

Introdução a Algoritmos

Disciplina: Programação aplicada à engenharia cartográfica

Maurício C. M. de Paulo - D.Sc.

6 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

```
1 import math
2
3 def comprimento_linha(coords):
4     comprimento = 0.0
5     for i in range(len(coords) - 1):
6         x1, y1 = coords[i]
7         x2, y2 = coords[i + 1]
8         comprimento += math.dist((x1, y1), (x2, y2))
9     return comprimento
10
11
12 linha = [(0, 0), (3, 4), (6, 4)]
13 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:

```
1 projeto/  
2     main.py  
3     modelos/  
4         __init__.py  
5         pessoa.py
```

Importação:

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

Benefício:

- Código organizado e reutilizável

Exemplo OO — LineString

```
1 import math
2 class LineString:
3     def __init__(self, coords):
4         self.coords = coords
5
6     def comprimento(self):
7         comprimento = 0.0
8         for i in range(len(self.coords) - 1):
9             p1 = self.coords[i]
10            p2 = self.coords[i + 1]
11            comprimento += math.dist(p1, p2)
12            return comprimento
13
14 linha = LineString([(0, 0), (3, 4), (6, 4)])
15 comp = linha.comprimento()
```

Vantagem:

- Geometria encapsula dados e operações

Declaração de classes em Python

```
1 class NomeDaClasse:
2     def __init__(self, parametro):
3         self.atributo = parametro
4
5     def metodo(self):
6         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

```
1 class Ponto:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6 p1 = Ponto(10,20)
7 p2 = Ponto(-10,10)
8
9 print(p1) #<\_ \_main\_\_.Ponto object at 0x71e2a65bcdd0>
10 print(p2) #<\_ \_main\_\_.Ponto object at 0x71e2a642c770>
11
12 print(p2.x, p2.y) #-10 10
```

Observação:

- Cada objeto tem seu próprio estado

```
1 from dataclasses import dataclass
2
3 @dataclass
4 class Ponto:
5     x: float
6     y: float
7
8 p1 = Ponto(10,20)
9 p2 = Ponto(-10,10)
10
11 print(p1) # Ponto(x=10, y=20)
12 print(p2) # Ponto(x=-10, y=10)
```

Vantagens:

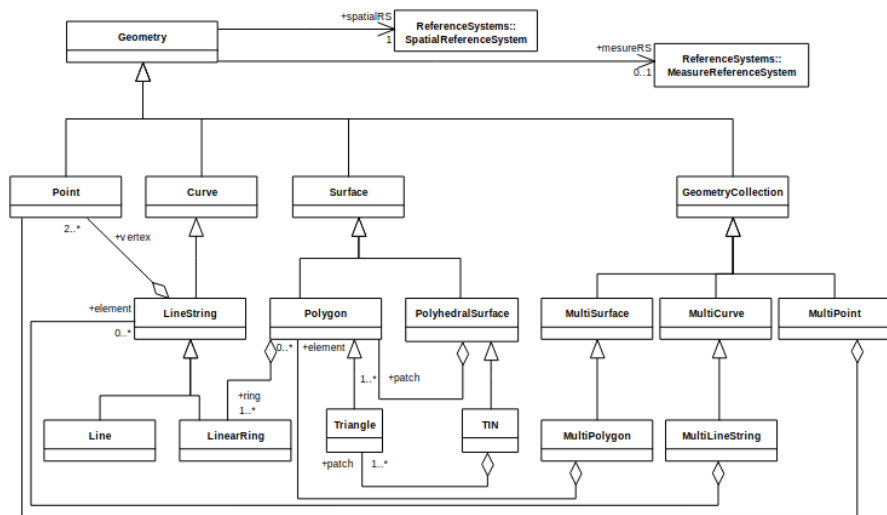
- Menos código repetitivo
- `__init__`, `__repr__` automáticos
- Ideal para classes de dados

Orientação a objetos com @dataclass

```
1 import math
2 from dataclasses import dataclass
3
4 @dataclass
5 class LineString:
6     coords: list[tuple[float, float]]
7
8     def comprimento(self) -> float:
9         comprimento = 0.0
10         for i in range(len(self.coords) - 1):
11             p1 = self.coords[i]
12             p2 = self.coords[i + 1]
13             comprimento += math.dist(p1, p2)
14         return comprimento
15
16 linha = LineString([(0, 0), (3, 4), (6, 4)])
17 comp = linha.comprimento()
```

Open Geospatial Consortium (OGC) - Simple features access

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



OGC Simple Feature Specification (SFS)

No OGC SFS:

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

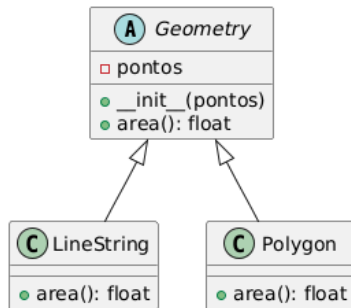
Exemplo em Python:

```
1 class Geometry:
2     def __init__(self, pontos):
3         self.pontos = pontos
4
5     def area(self) -> float:
6         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

LineString

```
1 class LineString(Geometry):
2     # Método herdado:
3     # __init__(self, pontos)
4
5     def area(self) -> float:
6         return 0.0
```

- Geometria unidimensional
- Não ocupa área

Polygon

```
1 class Polygon(Geometry):
2     # Método herdado:
3     # __init__(self, pontos)
4
5     def area(self) -> float:
6         area = 0.0
7         n = len(self.pontos)
8         for i in range(n):
9             x1,y1 = self.pontos[i]
10            x2,y2 = self.pontos[(i+1)%n]
11            area += x1*y2 - x2*y1
12        return abs(area) / 2.0
```

- Geometria bidimensional
- Implementa cálculo real de área

Polimorfismo com OGC SFS

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

```
1 l = LineString([(0,0), (1,1)])  
2 p = Polygon([(0,0), (1,0), (1,1), (0,1)])  
3  
4 imprimir_area(l)  
5 imprimir_area(p)
```

Resultado:

- Mesmo código
- Comportamento depende da classe concreta

Exercícios

- 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

Exemplo:

O endereço de uma folha no geoftp do IBGE é:

https:

//geoftp.ibge.gov.br/cartas_e_mapas/folhas_topograficas/
vetoriais/escala_1000mil/shapefile/g04_na19.zip

O server_url_format seria:

f"https://geoftp.ibge.gov.br/cartas_e_mapas/folhas_topograficas/vet

Downloader
+ server_url_format: string
+ destination_folder: string
+ __init__(server_url_format, destination_folder):
+ download_file(name): string

Mais exercícios

Exercícios para o pessoal que quiser praticar orientação a objetos:

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