



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 07 – REVISÃO E FUNÇÕES
AGREGADORAS

O QUE IREMOS APRENDER

- 01** LISTAS E TUPLAS
- 02** DICIONÁRIOS E SETS
- 03** FUNÇÕES
- 04** MÓDULOS E BIBLIOTECAS
- 05** MÃOS NO CÓDIGO
- 06** FUNÇÕES AGREGADORAS
- 07** O QUE SÃO FUNÇÕES AGREGADORAS
- 08** EXEMPLOS DE FUNÇÕES AGREGADORAS NO PYTHON
- 09** MÃOS NO CÓDIGO

Listas e Tuplas

Uma lista é uma coleção ordenada de elementos.

Os elementos em uma lista podem ser de **diferentes tipos**, como números inteiros, strings, booleanos, outras listas e até mesmo objetos personalizados.

As listas são **mutáveis**, o que significa que você pode adicionar, remover ou modificar elementos após a criação da lista.

Para criar uma lista em Python, você usa colchetes `[]` e separa os elementos com vírgulas.

Listas e Tuplas



```
1  minha_lista = [1, 2, "três", True]
2
3  tamanho = len(minha_lista)
4  print(tamanho)
```

len(lista): Retorna o número de elementos na lista.



```
1  minha_lista = [1, 2, "três", True]
2
3  minha_lista.append(4)
4  print(minha_lista) # Saída: [1, 2, "três", True, 4]
```

lista.append(elemento):
Adiciona um elemento ao final da lista.

Listas e Tuplas



```
1 minha_lista = [1, 2, "três", True]
2 outra_lista = [4,5,6]
3
4 minha_lista.extend(outra_lista)
5 print(minha_lista) # Saída: [1, 2, "três", True, 4, 5, 6]
```

lista.extend(iterável): Adiciona os elementos de um iterável (por exemplo, outra lista) ao final da lista.



```
1 minha_lista = [1, 2, "três", True]
2
3 minha_lista.insert(1,4)
4 print(minha_lista) # Saída: [1, 4, 2, "três", True]
5
```

lista.insert(posição, elemento): Insere um elemento em uma posição específica na lista.

Listas e Tuplas

```
lista = [1,2,'três',True,'três']  
lista.remove('três')  
print(lista) #Saída [1,2,True,três]
```

lista.remove(elemento): Remove a primeira ocorrência de um elemento específico da lista.



```
1 minha_lista = [1, 2, "três", True]  
2 minha_lista.pop(3)  
3  
4 print(minha_lista) # Saída: [1, 2, três]
```

lista.pop(índice): Remove e retorna o elemento na posição especificada. Se nenhum índice for fornecido, remove e retorna o último elemento.

Listas e Tuplas



```
1 minha_lista = [1,10, 5, 6, 7, 3, 2]
2 minha_lista.sort()
3
4 print(minha_lista) # Saída: [1, 2, 3, 5, 6, 7, 10]
```

lista.sort(): Ordena a lista em ordem crescente (altera a lista original).

```
lista = [1,10,5,6,7,3,2]
lista.reverse()
print(lista) #Saída [2,3,7,6,5,10,1]
```

lista.reverse(): Inverte a ordem dos elementos na lista

Listas e Tuplas

Uma tupla é semelhante a uma lista, mas é imutável, o que significa que seus elementos não podem ser alterados, adicionados ou removidos após a criação da tupla.

As tuplas também são uma coleção ordenada de elementos e podem conter tipos de dados mistos.

Para criar uma tupla em Python, você usa parênteses () e separa os elementos com vírgulas.



```
1 minha_tupla = (1, 2, "três", True)
```


Dicionários e Sets

Um conjunto é uma coleção não ordenada de elementos exclusivos.

Os elementos em um conjunto não têm índices e não são duplicados.

Você pode realizar operações matemáticas de conjunto, como união, interseção e diferença, em conjuntos.

Os conjuntos são definidos usando chaves `{}` ou a função `set()`

Dicionários e Sets

Operações comuns de conjunto incluem adição de elementos com `add()`, remoção com `remove()` ou `discard()`, verificação de pertencimento com `in`, entre outras.

```
1 meu_set = {'Infinity', 'School'}
2 print(type (meu_set))
3 #<class 'set'>
4
5 frutas = {"maçã", "banana", "cereja", "maçã"}
6
7 print(frutas)
8 #{'maçã', 'cereja', 'banana'}
```

Dicionários e Sets

Um dicionário é uma coleção de pares chave-valor, onde cada chave é única.

Os elementos em um dicionário são armazenados como pares chave-valor, onde a chave é usada para acessar o valor associado.


Os dicionários são muito eficientes para pesquisas de valor com base na chave.

Os dicionários são definidos usando chaves `{}` e pares chave-valor separados por `:`

Dicionários e Sets



```
1 dados_usuario= {  
2     'Nome': 'Raama',  
3     'Idade': 18,  
4     'Cidade': 'Salvador'  
5 }  
6  
7 print(dados_usuario['Idade'])
```




```
1 dados_usuario= {  
2     'Nome': 'Raama',  
3     'Idade': 18,  
4     'Cidade': 'Salvador'  
5 }  
6  
7 print(dados_usuario.get('Idade'))
```


Para acessar um valor de um dicionário basta printarmos o dicionário e colocar entre colchetes e aspas a chave que queremos. Também podemos utilizar a função `get()` para acessar um valor.

Funções

Uma função em programação é um bloco de código que executa uma tarefa específica quando chamado. As funções são usadas para organizar o código, reutilizar lógica e dividir um programa em partes menores e mais gerenciáveis. As funções são definidas usando a palavra-chave `def`.




```
1 def nome_da_funcao(parametros):  
2     #Corpo da função  
3     # Realize a lógica desejada aqui  
4     return resultado # Opcional, se a função retornar um valor
```



```
1 def soma(a, b):  
2     return a + b  
3  
4 resultado = soma(3, 5)  
5 print(resultado) # Saída: 8
```

Funções

Em Python, você pode definir funções com argumentos regulares (args) e argumentos de palavra-chave (kwargs) para torná-las mais flexíveis e capazes de lidar com diferentes situações.



```
1 def numero_par(numero):
2     if numero % 2 == 0:
3         return True
4     else:
5         return False
6
7 eh_par = numero_par (4)
8 print(eh_par) # Saída: True
```



```
1 def imprimir_mensagem(nome, mensagem):
2     print(f" {nome}: {mensagem}")
3
4 imprimir_mensagem("Alice", "Olá, como você está?")
```


Funções

Argumentos Regulares (Args):
Argumentos regulares são os parâmetros que você passa para uma função de forma posicional. Eles são acessados dentro da função na mesma ordem em que são passados na chamada da função.

```
1  def media(*args):  
2      '''  
3          Calcula a média de uma lista de números.  
4          :param args: Uma lista de números.  
5          :return: A média dos números.  
6          '''  
7      if not args:  
8          return 0  
9      return sum(args) / len(args)  
10  
11 resultado = media(2, 4, 6, 8, 10)  
12 print(resultado) # Saída: 6.0
```


Funções

Argumentos de Palavra-chave (Kwargs):

Argumentos de palavra-chave permitem que você passe argumentos para uma função usando o nome do parâmetro.

Isso torna a chamada da função mais legível e permite que você passe os argumentos em qualquer ordem, independentemente da ordem dos parâmetros na definição da função.

Funções

Argumentos de Palavra-chave (Kwargs):



```
1 def saudacao (nome, mensagem):  
2     return f"{mensagem}, {nome}!"  
3  
4 resultado = saudacao (mensagem="Olá", nome="Bob")  
5 print(resultado) # Saída: "Olá, Bob!"
```

Funções

Você pode usar argumentos regulares e de palavra-chave em uma única função, mas os argumentos regulares devem aparecer antes dos argumentos de palavra-chave na definição da função.

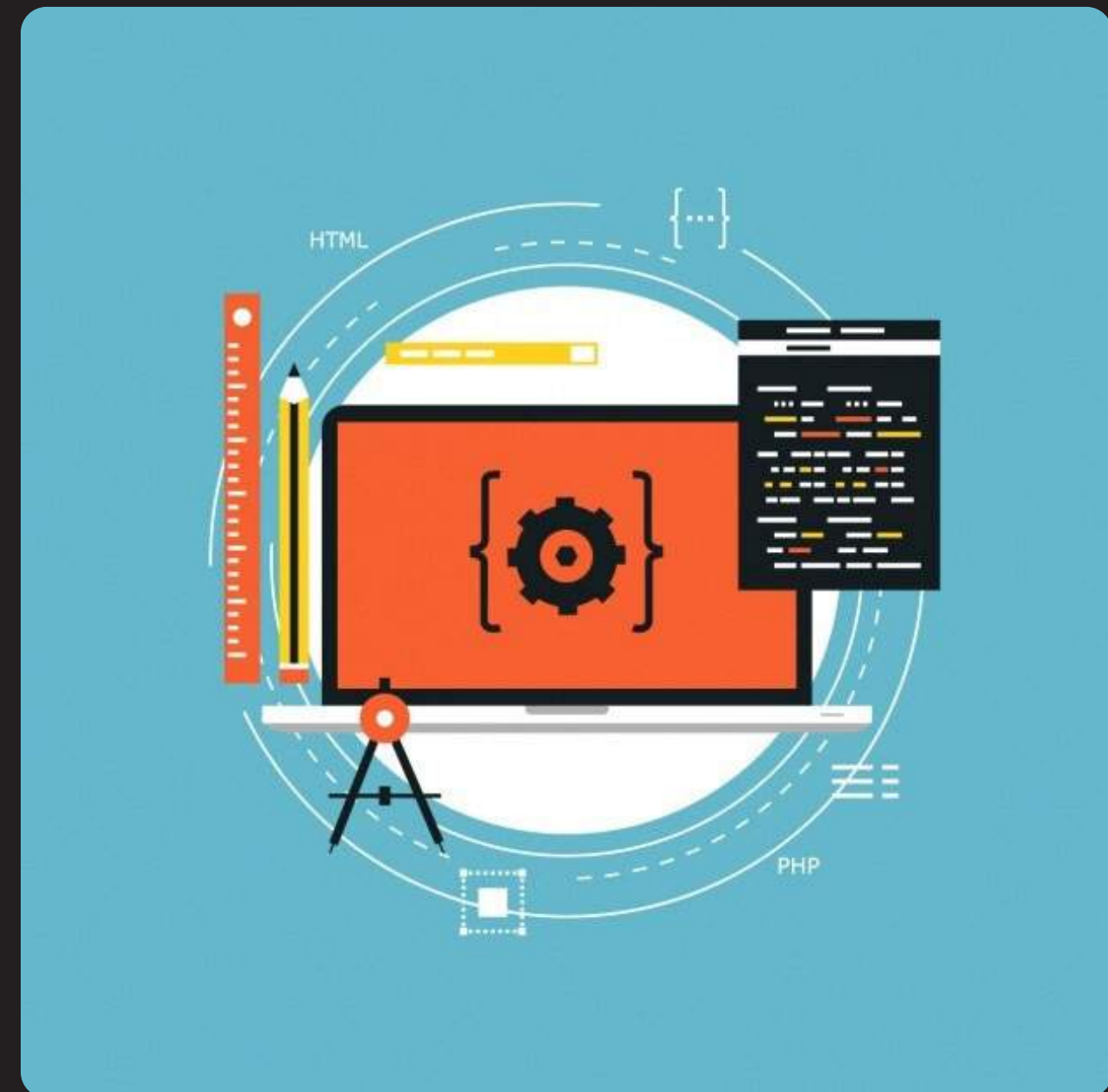
```
1 def saudacao (nome, *args, mensagem="01á", **kwargs):
2     saudacao_completa = f" {mensagem}, {nome}"
3     if args:
4         saudacao_completa += f" {'', '.join(args)}"
5     if kwargs:
6         saudacao_completa += f"({'', '.join(f'{{k}}={{v}}' for k, v in kwargs.items()))}"
7     return saudacao_completa
8
9 resultado = saudacao("Alice", "Bob", "Carol", mensagem="01", idade=30, cidade="Nova York")
10 print(resultado) # Saída: "Oi, Alice! Bob, Carol (idade=30, cidade Nova York)"
```

Módulos e Bibliotecas

Um módulo é um arquivo Python contendo funções, classes e variáveis.


Você pode criar seus próprios módulos escrevendo código Python em um arquivo com extensão `.py`.

Para usar um módulo em outro programa, você importa-o usando a instrução `import`.




Módulos e Bibliotecas

Criamos uma função chamada saudação que possui o parâmetro nome, no arquivo **meu_modulo.py**



```
1  # meu_modulo.py
2  def saudacao (nome):
3      return f"Olá, {nome}!"
```



```
1  import meu_modulo
2
3  mensagem = meu_modulo.saudacao("Alice")
4  print(mensagem) # Saída: "Olá, Alice!"
```

No arquivo **meu_arquivo** importamos o **meu_modulo**. Criamos uma variável **mensagem** que armazena a chamada do arquivo **meu_modulo** e busca a função saudação que espera um argumento como resposta.

Módulos e Bibliotecas

As bibliotecas de código são coleções de código predefinido que oferecem funcionalidades específicas e podem ser reutilizadas em vários programas. Elas facilitam o desenvolvimento de software, economizam tempo e evitam a necessidade de reinventar a roda ao realizar tarefas comuns.

```
1 import random
2 # Exemplo de uso de funções para geração de números aleatórios
3 # Gera um número aleatório entre 1 e 6
4 # (simulando um dado de seis faces)
5 numero_aleatorio = random.randint(1, 6)
6 print(numero_aleatorio)
7
8 # Embaralha uma lista
9 minha_lista = [1, 2, 3, 4, 5]
10 random.shuffle(minha_lista)
11 print(minha_lista)
12
13 # Escolhe um elemento aleatório de uma lista
14 opcoes = ["rock", "paper", "scissors"]
15 escolha = random.choice(opcoes)
16 print(escolha)
```

ATIVIDADE PRÁTICA 1

Dada uma lista de números, crie uma nova lista contendo apenas os números pares.

ATIVIDADE PRÁTICA 2

Crie um dicionário com informações de produtos, incluindo nome, preço e quantidade em estoque. Implemente funções para adicionar, remover e atualizar produtos no dicionário.

ATIVIDADE PRÁTICA 3

Crie um conjunto com nomes de cores. Implemente uma função que retorne as cores que têm mais de quatro letras.

ATIVIDADE PRÁTICA 4

Crie uma função que receba uma lista de strings e retorne uma nova lista contendo apenas as strings palíndromos.

Funções Agregadoras

As funções agregadoras desempenham um papel fundamental na análise de dados, As funções agregadoras são essenciais na resolução de problemas, permitindo resumir informações, extrair insights e tomar decisões informadas. Elas são ferramentas fundamentais na manipulação de dados e na aplicação de lógica de programação para resolver problemas do mundo real.



Funções Agregadoras

Funções agregadoras, também conhecidas como funções de agregação, são funções que são usadas para processar, resumir ou extrair informações de um conjunto de dados, geralmente coleções de valores, como listas, tuplas, dicionários, ou até mesmo bancos de dados. Essas funções permitem calcular estatísticas, encontrar valores extremos, contar elementos, e realizar outras operações que fornecem uma visão resumida dos dados.



Funções Agregadoras

01

sum()

Calcula a soma de todos os elementos em um objeto iterável, como uma lista.

02

max() e min()

Retorna o valor máximo e o valor mínimo em um objeto iterável.

03

any() e all()

`any()` retorna True se pelo menos um elemento de uma coleção for verdadeiro, enquanto `all()` retorna True se todos os elementos forem verdadeiros.

04

len()

Retorna o número de elementos em um objeto iterável, como uma lista, tupla ou string.

05

sorted()

Classifica os elementos de uma lista em ordem crescente (ou decrescente, se especificado). Não modifica a lista original, mas retorna uma nova lista ordenada.

06

join()

Esta é uma função de strings que é frequentemente usada para concatenar elementos de uma lista em uma única string, usando um delimitador específico.

Funções Agregadoras

sum()



```
1 valores = (2.5, 3.7, 1.8, 4.2)
2 soma = sum(valores)
3 print(soma) # Resultado: 12.2 (2.5 +3.7 + 1.8 + 4.2)
```

len()



```
1 frutas = ("maçã", "banana", "laranja", "uva")
2 quantidade = len(frutas)
3 print(quantidade) # Resultado: 4 (4 frutas na tupla)
```

max()



```
1 valores = (5.6, 9.2, 3.8, 12.7)
2 maximo = max(valores)
3 print(maximo) # Resultado: 12.7 (maior número na tupla)
```

min()



```
1 numeros = [12, 45, 78, 32, 19]
2 minimo = min(numeros)
3 print(minimo) # Resultado: 12 (menor número na lista)
```


Funções Agregadoras

join()



```
1 palavras = ["Python", "é", "uma", "linguagem", "de", "programação"]
2 frase = " ".join(palavras)
3 print(frase) # Resultado: "Python é uma linguagem de programação"
```

any()



```
1 valores = [False, True, False, True, False]
2 resultado = any (valores)
3 print(resultado) # Resultado: True (pelo menos um valor é verdadeiro)
```

sorted()



```
1 nomes = ("Alice", "Bob", "Charlie", "David", "Eve")
2 nomes_ordenados = sorted (nomes)
3 print(nomes_ordenados) # Resultado: ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
```

all()



```
1 valores = [True, True, True, True, True]
2 resultado = all(valores)
3 print(resultado) # Resultado: True (todos os valores são verdadeiros)
```

ATIVIDADE PRÁTICA 5

Dado um dicionário que representa as vendas de produtos, encontre o produto mais vendido (ou os produtos mais vendidos, se houver um empate).

ATIVIDADE PRÁTICA 6

Receba uma lista de números e use funções agregadoras para contar quantos valores são ímpares e quantos são pares.

ATIVIDADE PRÁTICA 7

Você possui dados de vendas trimestrais de uma empresa em uma lista. Cada trimestre é representado como uma lista de números, onde cada número representa o valor de vendas de um mês (janeiro a março, abril a junho, julho a setembro e outubro a dezembro).

Você deve realizar as seguintes tarefas:
Calcule a média de vendas por trimestre.

ATIVIDADE PRÁTICA 7

Encontre o trimestre com a maior soma de vendas.
Encontre o trimestre com a menor soma de vendas.
Calcule o total de vendas no ano inteiro.
- Construa seus dados fictícios

DESAFIO PRÁTICO

Análise de Produção anual

Você tem um conjunto de dados que contém informações sobre a produção anual de diferentes culturas em diversas fazendas ao longo de vários anos. O objetivo é realizar uma análise simples desses dados usando apenas as funções agregadoras.

Tarefas: Encontre o ano em que a produção total foi máxima e o ano em que foi mínima. Identifique a cultura que teve a maior produção total e a cultura com a menor produção total ao longo dos anos. Encontre a fazenda que obteve a produção máxima em um determinado ano e a fazenda com a produção mínima no mesmo ano.

- Construa seus próprios dados fictícios.

Material Complementar

- Exploração: Não tenha medo de explorar e testar diferentes códigos. A experimentação é uma grande aliada da aprendizagem.
- Perguntas: Faça perguntas, seja curioso! Entender o "porquê" das coisas ajuda a consolidar o conhecimento.
- Revisão: Revise o que aprendeu, tente explicar para si mesmo ou para outras pessoas. Ensinar é uma ótima forma de aprender.
- Prática: A prática leva à perfeição. Quanto mais exercícios fizer, mais fácil será lembrar e entender os conceitos.



SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

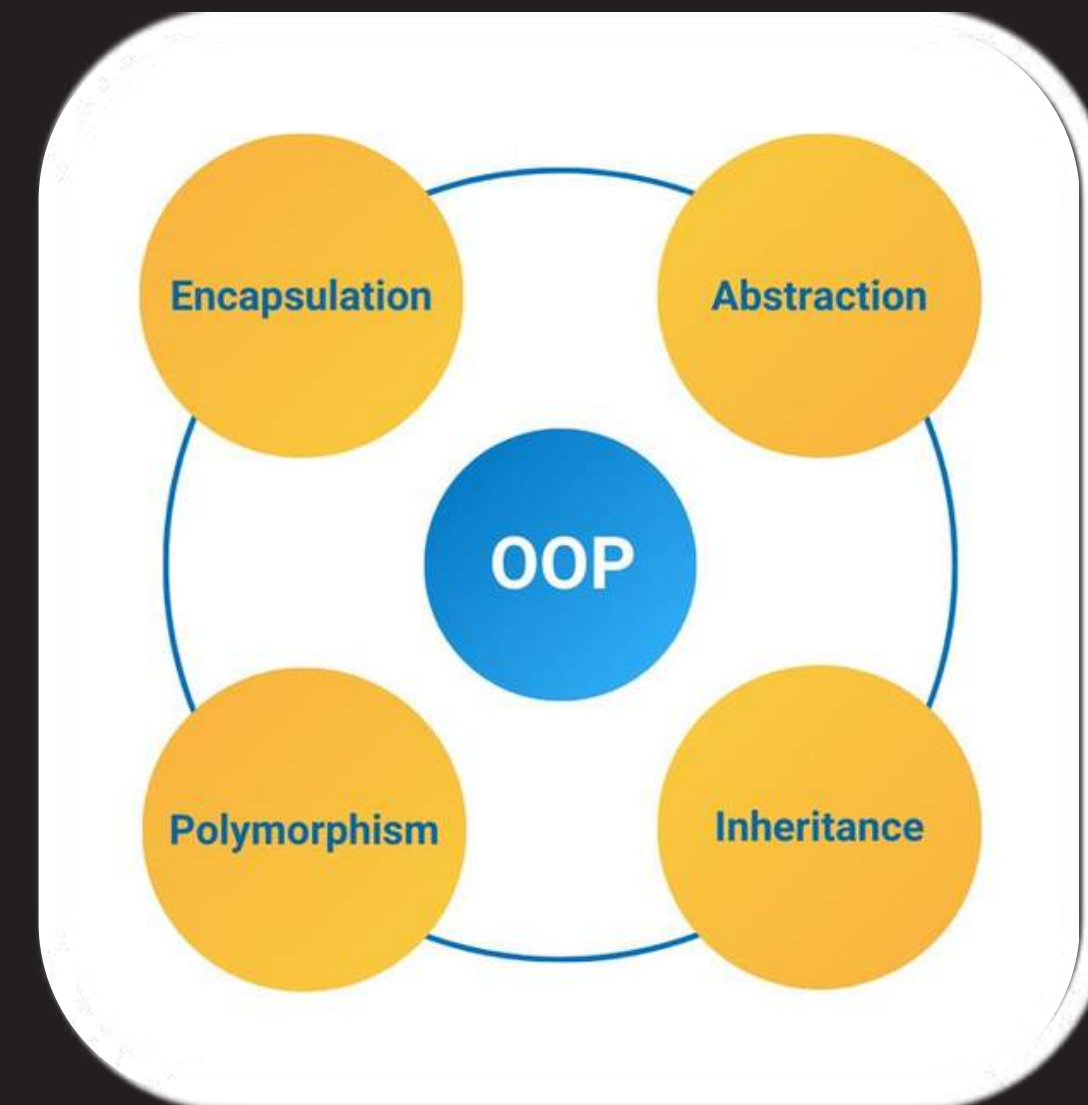
AULA 08 DE PYTHON:
POO I

The logo consists of the letters 'IN' in white, bold, sans-serif font, centered within a solid red square.

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

POO (Programação Orientada a Objetos)

Quem desenvolve softwares sabe o quanto o trabalho pode ser desafiador. Afinal de contas, programar é trazer soluções para os problemas dos clientes utilizando algoritmos em uma linguagem de programação. Portanto, a programação orientada a objetos trouxe agilidade no desenvolvimento de software, reduzindo o tempo para identificar e corrigir erros de programação. Isso revitalizou o paradigma da programação como um todo. E desse assunto que iremos tratar nesta aula.



Paradigma de Programação

Um paradigma de programação fornece e determina a visão que o **programador** possui sobre a estruturação e execução do programa. Por exemplo, em **programação orientada a objetos**, os programadores podem abstrair um programa como uma coleção de **objetos** que interagem entre si.





INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 07 – REVISÃO E FUNÇÕES
AGREGADORAS