FUNDAMENTOS DA LINGUAGEM PYTHON

AGENDA

- Breve Histórico, Versões e Modelo de Dados;
- Identificadores e Tipos de Dados;
- Comando Condicional (*if*);
- Função Range e Comando de Controle (*for*);
- Comando de Controle (*while*);
- Funções, Objetos e Classes.

BREVE HISTÓRICO

- A linguagem **Python** foi criada no início dos anos 1990, por **Guido van Rossum**, matemático e programador.

- Possui as seguintes características:
  - ✓ **Portabilidade** (Unix, Linux, Windows (todas as versões), macOS, BeOS, VMS, entre outras);
  - ✓ **Código livre** (*opensource*);
  - ✓ **Simplicidade com robustez;**
  - ✓ **Fácil de aprender;**
  - ✓ **Grande aplicabilidade.**

Fonte: Banin (2018)

VERSÕES DO PYTHON

- Há duas versões de **Python**, a **2.x** e **3.x**.

- No entanto, estas duas versões possuem diferenças, a ponto de a versão **3.x** representar uma quebra de compatibilidade em relação à versão **2.x**.

- **E agora? Qual versão deve ser escolhida?**

  - ✓ **RESPOSTA: "Python 2.x é legado, Python 3.x é o presente e o futuro da linguagem".**

Fonte: Banin (2018)

MODELO DE DADOS DO PYTHON

- O modelo de dados do Python adota como paradigma que todo dado em Python é representado por um objeto.

- **Todo objeto Python tem três aspectos**: um identificador, um tipo e um conteúdo.

  - **Identificador**: é o nome que o objeto tem no programa.

  - **Tipo**: determina o tipo dos dados, por exemplo, um número inteiro, um texto, e também as operações que são suportadas por esse tipo.

  - **Conteúdo**: é o valor (ou conjunto de valores) armazenado.

Fonte: Banin (2018)

IDENTIFICADORES

- **Identificador** é um nome dado para variáveis, constantes e funções.

- **Em Python**, um identificador pode conter apenas:
  - ✓ Letras de A a Z (maiúsculas e minúsculas);
  - ✓ Dígitos de 0 a 9;
  - ✓ Símbolo *underscore* _

- Deve começar obrigatoriamente com uma letra ou *underscore*.

- Não é permitido que um identificador comece com um número. Não deve conter espaços, caracteres especiais ou letras acentuadas.

Fonte: Banin (2018)

IDENTIFICADORES

- **ATENÇÃO:** Python faz diferenciação entre maiúsculas e minúsculas.
- Portanto, **fique atento**: nota é diferente de NOTA.
- A boa prática de programação diz que um identificador deve ser não apenas válido, mas também significativo/adequado.
- Um identificador deve descrever precisamente o dado que a variável armazena.

Fonte: Banin (2018)

OBJETOS IMUTÁVEIS E MUTÁVEIS

- Em **Python** os objetos podem ser classificados como **imutáveis** ou **mutáveis**.
- Um objeto imutável tem conteúdo fixo, ou seja, não pode ser alterado **sem que o objeto seja reconstruído**.
- Os tipos **int**, **float**, **string**, e **tupla** **são imutáveis**, de modo que, quando um novo conteúdo for atribuído ao objeto, sua instância anterior é removida, e uma nova instância, criada.
- Os tipos **lista**, **conjunto** e **dicionário** **são mutáveis**, de modo que podem ter seu conteúdo alterado, sem que sua instância seja recriada.

Fonte: Banin (2018)

TIPOS DE DADOS

1. Dados Simples
2. Dados Estruturados

Dados Simples:

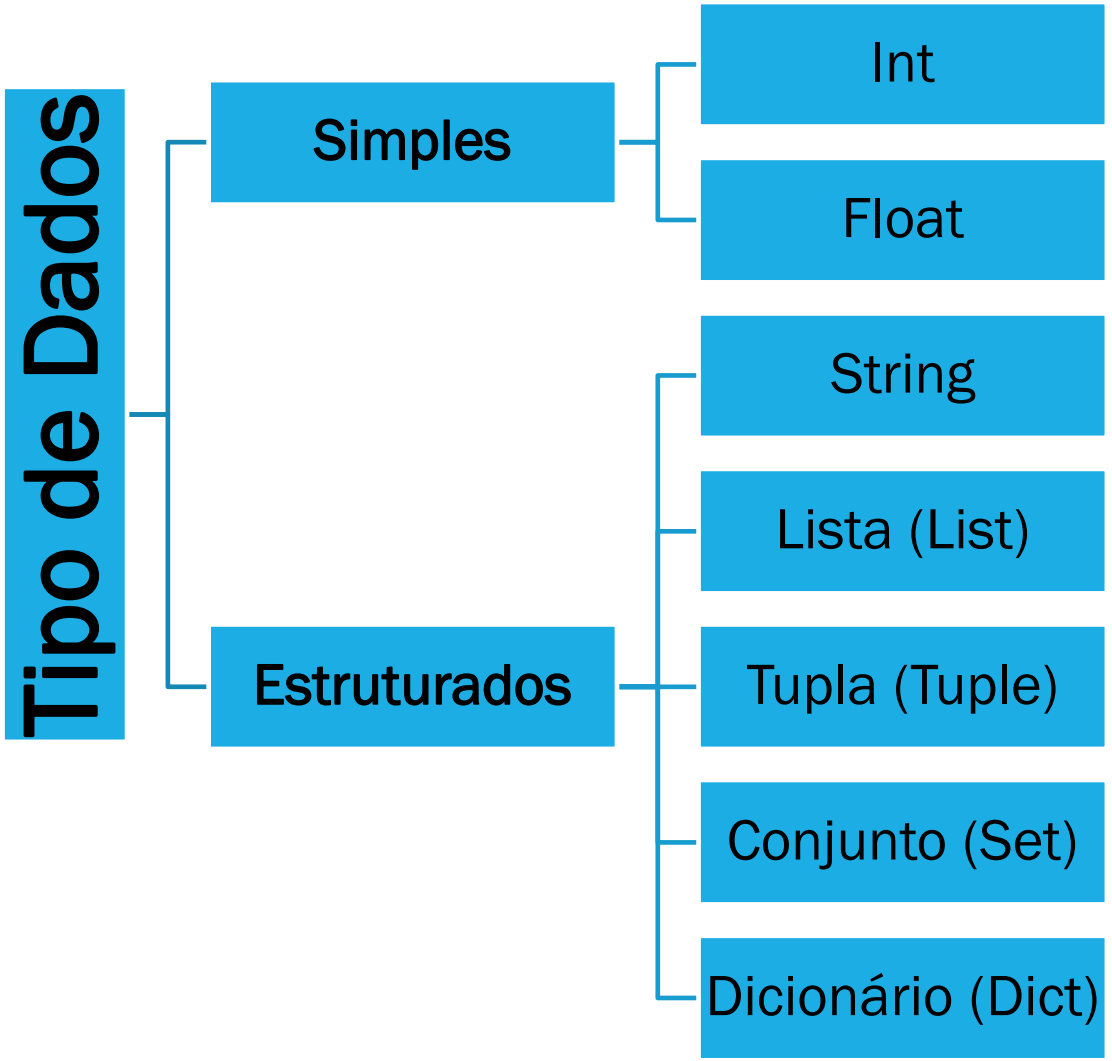
- ✓ **Integer (int):** armazena números inteiros positivos, zero ou negativos. **EXEMPLO: Idade = 21**.
- ✓ **Float (float):** armazena números reais positivos ou negativos, além do zero. O separador decimal é o ponto “.”. **EXEMPLO: Peso = 56.5**.

Dados Estruturados:

- ✓ Os tipos estruturados são compostos, ou seja, seu conteúdo é constituído por outros elementos.
- ✓ Assim sendo, tais tipos representam agregados de objetos que podem ser acessados e manipulados em conjunto ou isoladamente.

Fonte: Banin (2018)

TIPOS DE DADOS



TIPOS DE DADOS ESTRUTURADOS

TIPO	DESCRIÇÃO
String	Contêm qualquer sequência de letras, algarismos e caracteres especiais. Serve para armazenar quaisquer dados aos quais não se realizam operações aritméticas. As strings em Python, permitem acessar. É um tipo imutável.
Lista (list)	É um conjunto de itens entre colchetes e separados por vírgulas. Os itens não precisam ser todos do mesmo tipo e podem ser acessados e manipulados individualmente, todos de uma vez ou em grupos. É um tipo mutável.
Tupla (tuple)	Uma tupla caracteriza-se por ser um conjunto de itens entre parênteses e separados por vírgulas. É semelhante a uma lista, porém, com a principal característicade ser imutável. Ou seja, quando uma tupla é criada não é possível adicionar, alterar ou remover seus elementos. É um tipo imutável.

Fonte: Banin (2018)

TIPOS DE DADOS ESTRUTURADOS

TIPO	DESCRIÇÃO
Conjunto (set)	É uma coleção não ordenada de elementos **não duplicados** e caracteriza-se por itens entre chaves e separados por vírgulas. Tem diversos usos possíveis e suporta operações matemáticas típicas de conjuntos, como união, interseção e diferença, muito úteis em alguns algoritmos. É um tipo mutável.
Dicionário (dict)	É uma coleção de pares "**chave:valor**" não ordenados, com a obrigatoriedade de que as chaves sejam únicas (não duplicadas). Enquanto as listas e tuplas são indexadas por um número inteiro, os dicionários são indexados pela chave associada ao valor. Caracteriza-se por itens entre chaves, com chave, valor e separados por vírgulas. É um tipo mutável.

Fonte: Banin (2018)

COMANDO CONDICIONAL (IF)

- É comum em um algoritmo a tomada de decisões baseadas em valores contidos em objetos.
- Uma das maneiras de se conseguir isso é utilizar o comando condicional **if-else**.
- Para utilizá-lo é necessário criar uma condição cujo resultado será **falso** ou **verdadeiro**, e em função desse resultado o programa executará diferentes comandos em cada caso.
- **INDENTAÇÃO**:
  - ✓ em **Python**, todo conjunto de comandos subordinados deve estar indentado em relação ao seu comando proprietário.

Fonte: Banin (2018)

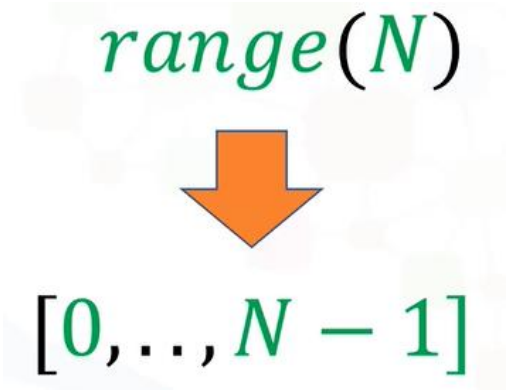
FUNÇÃO RANGE

- A função **range** é um gerador de sequências imutável de números inteiros.
- Segue uma regra de progressão aritmética definida pelos parâmetros utilizados.
- É utilizada com o comando **for**, que será visto a seguir.
- A sintaxe possui duas opções:
  - **class range (stop)**: é fornecido apenas o limite final, assumindo-se que o valor inicial é 0 e o incremento é 1.
  - **class range (start, stop, [step])**: são fornecidos o valor inicial e o limite final. Opcionalmente, pode ser fornecido também um terceiro parâmetro que é o passo.
  - **Os parâmetros devem, ser números inteiros**.

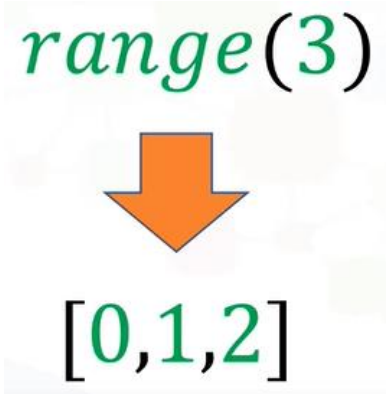
Fonte: Banin (2018)

FUNÇÃO RANGE

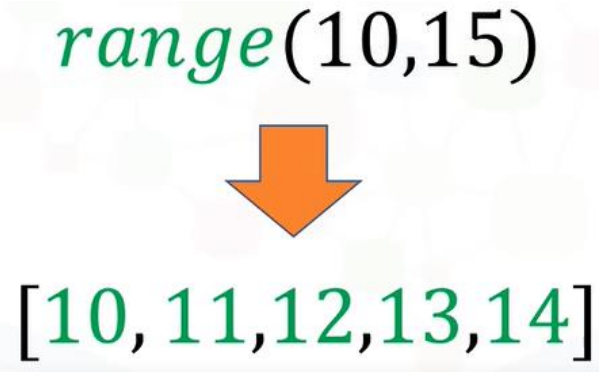
- A função **range** produz uma sequência ordenada como uma lista “i”.



- Se o **range** for um número inteiro positivo, a saída será uma sequência que contém o mesmo número de elementos que a entrada, mas iniciada em zero. **EXEMPLO**:



- Se se o **range** tiver duas entradas a saída é uma sequência que começa na primeira entrada. Em seguida, a sequência itera até, mas não incluindo o segundo número. **EXEMPLO**:



Fonte: IBM

COMANDO DE REPETIÇÃO (FOR)

- O comando **for** é uma opção disponível em **Python** para a construção de laços de repetição.

- Esse comando é utilizado para a iteração sobre os tipos sequenciais.

- Para cada valor atribuído ao objeto de controle, os comandos contidos no <*bloco de comandos*> serão executados. Desse modo, o laço será executado um certo número de vezes, que depende exclusivamente da quantidade de elementos contidos no tipo sequencial.

```
lista = [15, 8, 12, 19, 5]
for num in lista:
    print(num)
```

Fonte: Banin (2018)

COMANDO DE REPETIÇÃO (FOR)

Exemplo 1

Comando for em Java

```
for (int i = 0; i < 10; i++)
    ...
```

Equivalente em Python

```
for i in range(10):
    ...
```

Exemplo 2

Comando for em Java

```
for (int i = 10; i < 15; i++)
    ...
```

Equivalente em Python

```
for i in range(10, 15):
    ...
```

Exemplo 3

Comando for em Java

```
for (int i = 10; i < 50; i += 5)
    ...
```

Equivalente em Python

```
for i in range(10, 50, 5):
    ...
```

Fonte: Puc-MInas

COMANDO DE REPETIÇÃO (WHILE)

- Para implementar determinada lógica, é necessário repetir um trecho de programa um certo número de vezes.
- Para isso utiliza-se o comando de repetição **while**.
- Com o **while**, é possível repetir um conjunto de comandos enquanto uma condição especificada for verdadeira.
- O comando **while** em **Python** tem a seguinte construção básica: "**enquanto a condição for verdadeira, execute o conjunto de comandos**".
- Todo laço, para ser implementado, requer quatro elementos: **inicialização, condição, iteração e o corpo**.
- Os três primeiros dizem respeito à construção e ao controle do laço.

Fonte: Banin (2018)

COMANDO DE REPETIÇÃO (WHILE)

- ✓ **Inicialização**: constitui-se de todo código necessário para determinar a situação inicial do laço.
- ✓ **Condição**: é uma expressão lógica, que pode ser simples ou composta, cujo resultado é avaliado em falso ou verdadeiro, que determina se o laço termina ou prossegue.
- ✓ **Iteração**: é todo comando que modifica os objetos envolvidos na condição, a cada execução do laço.
- ✓ **Corpo do laço**: é constituído pelos comandos que devem ser executados repetidas vezes.

```
while (condição):
    bloco_comandos
```

```
# Utilizando o loop while

datas = [1982, 1980, 1973, 2000]

i = 0
ano = 0

while(ano != 1973):
    ano = datas[i]
    i = i + 1
    print(ano)

print("Levou",i," repetições para sair do loop.")
```

Fonte: Puc-MInas

FUNÇÕES

- **Função** é um bloco de código, ao qual se atribui um nome identificador que executa certa tarefa, e pode retornar algum resultado.

- As funções podem ser desenvolvidas, testadas e agrupadas em bibliotecas de modo a ficar disponíveis para uso em mais de um programa diferente, sempre que o programador necessite delas.

- Se houver parâmetros, estes são devidamente carregados com os valores passados na chamada da função, antes de se executar o primeiro comando interno.

- Após a execução de todos os comandos internos, a função ao final retorna para a instrução imediatamente seguinte ao ponto em que ocorreu a chamada.

Fonte: Banin (2018)

FUNÇÕES

❖ Em Python, uma função é definida por meio de um cabeçalho que contém quatro elementos:

- ✓ **A palavra reservada def**
- ✓ **Um nome válido**: o nome pode ser qualquer identificador válido.
- ✓ **Parâmetros entre parênteses**: devem ser fornecidos os parâmetros que a função receberá, caso existam.
- ✓ **O caractere dois pontos ":"** que indica ao interpretador o término do cabeçalho.

❖ O cabeçalho é seguido por um bloco de comandos indentados que constitui o corpo da função.

Fonte: Banin (2018)

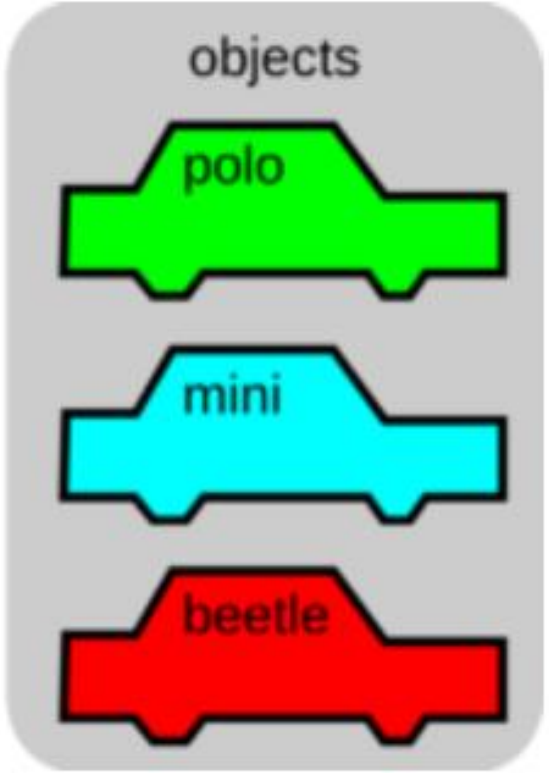
FUNÇÕES

❖ **Retorno das funções:**

- ✓ Em todas as linguagens é possível existir uma função que não retorna qualquer valor, bem como é possível retornar um ou muitos valores. **Em Python isso também é assim.**
- ✓ Para que uma função tenha retorno basta utilizar a instrução **return**, que produz dois efeitos: retorna o objeto que é colocado à sua frente e encerra a função imediatamente.
- ✓ Em funções que não têm retorno a instrução **return** não é utilizada. Nestes casos, uma vez chamada, sua execução prosseguirá desde a primeira até a última instrução de seu bloco de código.

Fonte: Banin (2018)

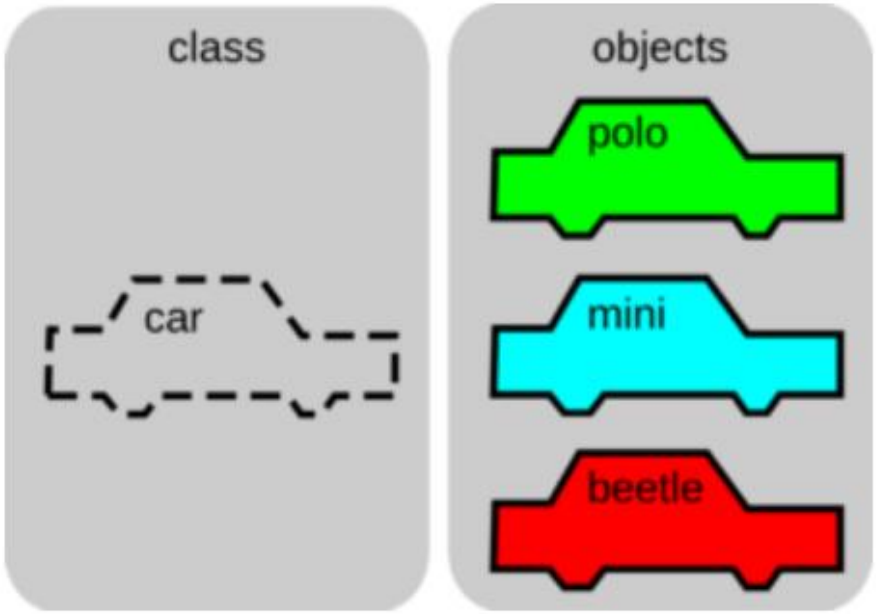
PROGRAMAÇÃO ORIENTADA A OBJETOS



- Um carro tem características, como um motor 2.0, azul escuro, quatro portas, câmbio automático. etc.
- Ele também possui comportamentos, como acelerar, desacelerar, acender os faróis, buzinar e tocar música.
- O carro é um *objeto*, onde as características são seus *atributos* e seus comportamentos são ações ou *métodos*.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

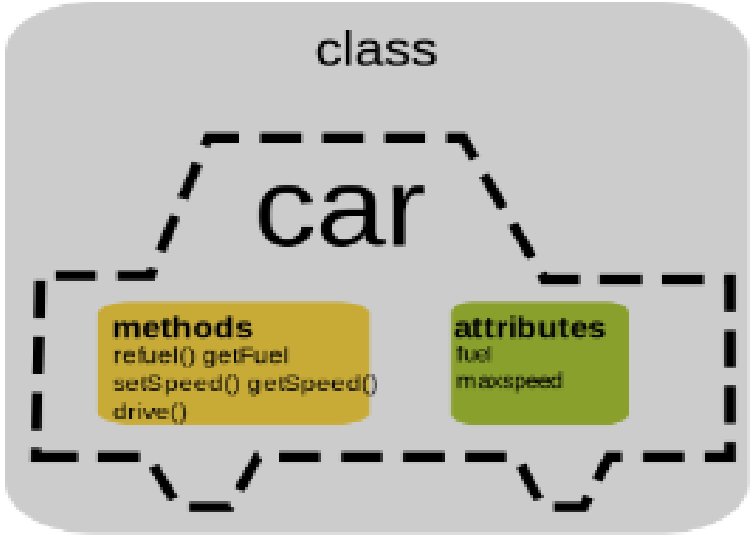
PROGRAMAÇÃO ORIENTADA A OBJETOS



- Pode-se dizer então que um objeto pode ser classificado (isto é, este objeto pertence à uma classe) como um carro.
- E que o carro do slide anterior nada mais é que uma instância dessa classe chamada "carro".
- Uma classe é um conjunto de características (atributos) e ações (métodos) que definem o conjunto de objetos pertencentes à essa classe.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

PROGRAMAÇÃO ORIENTADA A OBJETOS



- Classe é um conceito abstrato, um molde, que se torna concreto por meio da criação de um objeto que é a *instanciação da classe*.
- Aplicação do molde (classe) para criar um objeto.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

CLASSES E OBJETOS

Uma classe de objetos em Python é composta por:

- ✓ Construtor
- ✓ Atributos
- ✓ Variáveis
- ✓ Métodos

EXEMPLO DE CLASSE EM PYTHON

```python
class Carro:
    def __init__(self, modelo):
        self.modelo = modelo;
        self.velocidade = 0

    def acelerar(self):
        # Codigo para acelerar o carro

    def frear(self):
        # Codigo para frear o carro

    def acenderFarol(self):
        # Codigo para acender o farol do carro
```

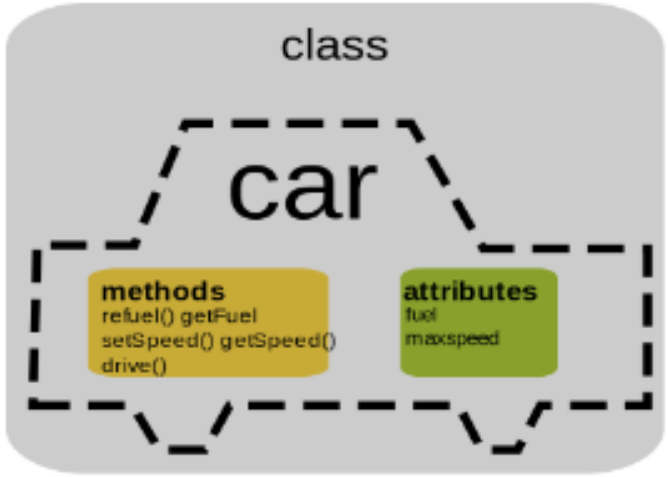
Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

PROGRAMAÇÃO ORIENTADA A OBJETOS

- As duas bases da POO são os conceitos de **classe** e **objeto**.
- Desses conceitos, derivam outros conceitos extremamente importantes ao paradigma:
  - ✓ **Encapsulamento**;
  - ✓ **Herança**;
  - ✓ **Polimorfismo**.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

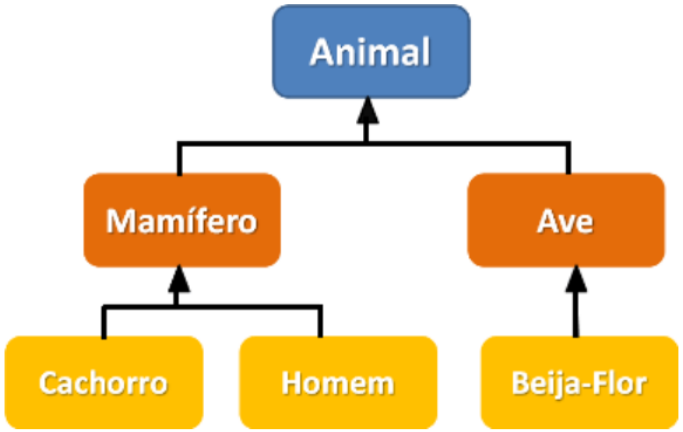
ENCAPSULAMENTO

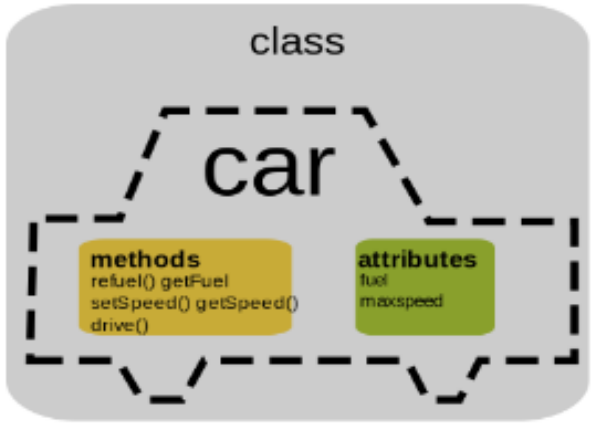


- No exemplo do carro, só sabemos que para acelerar, é necessário pisar no acelerador e de resto o objeto sabe como executar essa ação sem expor como o faz.
- Isso significa que a aceleração do carro está encapsulada, pois sabemos o que ele vai fazer ao executarmos esse método, mas não sabemos como.
- O mesmo vale para atributos, não sabemos como o carro calcula a velocidade, só precisamos saber que ele vai nos dar a velocidade certa.
- O encapsulamento impede o vazamento de escopo, onde um atributo ou método é visível por outro objeto ou classe.
- Ler ou alterar um atributo encapsulado pode ser feito a partir de getters e setters.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

HERANÇA





- Quando dizemos que uma classe A é um tipo de classe B, dizemos que a classe A herda as características da classe B e que a classe B é mãe da classe A, estabelecendo uma relação de herança entre elas.
- No caso do carro, dizemos então que um Honda Fit "Cross" é um tipo de Honda Fit, e o que muda são alguns atributos (paralama reforçado, altura da suspensão etc), e um dos métodos da classe (acelerar, pois agora há tração nas quatro rodas).
- Mas todo o resto permanece o mesmo, e o novo modelo recebe os mesmos atributos e métodos do modelo clássico.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

EXEMPLO DE HERANÇA EM PYTHON

Classe mãe

```python
class Carro:
    def __init__(self, modelo):
        self.modelo = modelo;
        self.velocidade = 0

    def acelerar(self):
        # Codigo para acelerar o carro

    def frear(self):
        # Codigo para frear o carro

    def acenderFarol(self):
        # Codigo para acender o farol do carro
```

```python
# As classes dentro do parênteses são as classes mãe da classe sendo definida
class HondaFit(Carro):

    def __init__(self, mecanismoAceleracao):
        modelo = "Honda Fit"
        # chama o construtor da classe mãe, ou seja, da classe "Carro"
        super().__init__(self, modelo, mecanismoAceleracao)
```

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

POLIMORFISMO

"Polimorfismo" vem do grego poli = muitas, morphos = forma).

- Vamos dizer que um dos motivos para comprar um carro é a qualidade do som dele que é feita via rádio ou bluetooth, enquanto que um carro mais antigo, era feita apenas via cartão SD e pendrive.
- Em ambos os carros está presente o método "**tocar música**" mas, como o sistema de som é diferente, a forma como o carro toca as músicas é diferente.
- Neste caso, o método "**tocar música**" é uma forma de polimorfismo, pois dois objetos, de duas classes diferentes, têm um mesmo método que é implementado de formas diferentes.
- Ou seja, um método possui várias formas, várias implementações diferentes em classes diferentes, mas que possuem o mesmo efeito.

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

EXEMPLO DE POLIMORFISMO EM PYTHON

```
def main():
    moto = Moto("Yahama XPTO-100", MecanismoDeAceleracaoDeMotos())
    carro = Carro("Honda Fit", MecanismoDeAceleracaoDeCarros())
    listaAutomoveis = [moto, carro]
    for automovel in listaAutomoveis:
        automovel.acelerar()
        automovel.acenderFarol()
```

Fonte: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=CjwKCAjwvNaYBhA3EiwACgndgt8J6yyQO2kMeaNL9jkXDr2cpPQDKAC8Q3FsQ-LBWmYPaiDopbbUrRoCAFQQAvD_BwE

BIBLIOTECAS PARA LIMPEZA E PRÉ-PROCESSAMENTO DE DADOS

"Os quatro cavaleiros do apocalipse de uma limpeza bem feita"

Fonte: https://medium.com/turing-talks/como-fazer-uma-limpeza-de-dados-completa-em-python-7abc9dfc19b8

BIBLIOTECAS PARA LIMPEZA E PRÉ-PROCESSAMENTO DE DADOS

- **Pandas**: é uma das bibliotecas Python mais conhecidas na Ciência de Dados. Possui ferramentas de análises de dados e estruturas de dados de alta performance e fáceis de utilizar. Possui muitos métodos internos para agrupar, filtrar e combinar dados.

- **Numpy**: é uma biblioteca utilizada principalmente para realizar cálculos e manipulação de *arrays*.

- **Matplotlib e Seaborn**: são bibliotecas utilizadas para visualizações dos dados (diagramas e gráficos). Recursos que são extremamente necessários na análise e interpretação dos dados.

REFERÊNCIAS

BANIN, S. L. Python 3: conceitos e aplicações - uma abordagem didática. São Paulo: Érica, 2018 (e-book).

Site Oficial do Python: www.python.org

Documentação do Python: https://docs.python.org/3/

Python Brasil: https://python.org.br/

DÚVIDAS

OBRIGADA!!!!

Profa. Carla Oliveira

E-mails: carla.olivei@gmail.com e carlaos@unicid.edu.br