Classes e Objetos

Estruturas de Dados

Universidade de Brasília
Departamento de Ciência da Computação

Exercício

Dado que uma pessoa vai ao mercado e deseja usar seu vale-alimentação para pagar por suas compras, ela precisa saber quanto será efetivamente descontado do seu vale. No entanto, vale ressaltar que apenas produtos alimentícios são cobrados do vale-alimentação.

Escreva uma função calcular_desconto_vale que receba três listas: nomes dos produtos, valor de cada produto, tipo do produto (alimentício ou não).

A função deve retornar o valor total que será descontado do vale-alimentação e imprimir o extrato do gasto no vale.

Exercício

```
def calcular_desconto_vale(nomes_produtos, valores_produtos, tipos_produtos):
    total = 0
    print("Extrato do Vale-Alimentação:")

    for nome, valor, tipo in zip(nomes_produtos, valores_produtos, tipos_produtos):
        if tipo == "ALIMENTICIO":
            total += valor
            print(f"{nome}: R${valor:.2f}")

    print(f"Total a ser descontado: R${total:.2f}")
    return total
```

Variáveis Dependentes

No exercício anterior, vemos que uma lista depende da outra.

```
nomes_produtos = ['Arroz', 'Feijao', 'Desinfetante']
valores_produtos = [12.34, 13.65, 7.45]
tipo_produtos = ['ALIMENTICIO', 'ALIMENTICIO', 'N_ALIMENTICIO']
```

Agregados Heterogêneos

O método zip nos permite fazer um agregado heterogêneo, representando um *produto*.

```
nomes_produtos = ['Arroz', 'Feijao', 'Desinfetante']
valores_produtos = [12.34, 13.65, 7.45]
tipo_produtos = ['ALIMENTICIO', 'ALIMENTICIO', 'N_ALIMENTICIO']
```

Agregados Heterogêneos

Assim, temos ao nosso dispor:

```
(
    ('Arroz', 12.34, 'ALIMENTICIO'),
    ('Feijao', 13.65, 'ALIMENTICIO'),
    ('Desinfetante', 7.45, 'N_ALIMENTICIO')
)
```

Escalabilidade

Mas e se de fato esse código fosse utilizado em um mercado de grande porte?

Escalabilidade

Mas e se de fato esse código fosse utilizado em um mercado de grande porte?

Imagine fazer uma operação *zip* de vários produtos, de vários caixas, em um mesmo computador/servidor

Escalabilidade

E se trabalhássemos com as listas separadas, sem zip, administrando apenas pelo índice em comum?

- Associação correta dependente do programador;
- Manipulação incorreta da lista invalida todos os dados;

Escalabilidade

E se trabalhássemos com as listas separadas, sem zip, administrando apenas pelo índice em comum?

- Associação correta dependente do programador;
- Manipulação incorreta da lista invalida todos os dados;

E mais...

E se eu precisasse ordernar os produtos por valor?

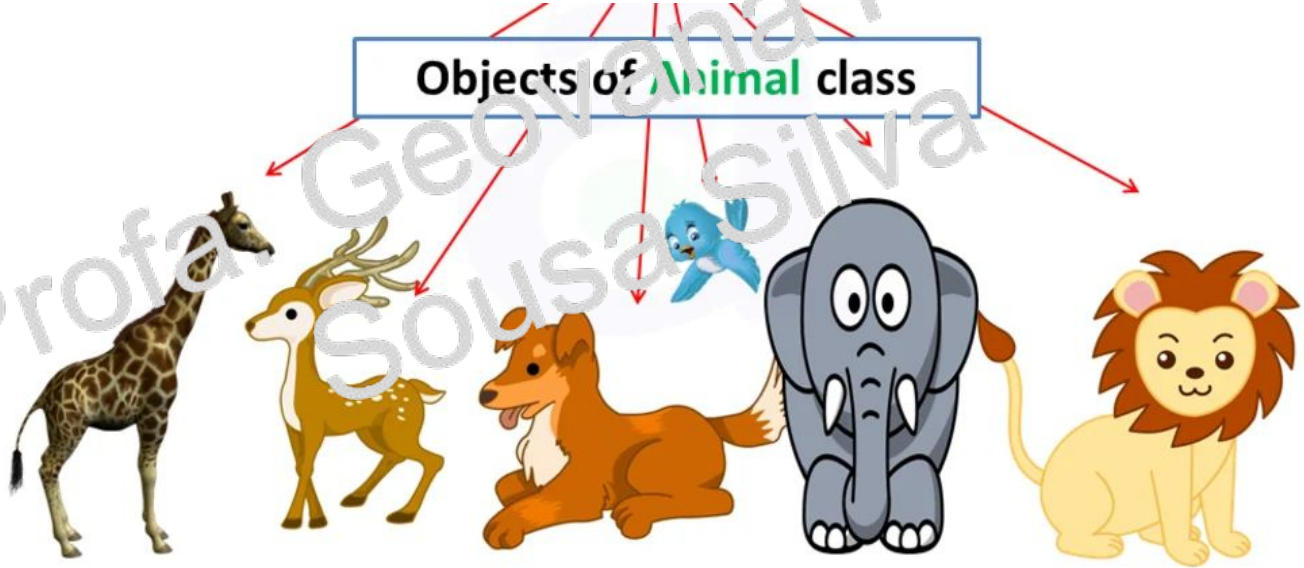
Classes e Objetos

Surgem para ajudar na manipulação de dados dependentes:

- Os objetos são adequados para modelar entidades ou conceitos do mundo real;
- Fornecem uma maneira de encapsular dados e comportamento em uma única unidade;
- Os objetos podem melhorar a organização, legibilidade e a reutilização do código;

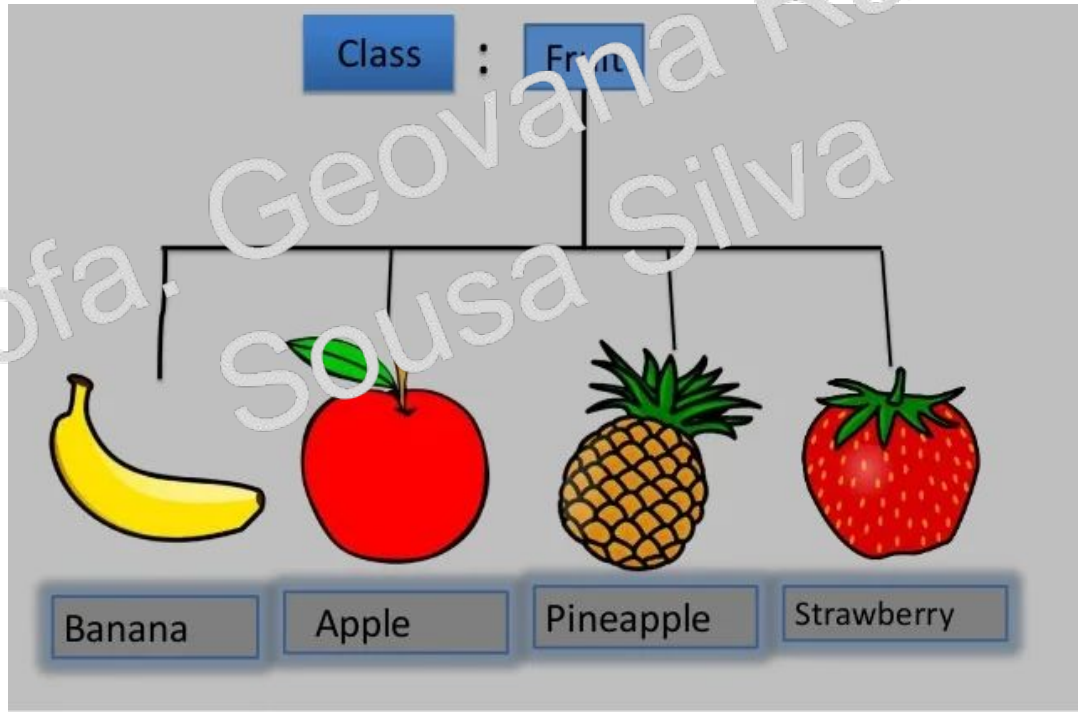
Classes e Objetos

Objetos são instâncias de classes.



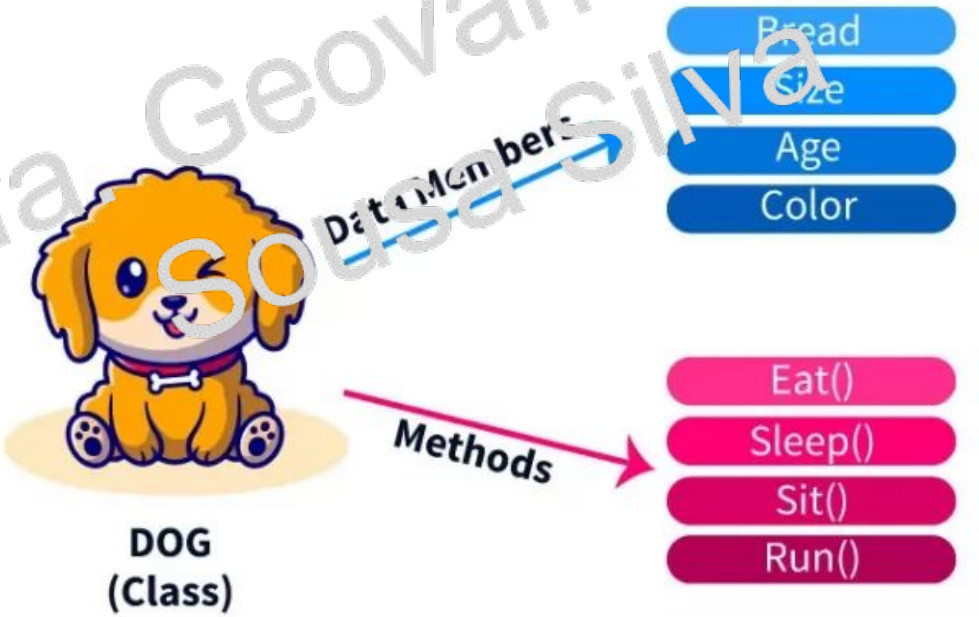
Classes e Objetos

Objetos são instâncias de classes.

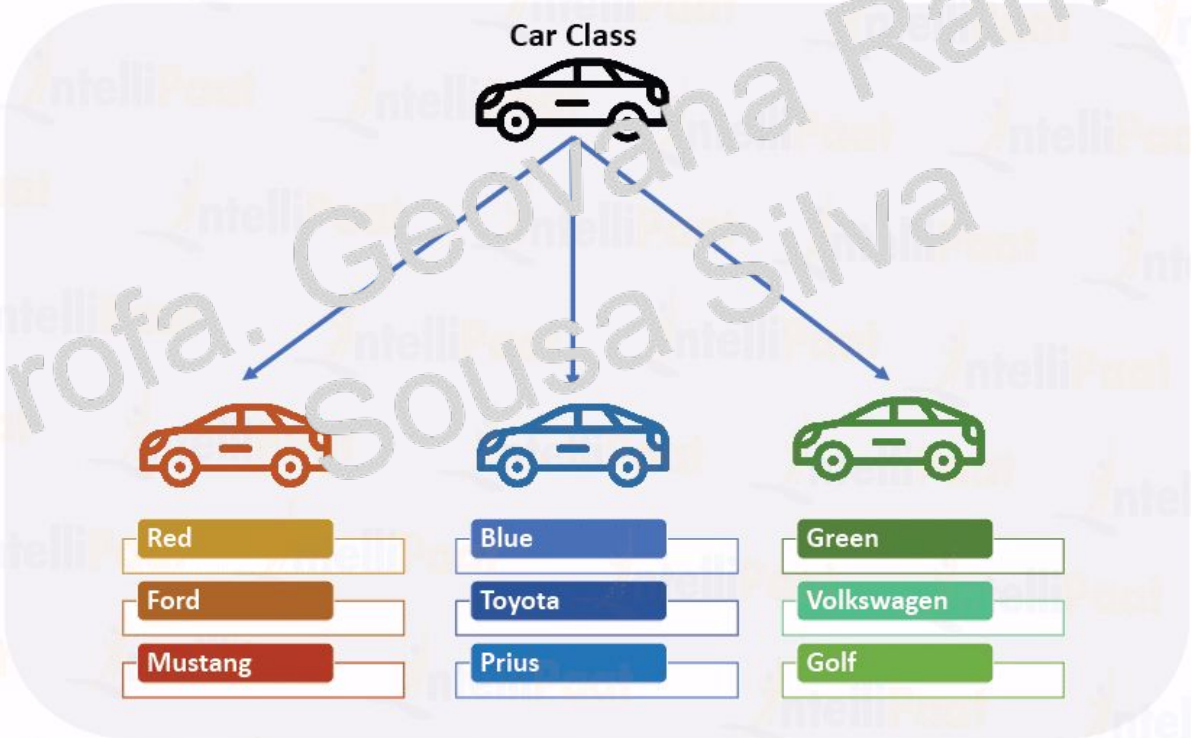


Classes e Objetos

Classes **definem** o que a entidade deve ter, e os objetos **concretizam** entidades.

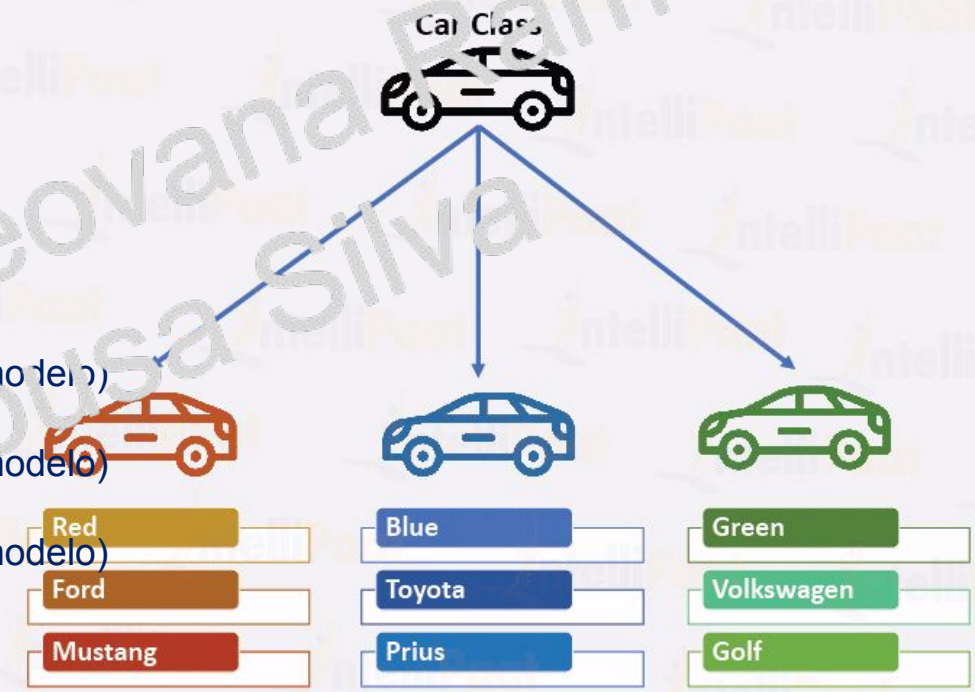


Classes e Objetos



Classes e Objetos

```
carro_1 = Carro('Red', 'Ford', 'Mustang')

carro_2 = Carro('Blue', 'Toyota', 'Prius')

carro_3 = Carro('Green', 'Volks', 'Golf')


print(carro_1.cor, carro_1.marca, carro_1.modelo)

print(carro_2.cor, carro_2.marca, carro_2.modelo)

print(carro_3.cor, carro_3.marca, carro_3.modelo)
```



Entidade Produto

O exercício inicial:

```
nomes_produtos = ['Arroz', 'Feijao', 'Desinfetante']
valores_produtos = [12.34, 13.65, 7.45]
tipo_produtos = ['ALIMENTICIO', 'ALIMENTICIO', 'N_ALIMENTICIO']
```

Seria:

```
produto_1 = Produto('Arroz', 12.34, 'ALIMENTICIO'),
produto_2 = Produto('Feijao', 13.65, 'ALIMENTICIO'),
produto_3 =  Produto('Desinfetante', 7.45, 'N_ALIMENTICIO')
```

Entidade Produto

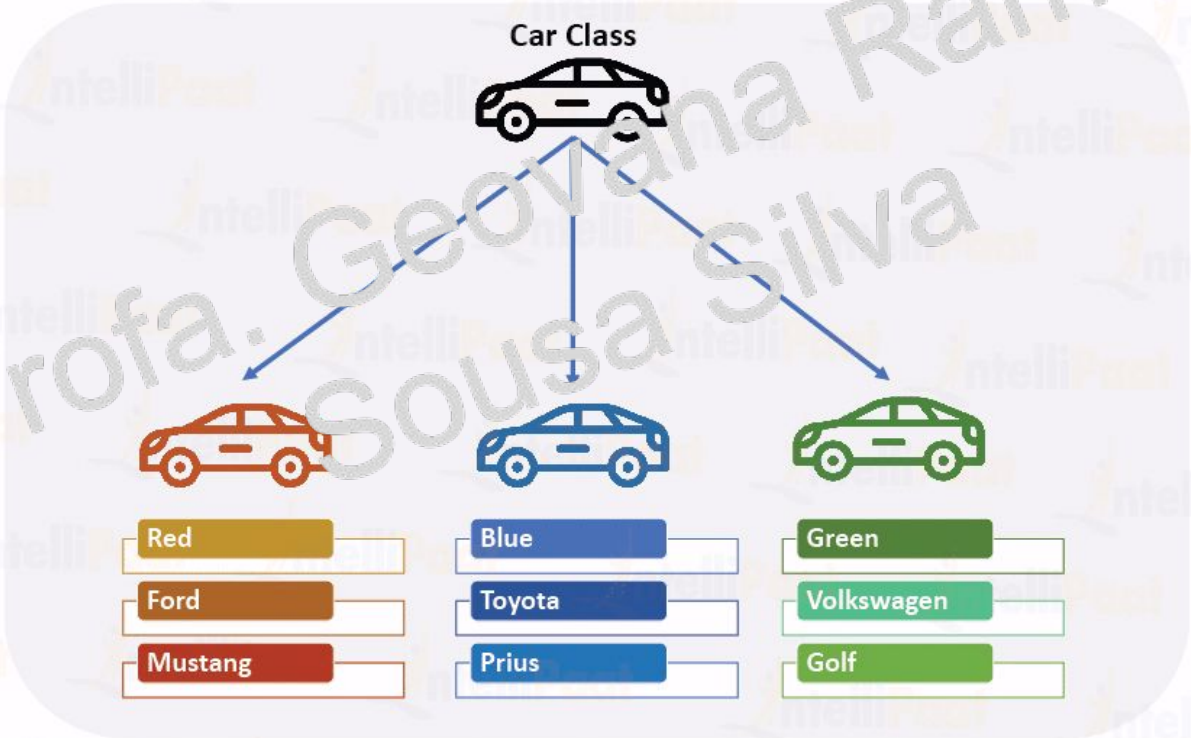
Em vez de três, podemos agora lidar com apenas uma lista:

```
produtos = [produto_1, produto_2, produto_3]
```

E podemos acessar seus atributos:

```
for produto in produtos:
    print(produto.nome, produto.valor, produto.tipo)
```

Definindo Classes



Definindo Classes

```
class Carro:

    def __init__(self, cor, marca, modelo):
        self.cor = cor
        self.marca = marca
        self.modelo = modelo
        self.ligado = False

    def ligar_carro(self):
        self.ligado = True
        print("Carro ligado!")

meu_carro = Carro(cor="vermelho", marca="Toyota", modelo="Corolla")
meu_carro.ligar_carro()
```

Exercício

Faça uma classe para a entidade produto do exercício da aula com os atributos nome, valor, tipo.

Adicione a classe um método que retorna se o produto é alimentício ou não e um método que print uma linha do extrato como: "Arroz R$ 12,43".

Exercício

```
class Produto:

    def __init__(self, nome, valor, tipo):
        self.nome = nome
        self.valor = valor
        self.tipo = tipo

    def eh_alimentício(self):
        return self.tipo == "ALIMENTÍCIO"

    def imprime_no_extrato(self):
        return f"{self.nome}\t\tR$ {self.valor:.2f}"

produto_1 = Produto(nome="Arroz", valor=12.43, tipo="alimentício")
```

Exercício

```
def calcular_desconto_vale(produtos):
        total = 0
        print("Extrato do Vale-Alimentação:")

        for produto in produtos:
            if produto.eh_alimenticio():
                total += produto.valor
                print(produto.imprime_no_extrato())

        print(f"Total a ser descontado: R$ {total:.2f}")
        return total
```

Resumo

Informações podem ser representadas na memória como diferentes **tipos de dados**.

Armazenar/organizar os dados pode **facilitar** acesso e modificações.

Classes e objetos podem ajudar a desenvolver sistemas mais complexos.