem na sequência. Por exemplo:

```
for fornecedor in fornecedores:
    for compra in compras:
```

- **While**

```
contador = 1
while contador < tam:
```

Estrutura de Repetição

- ***“goto”***

Por se tratar de uma linguagem altamente estruturada, Python não possui o comando *goto*. Ao invés disso, usa os comandos *for* e *while* já citados e outros de controle de loop como *break* e *continue*.

8. Tipos de Dados

Tipo de dado	Descrição	Exemplo da sintaxe
str, unicode	Uma cadeia de caracteres imutável	'Wikipedia', u'Wikipedia'
list	Lista heterogênea mutável	[4.0, 'string', True]
tuple	Tupla imutável	(4.0, 'string', True)
set,frozenset	Conjunto não ordenado, não contém elementos duplicados	set([4.0, 'string', True]) frozenset([4.0, 'string', True])
dict	conjunto associativo	{'key1': 1.0, 'key2': False}
int	Número de precisão fixa, é transparentemente convertido para long caso não caiba em um int.	42 2147483648L
float	Ponto flutuante	3.1415927
complex	Número complexo	3+2j
bool	Booleano	True ou False

Tipos de Dados

- Variáveis no interpretador Python são criadas através da atribuição e destruídas pelo coletor de lixo (garbage colector), quando não existem mais referências a elas.
- Existem vários tipos simples de dados pré-definidos no Python, tais como:
  - Números (inteiros, reais, complexos, ...)
  - Texto
- Existem tipos que funcionam como coleções. Os principais são:
  - Lista
  - Tupla
  - Dicionário

Tipos de Dados

- Em Python, alguns tipos de dados são imutáveis, como as strings e as tuplas. Esta abordagem, apesar de incomum a primeira vista, tem algumas vantagens:
  - Performance: idéia de uma string imutável traz consigo um tamanho imutável conhecido no momento da criação e durante o tempo de vida do objeto.
  - Memória: as strings imutáveis podem ser reutilizadas, duas strings iguais podem apontar para o mesmo lugar da memória.

Tipos de Dados

- Portanto, 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 tempos de execução.
- Além disso, Python é uma linguagem Case Sensitive.
- Não é possível declarar constantes de tipos mutáveis.

Tipos de Dados

- É possível usar a função *type()* para identificar na tela qual o tipo da variável.

```
>>> print type(var_x)
<type 'int'>

>>> print type(var_y)
<type 'float'>

>>> print type(var_z)
<type 'str'>
```

Tipos de Dados

- **Tipo numérico**
  - Suportam adição, subtração, multiplicação e divisão e também podem se relacionar. Mesmo os tipos não sendo declarados explicitamente, eles sempre irão assumir um tipo de dado.

  - Existem 4 tipos numéricos:
    - inteiro (int)
    - ponto flutuante (float)
    - booleano (bool)
    - complexo (complex)

Tipos de Dados

- Exemplo de operações matemáticas em Python:

```
>>> 3+3
6
>>>(50-5*6)/4
5
>>>10/3
3
>>>10/3.0
3.3333
```

Observação
A divisão por inteiro (/) sempre retorna um inteiro. A divisão inteira (//) retorna o inteiro imediatamente inferior.

Tipos de Dados

- **Texto:**
  - Em Python as strings podem ser declaradas utilizando aspas simples, duplas ou uma sequência de três aspas (simples ou duplas);
  - Como são imutáveis, não é possível adicionar, remover ou mesmo modificar algum caractere de uma string. Para realizar essas operações, é preciso criar uma nova string.

```
>>> var_str = 'abc' + 'def'
>>> var_str
'abcdef'
>>> var_str[2]
'c'
>>> var_str[0:2]
'ab'
>>> var_str[2:]
'cdef'
>>> var_str[-2:]
'ef'
```

Tipos de Dados

- **Operações com String**

```
1  # -*- coding: latin1 -*-
2
3  s = 'Linguagem'
4  # Concatenação
5  print 'Trabalho de ' + s + ' de programaçao!'
6  # Interpolação
7  print 'tamanho de %s => %d' % (s, len(s))
8  # String tratada como sequência
9  for ch in s: print ch
10 # Strings são objetos
11 if s.startswith('C'): print s.upper()
12 # o que acontecerá?
13 print 3 * s
14 # 3 * s é consistente com s + s + s
```

```
a20101016660@labgrad02 ~ $ python teste.py
Trabalho de Linguagem de programaçao!
tamanho de Linguagem => 9
L
i
n
g
u
a
g
e
m
LinguagemLinguagemLinguagem
```

Tipos de Dados

- **String**
  - Em python é possível fazer “fatias” da strings de forma rápida.

```
Python 2.7.5+ (default, Feb 27 2014, 19:37:08)
[GCC 4.8.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> string ="Linguagem de Programacao"
>>>
>>> string[:9]
'Linguagem'
>>>
>>> string[9:]
' de Programacao'
>>>
>>> string[9:12]
' de'
>>> string[12:-1]
' Programaca'
>>>
```

Tipos de Dados

- **Listas**

  - Pode ser escrita como uma lista de valores separados por vírgula e entre colchetes. Mais importante, os valores contidos na lista não precisam ser do mesmo tipo.

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

  - Da mesma forma que índices de string, índices de lista começam do 0. Listas também podem ser concatenadas e sofrer o operador de slice.

Tipos de Dados

- Ao contrário das strings, listas são mutáveis, o que significa que uma atribuição naturalmente válida:

```
29 # Criando uma lista qualquer
30 progs = ['LP', 'Aprovacao', 'final', 'sucesso', 'reprovado']
31 # Varrendo a lista inteira
32 for prog in progs:
33  print prog
34 # Trocando o último elemento
35 progs[-1] = 'aprovado'
36 # Incluindo
37 progs.append('Igor')
38 # Removendo
39 progs.remove('final')
40 # Ordena a lista
41 progs.sort()
42 # Inverte a lista
43 progs.reverse()
44 # Imprime numerado
45 for i, prog in enumerate(progs):
46  print i + 1, '=>', prog
47 # Imprime do segundo item em diante
48 print progs[1:]
49
```

```
a2010101660@labgrad02 ~ $ python teste.py
LP
Aprovacao
final
sucesso
reprovado
1 => sucesso
2 => aprovado
3 => LP
4 => Igor
5 => Aprovacao
['aprovado', 'LP', 'Igor', 'Aprovacao']
```

Tipos de Dados

- **Cópia de lista:**
  - As variáveis são nomes para os objetos. Ao atribuir à variável outra_lista a lista minha_lista, o interpretador utilizou o mesmo objeto minha_lista ao invés de criar uma nova lista.

```
>>> minha_lista = [1, 2.5, 5, 'string qualquer']
>>> outra_lista = minha_lista
>>> outra_lista[0] = 10
>>> minha_lista
[10, 2.5, 5, 'string qualquer']
>>>
```

Para de fato copiar: acrescentar [:] ao final de “minha_lista” (slicing)

Tipos de Dados

- **Tuplas**

Tuplas não são nada mais do que listas imutáveis, ou seja, logo após sua criação não se pode alterá-las sem criar uma nova tupla.

```
>>> minha_tupla = ('a', 'b', 2.5, 666)
>>> minha_tupla
('a', 'b', 2.5, 666)
>>> minha_tupla[1]
'b'
>>> tup = ('um', 2, 3)
>>> var_a, var_b, var_c = tup
>>> print var_a, var_b, var_b, var_c
um 2 2 3
```

Assim como em uma lista, elementos em uma tupla têm uma ordem e o slicing também é possível.

Tipos de Dados

- **Tuplas**
  - Uma tupla tem um desempenho melhor que uma lista e, por ser imutável, não apresenta efeito colateral;
  - Tuplas podem ser convertidas em listas e vice-versa:

```
>>> minha_tupla
('a', 'b', 2.5, 666)
>>> list(minha_tupla)
['a', 'b', 2.5, 666]
>>> minha_lista
[10, 2.5, 5, 'string qualquer']
>>> tuple(minha_lista)
(10, 2.5, 5, 'string qualquer')
```

Tipos de Dados

- **Dicionário**
  - Um dicionário em Python é uma estrutura que define uma relação de 1 : 1 entre as respectivas chaves e valores.

```
>>> meu_dicionario = {'nome': 'Alex', 'idade': 12}
>>> meu_dicionario['nome']
'Alex'
>>> meu_dicionario['idade']
12
>>> del meu_dicionario['nome']
>>> meu_dicionario['altura'] = 1.80
>>> meu_dicionario
{'idade': 12, 'altura': 1.8}
```

Tipos de Dados

- Alguns fatos importantes sobre dicionários em Python:
  - Eles podem conter dados heterogêneos, você pode misturar inteiros com strings, etc
  - As chaves de um dicionário são case-sensitive
  - Um dicionário vazio pode ser declarado como:

    >>> meu_dicionario = {}
  - Você pode limpar um dicionário usando o método clear():

    >>> meu_dicionario.clear()
  - Dicionários requerem que os tipos das chaves implementem um método hash, reduzindo assim a complexidade de pesquisa em um dicionário em O(1).

Tipos de Dados

Booleano:

- O tipo booleano (bool) é uma especialização do tipo inteiro (int). O verdadeiro é chamado True e é igual a um, 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).

- Os objetos não citados são considerados verdadeiros

9. Funções

- Em Python, as funções são declaradas usando a palavra reservada def (de definition). Exemplo da declaração de uma função que fará a soma de dois parâmetros:

```
>>> def soma(a,b):
...     return a+b
...
>>> soma
<function soma at 0x7fdaa64a3668>
```

Note também que a função soma() também é um objeto.

Funções

- **Argumentos padronizados**
  - Além de podermos usar parâmetros padronizados, também podemos realizar chamadas para funções nomeando parâmetros:

```
>>> def adiciona_pessoa(nome, idade=28, sexo='M'):
...     print nome, idade, sexo
...
>>> adiciona_pessoa('alex')
alex 28 M
>>> adiciona_pessoa('Scarlett', sexo='F')
Scarlett 28 F
```

Funções

- **Lista de Argumentos Arbitrários**
  - Uma função pode ser construída de forma que aceite uma lista de tamanho arbitrário de argumentos:

```
>>> def argumentos_arbitrarios(*args):
...     print 'Argumentos:', args
...
>>> argumentos_arbitrarios('teste', 1, 2.0, [1, 2, 3])
Argumentos: ('teste', 1, 2.0, [1, 2, 3])
```

Funções

- **Algumas notas sobre funções:**
  - Todas funções retornam um valor e, na ausência de return, elas retornarão None.
  - A primeira linha após a definição de uma função pode ser uma string, neste caso esta string representará a documentação da função (chamada de docstring):

```
>>> def soma(a,b):
...     """Função de soma"""
...     return a+b
```

10. Módulos

- Qualquer arquivo contendo código-fonte Python é chamado de Módulo. Um módulo pode ser importado por outros módulos, de forma que as classes, funções, variáveis globais e outros objetos deste módulo possam ser acessados por outro módulo. A sintaxe para importação de um módulo é feita através do importação utilizando a palavra reservada import:

```
# Arquivo minhas_funcoes.py
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)

# Arquivo usa_modulo.py
import minhas_funcoes
resultado = minhas_funcoes.fibonacci(8)
```

Módulos

- Também podemos importar objetos específicos diretamente para o namespace atual:

```
# Importa todas funções (má prática)
from minhas_funcoes import *
resultado = fibonacci(8)

# Importa apenas a função soma (boa prática)
from minhas_funcoes import fibonacci
resultado = fibonacci(8)

# Importa módulo usando outro nome
import minhas_funcoes as mf
resultado = mf.fibonacci(2, 3)
```

Módulos

- Um pacote em Python não é nada além de uma estrutura de diretório contendo módulos.

```
# Importando modulo 'modulo_a'
>>> from frontend.desktop import modulo_a
>>> modulo_a.funcao()
```

11. Escopo de nomes

Solução: uso da declaração *global* (ou não usar variáveis globais!).

12. Biblioteca Padrão

- Python apresenta uma vasta biblioteca de módulos e pacotes que é distribuída com o interpretador. Alguns módulos importantes da biblioteca padrão são:

  - Matemática: math, cmath, decimal e random.
  - Sistema: os, glob, shutils e subprocess.
  - Threads: threading.
  - Persistência: pickle e cPickle.
  - XML: xml.dom, xml.sax e elementTree
  - Configuração: ConfigParser e optparse.
  - Tempo: time e datetime.

Biblioteca padrão

Exemplos:

```
import csv
import sys
import math
import itertools # to jump header in csv
import locale # 10,20 10.20
import operator # sort
```

13. Exceções

- O tratamento de erros em Python é realizado através de blocos utilizando as palavras-chaves *try* e *except*:

```
meu_dicionario = {'chave_a': 1, 'chave_b': 2}
try:
    print meu_dicionario['chave_c']
except KeyError:
    print 'Chave não encontrada !'
```

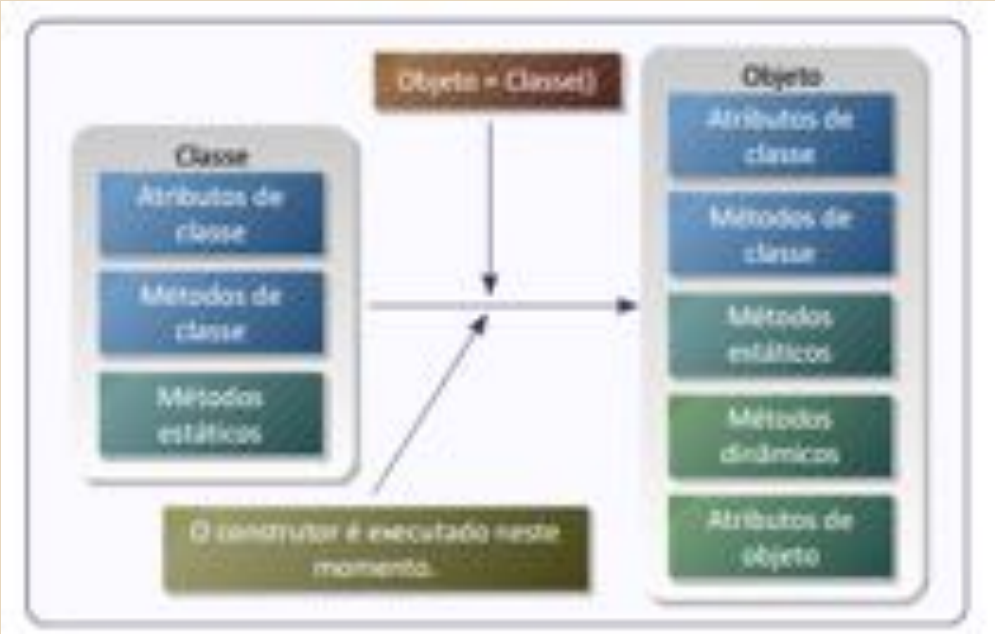
Exceções

- Para levantar exceções em Python, a palavra reservada *raise* deve ser utilizada:

```
def verifica_numero_primo(numero):
    if numero < 0:
        raise ValueError('O valor deve ser maior '
                         'ou igual a zero !')
    # (...)
```

14. Classes

- Em Python:
  - Quase tudo é objeto, mesmo os tipos básicos, como números inteiros;
  - Tipos e classes são unificados;
  - Os operadores são na verdade chamadas para métodos especiais;



Classes

- Novos objetos são criados a partir das classes através de atribuição. O objeto é uma instância da classe, que possui características próprias.

- Quando um novo objeto é criado, o construtor da classe é executado. Em Python, o construtor é um método especial, chamado __new__(). Após a chamada ao construtor, o método __init__() é chamado para inicializar a nova instância.

Classes

- Em Python, as classes são de□finidas utilizando a palavra reservada *class*;

```
# -*- coding: latin1 -*-

class Classe(supcl1, supcl2):
    """
    Isto é uma classe
    """
    clsvar = []

    def __init__(self, args):
        """
        Inicializador da classe
        """
        <bloco de código>

    def __done__(self):
        """
        Destrutor da classe
        """
        <bloco de código>

    def metodo(self, params):
        """
        Método de objeto
        """
        <bloco de código>
```

Classes

- Métodos de objeto podem usar atributos e outros métodos do objeto. A variável *self* que representa o objeto também precisa ser passado de forma explícita.

- Métodos estáticos são aqueles que não tem ligação com atributos do objeto ou da classe. Funcionam como as funções comuns.

- Um objeto continua existindo na memória enquanto existir pelo menos uma referência a ele. O interpretador Python possui um recurso chamado coletor de lixo (Garbage Collector) que limpa da memória objetos sem referências. Quando o objeto é apagado, o método especial __done__() é evocado.

15. Coletor de lixo

- Python utiliza um mecanismo bem simples como coletor de lixo: contagem de referências.

- Isso significa que para cada objeto é mantido um registro do número de referências existentes para ele. Quando esse número chegar a zero, ou seja, quando o objeto não estiver mais sendo usado, ele é destruído.

16. Polimorfismo

- As chamadas de função em Python são universais ou genéricas sem determinação de tipo. Por isso, sobrecarga não é suportada na linguagem.

- Graças à tipagem dinâmica, o polimorfismo de tipos acontece à todo momento!

17. Herança

Simples

```
class ClasseBase(object):
    def nome(self):
        print 'Sou ClasseBase !'
    def base(self):
        print 'Método da ClasseBase !'

class ClasseDerivada(ClasseBase):
    def nome(self):
        print 'Sou ClasseDerivada !'
```

Herança

Múltipla

```
class ClasseA(object):
    def metodo_a(self):
        print 'metodo_a !'

class ClasseB(object):
    def metodo_b(self):
        print 'metodo_b !'

class ClasseC(ClasseA, ClasseB):
    pass
```

Herança

Em Python não há um conceito de atributos privados como há em outras linguagens. Python encoraja os desenvolvedores a serem responsáveis.

18. Concorrência

- GIL (Global Interpreter Locker)
  - mecanismo de exclusão mútua do interpretador;
  - apenas uma thread pode executar instruções por vez;

- threading
  - Presente na biblioteca padrão;
  - Classes de alto nível de abstração;
  - Módulo *thread*, que implementa as rotinas de baixo nível e geralmente não é usado diretamente;

Concorrência

- Stackless Python
  - Concorrência baseada em tasklets (green thread);
  - Também utiliza canais síncronos para comunicação e sincronização de tasks;

- Multiprocessing
  - paralelismo através de múltiplos processos Python;
  - Comunicação principalmente através de envio de mensagens;
  - Maior overhead de sincronização;

- Green threads
  - Threads no user space;
  - Representam uma abstração às threads do sistema operacional (kernel space);

19. Avaliação da LP

Aplicabilidade

- Python, apesar de tipicamente usado em aplicações web e como linguagem de scripting para administração de sistemas, é uma linguagem de uso geral que pode ser empregada em vários tipos de problemas. A biblioteca padrão inclui módulos para processamento de texto e expressões regulares, protocolos de rede (HTTP, FTP, SMTP, POP, XML-RPC, IMAP), acesso aos serviços do sistema operacional, criptografia, interface gráfica etc. Além da biblioteca padrão, existe uma grande variedade de extensões adicionais para todo tipo de aplicação.

Facilidade de aprendizado

- É uma linguagem de alto nível, com alta capacidade de abstração o que torna essa simplicidade poderosa e expressiva. Python possui um conjunto reduzido de estruturas de controle, de forma a reduzir a complexidade da linguagem. Além disso a forma de expressar um loop for para percorer elementos de uma lista ou dicionário ou qualquer objeto que implemente alguns métodos especiais é simples e poderosa.

Avaliação da LP

Legibilidade:

- Python possui uma sintaxe clara e concisa, que favorece a legibilidade do código fonte, tornando a linguagem mais produtiva.
- Uso da indentação para delimitar.

Redigibilidade:

- Tipagem dinâmica (não é preciso explicitar tipos);
- Controle de laços é feito por indentação, não necessita de chaves;

Eficiência:

- Por ser interpretada, é mais lenta que C e C++;
- Em geral é uma linguagem eficiente, mas dependente da aplicação;
- Verificação dinâmica de tipos;
- Passagem de parâmetros normal (eager);

Avaliação da LP

Confiabilidade:

- É uma linguagem dinâmicamente tipada e ao mesmo tempo fortemente tipada e também possui inferência a tipos.
- O sistema de verificação de tipos não realiza a conversão implícita de um tipo de dados para outro, levantando uma exceção quando tipos inconsistentes são utilizados.

Custo:

- O custo de treinamento e para escrever programas em Python é baixo, uma vez que a que capacidade de escrita e a legibilidade são altas.
- O tempo do entendimento de códigos na aprendizagem e para a manuntebilidade é baixo.

Avaliação da LP

Portabilidade:

- A implementação padrão do Python é escrita em ANSI C portável, compila e executa em praticamente todas as principais plataformas em uso atualmente. Por exemplo, os programa em Python são executados em tudo, de PDAs até supercomputadores.

Reusabilidade:

- Oferece reuso de funções, tipos e variáveis distribuídas em bibliotecas.
- Oferece também o conceito de classes e possuem mecanismo de pacotes.
- O polimorfismo universal também auxilia na criação de código reusável e frameworks.

Avaliação da LP

Integração

- A facilidade de integração com C faz de Python uma linguagem embutida atrativa em aplicações de maior porte. Atualmente, o código Python pode chamar bibliotecas de C/C++, pode ser chamado a partir de programas em C/C++, pode ser integrado com componenetes Java, pode se comunicar pela rede.

Expressões e comandos

- Apresenta uma ampla variedade de comandos e expreções.

Tipos primitivos e compostos

- Apesar de não exigir declaração de tipos, apresenta ampla variedade.

Avaliação da LP

Gerenciamento de memória

- A linguagem Python disponibiliza um mecanismo de gerência de memória automático que é responsável por alocar memória para seus objetos e desalocá-la quando esses objetos não possuem mais referência para eles.

Passagem de parâmetros

- Variável em Python nada mais é do que uma referência para algum objeto da memória.

Encapsulamento e proteção

- Oferece através dos mecanismos de classes e pacotes.

Avaliação da LP

Verificação de tipos

- Python utiliza tipagem dinâmica, o que significa que o tipo de uma variável é inferido pelo interpretador em tempo de execução (isto é conhecido como Duck Typing).

Polimorfismo

- Python apresenta polimorfismo universal por inclusão que é caraterizado por herança simples e múltipla.

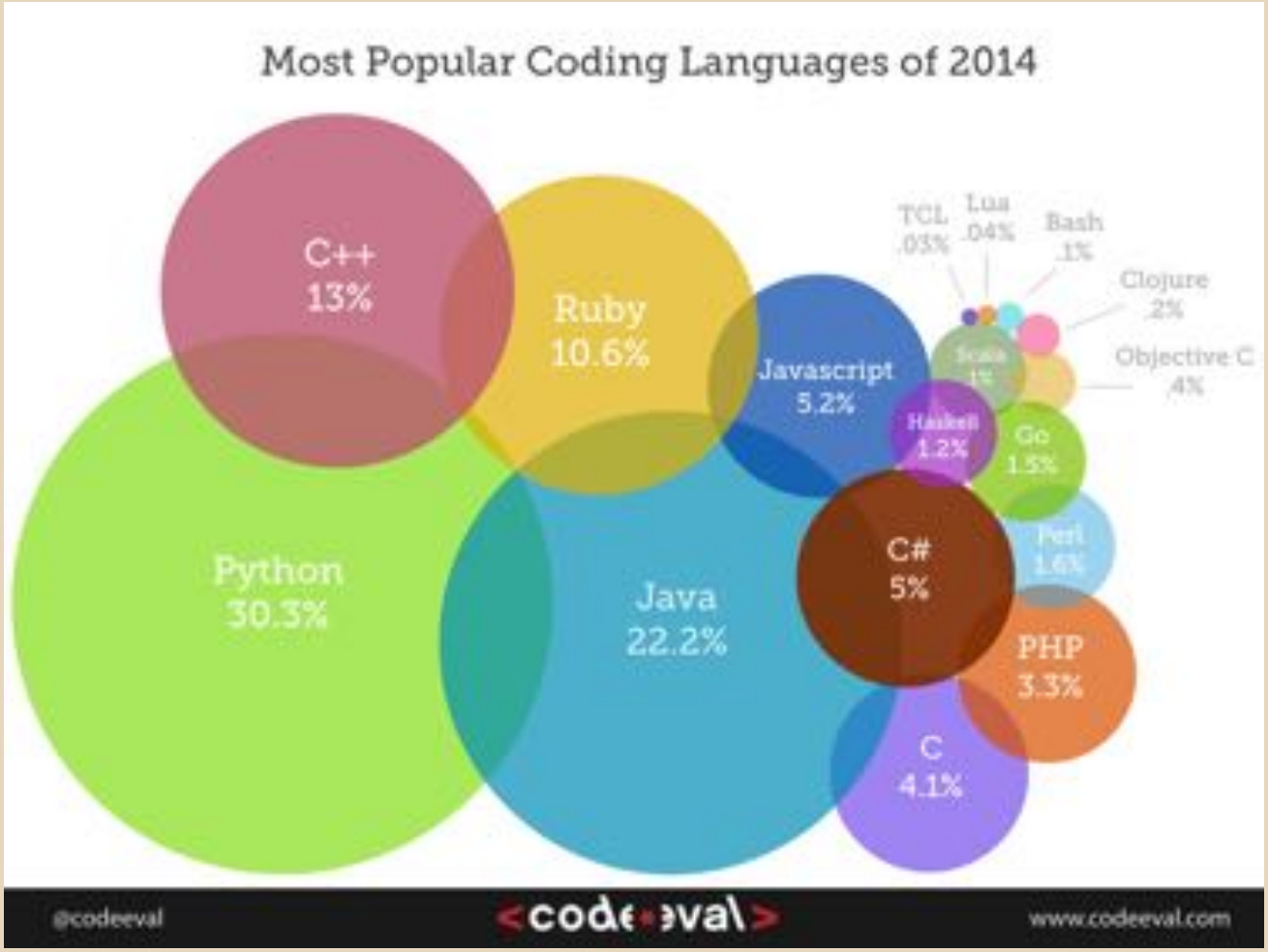
Exceções

- Python apresenta mecanismos embutidos para tratamento de exceções;

Concorrência

- Python apresenta bibliotecas para uso de concorrência;

20. Curiosidades



Fonte: http://blog.codeeval.com/codeevalblog/2014

Curiosidades

Oct 2014	Oct 2013	Change	Programming Language	Ratings	Change
1	1		C	17.925%	+0.43%
2	2		Java	13.939%	-2.63%
3	3		Objective-C	10.089%	+1.10%
4	4		C++	6.866%	-3.40%
5	8	^	C#	4.708%	-0.97%
6	7	^	Basic	3.507%	-1.31%
7	5	v	PHP	2.942%	-3.29%
8	9		Python	2.333%	-0.77%
9	12	^	Perl	2.129%	+0.53%
10	6	v	Transact-SQL	2.022%	-0.92%
11	17	^	Delphi/Object Pascal	1.852%	+1.11%
12	10	v	JavaScript	1.771%	-0.27%
13	11	v	Visual Basic .NET	1.752%	-0.18%
14	-	^	Visual Basic	1.564%	+1.56%
15	31	^	R	1.515%	+0.97%
16	13	v	Ruby	1.139%	-0.13%
17	81	^	Dart	1.118%	+1.08%
18	24	^	Ada	0.868%	+0.37%
19	-	^	Swift	0.781%	+0.78%
20	18	v	Pascal	0.736%	-0.03%

Fonte: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

21. Aplicações

22. Referências

- http://www.dcc.ufrj.br/~fabiom/mab225/tutorialpython.pdf
- http://www.python.org.br/wiki/Tutorial_Python
- http://dinomagri.com/wp-content/uploads/2014/04/Aula1.pdf
- http://ark4n.files.wordpress.com/2010/01/python_para_desenvolvedores_2ed.pdf
- http://turing.com.br/pydoc/2.7/tutorial/index.html
- http://slideplayer.com.br/slide/2262855/
- http://pt.slideshare.net/gobila/introduo-a-programao-em-python-d2
- http://homepages.dcc.ufmg.br/~joaoreis/Site%20de%20tutoriais/aprendendopython/index.html
- http://pt.slideshare.net/perone/introduo-bsica-a-linguagem-python
- http://lpunb.wikia.com/wiki/Linguagem_Python_-_2/2012_Grupo_2
- http://p.souza.cc/concorrencia-em-python/?full#Cover