

# Python orientado a objetos

Disciplina: Programação aplicada à engenharia cartográfica

Maurício C. M. de Paulo - D.Sc.

13 de fevereiro de 2026

# Paradigmas de programação em Python

Python suporta múltiplos paradigmas:

- Programação funcional
- Programação orientada a objetos (OO)

## Funcional

- Ênfase em funções
- Dados passados como argumentos
- Simples para tarefas pequenas

## Orientado a Objetos

- Dados + comportamento juntos
- Modela entidades do mundo real
- Facilita manutenção e reutilização

# Por que usar Orientação a Objetos?

## Principais vantagens:

- Organização do código em entidades
- Reutilização por meio de classes
- Facilita leitura e manutenção
- Escala melhor para sistemas grandes

## Quando OO é mais indicada:

- Projetos grandes
- Sistemas com múltiplas responsabilidades
- Modelagem de objetos reais (usuários, mapas, sensores)

# Exemplo funcional — LineString

```
import math

def comprimento_linha(coords):
    comprimento = 0.0
    for i in range(len(coords) - 1):
        x1, y1 = coords[i]
        x2, y2 = coords[i + 1]
        comprimento += math.dist((x1, y1), (x2, y2))
    return comprimento

linha = [(0, 0), (3, 4), (6, 4)]
comp = comprimento_linha(linha)
```

## Características:

- Dados são estruturas externas
- Funções operam sobre listas de coordenadas
- Pouca semântica explícita

# Módulos e pacotes em Python

## Estrutura de arquivos:

```
projeto/  
    main.py  
    modelos/  
        __init__.py  
        pessoa.py
```

## Importação:

```
from modelos import pessoa #importa todo o módulo
```

## Benefício:

- Código organizado e reutilizável

# Exemplo OO — LineString

```
import math
class LineString:
    def __init__(self, coords):
        self.coords = coords

    def comprimento(self):
        comprimento = 0.0
        for i in range(len(self.coords) - 1):
            p1 = self.coords[i]
            p2 = self.coords[i + 1]
            comprimento += math.dist(p1, p2)
        return comprimento

linha = LineString([(0, 0), (3, 4), (6, 4)])
comp = linha.comprimento()
```

## Vantagem:

- Geometria encapsula dados e operações

# Declaração de classes em Python

```
class NomeDaClasse:  
    def __init__(self, parametro):  
        self.atributo = parametro  
  
    def metodo(self):  
        print(self.atributo)
```

## Elementos principais:

- class: define uma classe
- self: referência ao objeto
- Métodos: funções da classe

## Função do \_\_init\_\_:

- Executado na criação do objeto
- Inicializa atributos

# Instanciação de objetos

```
class Ponto:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
p1 = Ponto(10,20)  
p2 = Ponto(-10,10)  
  
print(p1) #<__main__.Ponto object at 0x71e2a65bcdd0>  
print(p2) #<__main__.Ponto object at 0x71e2a642c770>  
  
print(p2.x, p2.y) #-10 10
```

## Observação:

- Cada objeto tem seu próprio estado

# @dataclass

```
from dataclasses import dataclass

@dataclass
class Ponto:
    x: float
    y: float

p1 = Ponto(10,20)
p2 = Ponto(-10,10)

print(p1) # Ponto(x=10, y=20)
print(p2) # Ponto(x=-10, y=10)
```

## Vantagens:

- Menos código repetitivo
- `__init__`, `__repr__` automáticos
- Ideal para classes de dados

# @dataclass

## Orientação a objetos com @dataclass

```
import math
from dataclasses import dataclass

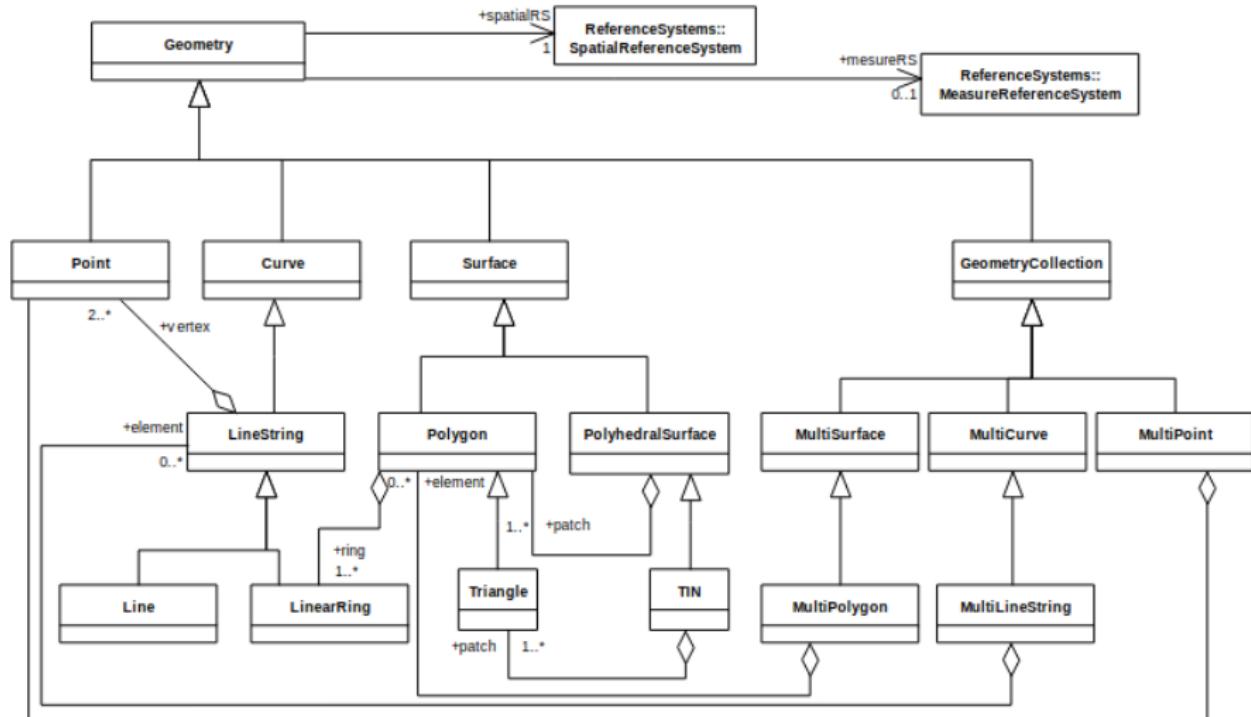
@dataclass
class LineString:
    coords: list[tuple[float, float]]

    def comprimento(self) -> float:
        comprimento = 0.0
        for i in range(len(self.coords) - 1):
            p1 = self.coords[i]
            p2 = self.coords[i + 1]
            comprimento += math.dist(p1, p2)
        return comprimento

linha = LineString([(0, 0), (3, 4), (6, 4)])
comp = linha.comprimento()
```

# Open Geospatial Consortium (OGC) - Simple features access

Especificação: <https://www.ogc.org/publications/standard/sfa/>



# OGC SFS — Herança de classes

## OGC Simple Feature Specification (SFS)

No OGC SFS:

- Geometry é a classe base
- Tipos geométricos específicos herdam de Geometry

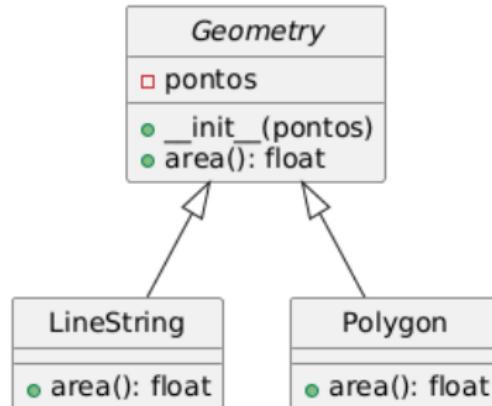
### Exemplo em Python:

```
class Geometry:  
    def __init__(self, pontos):  
        self.pontos = pontos  
  
    def area(self) -> float:  
        raise NotImplementedError
```

### Conceito-chave:

- Classes filhas herdam interface e comportamento

# OGC SFS — Diagrama de classes (simplificado)



## Objetivo do diagrama:

- Visualizar herança
- Relacionar código com modelo conceitual

# OGC SFS — Classes derivadas

## LineString

```
class LineString(Geometry):
    # Método herdado:
    # __init__(self, pontos)

    def area(self) -> float:
        return 0.0
```

## Polygon

```
class Polygon(Geometry):
    # Método herdado:
    # __init__(self, pontos)

    def area(self) -> float:
        area = 0.0
        n = len(self.pontos)
        for i in range(n):
            x1,y1=self.pontos[i]
            x2,y2=self.pontos[(i+1)%n]
            area += x1*y2 - x2*y1

        return abs(area) / 2.0
```

- Geometria unidimensional
- Não ocupa área

- Geometria bidimensional
- Implementa cálculo real de

# Polimorfismo com OGC SFS

```
def imprimir_area(geom: Geometry):
    print(f"Área = {geom.area()}")
```

```
l = LineString([(0,0), (1,1)])
p = Polygon([(0,0), (1,0), (1,1), (0,1)])

imprimir_area(l)
imprimir_area(p)
```

## Resultado:

- Mesmo código
- Comportamento depende da classe concreta

# Exercícios - Downloader

- Implementem uma classe que baixe dados de um servidor.
- Para criar um objeto “Downloader” devem ser passados o endereço de download e uma pasta de destino.

Link da dica

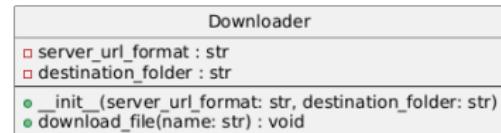
O endereço de uma folha no geoftp do IBGE é:

[https:](https://geoftp.ibge.gov.br/cartas_e_mapas/folhas_topograficas/vetoriais/escala_1000mil/shapefile/g04_na19.zip)

//geoftp.ibge.gov.br/cartas\_e\_mapas/folhas\_topograficas/  
vetoriais/escala\_1000mil/shapefile/g04\_na19.zip

Exemplo:

```
parte1 = "https://geoftp.ibge.gov.br/cartas_e_mapas/folhas_topograficas/"  
parte2 = f"vetoriais/escala_1000mil/shapefile/{name}.zip"  
server_url_format = parte1 + parte2
```



# Exercícios - Downloader

```
import requests, os

class Downloader:
    def __init__(self, server_url_format, destination_folder):
        # Inicialização dos atributos da classe
        self.server_url_format = server_url_format
        self.destination_folder = destination_folder
        #verifica se a pasta existe, se não existe, cria.
        os.makedirs(self.destination_folder, exist_ok=True)

    def download_file(self, name):
        # Esta não é a melhor forma de fazer isso. É um exemplo.
        url = self.server_url_format.format(name)
        response = requests.get(url)
        file_Path = f'{self.destination_folder}/{name}.zip'

        if response.status_code == 200:
            with open(file_Path, 'wb') as file:
                file.write(response.content)
                print('File downloaded successfully')
        else:
            print('Failed to download file')
```

# Exercícios - Downloader

## Utilizando a classe:

```
# Esta parte do código só será executada se este for o arquivo principal.  
# Ou seja: python downloader.py  
if __name__=="__main__":  
    # Cria o objeto da classe Downloader  
    obj_teste = Downloader( \  
        "https://geoftp.ibge.gov.br/cartas_e_mapas/folhas_topograficas/vetoriais/escala_1:500000/  
        "/home/mauricio/Desktop/progcart/aulas" )  
  
    # Lista os atributos e métodos do objeto.  
    print(dir(obj_teste))  
  
    # Chama uma função do objeto criado.  
    obj_teste.download_file("g04_na19")
```

# Mais exercícios

Exercícios para o pessoal que quiser praticar orientação a objetos:

<https://www.w3resource.com/python-exercises/oop/index.php>