

Linguagem de Programação Python

Carolina de Lima Silva¹, Mateus Gontijo¹

¹Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte - MG - Brasil

carolina.silva.1026582@sga.pucminas.br, mateus.gontijo.1266715@sga.pucminas.br

Abstract. *The Python language was created in 1991 and since then it has become increasingly popular. The history and development of Python from its creation to the present day will be presented in this work, addressing its main characteristics. It will also show success cases using it and its amazing libraries and frameworks.*

Resumo. *A linguagem Python foi criada em 1991 e desde então vem se tornando cada vez mais popular. Será apresentado neste trabalho a história e desenvolvimento do Python desde sua criação até os dias atuais, sendo abordado suas principais características. Também será mostrado casos de sucesso com a utilização dela e de suas incríveis bibliotecas e frameworks.*

1. Introdução

Python é uma poderosa linguagem de programação de uso geral. É usado em desenvolvimento web, ciência de dados, criação de protótipos de software e assim por diante. Felizmente para iniciantes, o *Python* tem uma sintaxe simples e fácil de usar. Isso torna o *Python* uma excelente linguagem para aprender a programar para iniciantes [LPP 2022].

Atualmente, o *Python* é a linguagem de programação multifuncional e de alto nível mais usada. Ele permite programar nos paradigmas Orientado a Objetos e Procedimental. Programas Python geralmente são menores do que outras linguagens de programação, como Java.

Os programadores têm que digitar relativamente menos quando programam nesta linguagem e o requisito de recuo da mesma os torna legíveis o tempo todo. *Python* está sendo usado por quase todas as empresas gigantes da tecnologia como – *Google, Amazon, Facebook, Instagram, Dropbox, Uber*, entre outros.

Este artigo tem como objetivo apresentar a história e desenvolvimento do *Python*, sendo abordado suas principais características.

2. História

2.1. Criação

Python foi concebido na década de 1980 por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI). Ele é carinhosamente conhecido como O Ditador Benevolente de Python para a Vida.

O nome *Python*, como muitos podem pensar à primeira vista, não possui relação direta com a espécie de serpentes, mas sim com o seriado *Monty Python's Flying Circus*,

na qual o criador da linguagem era um grande fã. Segundo o próprio autor o *Python* foi escolhido como provisório para o projeto, por estar com um humor um pouco irreverente e por ser um grande fã do Circo Voador de Monty Python.

2.2. Desenvolvimento

A linguagem foi feita com base na linguagem ABC, que é uma linguagem de programação da CWI utilizada para propósitos gerais que pode ser usada no lugar de *Basic*, *Pascal* ou *AWK*. Seu principal foco era aumentar a produtividade do programador [Wikiwand].

Em 1991, Guido van Rossum publicou o código (versão 0.9.0) em um grupo de discussão da internet. 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 da cláusula else.

Em 1994 foi formado o principal fórum de discussão do *Python*, *CompLangPython*, um marco para o crescimento da base de usuários da linguagem. A versão 1.0 foi lançada em janeiro deste ano. Novas funcionalidades incluíam ferramentas para programação funcional como lambda, map, filter e reduce.

Em 2000, *Python 2.0* implementou a list comprehension, uma relevante funcionalidade de linguagens funcionais como SETL e Haskell. A sintaxe da linguagem para essa construção é bastante similar a de Haskell, exceto pela preferência do Haskell por caracteres de pontuação e da preferência do *Python* por palavras reservadas alfabéticas. Essa versão 2.0 também introduziu um sistema coletor de lixo capaz de identificar e tratar ciclos de referências.

A terceira versão da linguagem foi lançada em dezembro de 2008, chamada *Python 3.0* ou *Python 3000*. Com noticiado desde antes de seu lançamento, houve quebra de compatibilidade com a família 2.x para corrigir falhas que foram descobertas neste padrão, e para limpar os excessos das versões anteriores. A primeira versão alfa foi lançada em 31 de agosto de 2007, a segunda em 7 de dezembro do mesmo ano.

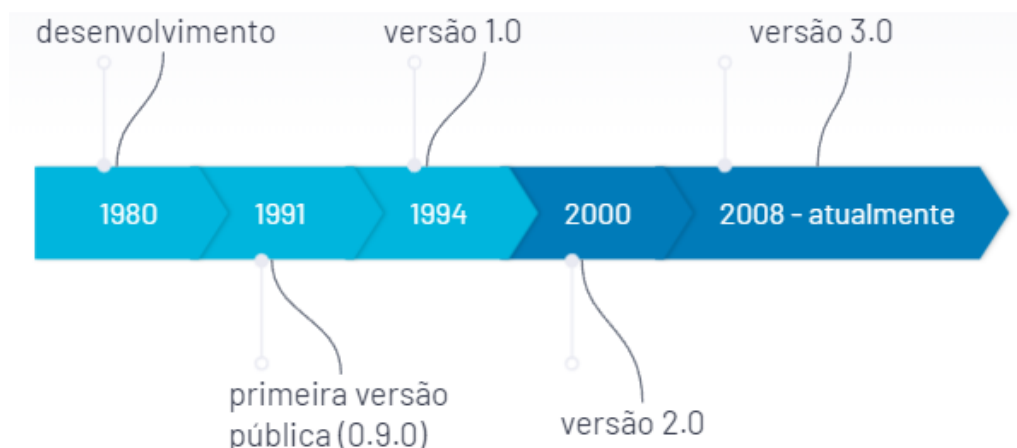


Figura 1. Timeline das versões Python

Uma das várias mudanças ocorreu na função print, que anteriormente era tido

como uma declaração em que não era necessário a utilização de parênteses para sua utilização.

```
Python 3.10 (64-bit)
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Olá, mundo!'
File "<stdin>", line 1
    print 'Olá, mundo!'
    ^^^^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
>>>
```

Figura 2. Função print Python 2

```
Python 3.10 (64-bit)
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Olá, mundo!')
Olá, mundo!
>>> _
```

Figura 3. Função print Python 3

Outra mudança que vale ser ressaltada pode ser observada na conversão automática de tipo que passou a existir na divisão por inteiros, que na última versão realiza a conversão de tipo inteiro para float, para um resultado mais preciso.

```
Python 3.10 (64-bit)
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 4/2
2.0
>>> _
```

Figura 4. Divisão por inteiros Python 3

2.3. Filosofia

Parte da cultura da linguagem gira ao redor de *The Zen of Python*, que é uma coleção de 19 princípios orientadores, na forma de poema, com uma série de aforismos, para escrever programas de computador que influenciam o design da linguagem de programação *Python*.

Este poema foi escrito pelo programador Tim Peters, que o publicou na lista de discussões do *Python* em 1999. A lista de Peters deixou em aberto o vigésimo princípio “para Guido preencher”, referindo-se ao autor original da linguagem.

O *The Zen of Python* foi incluído como entrada nas propostas oficiais de aprimoramento do *Python* (PEP), lançadas em domínio público. Ele também está incluído como um easter egg no interpretador *Python*, que pode ser exibido digitando:

```
>>> import this
```

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *right* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!
```

Figura 5. The Zen of Python

3. Características da Linguagem

Python foi criada para ser uma linguagem simples e de fácil aprendizado e mantém essa característica até os dias atuais e apresenta tipagem forte e dinâmica, além de ser multi-plataforma.

Possui indentação como parte funcional do código, sendo ela essencial para definição de blocos e escopos, característica implementada para maior uniformidade e legibilidade dos códigos escritos por diferentes programadores. Também possui inúmeros frameworks e bibliotecas de terceiros que permitem que o *Python* se apresente de forma ainda mais incrível.

4. Paradigmas da Linguagem

É uma linguagem multiparadigma fazendo com que ela seja flexível em diversas áreas de atuação na computação. Apresenta diversos tipos de dados implementados nativamente fazendo com que exista diversas possibilidades de aplicação. *Python* suporta quatro tipos de paradigmas de programação:

1. Paradigma de programação orientada a objetos
2. Paradigma de programação procedural
3. Paradigma de programação funcional
4. Paradigma de programação imperativo

4.1. Paradigma de programação orientada a objetos

O paradigma orientado a objetos tem como principal característica a utilização de estruturas no código para representação de objetos e é o mais presente no Python. Todos os dados no programa são representados por objetos ou por relações entre objetos. Nele são utilizadas as classes, assim como em muitas outras linguagens, porém apresenta algumas

diferenças. Não é possível a diferenciação de métodos públicos, privados e protegidos pela própria linguagem, mas a comunidade utiliza como convenção dois underscores antes do nome das propriedades e métodos considerados internos na classe. Após a declaração da classe é possível instanciar objetos daquela classe, como podemos ver no exemplo a seguir:

```
1 class Cliente:
2     def __init__(self, nome, email, plano):
3         self.nome = nome
4         self.email = email
5         lista_planos = ["basic", "premium"]
6         if plano in lista_planos:
7             self.plano = plano
8         else:
9             raise Exception("Plano Invalido")
10
11 cliente = Cliente("Lira", "lira@gmail.com", "blabla")
12 print(cliente.nome)
```

4.2. Paradigma de programação procedural

O paradigma de programação procedural é um subtipo de programação imperativa em que as instruções são estruturadas em procedimentos (também conhecidos como sub-rotinas ou funções). A composição do programa é mais uma chamada de procedimento em que os programas podem residir em algum lugar do universo e a execução é sequencial, tornando-se assim um gargalo para a utilização de recursos. A programação procedural segue o modelo stateful. O paradigma de programação procedural facilita a prática de um bom projeto de programa e permite que os módulos sejam reutilizados na forma de bibliotecas de código. No trecho de código abaixo pode-se ver um exemplo em que é declarada uma função stringify que recebe como parâmetro uma lista de caracteres e retorna a junção deles:

```
1 def stringify ( caracteres ) :
2     string = ''
3     for c in caracteres:
4         string = string + c
5     return string
6
7 caracteres = [ 'P', 'y', 't', 'h', 'o', 'n' ]
8 string_final = stringify(caracteres)
9 print(string_final)
```

4.3. Paradigma de programação funcional

O paradigma de programação funcional trata a computação de programas como a avaliação de funções matemáticas baseadas em cálculo lambda. O cálculo lambda é um sistema formal em lógica matemática para expressar computação com base na abstração e aplicação de funções usando associação e substituição de variáveis. Segue o “o que fazer solve” – ou seja, expressa a lógica sem descrever seu fluxo de controle – por isso também é classificado como o modelo de programação declarativa.

Esse paradigma promove funções sem estado, mas é importante notar que a implementação de programação funcional do Python se desvia da implementação padrão. Diz-se que Python é uma linguagem funcional impura porque é possível manter o estado e criar efeitos colaterais se você não for cuidadoso. Dito isto, a programação funcional é útil para processamento paralelo e é supereficiente para tarefas que exigem recursão e execução simultânea.

No *Python*, o paradigma funcional é implementado nas funções anônimas `lambda`, e nos métodos `map` e `reduce`. A implementação de programação funcional reduz as linhas de código e simplesmente faz seu trabalho em uma única linha, com exceção de usar o módulo `functools` e o método `reduce`. As três palavras-chave — `functools`, `reduce` e `lambda` — são definidas da seguinte forma:

- `functools`: é um módulo para funções de ordem superior e fornece funções que atuam ou retornam outras funções. Ele incentiva a escrita de código reutilizável, pois é mais fácil replicar funções existentes com alguns argumentos já passados e criar uma nova versão de uma função de maneira bem documentada.
- `reduce`: é um método que aplica uma função de dois argumentos cumulativamente aos itens em sequência, da esquerda para a direita, para reduzir a sequência a um único valor. Por exemplo:

```
1 import functools
2
3 numeros = [1, 2, 3, 4, 5]
4 potencia = lambda x, y = 2: x ** y
5 lista = list(map(potencia, numeros))
6 print(lista)
7
8 # Soma de valores
9 lista = [1, 2, 3, 4, 5]
10 soma = functools.reduce(lambda x, y: x + y, lista)
11 print(soma)
```

As funções `lambda` são funções pequenas e anônimas (ou seja, sem nome) que podem receber qualquer número de argumentos, mas devolvem apenas um valor. Eles são úteis quando são usados como argumento para outra função ou residem dentro de outra função; portanto, eles devem ser usados apenas em uma instância por vez.

4.4. Paradigma de programação imperativo

O paradigma de programação imperativo usa o modo imperativo da linguagem natural para expressar direções. Ele executa comandos passo a passo, assim como uma série de comandos verbais.

Seguindo a abordagem “como resolver”, ele faz alterações diretas no estado do programa; portanto, também é chamado de modelo de programação com estado. Usando o paradigma de programação imperativa, você pode escrever rapidamente um código muito simples, mas elegante, e é super útil para tarefas que envolvem manipulação de dados. Devido à sua estratégia de execução comparativamente mais lenta e sequencial, não pode ser usado para cálculos complexos ou paralelos.

No exemplo abaixo, onde o objetivo é pegar uma lista de caracteres e concatená-la para formar uma string, uma maneira de fazer isso em um estilo de programação imperativo seria algo como:

```
1>>> exemplo = ['p','y','t','h','o','n']
2>>> exemplo_string = ''
3>>> exemplo_string
4''
5>>> exemplo_string = exemplo_string + exemplo[0]
6>>> exemplo_string
7'p'
8>>> sample_string = exemplo_string + exemplo[1]
9>>> exemplo_string
10'py'
11>>> sample_string = exemplo_string + exemplo[2]
12>>> exemplo_string
13'pyt'
14>>> sample_string = exemplo_string + exemplo[3]
15>>> exemplo_string
16'pyth'
17>>> sample_string = exemplo_string + exemplo[4]
18>>> exemplo_string
19'pytho'
20>>> sample_string = exemplo_string + exemplo[5]
21>>> exemplo_string
22'python'
23>>>
```

A variável exemplo_string também é como um estado do programa que está sendo alterado após a execução da série de comandos e pode ser facilmente extraída para acompanhar o progresso do programa. O mesmo pode ser feito usando um loop for (também considerado programação imperativa) em uma versão mais curta do código acima:

```
1>>> exemplo = ['p','y','t','h','o','n']
2>>> exemplo_string = ''
3>>> exemplo_string
4>>> for c in exemplo:
5...     exemplo_string = exemplo_string + c
6...     print(exemplo_string)
7...
8p
9py
10pyt
11pyth
12pytho
13python
14>>>
```

5. Tipos de Dado

Todo valor em *Python* tem um tipo de dado. Como tudo é um objeto na programação *Python*, os tipos de dados são na verdade classes e as variáveis são instâncias (objetos) dessas classes.

Existem vários tipos de dados em Python. Alguns dos tipos importantes estão listados abaixo.

Tipo	Descrição	Exemplo da sintaxe
bool	Booleano	True ou 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
list	Lista heterogênea mutável	[4.0, 'string', True]
tuple	Tupla imutável	(4.0, 'string', True)
range	Sequência de números imutável que pode ser transformada em lista	range(10) range(0, 10) range(0, 10, 1)
set, frozenset	Conjunto não ordenado, não contém elementos duplicados	{4.0, 'string', True} frozenset([4.0, 'string', True])
str, unicode	Uma cadeia de caracteres imutável	'Wikipedia' u'Wikipedia'
bytes, bytearray, memoryview	Sequência binária	b'Wikipedia' bytearray(b'Wikipedia') memoryview(b'Wikipedia')
dict	Conjunto associativo	{'key1': 1.0, 'key2': False}

5.1. Dados mutáveis

A linguagem divide seus tipos em dois grandes grupos, os mutáveis e os imutáveis. Os dados mutáveis são aqueles que após serem instanciados podem ter os seus valores alterados pelo programa. Esses dados são: bytearray, list, dict e set.

Bytearray é um array de bytes com range de 0 a 255 representados em hexadecimal. Pode ser utilizada uma string para seu construtor para criação do array.

```
1 array = bytearray(10)
2
3 array = bytearray("Alguma string qualquer", 'utf-8')
```

O tipo list é um array de objetos indexados que possui diversos métodos como: append para adicionar novos itens, count para contar e retornar a quantidade de ocorrências do valor passado como parâmetro, index para buscar valor passado como parâmetro e retornar seu índice ou erro caso o valor não seja encontrado, remove que recebe um valor como parâmetro e busca na lista para então removê-lo, além de vários outros métodos. Um simples exemplo de lista:


```
lista = [1, 2, 3, 4, 5]
```

O tipo dict representa um dicionário e é representado através de chave-valor em cada uma de suas posições. Como por exemplo:

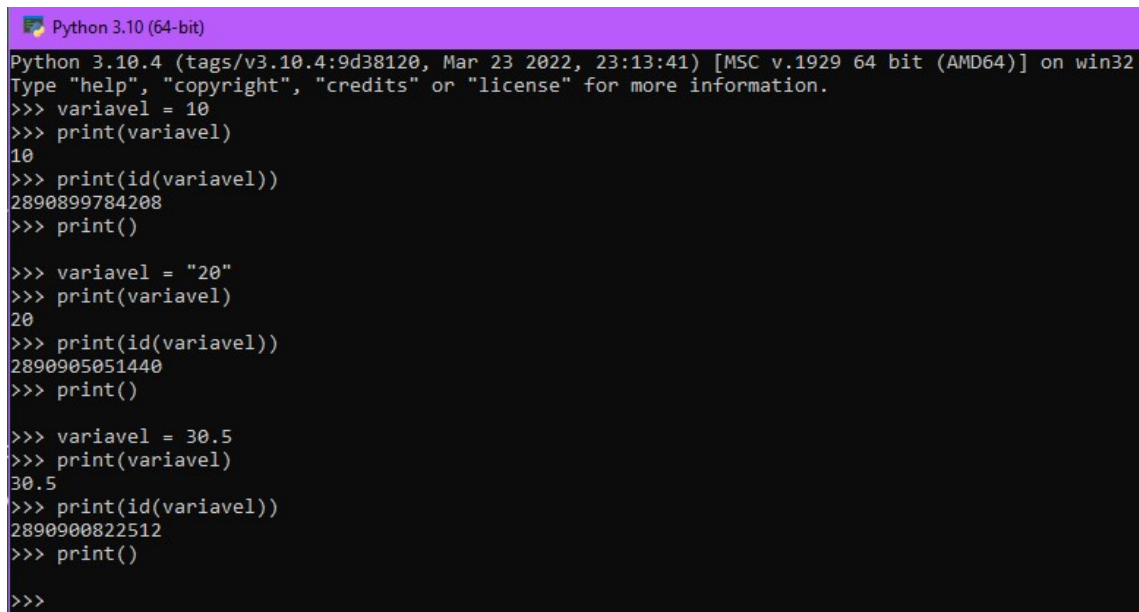
```
diccionario = {'a': 1, 'b': 2, 'c': 3, 'array': [1, 2, 3, 4, 5],  
              'lista': {'a': 1, 'b': 2, 'c': 3}}
```

O tipo set é um conjunto de dados similar a lista mas não possui índice e não pode possuir valores iguais em um mesmo conjunto. Os itens que podem estar contidos no conjunto são: int, float, str, tuple, e NoneType. Exemplo:

```
conjunto = {1, 2, 3, 4, 5, 6, "python", "programacao"}
```

5.2. Dados imutáveis

Os dados imutáveis, diferente dos dados mutáveis, não podem ter seus valores alterados. Apesar disso, as variáveis que possuem algum tipo qualquer podem ser declaradas novamente com um outro valor e tipo, mudando a referência que essa variável faz à memória. Podemos observar essa referência através da utilização da função interna do *Python* chamada *id*, que retorna a posição da memória que a variável referencia. No exemplo a seguir o valor da variável não foi alterado e sim sua referência.



```
Python 3.10 (64-bit)  
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> variavel = 10  
>>> print(variavel)  
10  
>>> print(id(variavel))  
2890899784208  
>>> print()  
  
>>> variavel = "20"  
>>> print(variavel)  
20  
>>> print(id(variavel))  
2890905051440  
>>> print()  
  
>>> variavel = 30.5  
>>> print(variavel)  
30.5  
>>> print(id(variavel))  
2890900822512  
>>> print()  
>>>
```

Figura 6. Referência em Python

Os dados imutáveis são: int, float, complex, bool, str, bytes, frozenset, range, NoneType, e tuple.

O tipo bool é um subtipo da classe int e possui dois possíveis valores: True e False.

O tipo str representa as strings em Python que possuem diversos métodos como upper para retornar todas as letras em maiúsculo, lower para retornar todas as letras em minúsculo, title para retornar a string com toda primeira letra de cada palavra em maiúsculo e muitos outros.

O tipo bytes é similar ao bytearray com a diferença que ele é imutável. Assim como acontece com o byte e o bytearray, o frozenset possui comportamento similar ao set, tendo como princípio a transformação de objetos iteráveis quaisquer em imutáveis.

O tipo range é utilizado para criar conjunto de valores inteiros especificando o início (opcional), o fim (obrigatório) e o passo (opcional).

O tipo NoneType, como o próprio nome já sugere, refere ao tipo do objeto None, que significa vazio, nulo. Não é possível instanciar novos objetos do tipo NoneType, apenas atribuir o valor None a variáveis.

O último tipo de tipo presente no Python que será abordado é a tupla. Tupla é um conjunto de valores semelhante ao tipo list. Por ser imutável, a tupla normalmente é utilizada para relacionar diferentes tipos de dados em uma mesma variável, como por exemplo a sigla e o nome de um estado.

6. Métodos

Na programação orientada a objetos, quando projetamos uma classe, usamos os três métodos a seguir:

1. O método de instância executa um conjunto de ações nos dados/valor fornecidos pelas variáveis de instância.
2. O método de classe é o método que é chamado na própria classe, não em uma instância de objeto específica. Portanto, ele pertence a um nível de classe e todas as instâncias de classe compartilham um método de classe.
3. O método estático é um método utilitário geral que executa uma tarefa isoladamente. Este método não tem acesso à variável de instância e classe.

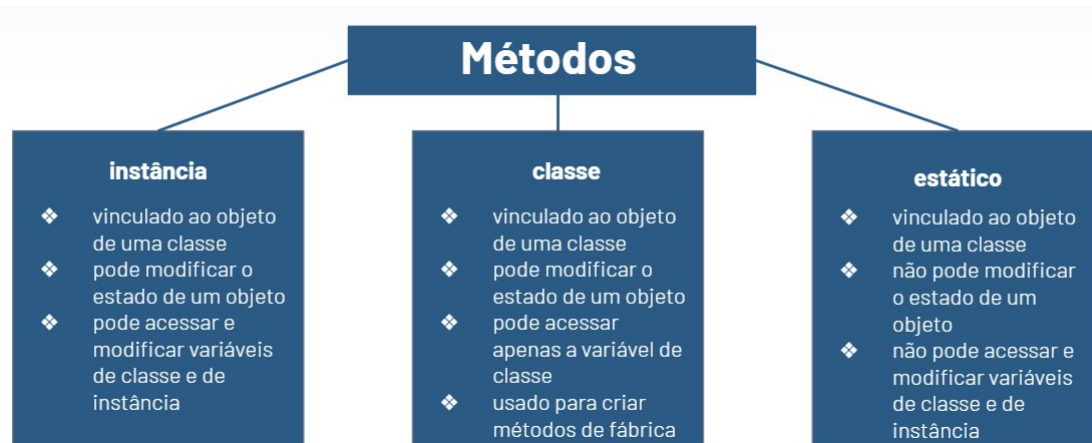


Figura 7. Método de instância X Método de classe X Método estático

6.1. Diferença número 1: Uso Primário

Método de classe pode modificar o estado da classe alterando o valor de uma variável de classe que se aplicaria a todos os objetos de classe. Os métodos de classe são usados como um método de fábrica. Métodos de fábrica são aqueles métodos que retornam um objeto de classe para diferentes casos de uso. Por exemplo, você precisa fazer algum pré-processamento nos dados fornecidos antes de criar um objeto.

Método de instância atua nos atributos de um objeto. Ele pode modificar o estado do objeto alterando o valor das variáveis de instância.

Método estático tem uso limitado porque não tem acesso aos atributos de um objeto (variáveis de instância) e atributos de classe (variáveis de classe). No entanto, eles podem ser úteis em questões como a conversão de um tipo para outro.

6.2. Diferença número 2: Definição do Método

Todos os três métodos são definidos dentro de uma classe e é bastante semelhante à definição de uma função regular.

Qualquer método que criarmos em uma classe será criado automaticamente como um método de instância. Devemos dizer explicitamente ao *Python* que é um método de classe ou método estático.

Usa-se o `@classmethoddecorador` ou a função `classmethod()` para definir o método de classe.

Usa-se o `@staticmethoddecorador` ou a função `staticmethod()` para definir um método estático.

Exemplo:

- Usa-se *self* como o primeiro parâmetro no método de instância ao defini-lo. O parâmetro *self* se refere ao objeto atual.
- Por outro lado, usa-se *cls* como o primeiro parâmetro no método de classe ao defini-lo. O *cls* refere-se à classe.
- Um método estático não usa instância ou classe como parâmetro porque não tem acesso às variáveis de instância e variáveis de classe.

O trecho de código abaixo deixa mais clara a ideia da utilização:

```
1 class Aluno:
2     # variaveis de classe
3     instituicao = 'PUC Minas'
4
5     # constructor
6     def __init__(self, name, age):
7         # variaveis de instancia
8         self.nome = name
9         self.idade = idade
10
11    # variaveis de instancia
12    def show(self):
13        print(self.nome, self.idade, Aluno.instituicao)
14
15    @classmethod
16    def troca_inst(cls, nome):
17        cls.inst_nome = nome
18
19    @staticmethod
20    def materia(materia_nome):
21        return ['LP ', 'Compila ', 'CG']
```

6.3. Diferença número 3: Chamada de Método

Métodos de classe e métodos estáticos podem ser chamados usando `ClassName` ou usando um objeto de classe.

O método `Instance` pode ser chamado apenas usando o objeto da classe.

Exemplo:

```
1 # cria um objeto
2 carol = aluna('Carol', 24)
3 # chama o instance method
4 carol.show()
5
6 # chama class method usando a classe
7 Aluno.troca_inst('UFMG')
8
9 # chama class method usando o objeto
10 carol.troca_inst('UFOP')
11
12 # chama static method usando a classe
13 Aluno.materia('LP')
14
15 # chama class method usando o objeto
16 carol.materia('CG')
```

6.4. Diferença número 4: Acesso ao Atributo

Tanto a classe quanto o objeto têm atributos. Atributos de classe incluem variáveis de classe e atributos de objeto incluem variáveis de instância.

- O método de instância pode acessar atributos de nível de classe e de objeto. Portanto, pode modificar o estado do objeto.
- Os métodos de classe só podem acessar atributos de nível de classe. Portanto, pode modificar o estado da classe.
- Um método estático não tem acesso ao atributo de classe e aos atributos de instância. Portanto, ele não pode modificar a classe ou o estado do objeto.

6.5. Diferença número 5: Limite de Classe e Limite de Instância

- Um método de instância é vinculado ao objeto, para que possamos acessá-lo usando o objeto da classe.
- Métodos de classe e métodos estáticos são vinculados à classe. Portanto, devemos acessá-los usando o nome da classe .

7. Interoperabilidade

Um outro ponto forte da linguagem é sua capacidade de interoperar com várias outras linguagens, principalmente código nativo. A documentação da linguagem inclui exemplos de como usar a Python C-API para escrever funções em C que podem ser chamadas diretamente de código Python - mas atualmente esse sequer é o modo mais indicado de interoperação, havendo alternativas tais como Cython, Swig ou cffi. A biblioteca Boost do C++ inclui uma biblioteca para permitir a interoperabilidade entre as duas linguagens, e pacotes científicos fazem uso de bibliotecas de alta performance numérica escritos em Fortran e mantidos há décadas.

8. Aplicações

Como dito ao longo deste artigo, *Python* é uma das linguagens de programação mais populares por ser completa e fácil de aprender. Uma consequência disso é uma vasta comunidade, que disponibiliza bibliotecas *Python* para diversas áreas da tecnologia.

A principal área é a data science (ciência de dados). A ciência de dados tem ganhado destaque, pois é uma ferramenta poderosa para as organizações, auxiliando na coleta de informações e na tomada de decisões estratégicas.

Então, vamos mostrar as principais bibliotecas Python que ajudam os cientistas de dados a executar seu trabalho. São elas:

- **OpenCV:** Open Source Computer Vision, é um pacote para processamento de imagens. Ele monitora funções gerais focadas na visão instantânea do computador. Embora o OpenCV não possua documentação adequada, de acordo com muitos desenvolvedores, é uma das bibliotecas mais difíceis de aprender. No entanto, ele fornece muitas funções embutidas através das quais você aprende a visão computacional facilmente.
- **Pandas:** Pandas é um pacote de software Python. É necessário aprender sobre ciência de dados e escrito exclusivamente para a linguagem Python. É uma plataforma rápida, demonstrativa e ajustável que oferece estruturas de dados intuitivas. Você pode manipular facilmente qualquer tipo de dado, como dados estruturados ou de séries temporais com este pacote incrível.
- **Scikit Learn:** O Scikit learn é uma biblioteca de aprendizado de máquina Python simples e útil. Está escrito em Python, Cython, C e C ++. No entanto, a maior parte é escrita na linguagem de programação Python. É uma biblioteca gratuita de aprendizado de máquina. É um pacote Python flexível que pode funcionar em completa harmonia com outras bibliotecas e pacotes python, como Numpy e Scipy.
- **TensorFlow:** O TensorFlow é uma biblioteca para Machine Learning de código aberto e gratuita. É muito fácil de aprender e possui uma coleção de ferramentas úteis. No entanto, não se limita apenas ao aprendizado de máquina; você também pode usá-lo para fluxo de dados e programas diferenciáveis. Você pode trabalhar facilmente com o TensorFlow instalando os Colab Notebooks em qualquer navegador.
- **Keras:** Pessoas que querem aprender redes neurais profundas, o Keras pode ser uma boa escolha para eles. Keras é uma biblioteca de rede neural profunda de código aberto. Está escrito em Python. Keras fornece uma política de inspeção eficaz em redes detalhadas. Os desenvolvedores que trabalham com Keras ficam impressionados com sua estrutura modular e fácil de usar.
- **Matplotlib:** O Matplotlib é uma biblioteca Python que usa o Python Script para escrever gráficos e plotagens bidimensionais. Frequentemente, aplicações matemáticas ou científicas exigem mais do que eixos únicos em uma representação. Essa biblioteca nos ajuda a criar várias ao mesmo tempo. No entanto, você pode usar o Matplotlib para manipular diferentes características das imagens.

Existem mais de 137.000 bibliotecas e 198.826 pacotes para *Python*, prontos para facilitar a programação dos desenvolvedores. Essas bibliotecas e pacotes destinam-se a uma variedade de soluções modernas, tais como: organização de matrizes, análise de expressões matemáticas, geração de gráficos, entre outros.

9. Parte Prática

Na parte prática será mostrado o gerenciador de versões *Python* chamado Anaconda e para a geração de gráficos utilizando o Matplotlib, mostraremos como funciona o google colab. Utilizaremos o VS Code para auxiliar na programação. Também serão mostrados exemplos de códigos dos mais simples aos mais elaborados, utilizando as principais bibliotecas para Data Science para que seja possível um overview geral sobre a linguagem.

Referências

- [1] Learn Python Programming. Disponível em: <<https://www.programiz.com/python-programming>>. Acesso em: 10 maio. 2022.
- [2] ABC (linguagem de programação) | Wikiwand. Disponível em: <[https://www.wikiwand.com/pt/ABC_\(linguagem_de_programa%C3%A7%C3%A3o\)](https://www.wikiwand.com/pt/ABC_(linguagem_de_programa%C3%A7%C3%A3o))>. Acesso em: 26 maio. 2022.
- [3] Code Style — The Hitchhiker's Guide to Python. Disponível em: <<https://docs.python-guide.org/writing/style/>>. Acesso em: 26 maio. 2022.
- [4] PEP 20 – The Zen of Python | peps.python.org. Disponível em: <<https://peps.python.org/pep-0020/>>. Acesso em: 26 maio. 2022.
- [5] Perceiving Python programming paradigms. Disponível em: <<https://opensource.com/article/19/10/python-programming-paradigms>>. Acesso em: 26 maio. 2022.
- [6] Welcome to Python.org. Disponível em: <<https://www.python.org/about/>>. Acesso em: 26 maio. 2022.
- [7] Python Data Types. Disponível em: <<https://pynative.com/python-data-types/>>. Acesso em: 26 maio. 2022.
- [8] Python Class Method vs. Static Method vs. Instance Method. Disponível em: <<https://pynative.com/python-class-method-vs-static-method-vs-instance-method/>>. Acesso em: 26 maio. 2022.
- [9] OLIVEIRA, M. As 30 melhores bibliotecas e pacotes Python para Iniciantes. Disponível em: <<https://terminalroot.com.br/2019/12/as-30-melhores-bibliotecas-e-pacotes-python-para-iniciantes.html>>. Acesso em: 26 maio. 2022.

10. Apêndice

Carolina Lima

Durante o desenvolvimento do trabalho, o meu foco maior sempre foi as importações das bibliotecas para Machine Learning que o *Python* disponibiliza. Aprendi a programar nessa linguagem de uma forma tão natural que nunca parei para ler a documentação. Quando surgia algum erro, apenas procurava em fóruns formas de resolvê-los.

A documentação oficial do Python além de ser possível encontrar detalhes profundos para uma melhor utilização da linguagem, é traduzida para o português brasileiro, o que facilita ainda mais na busca por informações.

Estou ansiosa para a parte prática, onde poderemos mostrar um pouquinho mais sobre tudo que o *Python* é capaz de fazer. Estou utilizando *Python* em diversas áreas nesse semestre e o que achei mais legal, além da área de Visão Computacional, é a forma como podemos tratar bases de dados gigantescas.

Eu poderia continuar falando sobre por que o Python é tão incrível, mas vamos nos limitar a alguns pontos. *Python* é uma das linguagens mais fáceis de aprender, apesar do fato de que a lógica de programação ainda é difícil.

Python também é uma das linguagens mais procuradas no setor de tecnologia hoje, usada por empresas como Google, Facebook, IBM, etc. Ele tem sido usado para criar aplicativos como Instagram, Pinterest, Dropbox e muito mais!

Mateus Gontijo

Ao decorrer do desenvolvimento do trabalho vários fatores em relação ao python me chamaram a atenção, o mais presente é a facilidade com que se encontram extensões para o uso da linguagem. Há uma abundância de bibliotecas disponíveis para uso livre e a maioria com documentação detalhada sobre como se utiliza. Não só isso, mas a quantidade de projetos de automação computacional existentes e de fácil implementação é um fato a se destacar a respeito da linguagem. Tudo isso atrelado a uma simplicidade de aprendizado do python devido à quantidade de programadores que consomem o python nas mais variadas tarefas do dia-a-dia.

Seu uso em interfaces web tem crescido muito durante os últimos anos, podemos citar o Django como um dos principais e mais utilizado. O Django é um framework de código aberto e escrito em Python que permite aos programadores implementar aplicativos complexos de maneira rápida e eficiente.

Python também vem crescendo sua popularidade através da grande evolução das IAs, suas implementações no machine learning são muito promissoras e não é atoa que essa linguagem tem sido destaque nessa área da computação.

Diante todos os itens estudados pude concluir que o python tem uma gama de aplicações muito vasta dentro do mundo da computação. Também pude ver que suas bibliotecas mais variadas ajudam diversas áreas da ciência a solucionarem problemas e até descobrirem coisas. Não menos importante a sua facilidade com o iniciante na programação devido à quantidade de conteúdo disponível na internet e também por ser relativamente simples comparada a outras linguagens.