

Programação Orientada ao Objeto



UNINASSAU

Prof^ª. Esp. Sônia Gomes de Oliveira

PAULISTA - 2024

O que é Programação Orientada a Objetos – POO?

A POO é um paradigma de programação que se propõe a abordar o design de um sistema em termos de entidades, os objetos, e relacionamentos entre essas entidades.

Objetivo

O objetivo é aproximar cada vez mais o mundo digital do mundo real.

Linguagens de Programação POO

C ++

Python

Java

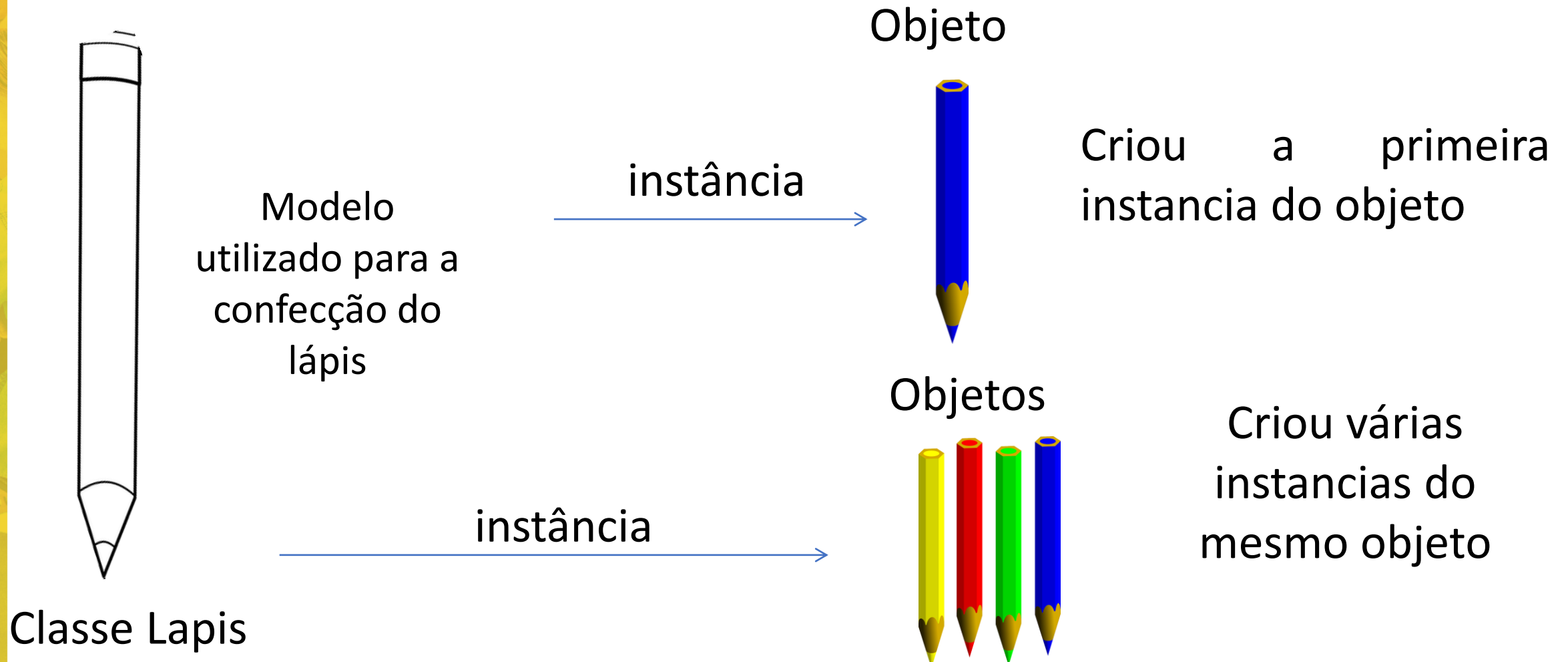
Ruby

PHP

visual basic

O que são classes?

Classes são consideradas modelos/moldes para construir um objeto. Tem como objetivo organizar os dados e funcionalidades no código.



O que é um objeto?

Um objeto é algo material ou abstrato que pode ser definido através de suas características, comportamentos e estados.

EXEMPLO:



Características:

- Modelo
- Botões
- Fabricante

Comportamento/Ações

- Ligar()
- Desligar()
- Pausar()

Estado/status

- Bom uso
- Ligando
- Desligando

Representação de Classes e Objetos

classe

Cachorro

Modelo: Classe

cor: str
olhos: str
comprimento: float
peso: float
raca : str

Características = atributos

latir()
correr()
sentar()
comer()

Comportamentos = métodos

latindo
correndo
comendo

Estados = status

OBJETO



Nomenclaturas importantes

Classes: Modelos ou moldes que sendo instanciada consegue criar um objeto ou vários objetos.

Objetos: É uma instância da classe.

Atributos: Características ou atributos do objeto

Métodos: São os comportamentos do objeto/ações.

Como definir uma classe?

```
class Pessoa: #nome da classe
```

```
'''
```

```
    DOCSTRING (opcional, mas recomendado)  
    utilizada para definir o que é a classe, seus  
    atributos e o que ela faz
```

```
'''
```


O que é um construtor?

O construtor é método tem como objetivo inicializar um objeto de classe quando o objeto é criado. Geralmente são usados para fazer a instancia.

Em Python, o método `__init__` é chamado de construtor e é sempre chamado quando um objeto é criado.

Tipos de Construtores

Construtor padrão: O construtor padrão é um construtor simples que **não aceita nenhum argumento**. Sua definição tem apenas um argumento obrigatório que é uma referência à instância, definida como **self**.

Sintaxe do construtor:

```
def __init__(self):
```

Quando uma função está dentro de uma classe, ela é chamada de **método de classe**. Podemos dizer que um método é uma função interna de uma classe.

Quando a classe inclui o método **__init__**, é automaticamente invocada quando a classe é instanciada.

Tipos de Construtores

Construtor parametrizado: É um construtor que recebe os parâmetros já definidos pelo programador. Também utiliza o self como o primeiro argumento.

Sintaxe do construtor parametrizado:

```
def __init__(self, nome, idade, profissao):
```

O identificador Self

O self identifica a instancia sobre a qual o método é invocado. Também é usado para acessar os membros que pertencem a classe.

Deve ser o primeiro parâmetro usado na definição do método.

EXEMPLO:

```
def __init__(self, nome, idade):  
    self.nomePessoa = nome  
    self.idadePessoa = idade
```

Exemplo – Classe com Construtor

Quando fazemos a criação de uma classe, devemos sempre iniciar com a primeira letra do nome da classe em maiúscula, e o restante minúscula.

EXEMPLO:

```
class Lapis:
    def __init__(self, modelo, cor, tamanho, ponta):
        self.modelo = modelo
        self.cor = cor
        self.tamanho = tamanho
        self.ponta = ponta
        print(self.modelo, self.cor, self.tamanho, self.ponta)

    def riscar(self):
        print('Está riscando!')

    def pintar(self):
        print('Está Pintando!')

    def escrever(self):
        print('Está escrevendo!')
```

Classe com Construtor - continuação

Fora da classe instanciamos:

```
modelo = input('Informe a modelo: ')\ncor = input('Informe a cor: ')\ntamanho = input('Informe a tamanho: ')\nponta = input('Informe a ponta: ')\nobjeto = Lapis(modelo, cor, tamanho, ponta)\nobjeto.riscar() #chamando o método
```


Exemplo – Classe sem Construtor

```
class Lapis:
    def recebe (self, modelo, cor, tamanho, ponta):
        self.modelo = modelo
        self.cor = cor
        self.tamanho = tamanho
        self.ponta = ponta
        print(self.modelo, self.cor, self.tamanho, self.ponta)

#----- INSTANCIANDO A CLASSE LAPIS -----

objeto = Lapis()
objeto.recebe('a', 'a', 10, True)
```

Exemplo 2 – Classe e métodos

Vamos criar a classe, construtor e o método

```
class Bola:
    def __init__(self, cor, marca):
        self.cor = cor
        self.marca = marca
        print('Valores iniciais: ', self.cor, self.marca)

    def mudarCor(self):
        self.cor_bola = input('Informe uma nova cor: ')
        print(f'A cor anterior é: {self.cor}')
        print(f'A nova cor é: {self.cor_bola}')

objeto = Bola('amarela', 'bic')
objeto.mudarCor()
```



**Como receber uma instancia
sem ser dentro da Classe?**

Como receber uma instancia sem ser dentro de uma Classe?

Primeiro você vai ter que criar o arquivo com a classe e depois você irá criar outro arquivo para chamar a classe especifica.

Como faz isso??

Utilizando o comando

```
from nome_arquivo import nome_classe
```

Praticando

Inicialmente vamos criar a classe Pessoa

teste.py

#nome do meu arquivo que criei a classe

```
class Pessoa:
    def __init__(self, nome, idade, altura, peso):
        self.nome = nome
        self.idade = idade
        self.altura = altura
        self.peso = peso
        print(self.nome, self.idade, self.altura, self.peso)
```

Praticando

Agora vamos criar outro arquivo e chamar a classe Pessoa

```
from teste import Pessoa

nome = input('Informe o nome: ')
idade = int(input('Informe o idade: '))
altura = float(input('Informe o altura: '))
peso = float(input('Informe o peso: '))

objeto = Pessoa(nome, idade, altura, peso)
```


Sobrecarga de métodos

É considerada sobrecarga de métodos quando você utiliza o método de uma classe várias vezes, passando parâmetros diferentes. Tem como objetivo evitar poluição no código.

EXEMPLO:

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
        print(self.nome, self.idade)
```

```
Pessoa('ana', 24)
Pessoa('maria', 50)
Pessoa('carlos', 34)
```

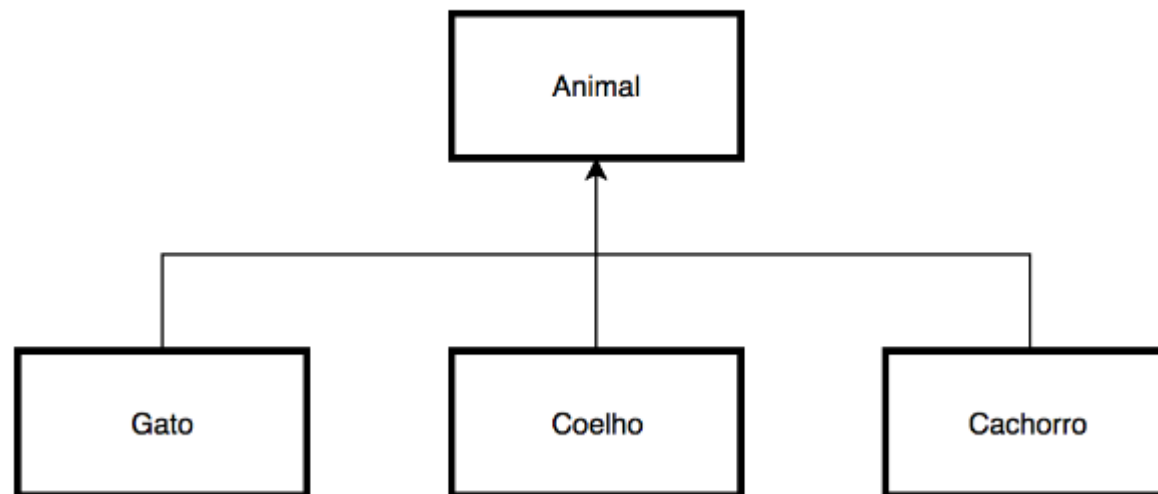


Herança em Python

POO - Herança

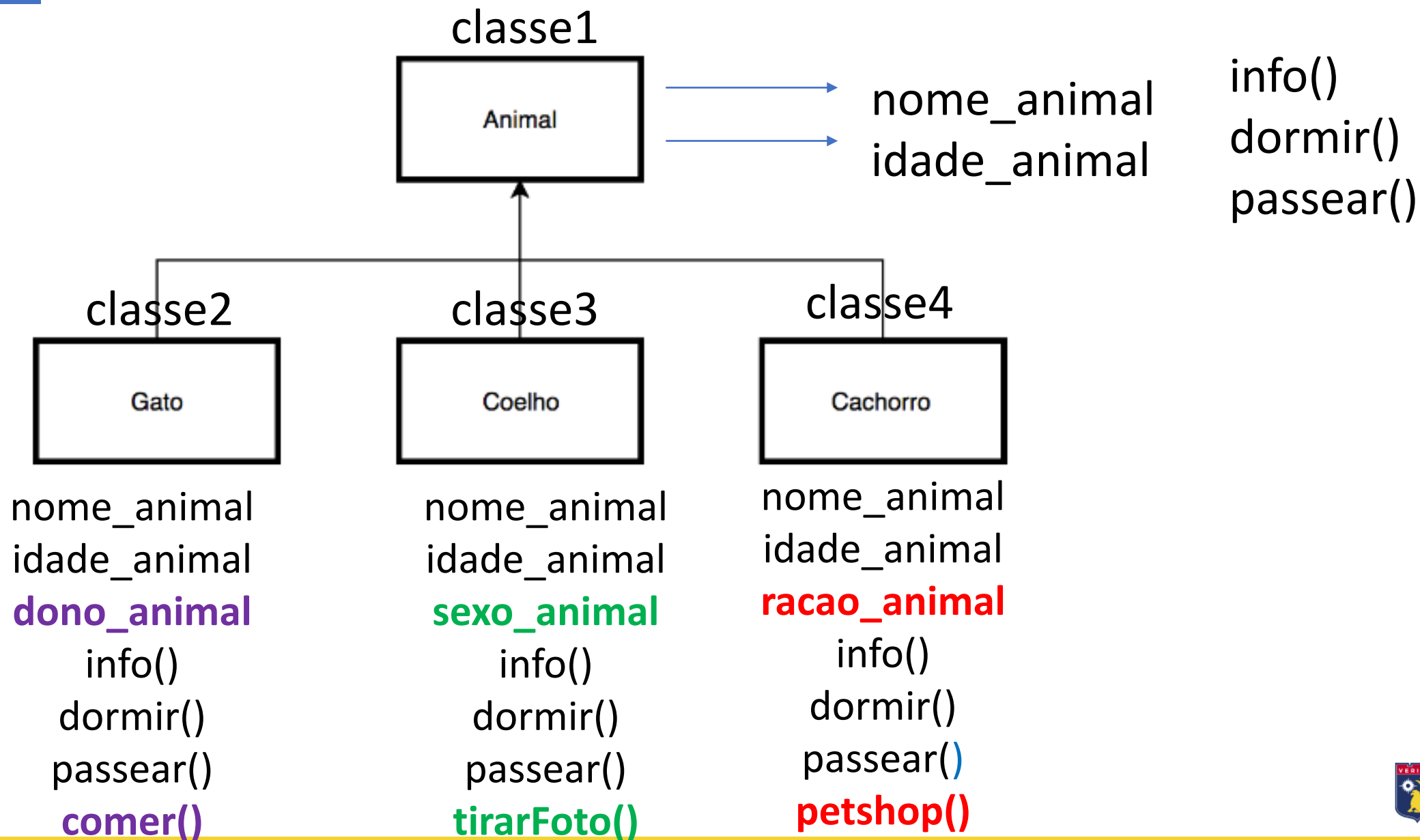
A Herança é um paradigma de orientação ao objeto que tem como objetivo permitir que classes (filhas) possam herdar da classe (pai) seus métodos e atributos, de modo que, reaproveite os códigos para não ficar repetindo desnecessariamente.

EXEMPLO:



Classes (filhas)

Herança – Entendendo a herança



POO - Herança

módulo classe2: Herdando as características da classe Pai.

```
classe2.py x
1  from classe1 import Animal
2
3  class Gato(Animal):
4      def __init__(self, nome_animal, idade_animal, dono):
5          Animal.__init__(self, nome_animal, idade_animal)
6          self.dono = dono
7
8      def mostrar_dono(self):
9          print(f'Seu dono é: {self.dono}')
10
11     def comer(self):
12         print('Gatinho que gosta de comer apenas ração')
13
```

POO - Herança

módulo classe2: Fora da classe Gato, iremos instanciar o objeto

```
#fora da classe Gato iremos fazer a instância do objeto
nome = input('Nome do animal: ')
idade = input('Idade do animal: ')
dono = input('Nome do dono do animal: ')

objeto = Gato(nome, idade, dono)
objeto.info() #método da classe pai
objeto.mostrar_dono() #método da classe filha
objeto.comer() #método da classe filha
objeto.passear() #método da classe pai
```

Console

```
Nome do animal: Nico
Idade do animal: 2
Nome do dono do animal: maria
Nome:Nico
Idade:2 ano(s)
Seu dono é: maria
Gatinho que gosta de comer apenas ração
Meu animal gosta de passear
```


POO - Herança

módulo classe3: Herdando apenas os métodos da Classe Pai

```
1  from classe1 import Animal
2
3  class Coelho(Animal):
4
5      def __init__(self, sexo_animal):
6          self.sexo = sexo_animal
7
8      def tirarFoto(self):
9          print('Tirando fotos do meu animal de estimação!')
10
11
12  #fora da classe coelho iremos instanciar o objeto
13
14  objeto = Coelho('femêa')
15  objeto.tirarFoto()
16  objeto.passear()
17  objeto.dormir()
```

Herança da classe Animal

Não foi utilizado o construtor da classe Pai, porque queríamos só os métodos.

Console

```
Tirando fotos do meu animal de estimação!
Meu animal gosta de passear
Animal que gosta muito de dormir
```



Classes no mesmo módulo e Polimorfismo

Classe no mesmo módulo

No Python é possível
você criar várias classes
no mesmo arquivo.

Chamando o método
da classe Pai e filha.

```
class Animal:
    def __init__(self, nome_animal, idade_animal):
        self.nome = nome_animal
        self.idade = idade_animal

    def info(self):
        print(f'Nome:{self.nome}\nIdade:{self.idade} ano(s)')

    def dormir(self):
        print('Animal que gosta muito de dormir')

    def passear(self):
        print('Meu animal gosta de passear')

class teste(Anilal):
    def print_teste(self):
        print('Apenas mostrando um método de teste')

objeto = teste('pitbul', 5)
objeto.info()
objeto.print_teste()
```

Polimorfismo

Polimorfismo é a capacidade de uma ou mais subclasses terem os mesmos métodos de uma superclasse, só que com comportamentos diferentes.

EXEMPLO:

```
classe4.py x
1 class Principal:
2
3     def metodoPrincipal(self):
4         print('Este é um método da SuperClasse\n')
5
6
7 class Secundaria(Principal):
8
9     def metodoPrincipal(self):
10        print('Agora eu tenho o mesmo método da SuperClasse!')
11        print('Só que com instruções diferentes :)')
12        print('Isto é chamando de Polimorfismo')
13
14 objeto = Principal()
15 objeto.metodoPrincipal()
16
17 objeto1 = Secundaria()
18 objeto1.metodoPrincipal()
```

Exemplo de Classes
no mesmo módulo



Encapsulamento

Encapsulamento - Conceito

O encapsulamento é a proteção dos atributos ou métodos de uma classe. O Encapsulamento tem como objetivo esconder os elementos para não ficar visível para as outras classes ou projetos.

No Python existe o **Public**, **Protected** e o **Private**.

Encapsulamento - Public

Exemplo1: Primeiro exemplo de uma classe que possui apenas métodos e atributos públicos.

Então todos os dados podem ser acessados livremente.

Com o public é possível alterar variáveis locais e Instanciar os métodos, pois todos são públicos.

```
class Funcionario:
    def __init__(self, nome, cargo):
        self.nome = nome
        self.cargo = cargo

        self.salario = 0.0

    def mostrar_info(self):
        print(self.nome, self.cargo)

    def mostrar_salario(self):
        print(self.salario)

nome = input('Insira o nome: ')
cargo = input('Insira o cargo: ')
objeto = Funcionario(nome, cargo)
objeto.salario = 3000
objeto.mostrar_info()
objeto.mostrar_salario()
```

Encapsulamento Public – Acessando dados de uma subClasse

A subclasse pode manipular qualquer atributo público ou protegido da classe Pai

```
class Funcionario:
    def __init__(self, nome, cargo):
        self.nome = nome
        self.cargo = cargo

        self.salario = 0.0

    def mostrar_info(self):
        print(self.nome, self.cargo)

    def mostrar_salario(self):
        print(self.salario)

nome = input('Insira o nome: ')
cargo = input('Insira o cargo: ')
objeto = Funcionario(nome, cargo)
objeto.salario = 3000
objeto.mostrar_info()
objeto.mostrar_salario()
```

```
from encap2 import Funcionario

class Subclasse:

    def metodoSubclasse(self):
        pass

nome = input('Insira o nome: ')
cargo = input('Insira o cargo: ')
objeto = Funcionario(nome, cargo)
objeto.salario = 1500
objeto.nome = 'ana'
objeto.cargo = 'Gerente'
objeto.mostrar_info()
objeto.mostrar_salario()
```

Encapsulamento - Protected

Exemplo2: o comando `protected` utiliza o (**`_underscore`**) ele permite que a classe Pai tenha acesso, assim como suas subclasses.

Classes ou pacotes que não são derivadas da classe Pai não tem acesso aos métodos ou atributos protegidos.

Método protegido

```
class Funcionario:
    def __init__(self, nome, cargo):
        self.nome = nome
        self.cargo = cargo

        self._salario = 0.0

    def mostrar_info(self):
        print(self.nome, self.cargo)

    def _mostrar_salario(self):
        print(self._salario)

objeto = Funcionario('ana', 'Programadora')
objeto._salario = 1500
objeto._mostrar_salario()
```

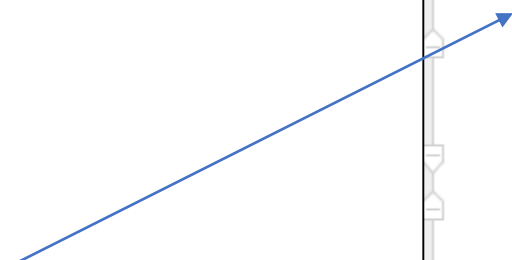
Encapsulamento Private – Acessando dados privados

Dados privados só pode ser visualizados dentro da própria classe que foi criada

Atributo privado não aparece fora da classe

```
class Funcionario:
    def __init__(self, nome, cargo):
        self.nome = nome
        self.cargo = cargo

        self.__salario = 100
```



f	nome	Funcionario
f	cargo	Funcionario
m	mostrarDados(self)	Funcionario
m	mostrar_info(self)	Funcionario
m	__init__(self, nome, cargo)	Funcionario
m	__str__(self)	object
f	__annotations__	object
p	__class__	object
m	__delattr__(self, name)	object
f	__dict__	object
m	__dir__(self)	object
m	__eq__(self, o)	object

objeto.
objeto.

Encapsulamento Private – Acessando dados privados

Dados privados só pode ser visualizados dentro da própria classe que foi criada

Não posso modifica-lo fora da classe

Método privado

Para conseguir ver o valor de um atributo privado, ele deverá está dentro de um método público.

```
class Funcionario:
    def __init__(self, nome, cargo):
        self.nome = nome
        self.cargo = cargo

        self.__salario = 100

    def mostrar_info(self):
        print(self.nome, self.cargo)

    def __mostrar_salario(self):
        print(self.__salario)

    def mostrarDados(self):
        print(self.__salario)

objeto = Funcionario('ana', 'Programadora')
objeto.mostrarDados()
```

Método super() do construtor

O método `super()` é uma referência a superclasse e também consegue herdar os atributos e métodos privados da superclasse.

```
class Animal:
    def __init__(self, nome_animal, idade_animal):
        self.nome = nome_animal
        self.idade = idade_animal
    def __info(self):
        print(f'Nome:{self.nome}\nIdade:{self.idade} ano(s)')

    def dormir(self):
        print('Animal que gosta muito de dormir')

class teste_super(Antimal):
    def __init__(self, nome_animal, idade_animal, name):
        super().__init__(nome_animal, idade_animal)
        self.name = name

    def __info(self):
        print(f'Nome:{self.nome}\nIdade:{self.idade} ano(s) {self.name}')
    def mostrar(self):
        self.__info()

objeto = teste_super('animal 1', 2, 'gatinho')
objeto.mostrar()
```


REFERÊNCIAS

ACERVO LIMA. **Construtores em Python.2022**. Disponíveis em< <https://acervolima.com/construtores-em-python/>>

TREINAWEB. **Herança no Python**. 2022. Disponíveis em< <https://www.treinaweb.com.br/blog/utilizando-heranca-no-python#:~:text=A%20Heran%C3%A7a%20%C3%A9%20um%20conceito,haja%20muita%20repeti%C3%A7%C3%A3o%20de%20c%C3%B3digo.> >

WIKILIVROS. **O que é encapsulamento**. 2020. Disponível em: https://pt.wikibooks.org/wiki/Python/Conceitos_b%C3%A1sicos/>