CEFET/RJ

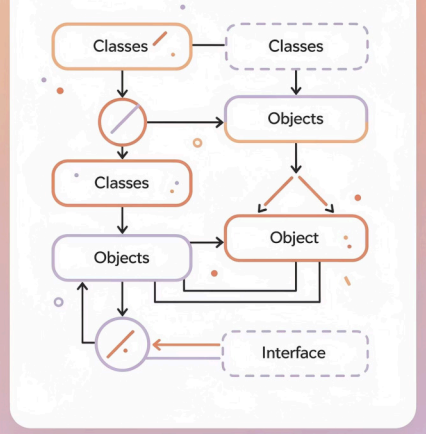
Classes e Objetos em Python

Programação orientada a objetos (OOP) em Python permite criar tipos próprios para modelar entidades do problema. Uma classe define dados (atributos) e comportamentos (métodos).

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br

https://eic.cefet-rj.br/~eogasawara



CONCEITO FUNDAMENTAL

Objeto com Atributos

Um objeto é uma instância de uma classe, contendo atributos que guardam estado. A classe define como o objeto é criado e quais validações são aplicadas.

Atributos ficam acessíveis via `obj.atributo`.

```
class Polygon:
 def __init__(self, n: int):
 self.n = n

p = Polygon(5)
print(p.n)
```

Construtor com Validação

Criação Consistente

O construtor `__init__` cria e inicializa objetos de forma consistente.

Validação de Entrada

Evita estados inválidos (por exemplo, número de vértices ≤ 0).

Exceções Claras

Se a validação falhar, lance uma exceção com mensagem clara.

```
class Polygon:
 def __init__(self, n: int):
 if n <= 0:
 raise ValueError("número de vértices deve ser maior que zero")
 self.n = n

p = Polygon(5)
print(p.n)
```

HERANÇA

Especializando Polygon em Rectangle

Polygon

Classe base com `n` vértices

Rectangle

Especialização com 4 vértices e atributos `w`, `h`

Herança permite reutilizar código e especializar comportamento. A relação "é-um" fica explícita: Rectangle **é um** Polygon.

```
class Rectangle(Polygon):
 def __init__(self, w: float, h: float):
 super().__init__(4)
 self.w = w
 self.h = h

r = Rectangle(3, 10)
print(r.n, r.w, r.h)
```

Interface de Impressão com __str__

Polimorfismo em Ação

Em Python, a forma padrão de controlar `print(obj)` é implementar `__str__`.

Cada classe pode definir sua própria representação textual. O mesmo `print()` exibe coisas diferentes conforme o tipo do objeto.

```
class Polygon:
 def __str__(self) -> str:
 return f"{self.n}"

class Rectangle(Polygon):
 def __str__(self) -> str:
 return f"{self.w}, {self.h}"

p = Polygon(5)
r = Rectangle(3, 10)
print(p) # 5
print(r) # 3, 10
```

Polimorfismo

Mesmo Comando

O comando `print()` funciona para diferentes tipos de objetos.

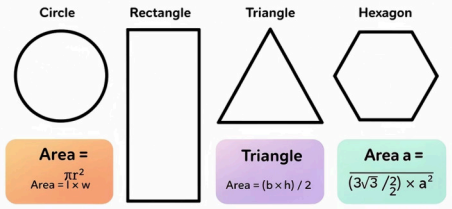
Resultados Diferentes

Cada classe define seu próprio `__str__`, produzindo saídas específicas.

Comportamento Dependente

Esse comportamento dependente do tipo é chamado de **polimorfismo**.

```
p = Polygon(6)
r = Rectangle(2, 4)
print(p) # usa Polygon.__str__()
print(r) # usa Rectangle.__str__()
```



NOVA INTERFACE

Criando a Interface area()

Uma interface é uma função que funciona para vários tipos de objetos. Em Python, isso é feito definindo um método com o mesmo nome nas classes.

Cada classe implementa sua própria versão de `area()`.

```
class Polygon:
 def __init__(self, n: int):
 self.n = n

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

Método Padrão e Método Específico

1

Método Padrão

Trata casos genéricos na classe base

2

Sobrescrita

Classes especializadas sobrescrevem o método

3

Comportamento Próprio

Cada classe implementa sua lógica específica

```
class Rectangle(Polygon):
    def __init__(self, w: float, h: float):
        super().__init__(4)
        self.w = w
        self.h = h

    def area(self) -> float:
        return self.w * self.h
```

Isso substitui o `UseMethod()` do R de forma natural em Python.

Descobrindo Interfaces Disponíveis

Inspeção de Objetos

Em Python, podemos inspecionar um objeto para ver seus métodos disponíveis.

Função dir()

A função `dir()` lista atributos e métodos públicos de um objeto.

Descoberta de Interfaces

Isso ajuda a descobrir quais "interfaces" um objeto oferece.

```
p = Polygon(5)
r = Rectangle(3, 10)
print(dir(p))
print(dir(r))
```

Usando os Objetos na Prática

Criação e Uso

- Criamos objetos de tipos diferentes
- Chamamos os mesmos métodos sobre eles
- Cada classe responde de forma adequada ao seu tipo

```
a = 3
p = Polygon(5)
r = Rectangle(3, 10)

print(p) # 5
print(r) # 3, 10
print(p.area()) # 0.0
print(r.area()) # 30
```

EXEMPLO

Estendendo o suporte a polígonos

Square

Deve herdar de Rectangle com lado igual

```
import math

class Square(Rectangle):
 def __init__(self, side: float):
 super().__init__(side, side)
```

Hexagon

Deve herdar de Polygon e implementar sua própria área

```
class Hexagon(Polygon):
 def __init__(self, side: float):
 super().__init__(6)
 self.side = side

 def area(self) -> float:
 return (3 * math.sqrt(3) / 2) * self.side ** 2
```

```
s = Square(4)
h = Hexagon(2)
print(s.area())
print(h.area())
```

O Que Vem a Seguir

Fundamentos Dominados

Você já domina os fundamentos de classes e objetos em Python.

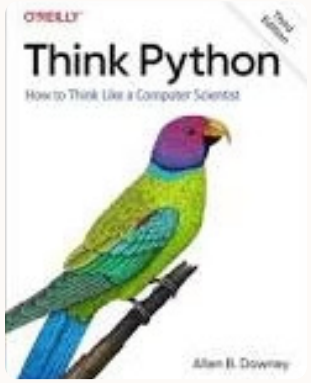
Construtores e Herança

Sabe criar construtores, implementar herança e aplicar polimorfismo.

Próximos Passos

Esses conceitos são base para trabalhar com dados reais em Python.

Referências



Think Python

Downey, A. *Think Python: How to Think Like a Computer Scientist.* O'Reilly Media.

An essential introduction to programming fundamentals and computational thinking using Python.

Python Data Science Handbook

VanderPlas, J. *Python Data Science Handbook.* O'Reilly Media.

A comprehensive guide to essential tools for working with data in Python, including NumPy, Pandas, and visualization libraries.

Data Science from Scratch

Grus, J. *Data Science from Scratch.* O'Reilly Media.

Learn data science fundamentals by building algorithms and tools from the ground up using Python.