

Introdução a Programação Orientada a Objetos

Modulo 02 - Programação Orientada a Objetos

ANTÔNIO DAVID VINISKI

antonio.david@pucpr.br

PUCPR

Agenda

- Programação Orientada a Objetos
- Classes
- Objetos
- Atributos
- Métodos



The background of the slide features a dark blue, stylized illustration of a person's hands and arms working at a desk. The person is using a laptop, with their hands positioned over the keyboard. To the left of the laptop is a small cup of coffee on a saucer. To the right is a notebook with a pen resting on it. The overall aesthetic is clean and professional, with a focus on the workspace.

Programação Orientada a Objetos

Introdução

- POO é caracterizado pelo uso de um conjunto de objetos interagentes, cada qual responsável pelo gerenciamento de seu **estado interno**
 - Os objetos interagem uns com os outros através da troca de mensagens.
 - Cada objeto é responsável pela **inicialização** e **destruição** de seus dados internos.



Programação Orientada a Objetos

- Na programação orientada a objetos:
 - Dados e procedimentos são encapsulados em um só elemento denominado **objeto**.
 - O estabelecimento de comunicação entre objetos (envio e recebimento de mensagens) caracteriza a execução do programa.



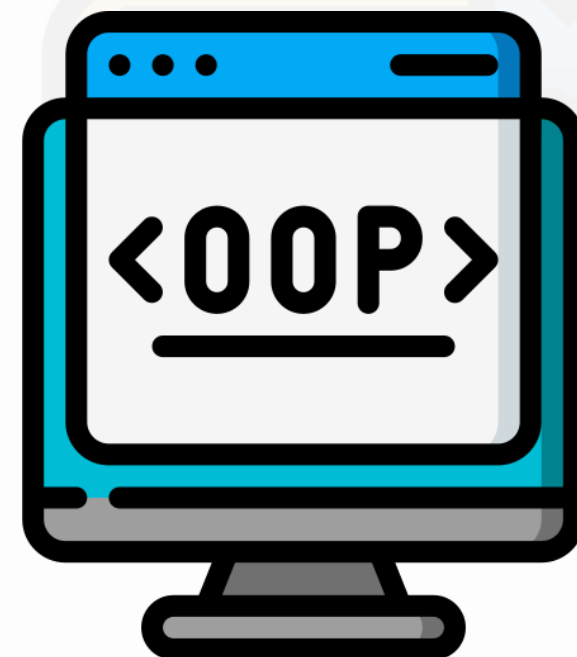
Programação Orientada a Objetos II

- Vantagens da POO em relação à programação estruturada
 - Maior índice de reaproveitamento de código.
 - Maior facilidade de manutenção.
 - Menos código gerado.
 - Maior confiabilidade no código.
 - Maior facilidade de gerenciamento do código (reduz grandes problemas para problemas menores).
 - ...



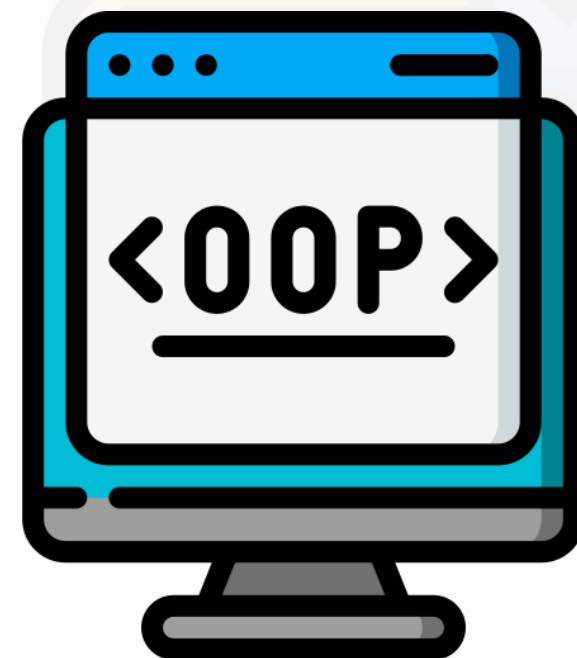
Programação Orientada a Objetos III

- Programação estruturada é baseada na definição de ações (funções)
 - Verbos
- Programação orientada a objetos é focada na definição de coisas ou objetos.
 - Substantivos
 - Mais próximo da percepção que o programador tem do mundo real



Paradigma da Orientação a Objetos (OO)

- Um **paradigma** é uma forma de abordar um problema
- O paradigma da **orientação a objetos** surgiu no fim dos anos 60
- Hoje em dia, praticamente suplantou o paradigma anterior, o paradigma estruturado..



Paradigma OO

- O paradigma OO foi formulado a partir de uma análise biológica
 - “Como seria um sistema de software que funcionasse como um ser vivo?” (Alan Key)



Paradigma OO – Analogia Biológica

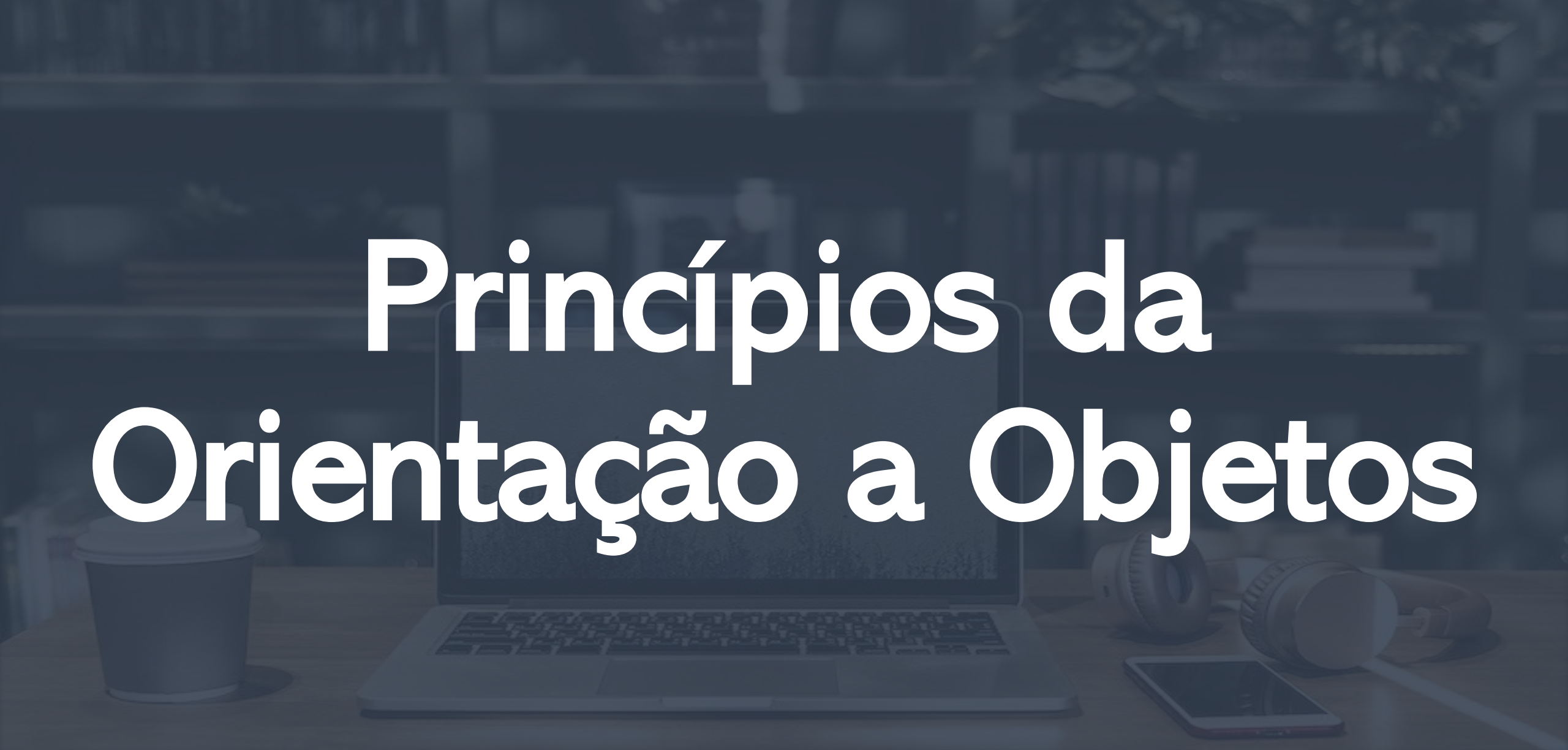
- Cada “**célula**” interage com outras células através do envio de **mensagens** para realizar um objetivo em comum
 - Cada célula se comporta como uma **unidade autônoma**
- De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de **agentes autônomos** que **interagem entre si**
- Com isso, estabeleceu os princípios da orientação a objetos



Análise e Programação OO

- Análise orientada a objetos
 - Exige capacidade de abstração do programador para que seja possível elencar todos os objetos da aplicação
- Programação orientada a objetos
 - Consiste em utilizar objetos computacionais para implementar as funcionalidades de um sistema.





Princípios da Orientação a Objetos

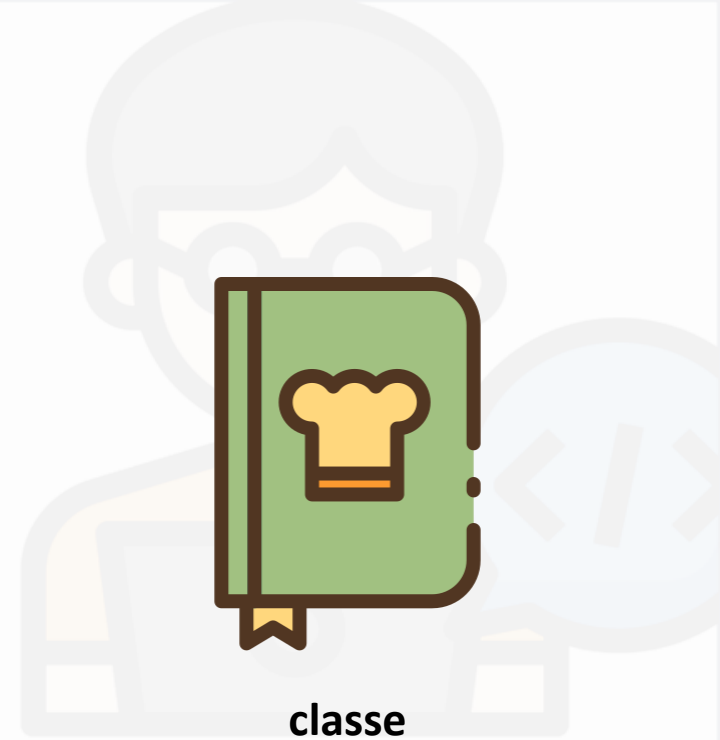
Princípios OO

Tudo é um objeto!



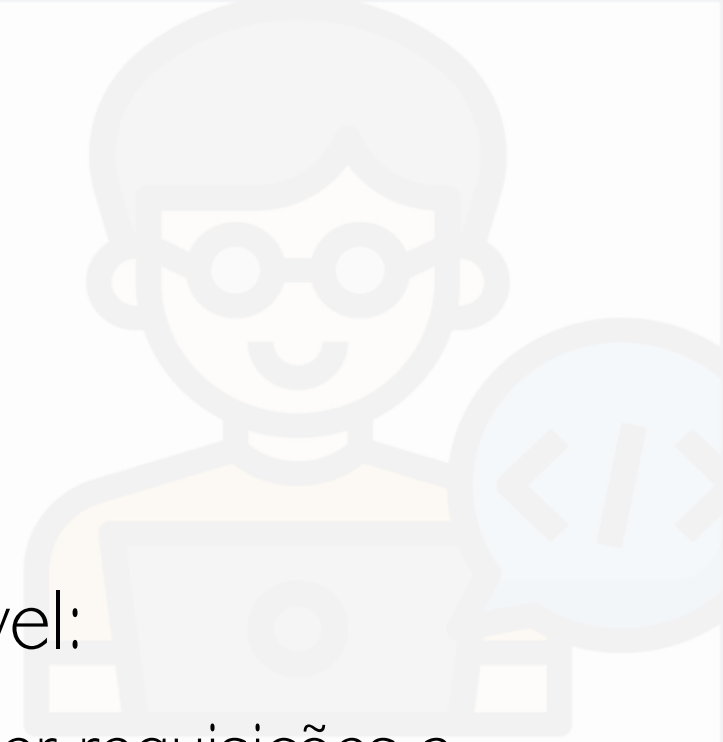
Glossário

- Classe (estrutura de um objeto)
- Objeto (instância de uma classe)
- Instanciação (criação de um objeto)
- Atributos (dados do objeto)
- Métodos (comportamento do objeto)



Princípios OO

- Tudo é um objeto
- Pense em um objeto como uma super variável:
 - O objeto armazena dados, também pode-se fazer requisições a esse objeto, pedindo que ele execute operações
- Elementos conceituais no problema que você está tentando resolver (cachorros, livros, sócios, empréstimos, etc.) como um objeto de um programa



Princípios OO

- Um programa é uma coleção de objetos dizendo uns aos outros o que fazer
- Para fazer uma requisição a um objeto envia-se uma mensagem para este objeto
- Uma mensagem é uma chamada de um método pertencente a um objeto em particular





Classes, Objetos e Instância

Classes e Objetos

- Quando preparamos um bolo, geralmente seguimos uma receita que define os ingredientes e o modo de preparação.
 - E a "receita" no mundo OO recebe o nome de **classe**.
 - Ou seja, antes de criarmos um **objeto**, definiremos uma classe.
 - O **objeto** representa o bolo finalizado, seguindo a receita pré-definida



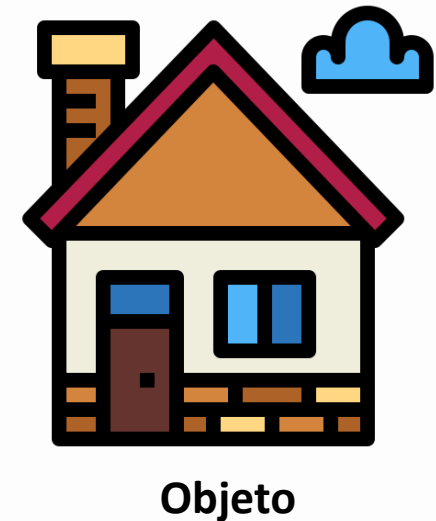
classe



Objeto

Classes e Objetos II

- Outra analogia que podemos fazer é entre o **projeto de uma casa** (a planta da casa) e a **casa em si**.
 - O projeto é a **classe**
 - A casa, construída a partir desta planta, é o **objeto**



Classes e Objetos III

- Pense em um conta bancária.
 - para criar uma conta, precisa ser preenchido um formulário contendo algumas informações
 - esse formulário representa parte da estrutura de uma conta, no caso de POO, a nossa **classe**
- O **objeto** será a conta efetivamente criada após o preenchimento do formulário de abertura de conta



classe



Objeto

Classes

- Uma classe descreve um conjunto de objetos semelhantes
 - Possui **Atributos** e **métodos** que resumem as características comuns de vários objetos.
- Diferença entre classe e objeto
 - **Objeto constitui uma entidade concreta com tempo e espaço de existência**
 - **Classe é tão-somente uma abstração**



Classes II

- Em termos de programação
 - definir uma classe significa formalizar um **tipo de dado** e todas as operações associadas a esse tipo
 - declarar objetos significa **criar variáveis** do tipo definido.



Classes - Python

■ No python definimos as nossas classes utilizando o comando **class**

```
class Bolo:  
    pass
```

```
class Casa:  
    pass
```

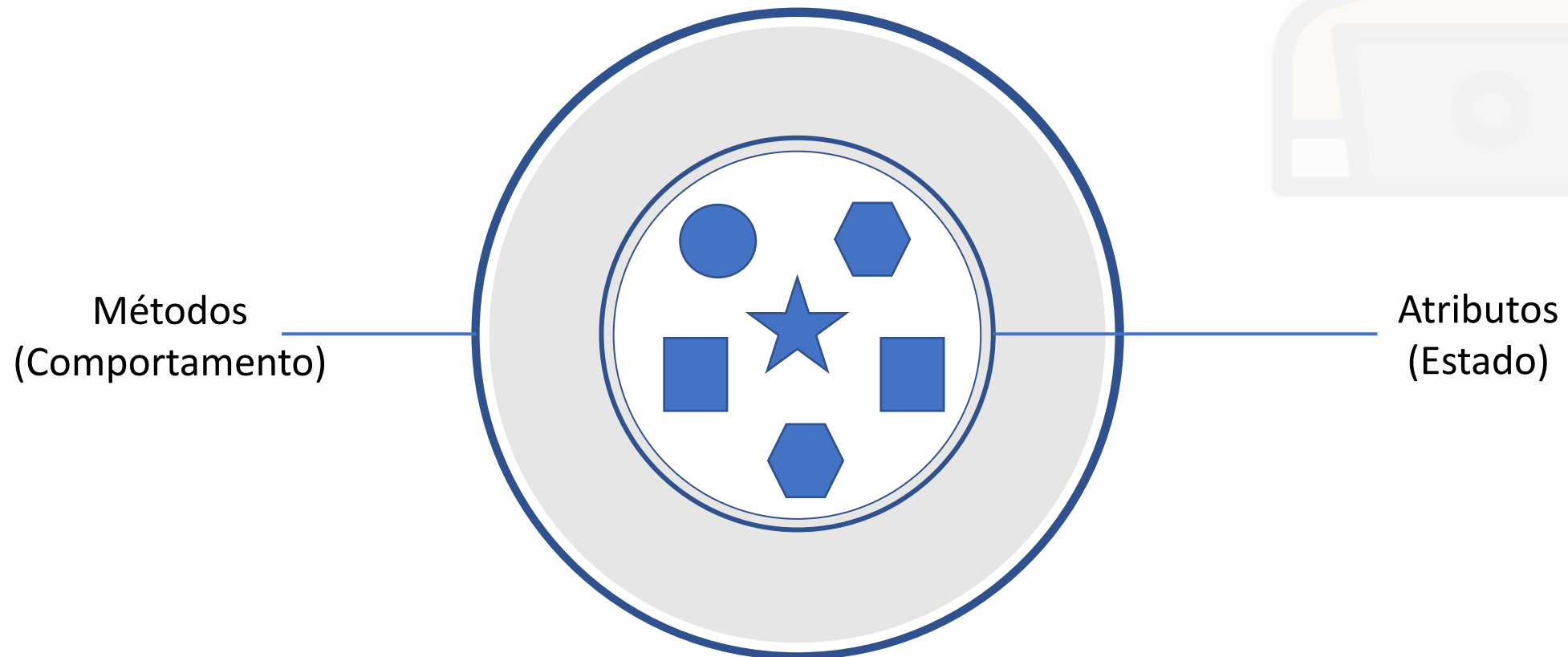
```
class Conta:  
    pass
```

Objeto

- Um objeto é uma **entidade** que formaliza o modo pelo qual compreendemos algo no **domínio do problema**.
 - Reflete a capacidade do sistema de guardar informações sobre o **elemento abstraído** e interagir com ele.
 - Entidade o mais próximo possível das **entidades do mundo real** - aquilo que é tangível ou visível.
 - Dessa forma, os objetos são uma forma de diminuir o **gap semântico**.
 - Diferença entre o domínio de problemas e soluções



Objetos – Representação



Objetos - Características

- A um objeto sempre estarão associados.

- Estado:

- Definido pelas propriedades (**atributos**) que ele possui e pelos valores que elas estão assumindo.

- Comportamento:

- Definido pela forma como ele age e reage, em termos de mudança de seu estado e o relacionamento com os demais objetos do sistema (métodos).

- Identidade:

- Cada objeto é único



Objetos - Exemplos

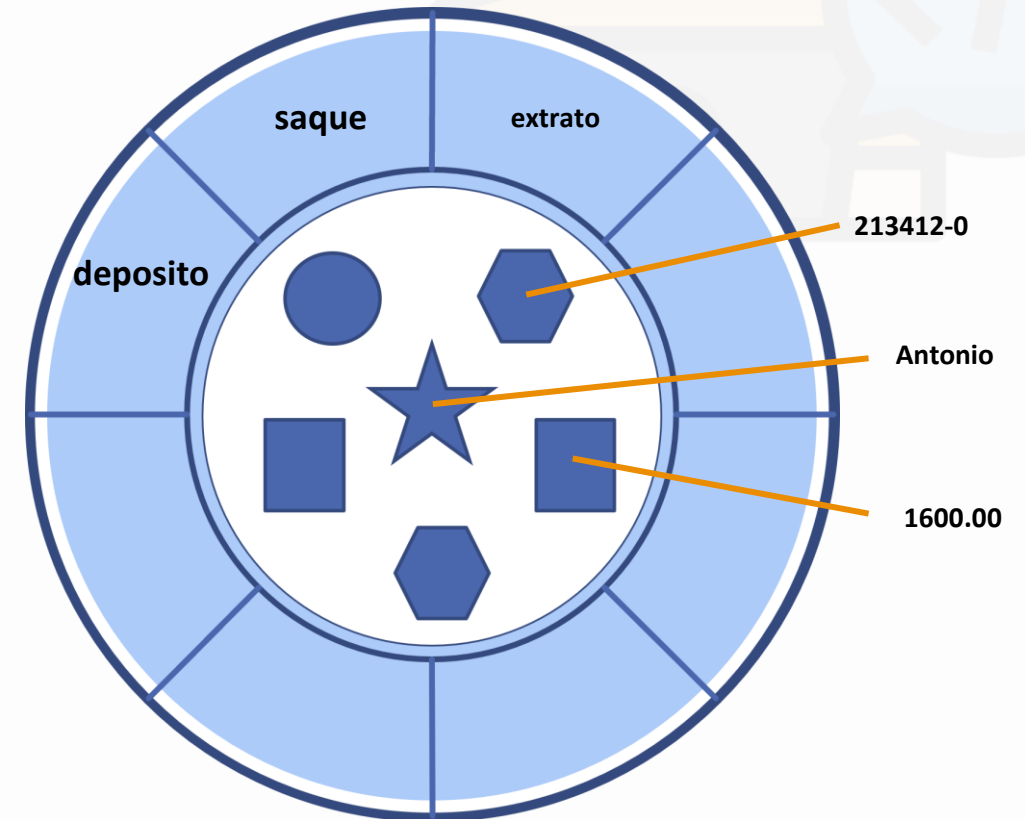
- Exemplos de objetos no mundo real:
 - Cachorro, mesa, televisão, bicicleta, lâmpada, ...
 - Lâmpada:
 - Atributos: ligada, desligada.
 - Métodos: ligar, desligar
 - Rádio:
 - Atributos: ligado, desligado, volume, estação, ...
 - Métodos: ligar, desligar, aumentar/abaixar volume, sintonizar,...
 - Cachorro:
 - Atributos: nome, cor, raça, peso, ...
 - Métodos: latir, morder, correr, dormir, ...



Objetos – Exemplos II

Conta:

- Atributos: numero, titular, saldo, ...
 - Métodos: deposito, saque, extrato, ...
- Objetos podem conter objetos como atributos.



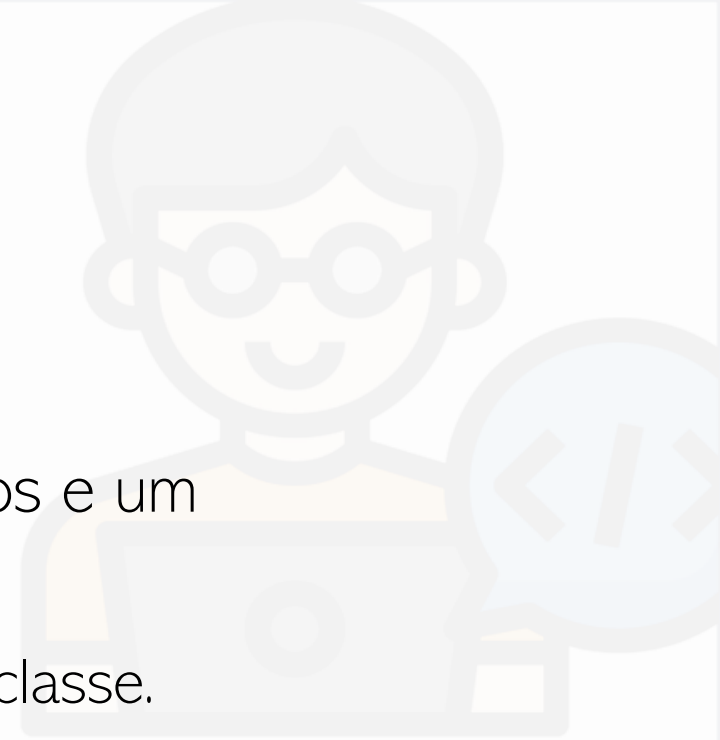
Instância

- Classe é um *template* (“forma”) para a criação de objetos.
 - Uma classe especifica os tipos de dados (**atributos**) e operações (**métodos**) suportadas por um conjunto de objetos
- Um objeto é uma instância de uma classe
 - Criação de um objeto a partir de uma classe é chamada de instanciação
 - É muito comum que em um programa existam várias instâncias de uma mesma **classe**
 - O que diferencia cada uma?



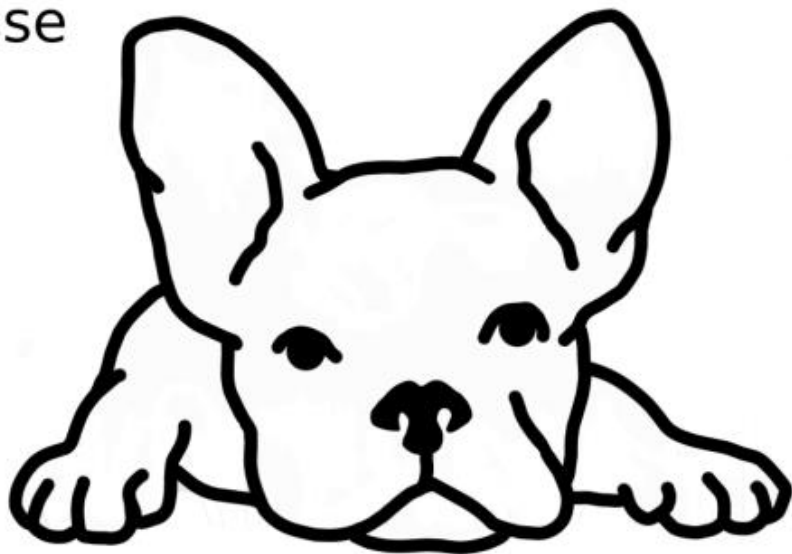
Instância II

- Cada instância é formada por valores de atributos únicos e um comportamento comum definido pela classe.
 - Inúmeras instâncias podem ser criadas a partir de uma classe.
 - O estado de cada instância é representado pelos valores de seus atributos, que podem ser diferentes.
 - Diferentes objetos de uma mesma classe possuem suas próprias cópias de cada atributo
 - A menos que isso seja desejado e explicitamente declarado.
 - Neste caso, um único atributo pode ser compartilhado para todas as instâncias.

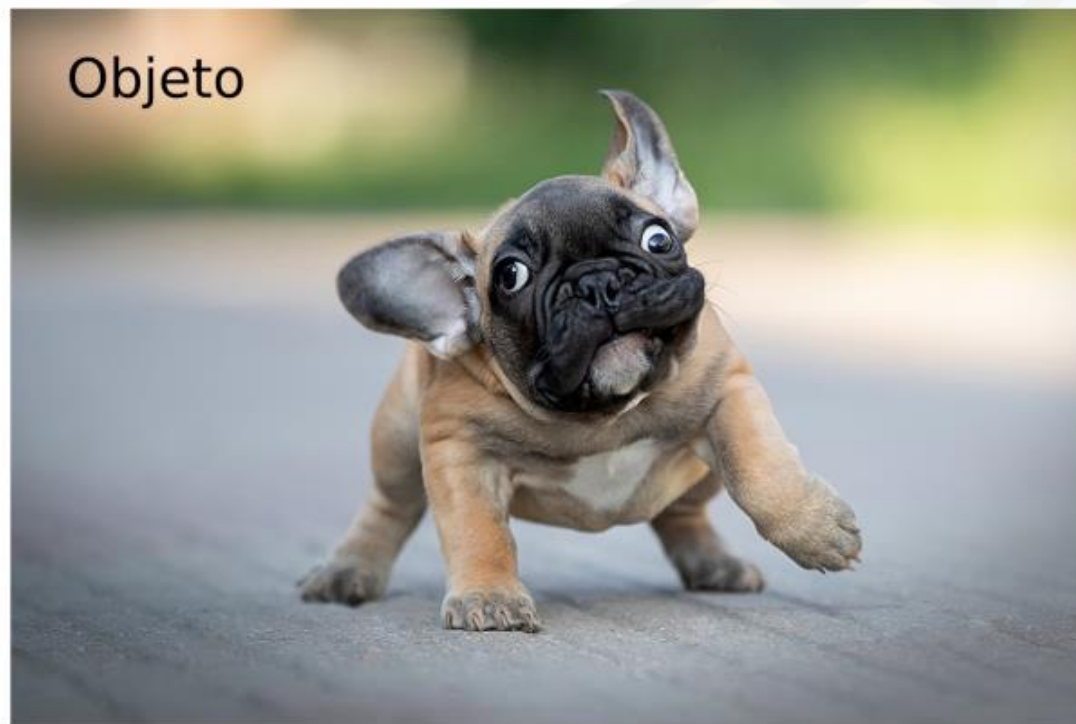


Instância III

Classe



Objeto



Objeto - Instância de uma classe em Python

Objeto bolo

```
class Bolo:  
    pass  
bolo = Bolo()
```

Objeto casa

```
class Casa:  
    pass  
casa = Casa()
```

Objeto conta

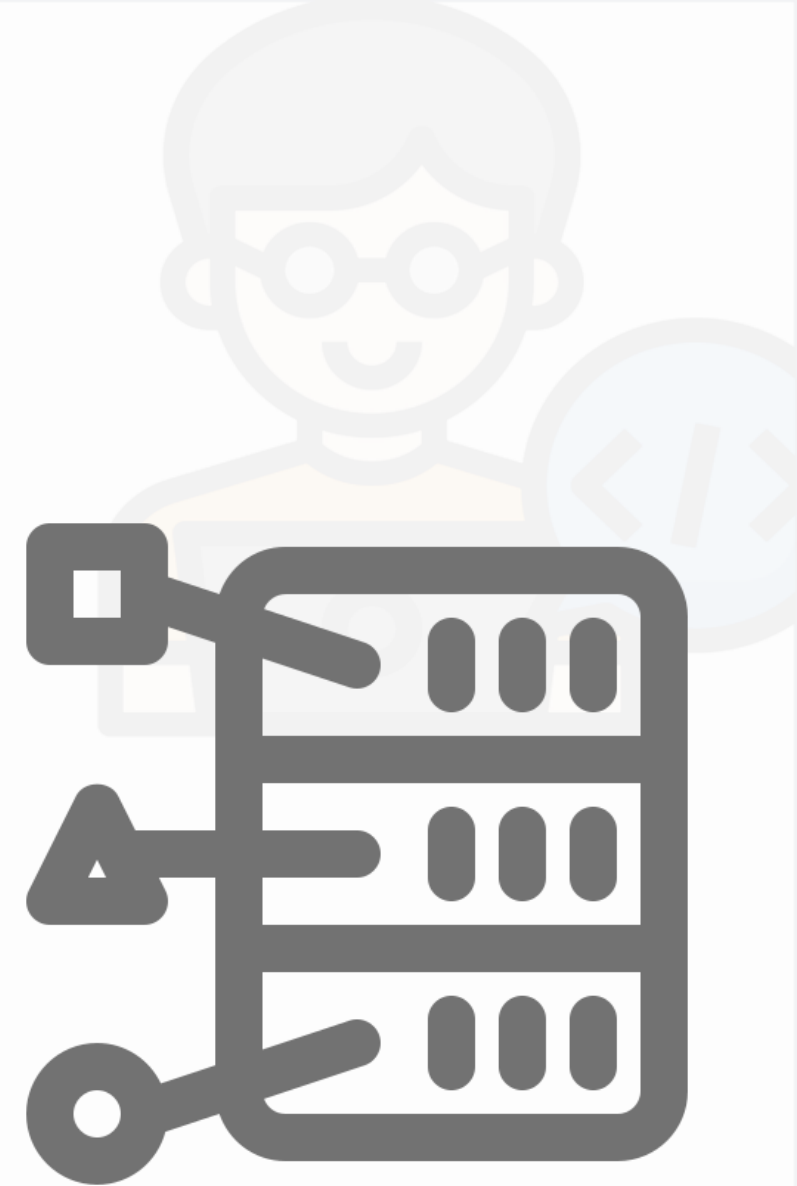
```
class Conta:  
    pass  
conta = Conta()
```

A dark blue background with a faint, stylized illustration of a person's hands typing on a laptop. To the left of the laptop is a cup of coffee on a saucer. To the right is a notebook with a pen resting on it. The title 'Atributos e Métodos' is written in large, white, sans-serif font across the center.

Atributos e Métodos

Atributos

- Representam um **conjunto de informações**, ou seja, **elementos de dados** que caracterizam um objeto.
- Descrevem as informações que ficam escondidas em um objeto para serem exclusivamente manipuladas pelas operações (**métodos**) daquele objeto.
- São variáveis que **definem o estado de um objeto**, ou seja, caracterizam os objetos.
- Cada objeto possui seu próprio conjunto de atributos.



Atributos em Python

☰ Atributos de uma classe

```
class Classe:  
    text: str # Atributo do tipo str  
    booleano: bool # Atributo do tipo bool  
    inteiro: int # Atributo do tipo str  
    decimal: float # Atributo do tipo float
```

```
classe = Classe() # instância da classe Classe / variável classe que referencia um objeto
```

```
classe.text = "Texto"  
classe.booleano = True  
classe.inteiro = 22  
classe.decimal = 12.79
```

Atributos em Python – classe Curso

■ Vamos criar uma classe chamada curso, contendo os atributos

■ nome: str # Atributo do tipo str

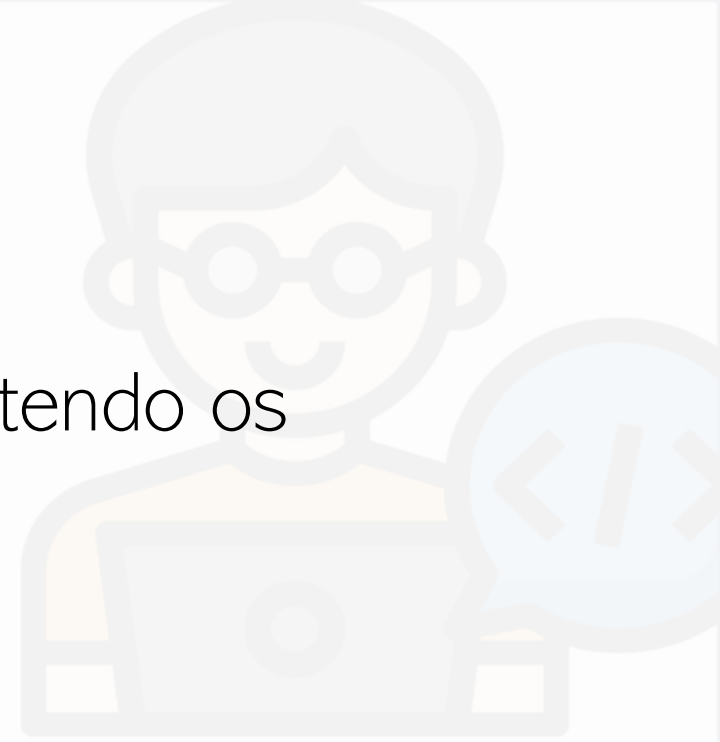
■ data: str # Atributo do tipo str

■ professor: str # Atributo do tipo str

■ horas: int # Atributo do tipo int

■ valor: float # Atributo do tipo float

■ Instanciar um objeto da classe Curso



Atributos em Python – classe Curso

🗄️ Classe Curso

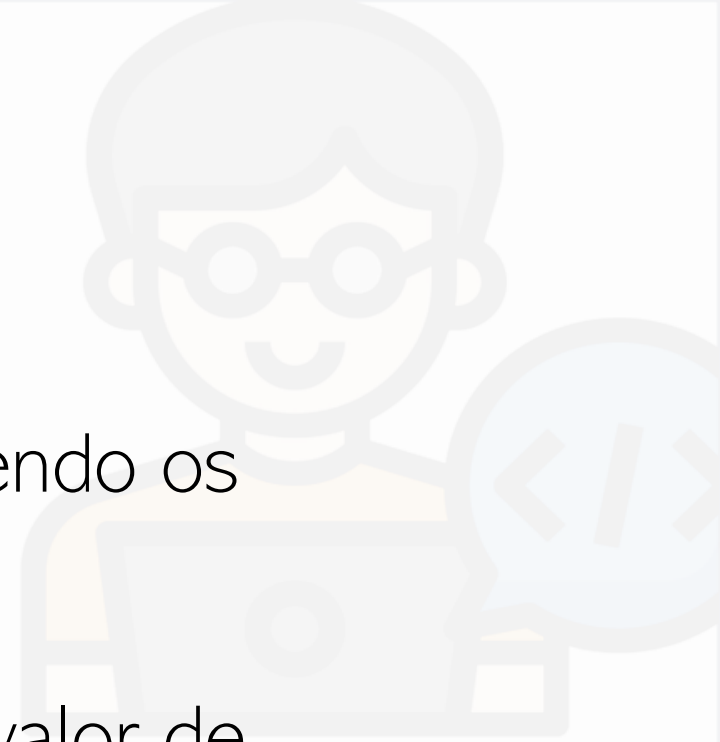
```
class Curso:
    nome: str # Atributo do tipo str
    data: str # Atributo do tipo str
    professor: str # Atributo do tipo str
    horas: int # Atributo do tipo int
    valor: float # Atributo do tipo float

curso = Curso() # instância da classe Curso / variável curso que referencia um objeto

curso.nome = "Programação Orientada a Objetos"
curso.data = "07/10/2023"
curso.professor = "Antonio David Viniski"
curso.horas = 24
curso.valor = 1000.00
```

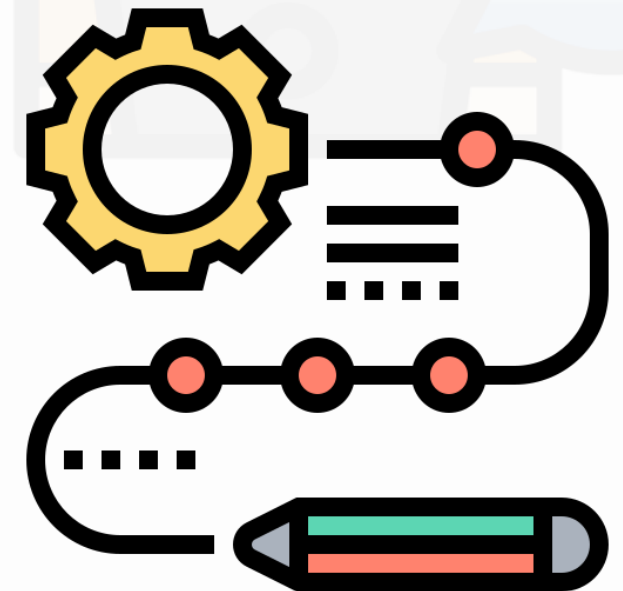
Exercício 1

- Crie a classe para representar a conta, contendo os atributos: numero, titular, saldo, limite, ativa
- Crie dois objetos do tipo conta, definindo o valor de seus atributos
- OBS: você pode utilizar como base o exemplo da criação de conta utilizando dicionários.



Método de Inicialização e Construtor

- Utilizado para definir o estado de um objeto (valor de seus atributos) durante a instanciação.
- O método construtor constrói o objeto (`__new__()`).
- O método de inicialização define o estado inicial do objeto construído (`__init__()`).



Inicializando os objetos em Python

- Em Python, alguns nomes de métodos estão reservados para o uso da própria linguagem.
- Um desses métodos é o `__init__()` que vai inicializar o objeto.
 - Seu primeiro parâmetro, assim como todo método de instância, é a própria instância.
 - Por convenção, chamamos este argumento de **self**.

```
class Classe:  
    def __init__(self, text: str, booleano: bool, inteiro: int, decimal: float):  
        self.text: str = text  
        self.booleano: bool = booleano  
        self.inteiro: int = inteiro  
        self.decimal: float = decimal
```

```
classe = Classe("Texto", True, 22, 12.79)
```

- Agora, quando um objeto é criado, todos os seus atributos serão inicializados pelo método `__init__()`.

Método Construtor

- Apesar de muitos programadores chamarem o método `__init__()` de construtor, ele não cria um objeto conta.
- Essa função é realizada pelo método `__new__()` que é chamado antes do `__init__()` pelo interpretador do Python, sendo quem realmente cria uma instância de Conta .
- O método `__init__()` é responsável por inicializar o objeto, tanto é que já recebe como argumento a própria instância (**self**) criada pelo construtor.

Inicialização – classe Curso

- Vamos definir o método de inicialização dos nossos atributos da classe Curso
- Instanciar um objeto da classe utilizando o método de inicialização criado



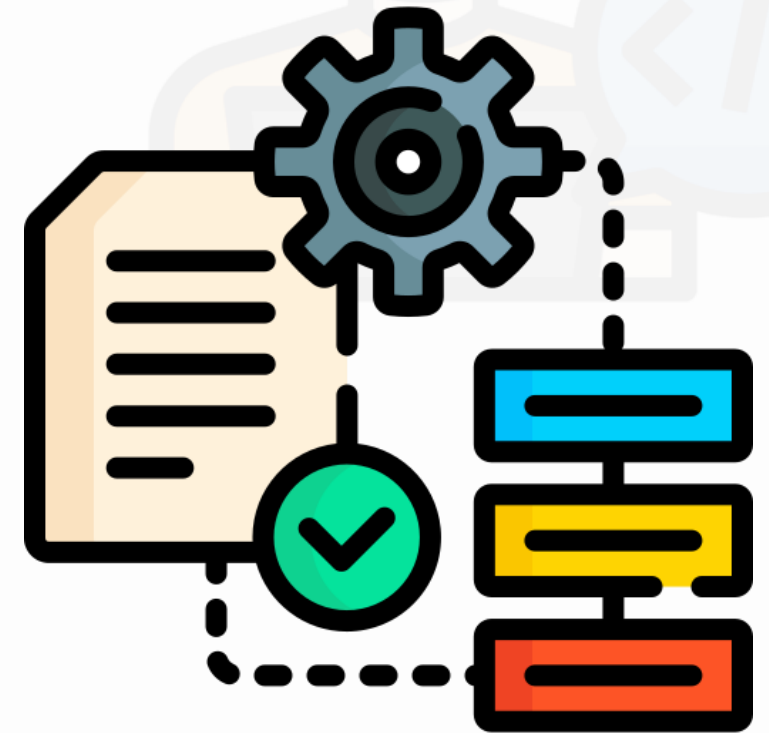
Inicialização – classe Curso - solução

```
class Curso:
    def __init__(self, nome: str, data: str, professor: str, horas: int, valor: float):
        self.nome: str = nome
        self.data: str = data
        self.professor: str = professor
        self.horas: int = horas
        self.valor: float = valor

curso = Curso("Programação Orientada a Objetos", "07/10/2023", "Antonio David Viniski", 24, 1000.00)
```

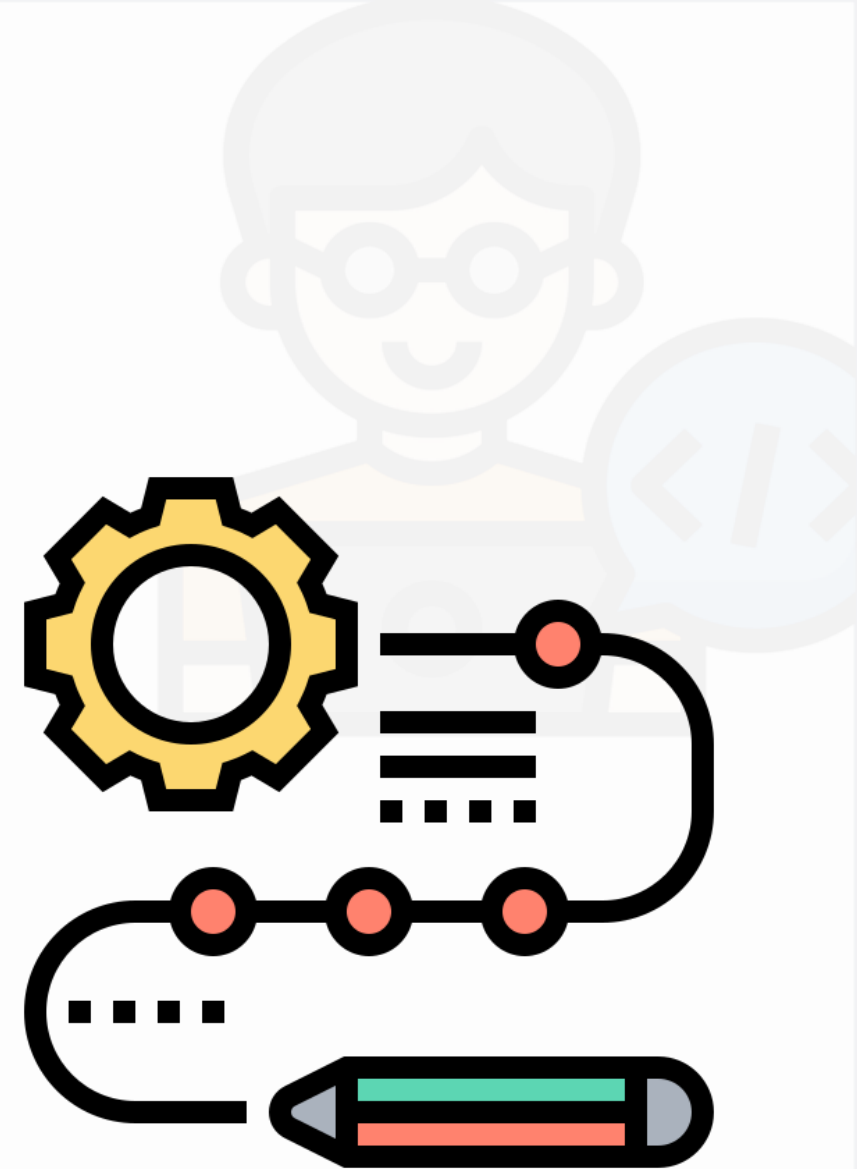
Métodos

- Os métodos são **operações** que podem ser executadas pelos objetos.
 - Valores dos atributos são (**normalmente**) acessados através dos métodos definidos pela classe.
 - Information-hiding.*
 - O serviço oferecido pelo método é um **comportamento** específico, residente no objeto, que **define como ele deve agir quando exigido**.



Métodos II

- São procedimentos definidos e declarados que atuam sobre um objeto ou sobre uma classe de objetos.
- Métodos são invocados por Mensagens.
- Cada objeto possui seu próprio conjunto de métodos.



Métodos em python

- A definição dos métodos das classes acontece de forma similar a definição de funções, utilizando o comando **def**
- A diferença é que o primeiro parâmetro sempre será a referência da própria classe (**self**)
- A chamada de método (troca de mensagens) ocorre utilizando a instância do objeto
 - `obj.metodo()`

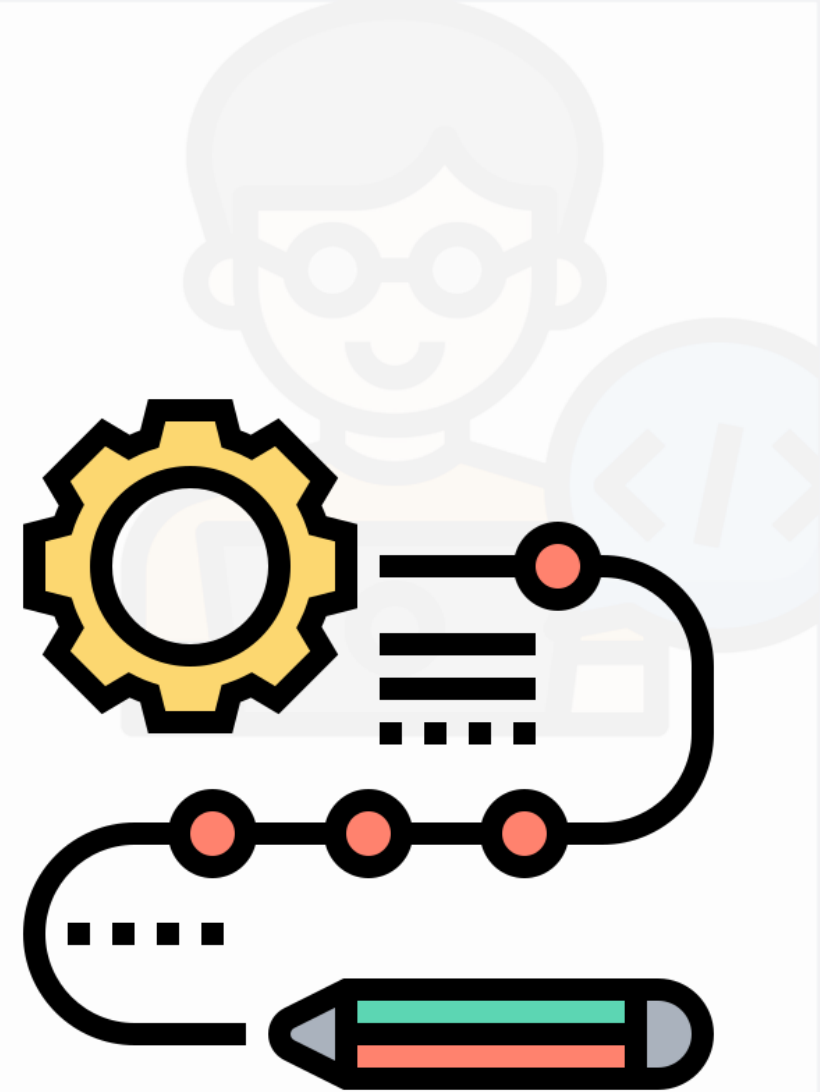
```
class Classe:
    # método __init__ omitido por conta do espaço

    def mudaTexto(self, novo):
        self.text = novo

classe = Classe("Texto", True, 22, 12.79)
classe.mudaTexto("Novo Texto")
```

Métodos - exemplo

- Considere as operações que podemos associadas a um curso:
 - adia
 - encarece
 - trocaProfessor
- Essas funcionalidades modificam o estado do curso, ou seja, o valor de seus atributos, assim, devem ser implementadas como **métodos** da classe Curso.



Métodos – exemplo curso - solução

```
class Curso:
    def __init__(self, nome: str, data: str, professor: str, horas: int, valor: float):
        self.nome: str = nome
        self.data: str = data
        self.professor: str = professor
        self.horas: int = horas
        self.valor: float = valor

    def adia(self, novadata):
        self.data = novadata

    def encarece(self, aumento):
        self.valor += aumento
```

Exercício 2

- Considerando a aplicação da conta:
 - Criar o método de inicialização dos atributos da conta
 - Criar o método de saque
 - Criar o método de depósito
 - Criar o método extrato.
 - Realizar o saque em uma conta
 - Mostrar o extrato
 - Realizar o depósito em uma conta
 - Mostrar o extrato



Exemplo – Transferência entre contas

- Diferente do que foi criado no paradigma estruturado, na orientação a objetos não precisamos passar duas contas e o valor para o método de transferência.
 - Todo método já recebe como primeiro parâmetro a instância da própria classe (self).
 - Assim, basta informarmos a conta que receberá a transferência e o valor.

Métodos com retorno

- Assim como nas funções, os métodos também podem retornar valores
- Suponha que precisamos verificar se existe saldo disponível na conta, caso exista, retornamos **True**, senão, retornaremos **False** para a operação de saque.

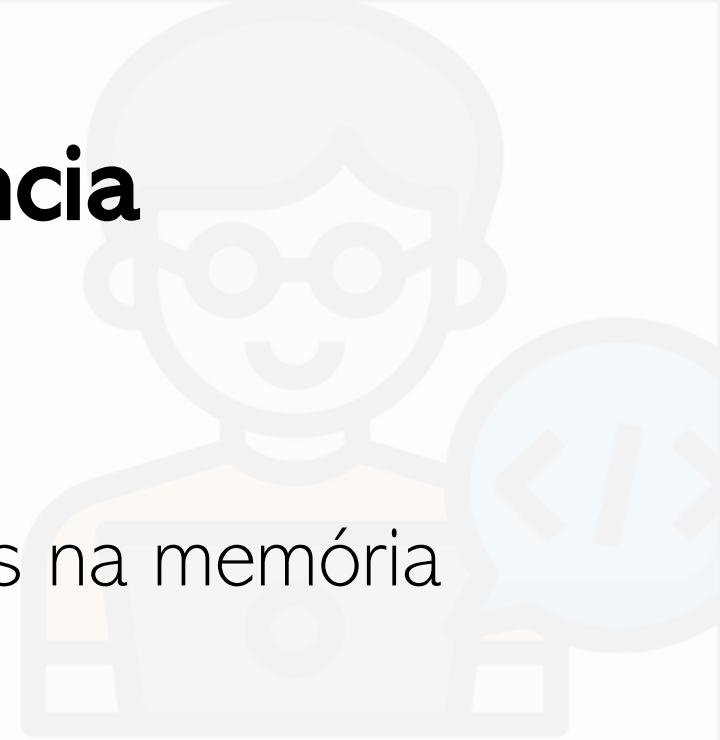


The background is a dark blue, textured surface. In the center, there is a faint illustration of a person's hands typing on a laptop keyboard. To the left of the laptop is a small, dark cup of coffee on a saucer. To the right is a dark notebook with a pen resting on it. The overall aesthetic is professional and focused.

Acessando os objetos

Objetos são acessados por referência

- Podemos manter vários Cursos armazenadas na memória durante o tempo de execução do programa.
- Quando criamos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma maneira de acessá-lo, chamada de **referência** (o **self**).



Copiando a referência de um objeto

```
c1 = Curso("Programação Orientada a Objetos", "07/10/2023", "Antonio David Viniski", 24, 1000.00)
c2 = c1
print(c2.valor)
# 1000.0
c1.encarece(100.0)
print(c1.valor)
# 1100.0
c2.encarece(30.0)
print(c2.valor)
# 1130.0
print(c1.valor)
# 1130.0
```

O que aconteceu aqui? O operador “=” copia o valor de uma variável. Mas qual é o valor da variável c1 ? É o objeto? Não. Na verdade, o valor guardado é a referência (endereço) de onde o objeto se encontra na memória principal.

Recuperando a referência de objetos

- Podemos utilizar a função `id` para recuperar a referência do objeto na memória.

```
print(id(c1))  
print(id(c2))
```

Verificando a referência dos objetos

■ Considere os seguintes cursos

```
curso1 = Curso("Programação Orientada a Objetos", "07/10/2023", "Antonio David Viniski", 24, 1000.00)  
curso2 = Curso("Programação Orientada a Objetos", "07/10/2023", "Antonio David Viniski", 24, 1000.00)
```

■ As contas são iguais?

```
if curso1 == curso2:  
    print("Os cursos são iguais!")
```

Exercício - Canvas

■ Considere um software de um jogo MOBA (*Multiplayer Online Battle Arena*).

■ Crie uma classe para representar cada um dos seguintes objetos:

■ Jogador

■ Personagem

■ Item

■ Informe os dados de cada um desses objetos (atributos)

■ Informe as operações sobre esses objetos (métodos)

■ Quais objetos adicionais você acha que fariam parte de um jogo?

