Python

Bruno Silva
Gustavo Monjardim

Sumário

- Introdução
- Sintaxe e Comandos;
- Identificadores e Sistema de Tipos;
- Módulos, Funções, Classes;
- Comandos iterativos;
- Polimorfismo
- Exceções
- Concorrência
- Avaliação da Linguagem

Histórico

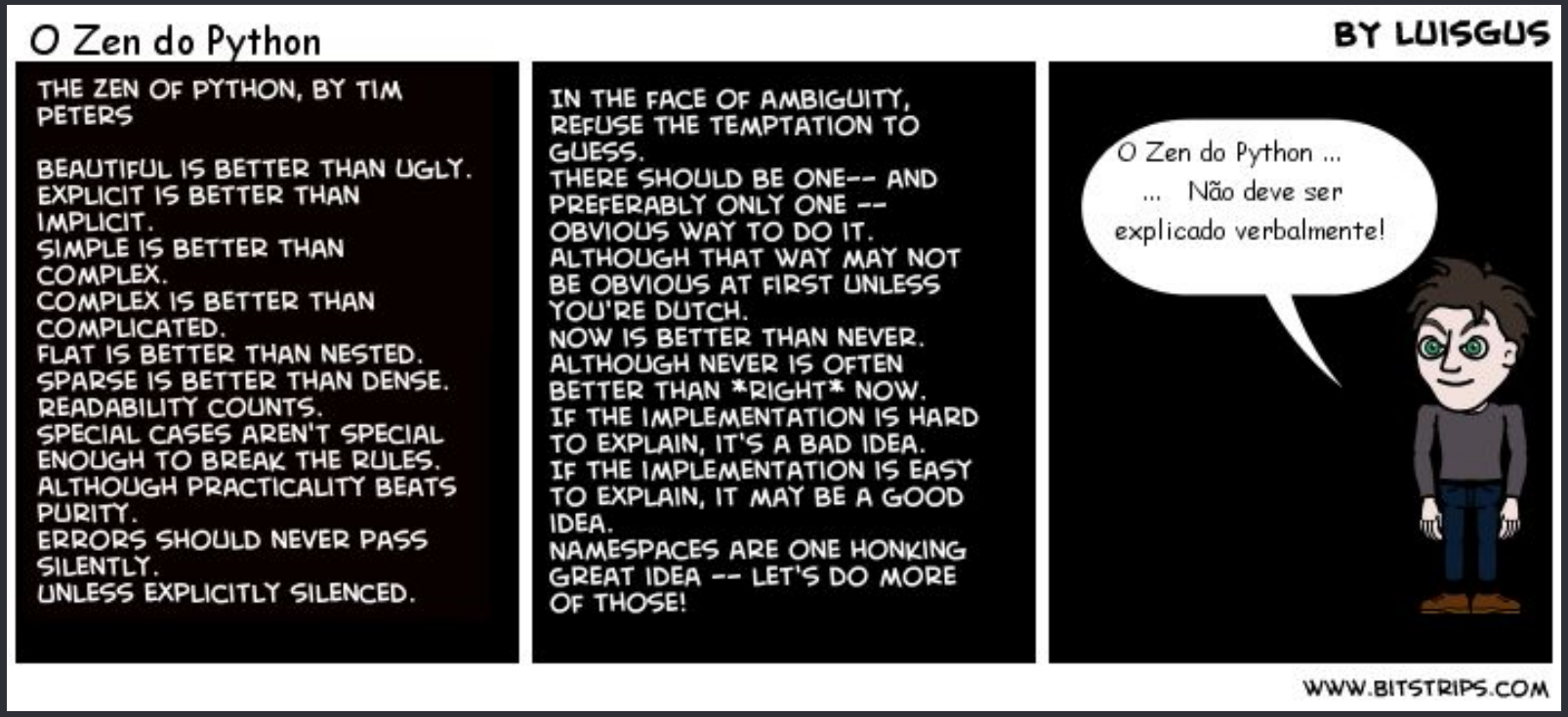
- Concebida no final da década de 1980 e lançada em 1991 por Guido Van Rossum;
- Derivada da linguagem ABC, que foi desenvolvida nos anos 80 com propósitos didáticos, mas adquiriu características de diversas outras linguagens, como C, C++, Modula, entre outras.
- LP open-source de alto nível;
- Portátil;
- Facilidade de aprendizado;
- Facilidade de integração com outras linguagens, como C e C++.

G. V. R. em 2006

O Zen do Python

- Escrita por Tim Peters, o Zen é um grupo de filosofias extremamente simples e que podem soar óbvias
  - Simples é melhor que complexo;
  - Legibilidade Conta;
  - Erros não devem passar em silêncio, a menos que sejam silenciados;
  - Agora é melhor que nunca;
  - Se a implementação é fácil de explicar, pode ser uma boa ideia;

O Zen do Python



Executando um programa

- Para executar um script em Python, basta abrir o terminal e executar o comando:

  >>> python modulo.py

- Modo interativo
  - O interpretador Python pode ser usado de forma interativa, nesse modo linhas de código são digitadas direto no terminal. Para iniciar o modo interativo basta digitar “python” no terminal.

Por que utilizar Python?

- Como podemos imprimir uma mensagem na tela?

C	Java	Python

#include <stdio.h>
 	
int main(void) {
printf("Hello World\n");
}
	public class Hello{
	public static void main(String args[])
	{
	System.out.println("Hello World);
	}
	}
		print("Hello World")

Aplicações

- Web;
- Gráficas;
- Administração de Sistemas;
- Data Science;
  - Tratamento e visualização de dados;
- Inteligência Artificial (Machine Learning);

Fonte: https://wiki.python.org.br/PerguntasFrequentes/SobrePython#Em_que_tipo_de_aplica.2BAOcA9Q-es_eu_usaria_Python.3F

Aplicações

- Ranking Interativo do IEEE Spectrum
  - Linguagens mais utilizadas
  - Em Julho/2017:

Fonte: https://www.tecmundo.com.br/software/119960-java-c-ranking-ieee-spectrum-revela-linguagem-usada-2017.htm . Acesso em 10/11/2017

Características

- Linguagem Interpretada (híbrida);
- Interpretador Python:
  - Versões mais recentes: 2.7.13 e 3.6.3
  - Não são totalmente compatíveis;
- Tipagem dinâmica;
- Fortemente tipada;
- Blocos definidos de acordo com a indentação;
  - ‘;’ separa comandos na mesma linha
  - ‘;’ ao final das linhas (se existir) é ignorado
- Estruturada, Orientada a Objetos e Funcional;

Interpretador

- Método híbrido:
  - O código é traduzido para bytecode (.pyc no Linux) antes de ser realmente interpretado;
  - Facilita a portabilidade;
  - Caso o bytecode já exista e o código não tenha sido modificado, o programa é executado diretamente;
- Implementação:
  - Principal: CPython (em linguagem C);
  - Jython

Sintaxe

- Python utiliza indentação para separação de blocos de código.
- Utiliza palavras ao invés de símbolos para algumas operações.
- # é utilizado para fazer um comentário de uma linha e aspas triplas são utilizados para comentários em múltiplas linhas.

Script helloworld.py	Saída no terminal

#Um exemplo de Hello World:
 	
if True and False:
  print(“Hello World”)
else:
  print(“Olá mundo”)
	Olá mundo

Alguns Comandos Básicos

- abs(x)
- print(x)
- print x
  - somente na versão 2
- eval(string):
  - executa o comando na string, modificada em tempo de execução

Alguns Comandos Básicos

- len(x):
  - tamanho de um container x (lista/vetor/mapa)
- getattr(objeto, nome) e setattr(objeto, nome, valor):
  - acessa (e modifica) o valor de dado atributo de um objeto de uma classe (o objeto e o atributo podem ser definidos em tempo de execução)
- import
- open
- issubclass
- isinstance

Palavras Reservadas

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

Operadores

- Aritméticos

  `+` `-` `*` `/` `%` `**` `//`

- Relacionais

  `==` `!=` `<>` `>` `<` `>=` `<=`

- Atribuição

  `=` `+=` `=+` `*=` `%=` `**=` `//=`

Operadores

- Operadores Bitwise
  & | ^ ~ << >>

- Operadores Lógicos
  and or not

- Operadores de Associação
  in / not in
- Operadores de Identidade
  is / not is

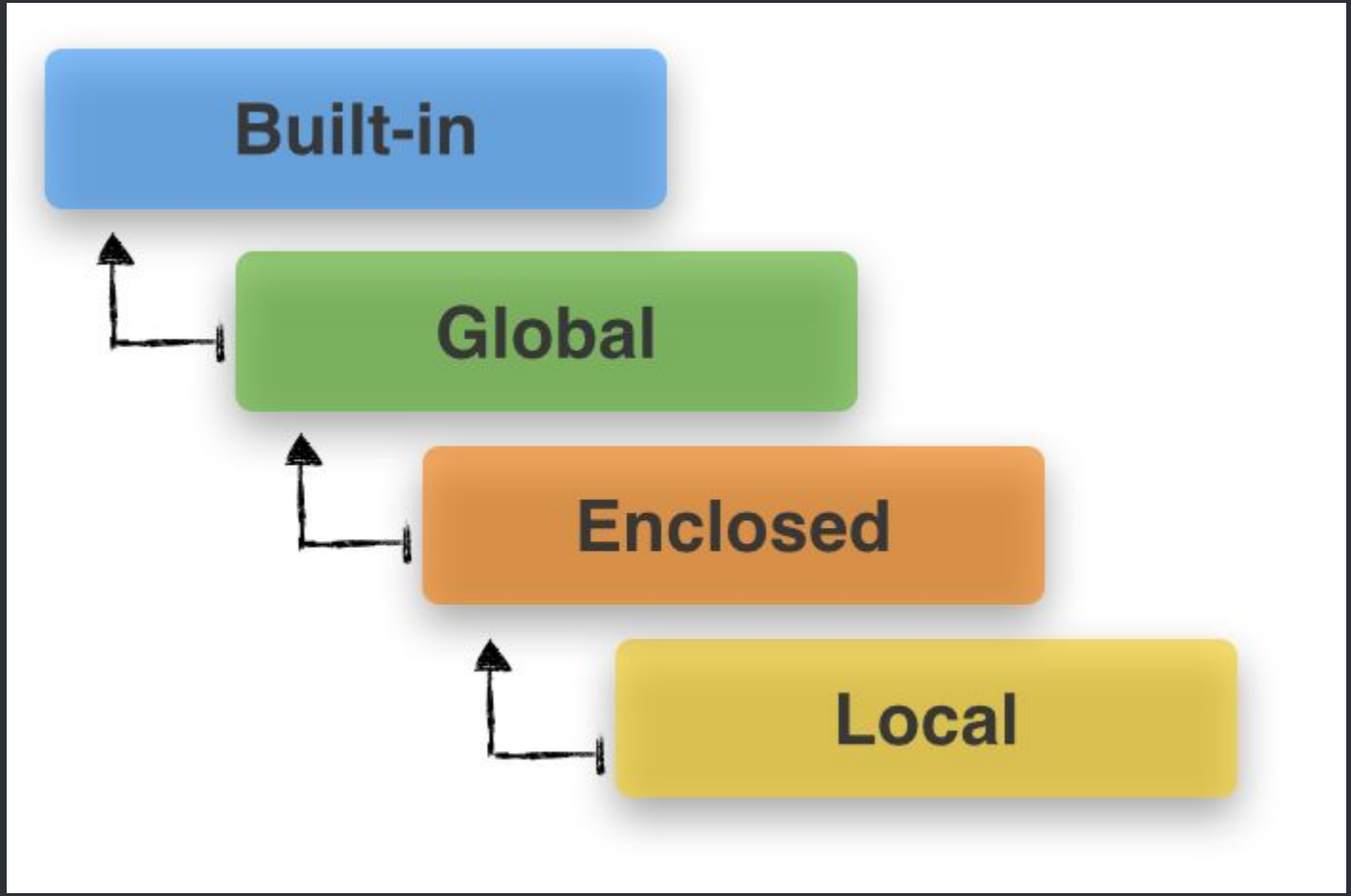
Amarrações

- Em Python tudo é um objeto.
- Os nomes de variáveis são referências a objetos, essas referências podem ser alteradas em tempo de execução.
- Amarrações entre nomes e objetos são feitas de forma dinâmica.

Escopo

- Escopo Estático
- São mantidos por Namespaces, que são mapeamentos que relacionam os nomes dos objetos aos objetos em si.
- Quando um nome não é encontrado em nenhum ambiente de amarração a exceção NameError é lançada. Se o nome foi encontrado mas ainda não aconteceu nenhum amarração a ele a exceção UnboundLocalError é lançada.

Escopo



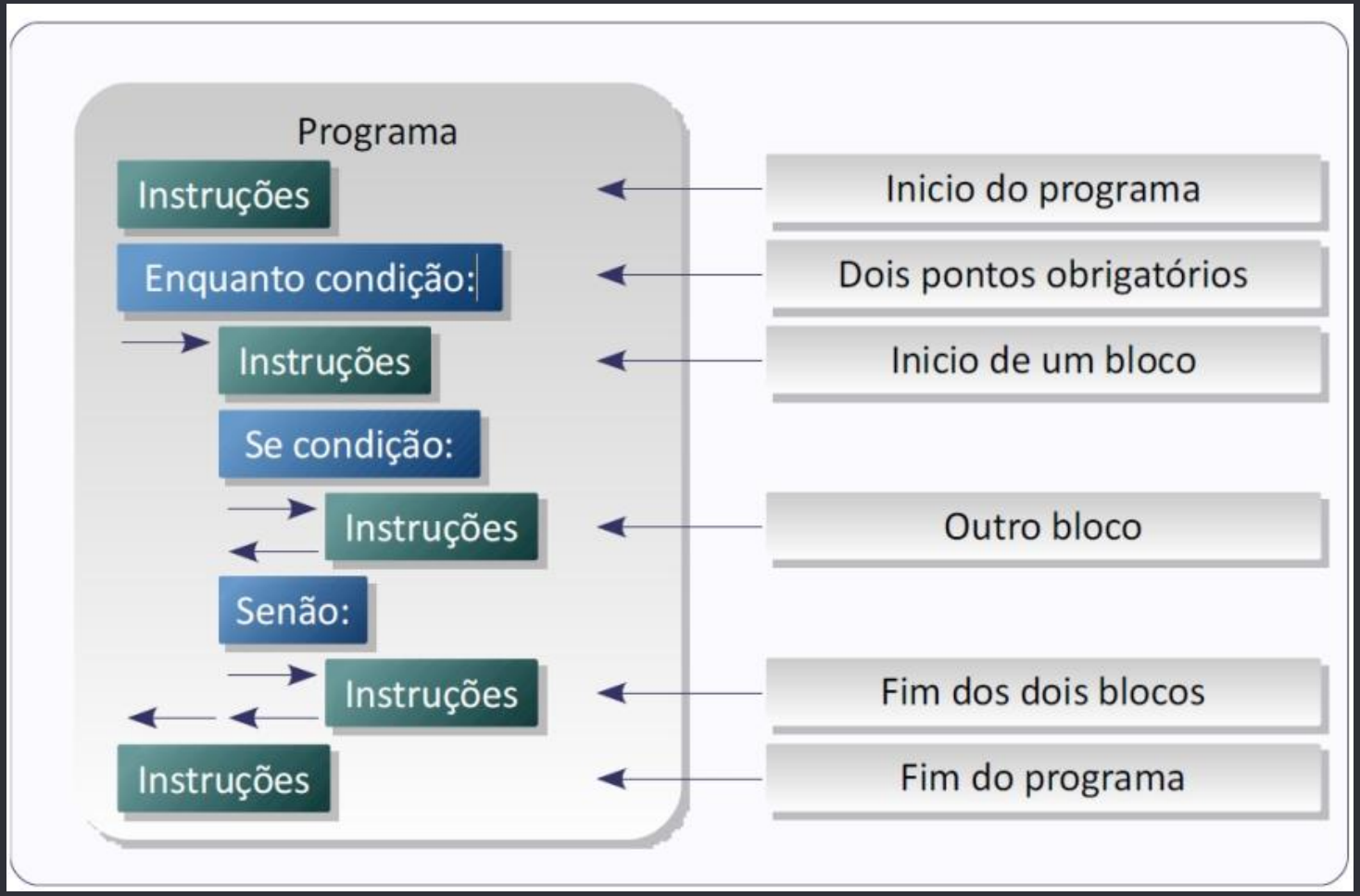
Escopo

- Variável Global

Script	Saída

a = 1
def f():
  a = 2
  print(a)
f()
print(a)
 	
a = 1
def f():
  global a
  a = 2
  print(a)
f()
print(a)
	2
	1
	
	2
	2

Blocos


Programa
Instruções
Enquanto condição:
Instruções
Se condição:
Instruções
Senão:
Instruções
Instruções
Inicio do programa
Dois pontos obrigatórios
Inicio de um bloco
Outro bloco
Fim dos dois blocos
Fim do programa

Identificadores e Tipos

- Tipagem Dinâmica.
- Fortemente tipada.
- Case-sensitive.
- Os caracteres válidos para identificadores são: letras maiúsculas e minúsculas (A a Z, a a z), o underscore (_) e dígitos (0 a 9) (com exceção do primeiro caractere).
- Podem ser mutáveis ou imutáveis.

Tipos de variáveis

- Alguns tipos ‘built-in’:
  - bool
  - int, long(apenas na versão 2)
  - float (precisão dupla)
  - complex (2 floats)
  - str (string, apenas na versão 2)
  - unicode (string)
  - list
  - tuple
  - range/xrange (apenas range na versão 3)
  - set
  - dict
  - file
  - Módulos/Funções/Classes

Tipos de variáveis

- Tipos numéricos
  - bool: and,or,not
  - int/float/complex: *, **, /, //, +, - , %:
    - -a: a negado
    - +a: a sem mudanças
    - permitem as operações ++a, --a (retornam a), -+a e +-a (retornam -a)
    - a++ não é permitido;
  - int/float/complex bit a bit: &, |, ^, <<, >>, ~
  - bool é subclasse de int:

Arquivo .py	Saída
a = True b = 10 print(a+b) #Conversão implícita	11

Tipos de variáveis

- Bool
  - Os seguintes valores são considerados falsos:
    - False.
    - None.
    - 0 (Zero).
    - “” (String vazia).
    - [] (Lista Vazia).
    - () (Tupla vazia).
    - {} (Dicionário Vazio).
  - Qualquer outro objeto é considerado verdadeiro.

Script	Saída
“Python” and 0 [] and True “” or () “false” or {}	0 [] () false

Tipos de variáveis

- List:
  - Mutável;
  - Armazena objetos (de qualquer classe);
  - Definida usando colchetes;
  - lista[x]: acessa elemento na posição x (inteiro);
  - Tamanho “ilimitado”;
  - Métodos: append, del, extend (+), *, count, sort, entre outros;
  - Operador in;

Script	Saída
a = int(3); b = float(3.4) c = []; c.append(a) c.append(b); d = c + c c.extend(c); e = [0]*5 print(c, d, e, e.count(0), 0 in e)	[3, 3.4, 3, 3.4] [3, 3.4, 3, 3.4] [0, 0, 0, 0, 0] 5 True

Tipos de variáveis

- Acesso a elementos em listas com T elementos:
  - Um elemento específico:
    - L[0]: Primeiro elemento da lista;
    - L[-1]: o último elemento da lista;
      - como se fosse a posição de número T-1;
  - Sublistas: através de “slices”:
    - L[a:b]: uma lista com os elementos da posição “a” até a posição anterior a “b” (a posição “b” não é incluída);
      - “a” deve ser menor que “b”, caso contrário uma lista vazia é retornada
      - Se a = 0 ou b = T-1, eles podem ser omitidos do comando

Tipos de variáveis

- Acesso a elementos em listas com T elementos:
  - Sublistas: através de “slices”:
    - L[:-1]: todos os elementos excluindo o último;
    - L[-2:]: os 2 últimos elementos;
    - L[-4:-3] é válido?

Tipos de variáveis

- Acesso a elementos em listas com T elementos:
  - através de “slices”:
    - L[:-1]: todos os elementos excluindo o último;
    - L[-2:]: os 2 últimos elementos;
    - L[-4:-3] é válido?
      - Sim! o comando é equivalente a L[T-4:T-3];
      - e T-4 <= T-3;

Script	Saída

L = [1,2,3,4,5] #T = 5
print(L[0])
print(L[0:4])
print(L[:-1])
print(L[-2:])
print(L[-4:-3]
	1
	[1, 2, 3, 4]
	[1, 2, 3, 4]
	[4, 5]
	[2]

Tipos de variáveis

- String / Unicode:
  - Tipo imutável;
  - Métodos: find, format (e operador %), lower, upper, isdigit, split, strip, entre outros;
  - Na versão 3, todas as strings são unicode;

Script	Saída
print('{}'.format(3.1415)) print('{:.4f}'.format(3.1415)) print('%.4f' % (3.1415)) print('' a b ''.strip()) x = “Python” print(x[0:4]) print(x[2:]) print(x[:]) print(x[-1])	3.1415 3.1415 3.1415 a b Pyth thon Python n

Tipos de variáveis

- Tupla:
  - Similar a listas, mas imutável;
  - Armazena objetos (de qualquer classe);
  - Definida usando parênteses e vírgula;
  - Tamanho “‘ilimitado”;

Script	Saída
tupla = (3,4.5,5,[]) print(tupla) print(tupla[0]) # Tupla é um tipo imutável # tupla[0] = 3 #não funciona	(3, 4.5, 5, []) 3

Tipos de variáveis

- xrange/range:
  - Na versão 2, range(100000) cria uma lista que vai de 0 até 100000;
  - range(y) utilizada principalmente em loops, mas desperdiça memória;
  - Para evitar desperdício, a função xrange foi criada;
    - Utiliza a mesma quantidade de memória independente do tamanho do loop;
  - A partir da versão 3, a antiga função range não existe mais, e a xrange teve seu nome trocado para range;

Script	Saída
for i in xrange(5): print(i)	1,2,3,4,5

Tipos de variáveis

- Set:
  - Conjunto não ordenado de objetos iteráveis (listas, tuplas, etc)
  - Métodos: update, intersection, union, issubset, entre outros...
  - Operador in;
- Frozenset
  - Equivalente a Set, mas imutável

Script	Saída
a = set([1]); b = set([3]) a.update([2]); b.update([2]) print(a,b); print(a.intersection(b)) print(a.union(b)); print(1 in a)	(set([1, 2]), set([2, 3])) set([2]) set([1, 2, 3]) True

Tipos de variáveis

- dict:
  - Representa um conjunto de pares chave-objeto;
  - Chaves armazenadas em tabela hash com tamanho variável;
  - Não pode ser ordenado;
  - Chaves devem ter método __hash__();
  - Definidos usando chaves {} e pares chave:valor
  - Se chave já existir, sobrescreve o objeto;
  - Operador in para descobrir se chave está no dict;

Script	Saída
dic = {}; dic['b'] = 5; dic[1] = 2; dic['a'] = 0; dic[1] = 100; key = 1 print(dic, key in dic); print(dic.keys()) print(dic.values())	{'a': 0, 1: 100, 'b': 5} True ['a', 1, 'b'] [0, 100, 5]

Tipos de variáveis

- File:
  - Parecido com o tipo FILE em C;
  - Função open(nome, modo) retorna um objeto File
  - Métodos: close, read, readline, write, mode, entre outras;

Script	Saída

arquivo = open('teste.txt','w')
arquivo.write("teste\n")
 	
#Nao esquecer de fechar o arquivo
#Caso contrário, não há garantia
#que tudo foi escrito
arquivo.close()	

Módulos, Funções e Classes

- Módulos:
  - Um arquivo python com definições de funções e classes é um módulo, e pode ser importado usando o comando “import arquivo”;
  - Um módulo pode ser importado por completo
    - import numpy
    - from numpy import *
  - Ou apenas funções podem ser importadas...
    - from numpy import array
  - Ainda é possível dar um apelido ao módulo:
    - import numpy as np

Módulos, Funções e Classes

- Módulos:
  - “__builtin__” é um módulo que é importado quando o interpretador é chamado;
    - Contém funções abs(), print(), eval(), etc...
  - Existem diversos módulos prontos em python, alguns deles são:
    - sys
    - math
    - os
    - datetime
    - random
    - multiprocessing

Script	Saída
import math import random print(math.sin(math.pi/2)) print(random.random())	1.0 0.268623459427

Módulos, Funções e Classes

- Funções:
  - Criadas através de definição de funções;
  - Todo operador chama uma função (é possível definir operadores para classes do programador, mas não é possível redefinir o operador + de inteiros, por ex.);
  - Pode receber um ou mais argumentos;
  - Suporta valores default (assim como C++);
  - Argumentos não precisam estar em ordem se identificados;

Módulos, Funções e Classes

- Funções:

Script	Saída
def novaFuncao(arg1 = 1, arg2 = 2): return arg1 ** arg2 a = 2; b = 3 print(novaFuncao(a,b)) #2^3 print(novaFuncao(a)) #2^2 print(novaFuncao(arg2 = a, arg1 = b)) #3^2	8 4 9

Módulos, Funções e Classes

- Funções
  - Toda função retorna um objeto (ou uma tupla com vários objetos);
  - Caso não tenha valor definido pelo programador, retorna None.
  - Uma função por si só também é um objeto.

Módulos, Funções e Classes

- Funções

Script	Saída

def soma(x, y):
  return x + y
 	
type(soma)
s = soma
s(1, 2)
soma(1, 2)
 	
def f():
  return
print(f())
 	
type(None)
	<class ‘function’>
	3
	3
	None
	<class ‘NoneType’>

Módulos, Funções e Classes

- Funções:
  - Suporta parâmetros variáveis:
    - Sintaxe parecida com ponteiros de C (*args);
    - Considerados como tuplas;

Script	Saída

def func(x, *args):
  print x
  print args
 	
func(1, 2, 3, 4, 5)
	1
	(2, 3, 4, 5)

Módulos, Funções e Classes

- Closures:

Script	Saída

def criaClosure(x):
  def soma(y):
  return y + x
  return soma
 	
closure = criaClosure(1)
print(closure(2))
print(closure(2.3))
	3
	3.3

Módulos, Funções e Classes

- Funções lambda:

Script	Saída
lambFun = lambda x1, x2: x1 + x2 print(lambFun(1,2)) print(lambFun(1,2.3))	3 3.3

Parâmetros

- Correspondência real-formal:
  - Posicional e por palavra-chave.
  - Posicional obrigatoriamente à esquerda dos parâmetros com palavra-chave;
- Direção de passagem:
  - Unidirecional de entrada variável.
- Mecanismo de passagem:
  - Referência.
- Momento da passagem:
  - Normal.

Módulos, Funções e Classes

- Classes:
  - Não há modificadores de acesso;
  - Atributos/métodos “privados” são identificados com underscores:
    - no mínimo 2 no início e no máximo um no final:
    - __atributoPrivado é “privado”;
    - Na verdade o interpretador renomeia esse atributo para dificultar o acesso;
    - Pode ser acessado usando o comando objeto._Classe__atributoPrivado;
  - Tudo em python é um objeto de uma classe (incluindo classes);
    - metaclasse padrão “type”
  - A partir da versão 3, todas as classes herdam da classe object;
  - Classes abstratas usando o módulo abc (Abstract Base Class)
    - metaclasse ABCMeta

Módulos, Funções e Classes

- Classes:
  - horario é atributo estático, mas cada instância tem um horário diferente;
  - vaiAcabar é um método estático

```
class Seminario(object):
    horario = '7am'
    def __init__(self, linguagem):
        disciplina = "LP"
        linguagem = linguagem

    @staticmethod
    def vaiAcabar():
        print("Em breve!!")

Python = Seminario("Python");
Seminario.vaiAcabar();
Python.horario = '8am'
print(Python.horario); print(Seminario.horario)
```

```
Em breve!!
8am
7am
```

I/O

- Entrada

```
var = input(“Digite algo: ”)
num = int(input(“Digite um inteiro: ”)
```

- Saída

```
print(x)
print(“Hello World”)
```

Gerência de memória

- Python não oferece o uso de ponteiros
- Utiliza um mecanismo de contagem de referências como coletor de lixo.
- O coletor de lixo mantém um registro do número de referências existentes para cada Objeto. Quando esse número chegar a zero o objeto é destruído.
- Através do módulo “gc”, é possível:
  - Habilitar/Desabilitar o coletor;
  - Forçar uma varredura;

Comandos Condicionais

- If
  - Python não possui o comando switch.

Script	Saída

num = 3
 	
if num == 1:
  print(“Um”)
elif num == 2:
  print(“Dois”)
else:
  print(“Outro número”)
	Outro número

Comandos Iterativos

- For

Script	Saída

for x in range(4):
  print(x)
 	
for letter in “Casa”:
  print(letter)
 	
vehicle = [“Carro”, “Moto”, “Caminhão”]
for vehicle in vehicles :
  print(vehicle)
 	
for num in range(2,5):
  print(num)
else:
  print(“Ok!”)
	0
	1
	2
	3
	
	C
	a
	s
	a
	
	Carro
	Moto
	Caminhão
	
	2
	3
	4
	Ok!

Comandos Iterativos

- While
  - assim como “for”, aceita else;

Script	Saída

cont = 0
 	
while (cont < 5):
  print(cont)
  cont += 1
else:
  print(str(cont) + " não é menor que 5")
	0
	1
	2
	3
	4
	5 não é menor que 5

Comandos Iterativos

- goto - Python não possui o comando goto
- break - Termina a execução do loop, e executa a primeira instrução após o loop.
- continue - Pula para próxima iteração do loop.
- pass - É utilizado quando um comando é requerido sintaticamente mas você não quer executar nada.

Script	Saída

for x in range(5):
    if(x == 2):
        pass
    else:
        print(x)
	0
	1
	3
	4

Polimorfismo

- Python tem tipagem dinâmica e forte
- Ad-hoc
  - Coerção
    - Feita implicitamente pelo interpretador Python.
  - Sobrecarga
    - Não suporta sobrecarga de subprogramas.
    - Permite sobrecarga de operadores para classes definidas pelo programador.
- Universal:
  - Paramétrico
    - Embutido na linguagem.
  - Inclusão:
    - Extensão de classes

Polimorfismo

- Coerção:
  - Função coerce (python 2) converte os parâmetros para o mesmo tipo
  - Lança exceção TypeError caso algum erro ocorra

Script	Saída
a = 3.4 + 2 print(a) b = coerce(3.4,2) print(b) #coerce([2],1) #Lança TypeError	5.4 (3.4, 2.0)

Polimorfismo

- Sobrecarga:
  - Não permite sobrescrever operadores existentes
  - Operadores de novas classes:
    - Operadores chamam métodos que podem ser sobrescritos pelo programador

Operador	Função correspondente
+	__add__()
-	__sub__()
*	__mul__()
/	__div__()

Polimorfismo

- Sobrecarga:

Script	Saída

class Seminario():
  def __init__(self):
  self.notaInicial = 0.0
  def __add__(self,pontos):
  self.notaInicial = self.notaInicial + pontos
  return self
 	
seminarioPython = Seminario()
print(seminarioPython.notaInicial)
seminarioPython = seminarioPython + 5
print(seminarioPython.notaInicial)
	0.0
	
	5.0

Polimorfismo

- Paramétrico
  - Embutido na linguagem.
  - Qualquer objeto pode ser passado como parâmetro para uma função.
  - Caso o subprograma tente acessar atributos ou métodos que não pertencem ao objeto passado como parâmetro é lançada uma exceção

Polimorfismo

- Inclusão:
  - Python oferece herança simples e múltipla de classes;
  - Não é obrigatório chamar o(s) método(s) construtor(es) da(s) superclasse(s);
  - Simples:
    - Função super() pode ser utilizada

```
class Produto(object):
    def __init__(self,peso,preco):
        self.peso = peso
        self.preco = preco

class Notebook(Produto):
    def __init__(self,peso,preco,tamanhoTela):
        super(Notebook, self).__init__(peso,preco)
        self.tamanhoTela = tamanhoTela

macBook = Notebook(1,6000,13)
```

Polimorfismo

- Inclusão:
  - Herança múltipla:
    - Função super() só chama o construtor da primeira superclasse
    - Construtores devem ser chamados explicitamente

Polimorfismo

- Inclusão:
  - Herança Múltipla:

```
class Produto(object):
    def __init__(self,preco):
        self.preco = preco
    def vende(self):
        print("Produto Vendido!")

class Computador(object):
    def __init__(self,tamanhoTela):
        self.tamanhoTela = tamanhoTela
    def vende(self):
        print("Computador Vendido!")

class Notebook(Produto,Computador):
    def __init__(self,preco,tamanhoTela):
        Produto.__init__(self,preco)
        Computador.__init__(self,tamanhoTela)
```

Polimorfismo

- Inclusão:
  - Problemas na herança múltipla:
    - E as duas definições do método vende()?
    - Notebook().vende() chama o método vende() da primeira superclasse (isso também vale para atributos estáticos)
    - É possível chamar um método ou atributo estático das demais, criando métodos que os “envolvem”;
    - Atributos não-estáticos são definidos de forma diferente;
    - A última definição destes é a que vale (depende da ordem que os construtores são chamados);
  - Method Resolution Order (MRO);
    - “Lineariza” as classes em uma lista, executando o método da classe mais “próxima” primeiro

Polimorfismo

- Classes Abstratas:

```
from abc import ABCMeta, abstractmethod #Modulo para classes abstratas

class Publicacao:
    __metaclass__ = ABCMeta

    def __init__(self, ano = None, numero = None, pagInicial = None, pagFinal = None,
titulo = None, listaDocentes = []):
        self.ano = ano
        self.numero = numero
        self.pagInicial = pagInicial
        self.pagFinal = pagFinal
        self.titulo = titulo
        self.listaDocentes = listaDocentes
        self.__veiculo = None

    @abstractmethod
     def getClass(self):
         pass
```

Polimorfismo

- Classes Abstratas:
  - Necessário utilizar o módulo abc e a metaclasse ABCMeta
  - E ter um método abstrato
  - Não é possível criar um objeto da classe Publicacao

Exceções

- Indicam erros na execução do programa:
- Exceções built-in derivadas de BaseException;
- Exceções do usuário devem ser subclasses de Exception;
- Exemplos:
  - Operação não definida para o tipo (TypeError)
  - Divisão por zero (ZeroDivisionError);
  - Método abstrato não implementado (NotImplementedError);
  - Erro do OS (OSError);
  - Falta de memória (MemoryError)
  - Variável não definida (NameError)

Exceções

- Exceções podem ser lançadas (ou relançadas) com o comando raise;
- Só subclasses de Exception ou BaseException podem ser lançadas;
- Tratadas dentro de blocos try/except

Script (python 2)	Saída

num = 1; den = 0
try:
  div = num/den;
  a = range(99999999999999999999999999999999)
except (NameError,TypeError):
  print("Variável não existe ou tipo errado!!!")
except ZeroDivisionError:
  print("Divisão por zero!!!")
except:
  print("Outro erro qualquer!!!"); raise #Relança o erro
finally:
  print("Que pena!")
	Divisão por zero!!!
	Que pena!

Exceções

- Exceções podem ser lançadas (ou relançadas) com o comando raise;
- Só subclasses de Exception ou BaseException podem ser lançadas;
- Tratadas dentro de blocos try/except

Script (python 2)	Saída

num = 1; denom = 0
try:
  div = num/den;
  a = range(99999999999999999999999999999999)
except (NameError,TypeError):
  print("Variável não existe ou tipo errado!!!")
except ZeroDivisionError:
  print("Divisão por zero!!!")
except:
  print("Outro erro qualquer!!!"); raise #Relança o erro
finally:
  print("Que pena!")
	Variável não existe ou tipo errado!!!
	Que pena!

Exceções

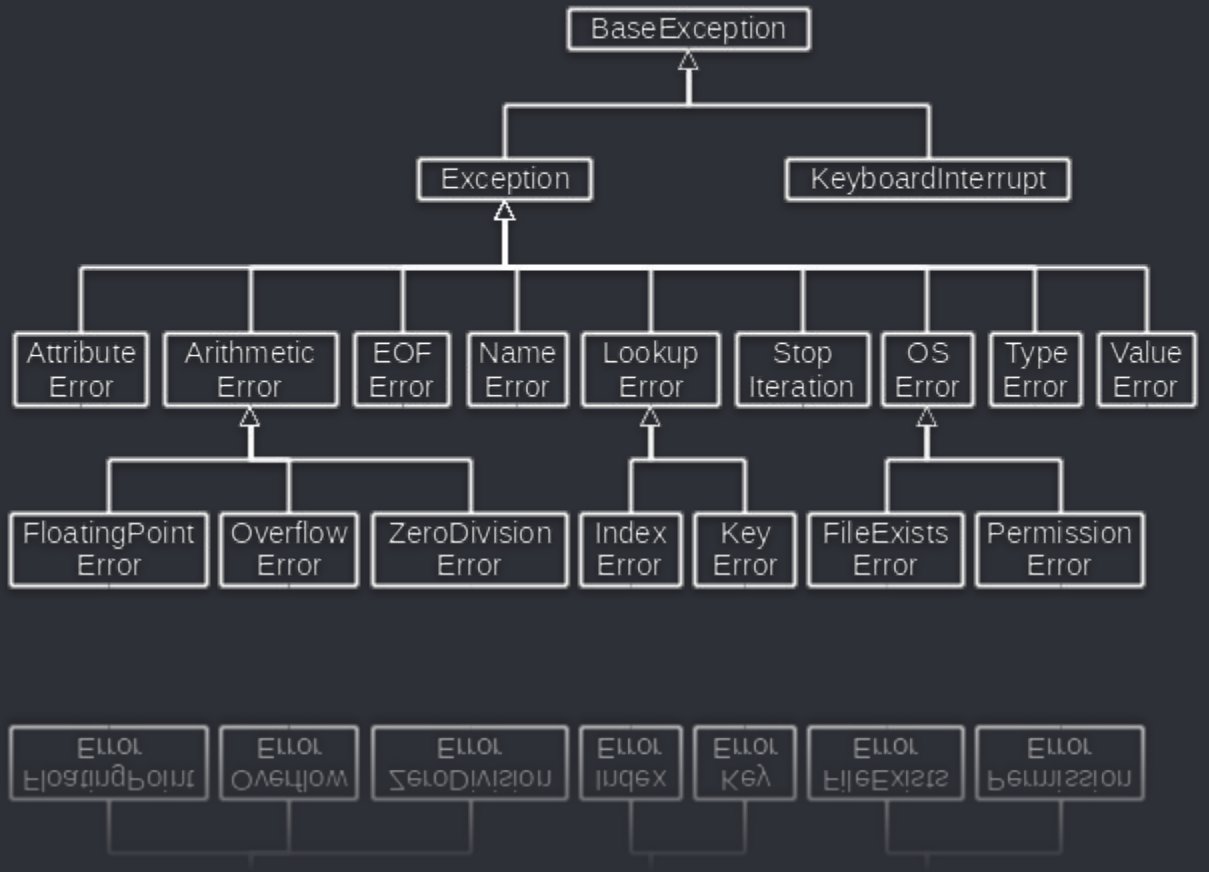
- Exceções podem ser lançadas (ou relançadas) com o comando raise;
- Só subclasses de Exception ou BaseException podem ser lançadas;
- Tratadas dentro de blocos try/except

Script (python 2)	Saída

num = 1; den = 1;
try:
  div = num/den;
  a = range(99999999999999999999999999999999)
except (NameError,TypeError):
  print("Variável não existe ou tipo errado!!!")
except ZeroDivisionError:
  print("Divisão por zero!!!")
except:
  print("Outro erro qualquer!!!"); raise #Relança o erro
finally:
  print("Que pena!")
	Outro erro qualquer!!!
	Que pena!
	
	OverflowError: range()
	result has too many items

Exceções

- Hierarquia de Exceções



Concorrência

- Python suporta programação concorrente através de módulos como:
  - threading
  - multiprocessing
- Threading:
  - classe Thread;
  - Semáforos e Monitores;
- Multiprocessing:
  - Utiliza processos ao invés de Threads;
  - Processos tem memória separada;

Concorrência

- Threading

Script	Saída

import threading
 	
class novaThread(threading.Thread):
  def __init__(self,i):
  threading.Thread.__init__(self)
  self.i = i
  def run(self):
  print(self.i)
 	
thread1 = novaThread("cachorro")
thread2 = novaThread("papagaio")
 	
thread1.start(); thread2.start()
thread1.join(); thread2.join()
	cachorro
	papagaio

Concorrência

- Threading

```
import threading

semaforo = threading.Semaphore()

semaforo.acquire() #lock

#Codigo com exclusão mútua

semaforo.release() #unlock
```

Concorrência

- Threading
  - Objeto Lock para exclusão mútua simples

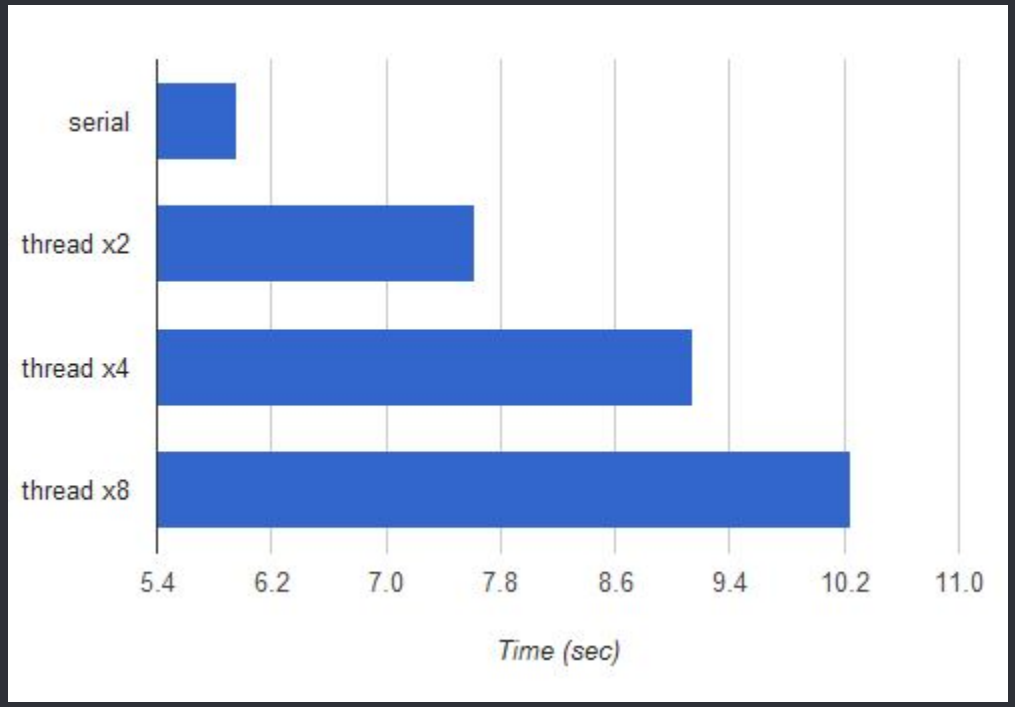
```
import threading

lock = threading.Lock()

with (lock):
    pass #O código só será executado por uma
única thread
            #E o lock será destruído quando o bloco
with acabar
```

Concorrência

- Threading
  - Aplicação: Fatoração de números
  - Fonte: https://eli.thegreenplace.net/2012/01/16/python-parallelizing-cpu-bound-tasks-with-multiprocessing
  - Implementação serial vs 2, 4 e 8 threads:

Concorrência

- Threading
  - Aplicação: Fatoração de números
  - Implementação serial vs 2, 4 e 8 threads:



Concorrência

- Threading
  - Em algumas implementações do interpretador Python, existe um objeto Lock global, que restringe acesso os objetos de Python em múltiplas threads;
    - Necessário em implementações que não são thread-safe
    - CPython não é;
    - Jython é;
  - Global Interpreter Lock (GIL) garante que acessos sejam mutuamente exclusivos;
  - GIL pode impedir que threads utilizem todas as vantagens de um processador com múltiplos núcleos;

Concorrência

- Multiprocessing
  - Utiliza processos ao invés de Threads;
  - Implementação similar, mas ao invés da classe Thread, se usa a classe Process;
  - Vantagens:
    - Usa memória separada;
    - Capaz de aproveitar melhor os múltiplos núcleos;
    - Evita desvantagens do GIL;
    - Processos podem ser interrompidos ou até mortos;
  - Desvantagens:
    - Comunicação entre processos é dificultada
    - Maior uso de memória

Concorrência

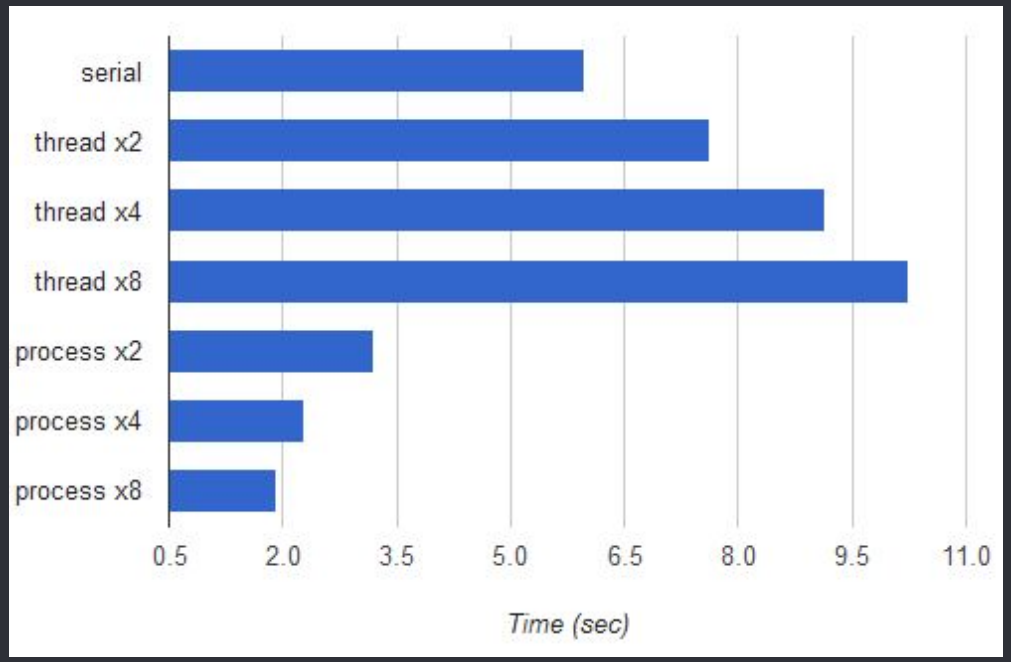
- Multiprocessing
  - Exemplo:

Script	Saída

from multiprocessing import Process, Lock
 	
def f(l, i):
  l.acquire()
  print 'hello world', i
  l.release()
 	
lock = Lock()
 	
for num in range(10):
  Process(target=f, args=(lock, num)).start()
	hello world 0
	hello world 1
	hello world 3
	hello world 2
	hello world 4
	hello world 5
	hello world 6
	hello world 7
	hello world 8
	hello world 9

Concorrência

- Multiprocessing
  - Aplicação: Fatoração de números
  - Implementação serial vs 2, 4 e 8 threads vs 2, 4 e 8 processos:



Avaliação da Linguagem

Critérios	Linguagem C	Linguagem Java	Linguagem Python
Aplicabilidade	Sim	Parcial	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim	Parcial	Parcial

Avaliação da Linguagem

Critérios	Linguagem C	Linguagem Java	Linguagem Python
Portabilidade	Não	Sim	Sim
Método de Projeto	Estruturado	OO	OO, estruturado e funcional
Evolutibilidade	Não	Sim	Sim
Reusabilidade	Parcial	Sim	Sim

Avaliação da Linguagem

Critérios	Linguagem C	Linguagem Java	Linguagem Python
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta
Escopo	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim

Avaliação da Linguagem

Critérios	Linguagem C	Linguagem Java	Linguagem Python
Tipos Primitivos e Compostos	Sim	Sim	Parcial
Gerenciamento de Memória	Programador	Sistema	Programador, Sistema
Persistência dos dados	Biblioteca de funções	JDBC, Biblioteca de classes,serialização	Biblioteca de classes,serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, por valor e por cópia de referência	Lista Variável, valores default, Cópia da Referência

Avaliação da Linguagem

Critérios	Linguagem C	Linguagem Java	Linguagem Python
Encapsulamento	Parcial	Sim	Sim
Sistema de Tipos	Não	Sim	Sim
Verificação de Tipos	Estática	Estática/Dinâmica	Dinâmica
Polimorfismo	Coerção e Sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim

Referências

- https://www.python.org
- https://docs.python.org/3/
- Como pensar como um cientista da Computação usando Python. Disponível em http://eltonminetto.net/docs/pythontut.pdf.
- https://stackoverflow.com/questions/1641219/does-python-have-private-variables-in-classes
- https://julien.danjou.info/blog/2013/guide-python-static-class-abstract-methods
- https://wiki.python.org/moin/GlobalInterpreterLock
- https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python
- https://eli.thegreenplace.net/2012/01/16/python-parallelizing-cpu-bound-tasks-with-multiprocessing