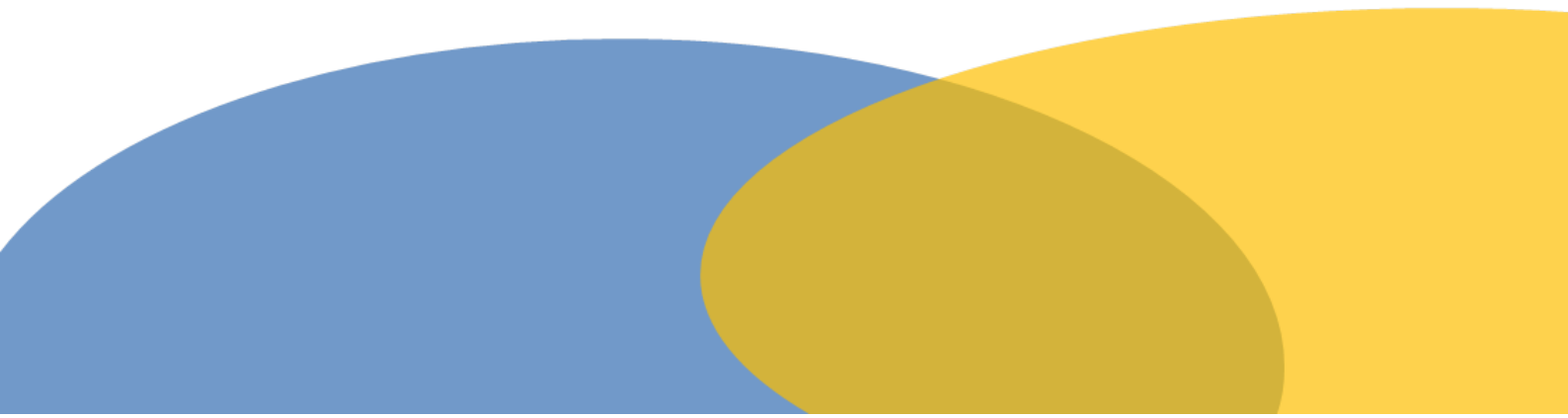
Introdução à Linguagem Python

Paradigmas de Linguagens de Programação

Alysson de Jesus Alcantara Alves
Ausberto S. Castro Vera

19 de setembro de 2021

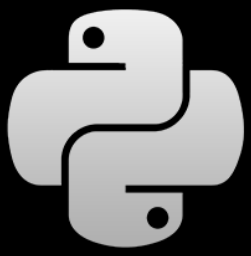


UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO

CCT - CENTRO DE CIÊNCIA E TECNOLOGIA
LCMAT - LABORATÓRIO DE MATEMÁTICAS
CC - CURSO DE CIÊNCIA DA COMPUTAÇÃO

Primeira edição, Maio 2019

Sumário



```
print “Hello World”
```

1. Introdução

> "As pessoas ainda são loucas pelo Python mesmo depois de vinte e cinco anos, o que é difícil de acreditar"—Michael Palin

Concebido originalmente no final dos anos 80, por por Guido van Rossum no Centrum Wiskunde & Informatica (CWI) na Holanda, veio como sucessor da linguagem ABC, a implementação se deu inicio em dezembro de 1989. Python foi lançado como sendo uma linguagem de propósito geral de alto nível, multiparadigma, suportando paradigmas orientados a objetos, imperativos, funcionais e procedurais, com uma tipagem dinâmica e de fácil leitura, característica por exigir poucas linhas se comparado a outras linguagens, por conta disso Python é tido como uma das linguagens mais populares da atualidade, considerado a terceira linguagem "mais amada"de acordo com uma pesquisa conduzida pelo site do Stack Overflow em 2018 e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk

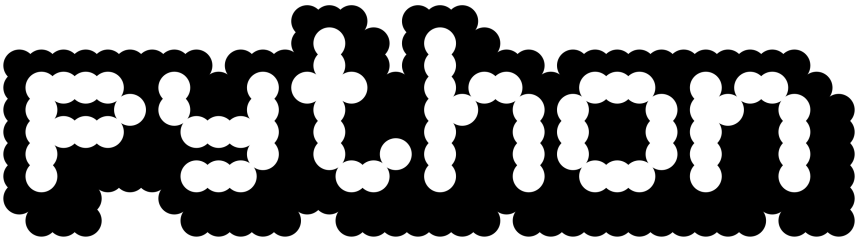
O nome Python teve a sua origem no grupo humorístico britânico Monty Python,[8] criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome (em português, píton ou pitão).

> Em fevereiro de 1991, Van Rossum publicou o código (rotulado como versão 0.9.0) no grupo de discussão da alt.sources. Nessa versão já estavam presentes classes com herança, tratamento de exceções, funções e os tipos de dado nativos list, dict, str, e assim por diante. Também estava presente nessa versão um sistema de módulos emprestado do Modula-3. O modelo de exceções também lembrava muito o do Modula-3, com a adição da opção else clause. Em 1994 foi formado o principal fórum de discussão do Python, comp.lang.python, um marco para o crescimento da base de usuários da linguagem. [Ven03]

1.1 Aspectos históricos da linguagem Python

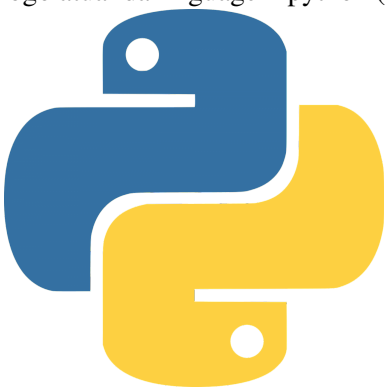
Python chegou a versão 1.0 em janeiro de 1994. As principais novas funcionalidades incluídas nesta versão foram as ferramentas de programação funcional lambda, map, filtere reduce. Van Rossum afirmou que "Python adquiriu lambda, reduz (), filtro () e mapa (), cortesia de um hacker Lisp que os perdeu e enviou patches de trabalho ". [vvR05] A última versão lançada enquanto Van Rossum estava no CWI foi o Python 1.2. Em 1995, Van Rossum continuou seu trabalho em Python na Corporation for National Research Initiatives (CNRI) em Reston , Virgínia , de onde lançou várias versões. Abaixo pode-se ver a evolução das logos do Python

Figura 1.1: Primeira logo da linguagem Python (1990s–2006)



Fonte: domínio público

Figura 1.2: Logo atual da linguagem python (2006 - atual)



Fonte: community logo python

1.2 Áreas de Aplicação da Linguagem

Por ter uma sintaxe simples e ser baseado em c, Python é uma linguagem que pode ser facilmente escalável e usada em diversas áreas diferentes.

Python pode servir como uma linguagem de script para aplicativos da web, por exemplo, via mod_wsgi para o servidor da web Apache Com a Web Server Gateway Interface, uma API padrão evoluiu para facilitar essas aplicações. Frameworks da Web como Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle e Zope apoiam os desenvolvedores no design e manutenção de aplicações complexas. Pyjs e IronPythonpode ser usado para desenvolver o lado do cliente de aplicativos baseados em Ajax. SQLAlchemy pode ser usado como um mapeador de dados para um banco de dados relacional. Twisted é um framework para programar comunicações entre computadores e é usado (por exemplo) pelo Dropbox [Res15]

1.2.1 Big Data

Python, possui bibliotecas voltada diretamente Big Data, como a Scikit-learn, que oferece funcionalidades para o pré-processamento, a classificação e a clusterização dos dados, a seleção de modelos e a aplicação de algoritmos de regressão. Das várias bibliotecas que o python possui para realizar cálculo de Big Data podemos citar a Imbalanced-learn, Jupyter, Pandas e Scikit-learn. No artigo de Ivan Eduardo o mesmo realiza uma série de aplicações de Big Data utilizando cada uma das bibliotecas [Kü15]

1.2.2 Orientação a objetos

[Woi03] A programação orientada a objetos (OOP) é um paradigma de programação baseado no conceito de "objetos ", que podem conter dados e código: dados na forma de campos (geralmente conhecidos como atributos ou propriedades), e código, na forma de procedimentos (freqüentemente conhecido como métodos).

Em Python, todo valor é na verdade um objeto. Seja uma tartaruga, uma lista, ou mesmo um inteiro, todos são objetos. Programas manipulam esses objetos realizando computações diretamente com eles ou chamando os seus métodos (ou seja, pedindo que esses objetos executem seus métodos). Para ser mais específico, nós dizemos que um objeto possui um estado e uma coleção de métodos que ele pode executar. O estado de um objeto representa as coisas que o objeto sabe sobre si mesmo. Por exemplo, como vimos com os objetos tartaruga, cada tartaruga possui um estado que representa a sua posição, sua cor, sua direção, etc. Cada tartaruga também tem a capacidade de se mover para a frente, para trás, ou virar para a direita ou esquerda. Cada tartaruga é diferente pois, embora sejam todas tartarugas, cada uma tem um estado diferente (como posições diferentes, ou orientações, etc).

Sabendo como eles funcionam podemos ver um exemplo prático na linguagem python criando um objeto ponto que possui o eixo x e o eixo y

Figura 1.3: Código de OOP em python, criando um ponto com dois eixos

```
1  class Point:
2      """ Point class for representing and manipulating x,y coordinates. """
3
4      def __init__(self):
5          """ Create a new point at the origin """
6          self.x = 0
7          self.y = 0
```

Fonte: Autor

1.2.3 Data Science

[Gru16] Pode-se dizer que um cientista de dados, são para propósitos práticos, estatísticos que entendem sobre ciência da computação, alguns são mais estatísticos e outros são mais engenheiros de software, mas ambos os perfis são cientistas de dados. A Ciência de dados serve para extrai todos os dados que estão bagunçados dentro da sociedade, e transforma-los em conhecimento, que servem como guia para melhorar nossa sociedade.

No exemplo abaixo é criado um histograma para agrupar dados unificados em agrupamentos (buckets) discretos e contar quantos pontos vão para cada um, no exemplo esses dados podem representar varias situações como a media diária em minutos que cada usuário passa no seu site.

Trecho de código em python para definir o histograma

```
def bucketize(point, bucket_size):
"""reduza o ponto para o pr ximo"""
"""m ltiplo mais baixo de bucket_size"""
return
    bucket_size * math.floor(point / bucket_size)

def make_histogram(points, bucket_size):
"""agrupa os pontos e conta quantos em cada bucket"""
return
    Counter(bucketize(point, bucket_size) for point in points)

def plot_histogram(points, bucket_size, title=""):
    histogram = make_histogram(points, bucket_size)
    plt.bar
     (histogram.keys(), histogram.values(), width=bucket_size)
    plt.title(title)
    plt.show()
```

Agora consideramos os seguintes conjuntos de dados

```
random.seed(0)
# uniforme entre   100   e 100

uniform = [200 * random.random() - 100 for _ in range(10000)]
# distribui  o normal com m dia 0, desvio padr o 57

normal = [57 * inverse_normal_cdf(random.random())
for _ in range(10000)]
```

Ambos possuem médias próximas a 0 e desvios padrões próximos a 58. No entanto, possuem distribuições bem diferentes. A Figura 10-1 mostra a distribuição de uniform:

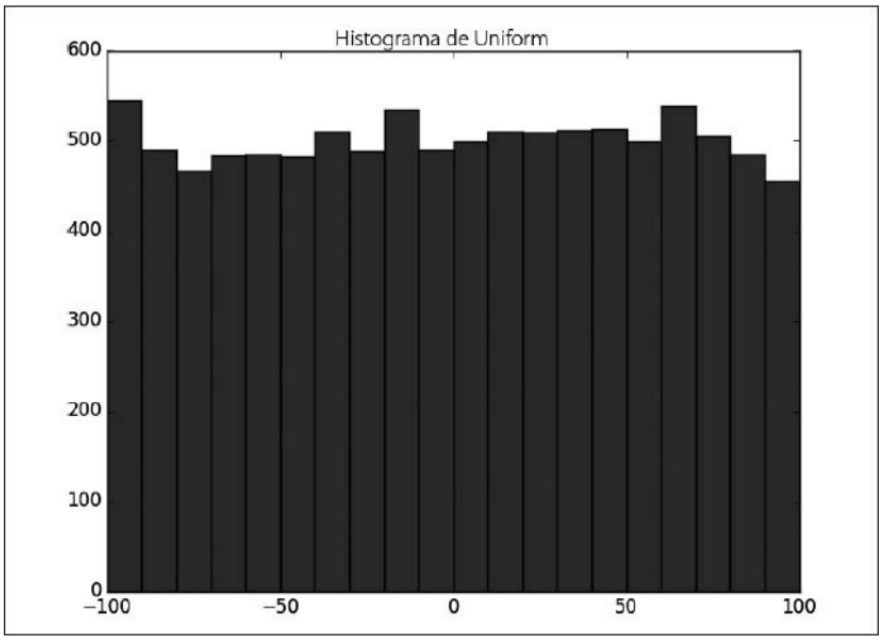
```
plot_histogram(uniform, 10, "Histograma de Uniform")
```

Já o trecho abaixo mostra a distribuição normal

```
plot_histogram(normal, 10, "Histograma Normal")
```

Nesse exemplo as duas possuem valores max e min que podem ser vistos no gráfico abaixo:

Figura 1.4: Histograma uniforme



Fonte: Livro - Data Science do zero: Primeiras regras com o Python

2. Conceitos básicos da Linguagem Python

A linguagem Python é extremamente versátil e cada vez mais usada não só na academia, mas também na indústria.Fundamentalmente falando, ela tem três características básicas.

Como o Python é uma linguagem interpretada, isso significa que funciona em um ambiente onde o interpretador interativo avalia cada expressão no contexto definido, com o efeito de calcular o resultado (ou mensagem de erro) e, finalmente, atualizar o contexto. Essa classificação parece ser o oposto das chamadas linguagens compiladas (como C, C ++ ou Fortran), onde o compilador analisa todo o programa e, se nenhum erro for detectado, gera um arquivo executável utilizável.

2.1 Variáveis e constantes

O conceito de variáveis em Python é sempre representado por um objeto (tudo é um objeto) e cada variável é uma referência. Na maioria das linguagens de programação, quando inicializamos uma variável e atribuímos um valor a ela, ela carrega o valor alocado na memória.Quando alteramos seu valor, também alteramos o valor na memória. No entanto, em Python, as variáveis armazenam endereços de memória em vez de valores.

O Código abaixo mostra a declaração de variaveis usando Python

```
>>> #criando uma variavel x
>>> x = [1,2,3]
>>> y = x
>>> # o m todo append() adiciona um elemento ao vetor x
>>> x.append(4)
>>> print(y)

[1, 2, 3, 4]
```

2.2 Tipos de Dados Básicos

Em Python, todo valor (objeto) pertence a um certo tipo (classe). Este tipo determina quais operações podem ser aplicadas ao valor (objeto). Portanto, é uma linguagem digitada, mas os tipos não são explícitos, o que torna o trabalho do usuário mais fácil (embora possa causar outros problemas na depuração de erros downstream). Os tipos básicos mais úteis disponíveis são: inteiro (int), número real (float) e valor lógico (bool). Os tipos básicos são imutáveis.

```
>>> type(1)
    int

>>> type(1.1)
    float

>>> type('1')
    str

>>> type(1==1)
    bool
```

O tipo bool disponibiliza as constantes True e False, bem como diversos operadores lógicos (AND, NOT, OR). O Python tem ainda diversos operadores de comparação que devolvem valores lógicos: <, <=, >, >=, ==, !=, is, is not, isinstance.

2.2.1 Números

A sintaxe de expressões do Python é usual: os operadores +, -, * e / funcionam da mesma forma que em outras linguagens tradicionais (por exemplo, Pascal ou C); parênteses (()) podem ser usados para agrupar expressões, uma divisão sempre vai retornar valores flutuantes. Por exemplo:

```
>>> 1 + 2
    3

>>> 50 - 5*6
    20

>>> 8 / 5
    1.6
```

2.2.2 String

Uma string é uma sequência de caracteres é uma sequência de caracteres tratada como um único item de dados. Para Python, uma string é uma matriz de caracteres ou qualquer grupo de caracteres escritos entre aspas duplas ou simples. Por exemplo

```
>>> x = 'Python'
>>> print (x)  Python
```

2.3 Estruturas de dados

Tipos de dados de coleção são junções de vários tipos de dados diferentes criando uma estrutura complexa que pode ser usada para servir em diversas ocasiões diferentes, as principais estruturas dentro do Python são Listas de dados, Sets de dados, dicionários, e Tuplas uma estrutura peculiar

do Python similar a uma lista.

2.3.1 Tipos Sequenciais

Em Python uma lista ou list é uma sequência de valores ou coleção ordenadas, onde cada valor se identifica por um índice começando a partir do 0. Esses valores são chamados de elementos ou itens no python, onde cada item pode ter tipos de dados diferente

```
 # como declarar uma lista
 # com tipos diferentes em python

>>> nome_da_lista = [2, "Ola", 3.1]
```

2.3.2 Tipos Conjunto

Um tipo conjunto é uma coleção de vários tipos de dados diferentes, similar a um vetor de string, porém pode receber diversos tipos de dados. Sets são uma coleção de itens desordenada, que é parcialmente imutável e não podem conter elementos duplicados. Em Python os tipos conjuntos são representados pelo tipo Sets de dados Como parte de serem parcialmente imutáveis os sets possuem funções de adição e remoção de elementos No exemplo abaixo, vamos declarar um set, e manipular os elementos de dentro dele

```
meu_set = {1, 2, 3, 4, 1}

# Adicionando elementos
meu_set.add(2)

# Atualizando set
meu_set.update([3, 4, 5, 6])

# Removendo elemento
meu_set.discard(2)
```

2.3.3 Tipos Mapeamento

Um mapa é uma coleção associativa desordenada. Um tipo mapeamento é composto por uma série de chaves e valores correspondentes, em Python o único tipo mapeamento nativo da linguagem é o dicionário, eles ajudam a implementar tipos de dados abstratos. No exemplo vamos criar e acessar um dict, pessoa, e atribuir a pessoa uma série de características

```
>>> pessoa = {
    'nome': 'Pythonico',
    'altura': 1.65,
    'idade': 21
    }
>>> print(pessoa['nome'])

Pythonico
```

2.4 Estrutura de Controle e Funções

As estruturas de controle servem para decidir os blocos de códigos que serão executados a partir de determinadas validações lógicas na linguagem. Para o Python é importante ressaltar que dentro de estruturas de controle a indentação do código é fundamental para que não ocorra erros na estrutura

2.4.1 O comando IF

Uma instrução else pode ser combinada com uma instrução if . Uma instrução else contém o bloco de código que é executado se a expressão condicional na instrução if for resolvida para 0 ou um valor FALSE.

A instrução else é uma instrução opcional e pode haver no máximo apenas uma instrução else após if .

A sintaxe básica para a estrutura condicional if é a seguinte

```
if expression:
   statement(s)
else:
   statement(s)
```

2.4.2 Laço FOR

Estruturas de repetição são extremamente uteis em varias situações, elas fazem com que um bloco de códigos seja executado por varias vezes, ate que uma condição se satisfaça, uma das mais utilizadas no Python e na programação em geral, é o FOR loop, o for assim como outros loops como while, também executa um bloco de código ate que determinada condição se satisfaça, porém com a vantagem de que não precisamos declarar uma variavel de contador para incrementar dentro do for, o que torna a escria mais simples e prática

A sintaxe do for é bem simples, ela começa com a chamada do loop for, em seguida a variavel que iremos iterar pelo loop, e por fim a condição final ate onde o loop vai, mostro com mais detalhes nos exemplos abaixo:

```
cidades = ['Rio de Janeiro', 'Curitiba', 'Salvador']

for cidade in cidades:
    print(cidade)
print("fim do for loop")

>>> Rio de Janeiro
>>> Curitiba
>>> Salvador
>>> fim do for loop
```

Pode parecer um pouco confuso, principalmente para quem vem de for's tradicionais como no caso da linguagem c, mas basicamente o que o Python quer dizer é que estamos usando uma variavel de iteração chamada cidade, onde cidade representa cada um dos items no array de cidades que declaramos acima, no python o for sabe exatamente quando parar, ele para assim que cidade percorre todos os elementos de cidades.

Mas e quando precisamos dizer com mais especificidade quando queremos que um loop acaba em Python ?
Para isso usamos os while loops, explicarei eles em seguida

2.4.3 Laço WHILE

Nos loops while, diferente do for, nós precisamos de uma condição para que aquele loop exista, e ele existira enquanto aquela condição for verdadeira. No exemplo vamos utilizar uma variavel nome que, enquanto ela for verdadeira o loop for ira existir

```
nome = input('insira seu nome: ')

while nome:
    input('insira um nome: ')

>>> insira um nome: Luiz
>>> insira um nome: Larissa
>>> insira um nome: Lara
>>> insira um nome: Lucas
>>> ........
```

No exemplo acabamos gerando um loop infinito dentro do while, um detalhe que se deve tomar muito cuidado, como a variavel nome, sempre sera entendida como válida pelo Python, o while nunca vai deixar de executar, gerando um loop infinito e comportamentos indesejados, por isso é preciso tomar cuidado ao usar while loop's

2.5 Módulos e pacotes

Trabalhar com módulos e pacotes, é algo fundamental para todo desenvolvedor Python, além de ser uma atividade bem commum durante o desenvolvimento, mas antes precisamos entender que para o Python, todos os scripts são entendidos como um módulo, e que um pacote, é basicamente um conjunto de módulos, ou seja, um conjunto de script's python

2.5.1 Módulos

Todo script de Python é entendido como um módulo, como diz a própria documentação oficial do Python.

> "Um módulo é um arquivo Python contendo definições e instruções. O nome do arquivo é o módulo com o sufixo .py adicionado. Dentro de um módulo, o nome do módulo (como uma string) está disponível na variável global"
>
> ```
> __name__
> ```

É importante lembrar que módulos podem possuir outros módulos importados dentro deles

2.5.2 Pacotes

Em dado momento, vamos começar a ter uma aplicação em Python com muitos módulos para serem gerenciados ao mesmo tempo, isso pode ocasionar além de conflitos de arquivos uma confusão para os desenvolvedores se encontrarem, por isso Python tem um recurso de pacotes, onde você pode empacotar uma série de módulos dentro de um único pacote, e pode utilizar o pacote em outros arquivos python, assim variáveis e arquivos que possuem nome similares não darão mais conflitos com o python, além de ficar tudo bem organizado para os desenvolvedores

3. Programação Orientada a Objetos com Python

3.1 Classes e Objetos

```
class NomeClasse:

def metodo1:

def Metodo2:
```

3.2 Operadores ou Métodos

3.3 Herança

3.4 Estudo de Caso:

4. Aplicações da Linguagem Python

Devem ser mostradas pelo menos CINCO aplicações completas da linguagem, e em cada caso deve ser apresentado:

- Uma breve descrição da aplicação
- O código completo da aplicação,
- Imagens do código fonte no compilador-interpretador,
- Imagens dos resultados após a compilação-interpretação do código fonte
- Links e referencias bibliográficas de onde foi obtido a aplicação

4.1 Operações básicas

4.2 Programas gráficos

4.3 Programas com Objetos

4.4 O algoritmo Quicksort - Implementação

4.5 Aplicações com Banco de Dados

5. Ferramentas existentes e utilizadas

Neste capítulo devem ser apresentadas pelo menos DUAS (e no máximo 5) ferramentas consultadas e utilizadas para realizar o trabalho, e usar nas aplicações. Considere em cada caso:

- Nome da ferramenta (compilador-interpretador)
- Endereço na Internet
- Versão atual e utilizada
- Descrição simples (máx 2 parágrafos)
- Telas capturadas da ferramenta
- Outras informações

5.1 Editor MNOP

5.2 Compilador XYZ

5.3 Interpretador UVW

5.4 Ambientes de Programação IDE MNP

6. Conclusões

Os problemas enfrentados neste trabalho ...

O trabalho que foi desenvolvido em forma resumida ...

Aspectos não considerados que poderiam ser estudados ou úteis para ...

Figura 6.1: Linguagens de programação modernas

Fonte: O autor

Referências Bibliográficas

[Gru16] Joel Grus. *Data Science do zero: Primeiras regras com o Python*. Alta Books, Rua Viúva Cláudio, 291 - Bairro Industrial do Jacaré, edição revista e ampliada edition, 2016. Citado na página 8.

[Kü15] Ivan Eduardo Metz Kühne. Aplicação de técnicas de big data analytics às smart grids como forma de descoberta de padrões relevantes. *Universidade Regional do Noroeste do Estado do Rio Grande do Sul*, 2015. Citado na página 7.

[Res15] Google Research. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *White Paper*, 2015. Citado na página 7.

[Ven03] Bill Venners. The making of python a conversation with guido van rossum, part i. *Commun. ACM*, 2003. Citado na página 5.

[vvR05] Guido van van Rossum. The fate of reduce() in python 3000. *Commun. ACM*, 2005. Citado na página 6.

[Woi03] Josué Labaki & E.R Woiski. Python orientado a objetos. *Grupo Python, UNESP-Ilha Solteira*, 2003. Citado na página 7.

Disciplina: *Paradigmas de Linguagens de Programação 2019*
Linguagem: *Linguagem Python*
Aluno: *Nome Completo do aluno*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	**Pontos**
Elementos básicos da linguagem (Máximo: 01 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Usos e áreas de Aplicação da Linguagem	
Cada elemento da linguagem (definição) com exemplos (Máximo: 02 pontos) • Exemplos com fonte diferenciada (Courier , 10 pts, azul)	
Mínimo 5 exemplos completos - Aplicações (Máximo : 2 pontos) • Uso de rotinas-funções-procedimentos, E/S formatadas • Menu de operações, programas gráficos, matrizes, aplicações	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 2 pontos) • Ferramentas utilizadas nos exemplos: pelo menos DUAS • Descrição de Ferramentas existentes: máximo 5 • Mostrar as telas dos exemplos junto ao compilador-interpretador • Mostrar as telas dos resultados obtidos nas ferramentas • Descrição das ferramentas (autor, versão, homepage, tipo, etc.)	
Organização do trabalho (Máximo: 01 ponto) • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia	
Uso de Bibliografia (Máximo: 01 ponto) • Livros: pelo menos 3 • Artigos científicos: pelo menos 3 (IEEE Xplore, ACM Library) • Todas as Referências dentro do texto, tipo [ABC 04] • Evite Referências da Internet	
Conceito do Professor (Opcional: 01 ponto)	
Nota Final do trabalho:	

Observação: Requisitos mínimos significa a *metade* dos pontos