Python

Componentes: Eduarda Coppo, Kaio Rosa, Jhones Gonçalves e Lucas Valentim

Conteúdo

- Introdução;
- Amarrações;
- Identificadores e sistemas de tipos;
- Módulos, funções e classes;
- Comandos interativos;
- Polimorfismo;
- Exceções;
- Concorrência;
- Avaliação de linguagem.

História

- Python é uma linguagem de programação criada por Guido van Rossum em 1991;
- Feita com base na linguagem ABC;
- Intuito de ser uma linguagem interpretada com comandos simples;
- O nome é uma homenagem á Monty Python's Flying Circus;
- Linguagem de propósito geral.

Fontes: http://mindbending.org/pt/a-historia-do-python

Popularidade

Oct 2018	Oct 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.801%	+5.37%
2	2		C	15.376%	+7.00%
3	3		C++	7.593%	+2.59%
4	5	^	Python	7.156%	+3.35%
5	8	^	Visual Basic .NET	5.884%	+3.15%
6	4	v	C#	3.485%	-0.37%
7	7		PHP	2.794%	+0.00%
8	6	v	JavaScript	2.280%	-0.73%
9	-	^^	SQL	2.038%	+2.04%
10	16	^^	Swift	1.500%	-0.17%
11	13	^	MATLAB	1.317%	-0.56%
12	20	^^	Go	1.253%	-0.10%
13	9	vv	Assembly language	1.245%	-1.13%

https://www.tiobe.com/tiobe-index/

Zen of Python

Poema criado por Tim Peters. É possível visualizar dentro da linguagem utilizando

O comando >>>import this

- Bonito é melhor do que feio;
- Simples é melhor do que complexo;
- Legibilidade conta;
- Erros nunca devem passar silenciosamente;
- Casos especiais não são especiais o suficiente para quebrar as regras.

Fontes: https://www.python.org/dev/peps/pep-0020/

Vantagens

- Facilidade;
- Desenvolvimento web;
- Inteligência artificial;
- Computação gráfica;
- Big data;

Características

- Linguagem interpretada;
- Tipagem dinâmica;
- Fortemente tipadas;
- Blocos definidos de acordo com a indentação;
- Multiparadigmal;
- Tudo é um objeto;
- “Baterias inclusas”.

Características

```
def olaMundo(n, lingua):
    if lingua == 'pt':
        a = 'Ola mundo'
    elif lingua == 'en':
        a = 'Hello world'
    i = 0;
    while i < n:
        a += '!';
        i+= 1
    print(a)

olaMundo(4, 'en')
```

Saída:
Hello World!!!!

Características

```
def soma():
    a = 1
    b = '2'
    print(a+b)

soma()

def soma():
    a = str(1)
    b = '2'
    print(a+b)

soma()
```

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

Saída:
12

Características

```
class Pessoa(object):
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def __str__(self):
        return 'nome: ' + self.nome + '\nidade: ' + str(self.idade)

def imprime():
    a = 'João'
    print(a)
    a = Pessoa(a, 15)
    print(a)
imprime()
```

Saída:
João
nome: João
Idade: 15

Escopo

- Existe dois escopos para variáveis em Python: Local e global.

```
def imprime():
    a = 5
    print(a)

a = 1
imprime()
print(a)
```

Saída:
5
1

Escopo

- Existe dois escopos para variáveis em Python: Local e global.

```
def imprime():
    global a
    a = 5
    print(a)

a = 1
imprime()
print(a)
```

Saída:
5
5

Operadores

- Aritméticos:

+ - * / % ** //

- Relacionais:

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

- Atribuição:

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

Palavras reservadas

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Fonte: https://docs.python.org/3/reference/lexical_analysis.html

Caracteres válidos

- Os mesmos no Python 2 e no Python 3
  - Letras maiúsculas e minúsculas de A a Z
  - Underscore _
  - Dígitos de 0 a 9
    - Porém, o nome do identificador não pode começar com um dígito

Tipos de variáveis

- int
  - Não tem tamanho limitado em Python 3. Em Python 2 equivale ao long de C
  - x = 10
- float
  - Equivale ao double de C
  - f = 5.2
- complex
  - complex(4, -2)

Tipos de variáveis

- string
  - Sequência de caracteres Unicode
  - s = “Hello, World 👋”
- bool
  - True ou False
  - 0, objetos vazios e None equivalem a False
  - Todo o resto equivale a True

Objetos mutáveis e imutáveis

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Tipos de variáveis

- Lista
  - lista = [1, “Hello”, -1.2, 5]
- Tupla
  - tupla = (1, “Hello”, -1.2, 5)
- Dicionário
  - dicionario = {‘nome’: Fulano, ‘idade’: 20, ‘altura’: 1.8}
  - dicionario[‘nome’] contém o valor armazenado no campo nome

Diferença entre lista e tupla

- A diferença é que a lista é mutável e a tupla não
- lista.append(9)
  - O inteiro 9 é adicionado ao final da lista
- tupla.append(9)
  - Dá erro
  - TypeError: 'tuple' object does not support item assignment

Tipos de variáveis

- set
  - Conjunto matemático
  - conjunto = set([1, 2, 2, 2, 3])
    - {1, 2, 3}
  - set([1,2,3]) - set([3,4])
    - set([1, 2])
  - set([1,2,3]) & set([3,4])
    - set([3])
  - set([1,2,3]) | set([3,4])
    - set([1, 2, 3, 4])

Funções

- Definida com a palavra chave ‘def’, seguida do nome da função e a lista de parâmetros entre parênteses

```
#Fibonacci numbers module

def fib(n): #write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

```
def fib2(n): #return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Funções

- Direção de passagem de parâmetros unidirecional de entrada variável
- Mecanismo de passagem por referência
- Momento da passagem normal
  - A avaliação do parâmetro real ocorre no momento da chamada da função
- Suporta valores default
  - def exemplo(a, b = 1)
- Correspondência parâmetro real-formal posicional e por palavra-chave
  - Posicional: A sequência em que os parâmetros são escritos representa a correspondência real-formal
  - Palavra-chave: correspondência explícita

Funções

```
def soma(a, b = 1):
    c = a + b
    print(c)

def mult(a, b, c, d):
    m = a * b * c * d
    print(m)

soma(3)
mult(b = 9, c = 5, d = 8, a = 2)
```

Módulos

- É um arquivo em python com definições (funções, classes)
- Este arquivo pode ser importado em outro arquivo com a palavra chave ‘import’
- import modulo
  - importa o módulo todo
- from modulo import imprime
  - Importa apenas a função imprime
- import modulo as m
  - Toda vez que o módulo for usado, ele será referido por m

Módulo - Exemplo

```
#fibo.py

def fib(n): #write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n): #return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Módulos - Exemplo

```
>>> import fibo
```

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Módulos - Exemplo

```
>>> from fibo import fib, fib2
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import *
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> import fibo as fib
>>> fib.fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Fonte: https://docs.python.org/3/tutorial/modules.html

Classes

- Definidas usando a palavra-chave class
- Toda classe herda de object
- Permite herança múltipla

```
class Base1:
    pass

class Base2:
    pass

class Derivada(Base1, Base2):
    pass
```

Comandos

- Objetivo é atualizar variáveis ou controlar o fluxo de controle;
- Podem ser primitivos ou compostos;

Tipo de Comandos

- Atribuições;
- Comandos colaterais;
- Comandos condicionais;
- Comandos iterativos;
- Comandos de desvio incondicional.

Atribuições

- Simples

```
>>> x = 5
>>> x
5
>>> x = 'Some string'
>>> print x
Some string
```

-

Multipla

```
>>> i = j = errorCount = 0
>>> i
0
>>> j
0
>>> errorCount
0
```

- Composta

```
>>> i = 1
>>> i += 3
>>> i
4
>>> i *= 5
>>> i
20
```

Comandos Colaterais

- Comandos que permitem processamento paralelo;

Nao existem em python

Comandos Condicionais

```
idade = 18
if idade < 16:
   print("Nao pode votar")
elif  idade >= 16 and idade < 18:
print("Voto facultativo")
else:
   print("Voto Obrigatorio")
```

Comandos Iterativos

```
# For Loop
amigos= [ "John", "Tom", "Micaela", "Mary" ]

for nome in amigos:
    print(amigos)
```

```
# For Loop
for i in range(1,10):
    print(i)
```

```
# While Loop
i = 1
while i < 6:
  print(i)
   i += 1
```

break - “quebra” a execução do loop e executa a próxima linha fora do loop

continue - Vai para a próxima iteração do loop.

pass - quando pass é chamado não faz nada. Quando nenhum código é necessário mas é requerido sintaticamente. (Null operator)

Comandos De Desvios Incondicionais

Nao possui goto

Polimorfismo

Tipos de Polimorfismo

- Ad-hoc :
  - Coerção
  - Sobrecarga
- Universal:
  - Paramétrico
  - Inclusão

Coerção

- Não faz a coerção em atribuições

  Ex.:

  num_int = 123
  num_str = "456"

  num_str = int(num_str)

- Em operações, não faz a coerção implícita quando strings e tipos numéricos estão envolvidos:

Sobrecarga

Não suporta method overloading

Operadores em Python trabalham com built-in classes

Funções Especiais em Python

- Começam com double undercore __method__

- O __init__() por exemplo é chamado toda vez que um novo objeto é criado

Operator	Special Method	Description
+	__add__(self, object)	Addition
-	__sub__(self, object)	Subtraction
*	__mul__(self, object)	Multiplication
**	__pow__(self, object)	Exponentiation
/	__truediv__(self, object)	Division
//	__floordiv__(self, object)	Integer Division
%	__mod__(self, object)	Modulus

==	__eq__(self, object)	Equal to
!=	__ne__(self, object)	Not equal to
>	__gt__(self, object)	Greater than
>=	__ge__(self, object)	Greater than or equal to
<	__lt__(self, object)	Less than
<=	__le__(self, object)	Less than or equal to
in	__contains__(self, value)	Membership operator
[index]	__getitem__(self, index)	Item at index
len()	__len__(self)	Calculate number of items
str()	__str__(self)	Convert object to a string

Parametrico

- Embutido na linguagem.
- Qualquer objeto pode ser passado como parâmetro para uma função.
- Lança exceção caso seja tentado acessar método ou atributo que não pertence ao objeto

Inclusão

- Herança Simples

```
class Triangle(Polygon):
    def __init__(self):
        Polygon.__init__(self,3)

    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is %0.2f'
%area)
```

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in
range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter
side "+str(i+1)+" : ")) for i in
range(self.n)]

    def dispSides(self):
        for i in range(self.n):

print("Side",i+1,"is",self.sides[i])
```

```
>>> t = Triangle()

>>> t.inputSides()
Enter side 1 : 3
Enter side 2 : 5
Enter side 3 : 4

>>> t.dispSides()
Side 1 is 3.0
Side 2 is 5.0
Side 3 is 4.0

>>> t.findArea()
The area of the triangle is 6.00
```

- Herança Multipla

```
class First(object):
  def __init__(self):
    super(First, self).__init__()
    print("first")

class Second(object):
  def __init__(self):
    super(Second, self).__init__()
    print("second")

class Third(Second, First):
  def __init__(self):
    super(Third, self).__init__()
    print("third")

Third();
```

Saida

first
second
third

Exceções

- O Tratamento de Exceções não é obrigatório.
- Exceções não tratados resultam em uma mensagem de erro.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Exceções

- Todas as Exceções são subclasses de BaseException. (A partir do Python 3)
- É encorajado que exceções definidas pelo usuário sejam descendentes de Exception.

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
```

```
      +-- BufferError
      +-- EOFError
      +-- MemoryError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      +-- ReferenceError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
```

Manipulando Exceções

- Semelhante ao que Java faz, com except no lugar de catch.
- Utiliza o Modo de Terminação para o Tratamento de Exceções.

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try again...")
...
```

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Manipulando Exceções

- Pode-se omitir os nomes das Exceções.

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

Manipulando Exceções

- Pode-se vincular uma variável a instância da exceção.

```
>>> try:
...     raise Exception('spam', 'eggs')
... except Exception as inst:
...     print(type(inst))    # the exception instance
...     print(inst.args)     # arguments stored in .args
...     print(inst)          # __str__ allows args to be printed directly,
...                          # but may be overridden in exception subclasses
...     x, y = inst.args     # unpack args
...     print('x =', x)
...     print('y =', y)
...
```

SAÍDA:

```
<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

Manipulando Exceções - else

- Pode-se utilizar opcionalmente a cláusula else.

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

Manipulando Exceções - finally

- Pode-se utilizar opcionalmente a cláusula finally.
- Não se pode utilizar uma declaração continue dentro da cláusula finally.

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
```

SAÍDA:

```
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Lançando Exceções

- Pode-se forçar uma exceção com uma instrução raise. É como o throw em C++.

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

Encadeamento de Exceções

- A cláusula from permite encadear exceções. (A partir do Python 3)

```
>>> try:
...     print(1 / 0)
... except Exception as exc:
...     raise RuntimeError("Something bad happened") from exc
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ZeroDivisionError: division by zero

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: Something bad happened
```

Encadeamento de Exceções

- O encadeamento pode ser suprimido passando None para from.

```
>>> try:
...     print(1 / 0)
... except:
...     raise RuntimeError("Something bad happened") from None
...
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: Something bad happened
```

Encadeamento de Exceções

- Encadeamento Implícito, lançada dentro de finally ou do tratador da exceção.

```
>>> try:
...     print(1 / 0)
... except:
...     raise RuntimeError("Something bad happened")
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ZeroDivisionError: division by zero

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: Something bad happened
```

Novas Exceções

- Programas podem criar suas próprias Exceções com novas classes.

```
class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class InputError(Error):
    """Exception raised for errors in the input.

    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """

    def __init__(self, expression, message):
        self.expression = expression
        self.message = message
```

Concorrência - Threads

- Implementadas a partir do módulo threading.
- A partir do Python 3.7 está sempre disponivel.
- Oferece uma implementação de alto nível, em contraste a _thread.
- Cria-se subclases da classe Thread.
- Substitui o método run.
- Inicia-se a thread com o método start.
- Global Interpreter Lock (GIL). (CPhython)
- Lock em nível do Interpretador.

Concorrência - Threads

```
import threading
import time
import random

def worker(number):
    sleep = random.randrange(1, 10)
    time.sleep(sleep)
    print("I am Worker {}, I slept for {} seconds".format(number, sleep))

for i in range(5):
    t = threading.Thread(target=worker, args=(i,))
    t.start()

print("All Threads are queued, let's see when they finish!")
```

SAÍDA:

```
All Threads are queued, let's see when they finish!
I am Worker 1, I slept for 1 seconds
I am Worker 3, I slept for 4 seconds
I am Worker 4, I slept for 5 seconds
I am Worker 2, I slept for 7 seconds
I am Worker 0, I slept for 9 seconds
```

Concorrência - Threads - Sincronização

- A declaração with permite a criação de trechos ‘Thread Safe’.
- Chama o método acquire quando entra no bloco, e realese quando sai.
- Funciona com: Locks, Conditions e Semaphores.
- Módulo **queue**.

```
with some_lock:
    # do something...
```

```
# Consume one item
with cv:
    while not an_item_is_available():
        cv.wait()
    get_an_available_item()

# Produce one item
with cv:
    make_an_item_available()
    cv.notify()
```

Concorrência - Multiprocessamento

- Módulo multiprocessing.
- Semelhante ao threading, porém contorna o GIL utilizando **subprocessos**.
- Novas API's como os objetos Pool.
- Processos são criados a partir de de objetos Process.
- Contextos de Iniciação: **spaw**, **fork** e **forkserver**. set_start_method() ou get_context().

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

```
import multiprocessing as mp

def foo(q):
    q.put('hello')

if __name__ == '__main__':
    ctx = mp.get_context('spawn')
    q = ctx.Queue()
    p = ctx.Process(target=foo, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

Concorrência - Multiprocessamento

- Suporta 2 tipos de canais de comunicação de processos: **Pipes** e **Queues.**
- Contém formas de sincronização equivalentes a todas existentes em threading.

```
from multiprocessing import Process, Lock

def f(l, n):
    l.acquire()
    try:
        print('hello world', n)
    finally:
        l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        Process(target=f, args=(lock, num)).start()
```

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim	Parcial	Parcial
Portabilidade	Não	Sim	Sim
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sir		
Portabilidade	Nã		
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Python é uma linguagem de propósito geral, ou seja, consegue ser aplicado nos diversos ramos da computação

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim		
Portabilidade	Não		
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Possui verificação de tipos e tratamento de exceções

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sir		
Portabilidade	Nã		
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Python possui indentação obrigatória e diversas outras facilidades, criando uma curva de aprendizado alta

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim	Parcial	Parcial
Portabilidade	Nã		
Método de projeto	Est		
Evolutibilidade	Não	Sim	Sim

Não é totalmente eficiente pois é uma linguagem fortemente tipada, logo verificações de tipos são constantes

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	**Sim**	**Sim**	**Sim**
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim		
Portabilidade	Não	Sim	Sim
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Possui interpretador próprio, permitindo a portabilidade para diversos sistemas

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim		
Portabilidade	Não		
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Python é uma linguagem multiparadigmal, permitindo escolher o que será usado para situações específicas

Avaliação de linguagem

Critérios gerais	C	Java	Python
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Sim
Eficiência	Sim	Parcial	Parcial
Portabilidade	Nã		
Método de projeto	Estruturado	OO	Estruturado, OO, Funcional
Evolutibilidade	Não	Sim	Sim

Python recebe atualizações constantes, atualmente está na versão 3.7.1

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Depende da ferramenta	Parcial
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Possui funções e sistema de classes	
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta		
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Pode invocar código compilado de outras linguagens

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Depende da ferramenta	Parcial
Escopo	S		
Expressões e comandos	S		
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Python é uma linguagem gratuita, porém projetos com ela podem se tornar caras pois existem poucos profissionais na área

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Depende da ferramenta	Parcial
Escopo	Sim	Sim	Sim
Expressões e comandos	S		
Tipos primitivos e compostos	S		
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Possui escopo bem definido, obrigando ao programador a explicitar o escopo de entidades em certos casos

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	D f		
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Oferece uma variedade de expressões, comandos e funções

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Depende da ferramenta	Parcial
Escopo	S		
Expressões e comandos	S		
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Sistema	Programador, sistema

Apesar de tudo ser objeto, possui todos os tipos primitivos, inclusive booleano

Avaliação de linguagem

Critérios gerais	C	Java	Python
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim
Custo	Depende da ferramenta	Depende da ferramenta	Parcial
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	S		
Gerenciamento de memória	Programador	Sistema	Sistema

Python, através de seu interpretador, possui coletor de lixo assim como Java

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e	Lista variável e cópia
Encapsulamento	Parcial		
Sistemas 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

Inclui operações de entrada e saída na linguagem

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial	Si	Si
Sistemas de tipos	Não		
Verificação de tipos	Estática		
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim

Partindo do princípio em que tudo em Python é um objeto, os parâmetros são cópias para referências

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial	Sim	Sim
Sistemas de tipos	Não		
Verificação de tipos	Estática		
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim

Atributos e métodos podem ser públicos ou privados

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial	Sim	Sim
Sistemas de tipos	Não	Sim	Sim
Verificação de tipos	Estática		
Polimorfismo	Coerção e sol		
Exceções	Não		
Concorrência	Não	Sim	Sim

Possui sistema de tipos bastante rigoroso, pois é uma linguagem fortemente tipada

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial		
Sistemas de tipos	Não		
Verificação de tipos	Estática	Estática/Dinâmica	Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim

Todas as variáveis são alocadas dinamicamente, sendo desnecessário a definição dos tipos das variáveis

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial	Si	Si
Sistemas de tipos	Não		
Verificação de tipos	Estática		
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim

Possui todos os polimorfismos e permite sobre escrita de operadores, apesar de não ser recomendado pela linguagem

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial	Sim	Sim
Sistemas de tipos	Não	Sim	Sim
Verificação de tipos	Estática		
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim

Oferece sistema para tratamento de exceções

Avaliação de linguagem

Critérios gerais	C	Java	Python
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de Parâmetros	Lista variável e valor	Lista variável, valor e cópia de referência	Cópia de referência
Encapsulamento	Parcial	Sim	Sim
Sistemas de tipos	Não	Sim	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Dinâmica
Polimorfismo	Coerção e sol		
Exceções	Não		
Concorrência	Não	Sim	Sim

Oferece threads e sincronização na API básica

Referencias

https://docs.python.org/3/reference/
https://inf.ufes.br/~vitorsouza/wp-content/uploads/academia-br-lp-slides05-expressoes.pdf
https://inf.ufes.br/~vitorsouza/wp-content/uploads/academia-br-lp-slides07-polimorfismo.pdf
https://data-flair.training/blogs/python-inheritance/
https://www.programiz.com/python-programming/inheritance
http://mindbending.org/pt/a-historia-do-python
https://www.python.org/dev/peps/pep-0020/
https://www.javatpoint.com/multiple-inheritance-in-python