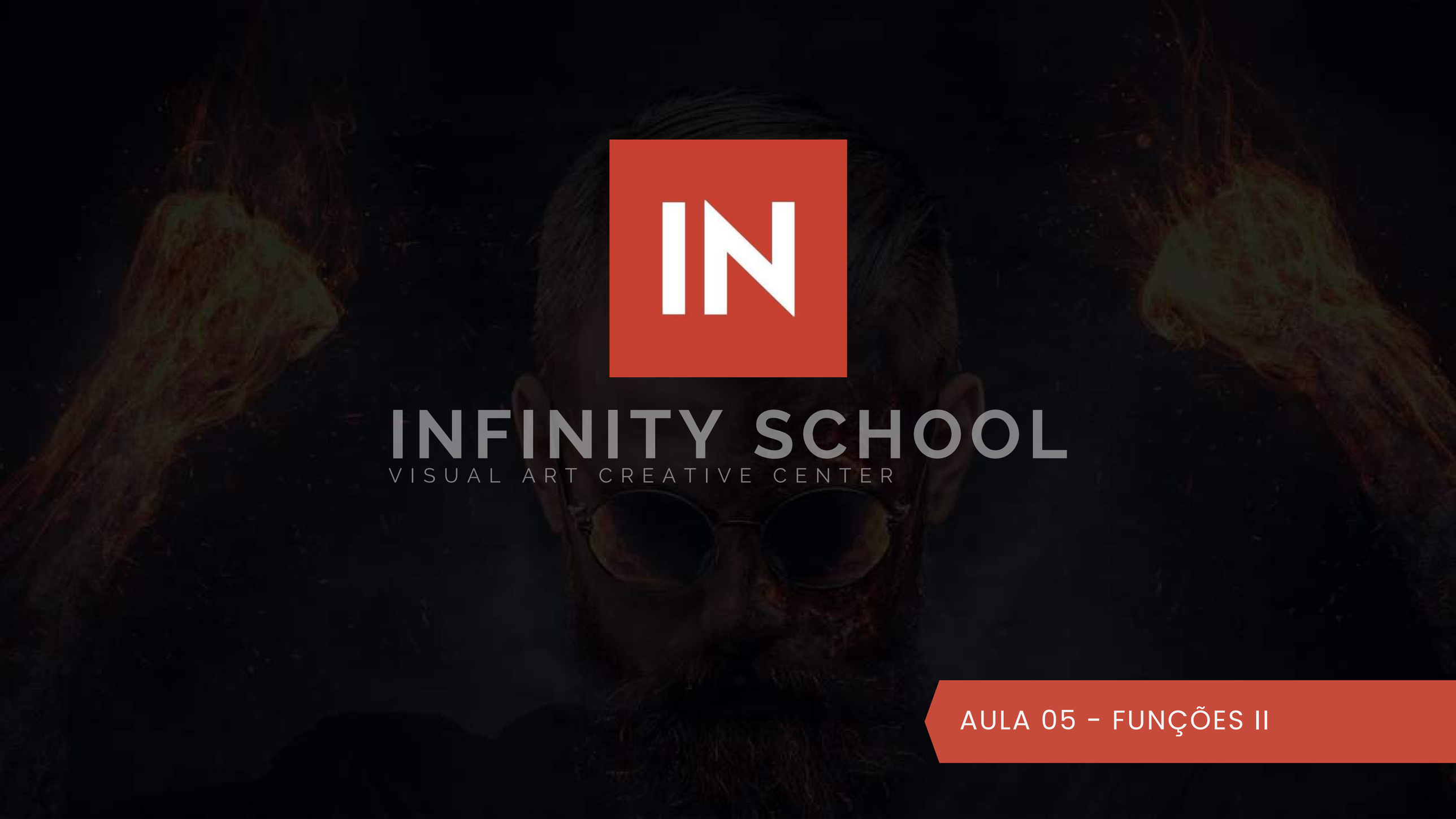


INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 05 – FUNÇÕES II

O QUE IREMOS APRENDER

01 RESUMO DA AULA PASSADA

02 ARGS

03 KWARGS

04 FUNÇÕES LAMBDA

05 ATIVIDADES PRÁTICAS

06 AMBIENTES VIRTUAIS

07 PROJETO

Resumo da aula passada

Funções são blocos de código em programação que realizam tarefas específicas e podem ser chamados ou invocados em vários pontos de um programa. Elas desempenham um papel fundamental na organização e reutilização de código.

```
def soma(a, b):
    return a + b

# Chamando a função e armazenando o resultado em uma variável
resultado = soma(5, 3)
print(resultado)  # Isso imprimirá "8"
```

Resumo da aula passada

Chamada de Função: Para executar uma função, você a chama pelo nome, passando os argumentos necessários, se houver. A função pode retornar um valor como resultado.

Parâmetros e Argumentos: Funções podem receber parâmetros, que são variáveis ou valores que a função usa em seu código. Quando você chama a função, você fornece argumentos, que
são os valores reais que correspondem aos parâmetros.

Retorno de Valor: Muitas funções retornam um valor como resultado. Esse valor pode ser usado em outras partes do programa onde a função foi chamada.

Escopo: Funções têm seu próprio escopo, o que significa que as variáveis definidas dentro de uma função não são visíveis fora dela, a menos que sejam retornadas.

Resumo da aula passada

```python
# Definição de uma função que calcula o quadrado de um número
def quadrado(numero):
    resultado = numero ** 2
    return resultado

# Chamando a função e armazenando o resultado em uma variável
resultado = quadrado(5)

# Imprimindo o resultado
print("O quadrado de 5 é:", resultado)
```

Neste exemplo, a função quadrado recebe um número como argumento, calcula o quadrado desse número e retorna o resultado.

Em seguida, a função é chamada com o argumento 5, e o resultado é armazenado na variável resultado e impresso na tela. O programa irá imprimir "O quadrado de 5 é: 25".

ATIVIDADE PRÁTICA 1

Crie um programa que solicita ao usuário que insira três notas e, em seguida, calcule a média dessas notas usando uma função. A função deve receber as três notas como argumentos e retornar a média. Por fim, o programa deve imprimir a média calculada.

ATIVIDADE PRÁTICA 2

Crie um programa que define uma função calcular_area_retangulo que recebe dois argumentos, comprimento e largura de um retângulo, e retorna a área desse retângulo. Em seguida, o programa deve solicitar ao usuário que insira o comprimento e a largura e imprimir a área calculada.

Contextualização da aula de hoje

Na aula passada, vimos como utilizar funções para resolver problemas. E vimos que uma função recebe seus parâmetros predefinidos, resolve um problema e devolve a solução. Mas e se não soubermos quantos parâmetros iremos enviar? ou quisermos enviar múltiplos parâmetros nomeados de uma vez só?

Nessas situações, conseguimos utilizar os “*args” e os “*kwargs” do python para passar uma quantidade indeterminada de parâmetros.

Args (Argumentos Posicionais Arbitrários)

- A notação *args permite que você passe um número variável de argumentos posicionais para uma função.
- Os argumentos são coletados em uma tupla dentro da função, que pode ser acessada pelo nome args (ou qualquer nome de sua escolha).
- O operador * antes de args é usado para indicar que todos os argumentos posicionais a seguir devem ser coletados na tupla args.

Args (Argumentos Posicionais Arbitrários)

```python
def somar_numeros(*args):
    resultado = 0
    for num in args:
        resultado += num
    return resultado

# Chamando a função com diferentes números de argumentos
print(somar_numeros(1, 2, 3))  # Isso imprimirá "6"
print(somar_numeros(10, 20, 30, 40, 50))  # Isso imprimirá "150"
```

A função somar_numeros aceita um número variável de argumentos posicionais (denominados args). Ela itera sobre esses argumentos e soma todos eles para produzir o resultado.

O operador *args permite que você chame a função com diferentes números de argumentos sem a necessidade de especificar quantos são.

```python
def somar(*args):
    resultado = 0
    for num in args:
        resultado += num
    return resultado

print(somar(1, 2, 3))  # Isso imprimirá "6"
```

Kwargs (Argumentos de Palavra Chave)

- A notação **kwargs permite que você passe um número indefinido de argumentos de palavra-chave para uma função.
- Os argumentos de palavra-chave são coletados em um dicionário dentro da função, que pode ser acessado pelo nome kwargs (ou qualquer nome de sua escolha).
- O operador ** antes de kwargs é usado para indicar que todos os argumentos de palavra-chave a seguir devem ser coletados no dicionário kwargs.

Kwargs (Argumentos de Palavra Chave)

```python
def mostrar_info(**kwargs):
    for chave, valor in kwargs.items():
        print(f"{chave}: {valor}")

mostrar_info(nome="João", idade=30, cidade="Exemplo")  # Isso imprimirá informações formatadas
```

```python
def mostrar_informacoes(**kwargs):
    for chave, valor in kwargs.items():
        print(f"{chave}: {valor}")

# Chamando a função com argumentos de palavra-chave arbitrários
mostrar_informacoes(nome="Alice", idade=30, cidade="Exemplo")

# Outro exemplo
mostrar_informacoes(curso="Python", nivel="Iniciante", plataforma="Online")
```

O operador **kwargs é útil quando você precisa criar funções flexíveis que podem lidar com várias informações de configuração ou opções, como na construção de funções genéricas e utilitárias.

Args e Kwargs

Você também pode combinar *args e **kwargs na definição de uma função para receber argumentos posicionais e nomeados em conjunto.

```
def minha_funcao(*args, **kwargs):
    for arg in args:
        print(arg)
    for chave, valor in kwargs.items():
        print(chave, valor)

minha_funcao("Curriculo", "Desenvolvedor", nome="Alice", idade=25)
```

A função minha_funcao( ) recebe tanto argumentos posicionais (“Curriculo” e “Desenvolvedor”) quanto argumentos nomeados (nome="Alice" e idade=25), e a função os manipula separadamente.

Funções Lambda

- As funções lambda, também conhecidas como funções anônimas, são funções pequenas e concisas que podem ser definidas em uma única linha de código. Elas são úteis quando você precisa de uma função simples que será usada apenas em um contexto específico.

- As funções lambda não têm um nome definido, pois são anônimas. Elas são usadas principalmente como argumentos de outras funções ou em situações em que você precisa de uma função temporária.

- A sintaxe geral de uma função lambda é a seguinte: lambda parâmetros: comando. Elas são definidas usando a palavra-chave lambda, seguida pelos argumentos da função, dois pontos :, e a expressão que será executada e retornada pela função.

```
variável = lambda parâmetro : comando
```

Funções Lambda

```
quadrado = lambda x : x ** 2   #← Função lambda para calcular o quadrado de um numero.
print(quadrado(5))

par = lambda x: x % 2 == 0    #← Função lambda para verificar se o numero é par.
print(par(10))

name_upperCase = lambda n : n.upper() # ← Função lambada em strings
print(name_upperCase("jose"))
```

Expressões condicionais Funções Lambda

- Você pode usar expressões condicionais em funções lambda para criar lógica condicional dentro da expressão. Nesse exemplo, a função lambda par_impar recebe um número x.

- A expressão condicional if x % 2 == 0 else "ímpar" verifica se x módulo de 2 é igual a zero. Se a condição for verdadeira, a função retorna a string "par"; caso contrário, retorna a string "ímpar".

Expressões condicionais Funções Lambda

```
# Função lambda usando condicional para verificar se um número é par ou ímpar
par_impar = lambda x: "par" if x % 2 == 0 else "ímpar"

# Exemplos de uso da função lambda
print(par_impar(5)) # Saída: ímpar
print(par_impar(-2)) # Saída: par
```

Também podemos usar expressões condicionais mais complexas dentro de funções lambda, como veremos a seguir:

Expressões condicionais Funções Lambda

```
# Função lambda usando condicional para classificar informações em três categorias de menssagens.
valida_usuarios = lambda user: "Erro: usuario precisa ser definido" if user == "" else ("usuario não
pode ter menos de 4 digitos" if len(user) < 4 else "usuario definido com sucesso!")

# Exemplos de uso da função lambda
print(valida_usuarios(""))
print(valida_usuarios("zé"))
print(valida_usuarios("josé"))
```

Nesse caso, a função lambda valida_usuarios recebe uma string user. A expressão condicional verifica se user é igual a “”; se for verdadeiro, retorna “erro: usuário precisa ser definido”. Caso contrário, ela verifica se o comprimento da string atribuída a user é menor que 4; se for verdadeiro, retorna “usuário não pode ter menos de 4 dígitos”. Se nenhuma das condições anteriores for atendida, a função retorna a string "usuário definido com sucesso".

Funções Lambda

As funções lambda também são frequentemente usadas em combinações de funções integradas ao Python. As funções map(), filter() e reduce() são muito úteis para manipulação de dados em Python. Elas permitem que você aplique transformações em elementos de uma sequência, filtre elementos com base em condições e reduza uma sequência a um único valor. Ao combinar essas funções com funções lambda, você pode escrever código mais conciso e expressivo.
A seguir vamos aprender como cada uma funciona.
A função map() recebe uma função e uma sequência (como uma lista) como argumentos e aplica a função a cada elemento da sequência. Ela retorna um objeto map que pode ser convertido em uma lista, se necessário. A função map() é útil quando você deseja aplicar uma determinada operação a todos os elementos de uma sequência.
Exemplo usando map() com uma função lambda:

```
numeros = [1, 2, 3, 4, 5]
quadrados = list(map(lambda x: x ** 2, numeros))
print(quadrados)
```

Funções Lambda

A função reduce() está disponível no módulo functools e recebe uma função e uma sequência como argumentos. Ela aplica a função cumulativamente aos elementos da sequência, de modo que cada aplicação sucessiva usa o resultado da aplicação anterior. Ela retorna um único valor como resultado final.

```
from functools import reduce

numeros = [1, 2, 3, 4, 5]
soma = reduce(lambda x, y: x + y, numeros)
print(soma)
```

Nessa aula aprendemos como usar args e kwargs nas funções, expressões condicionais em funções lambda para criar lógica condicional simples e também como usar funções agregadoras para manipulação de dados em Python.
Lembre-se de que as funções lambda são mais adequadas para tarefas simples, expressões mais complexas podem se tornar difíceis de ler e entender.

ATIVIDADE PRÁTICA 3

Crie uma função chamada concatenar_strings que aceita um número variável de strings como argumentos posicionais (usando *args). A função deve concatenar todas as strings em uma única string e retorná-la.

ATIVIDADE PRÁTICA 4

Crie uma função que aceita uma lista de números e use a função map para retornar uma nova lista contendo o dobro de cada número na lista de entrada.

ATIVIDADE PRÁTICA 5

Crie uma função que aceita uma lista de números e use a função filter para retornar uma nova lista contendo apenas os números pares da lista de entrada.

ATIVIDADE PRÁTICA 6

Crie uma função que aceita uma lista de strings e use a função reduce (importada de functools) para encontrar a maior string na lista.

ATIVIDADE PRÁTICA 7

Crie uma função chamada criar_lista_de_compras que aceita um número variável de itens de compras como argumentos posicionais (usando *args). A função deve criar e retornar uma lista de compras que contenha todos os itens fornecidos.

ATIVIDADE PRÁTICA 8

Crie uma função que aceite dois números e uma operação (por exemplo, adição, subtração, multiplicação, divisão) como argumentos e use funções lambda para realizar a operação especificada. A função deve retornar o resultado da operação.

DESAFIO PRÁTICO

Processador de Texto - passo 1

Crie um processador de texto simples que realiza várias operações em um texto de entrada, como contar palavras, contar letras, inverter o texto e substituir palavras-chave.

Requisitos:

- Crie uma função chamada processador_texto que aceite uma string de texto como argumento.

DESAFIO PRÁTICO

Processador de Texto - passo 2

- A função deve aceitar uma série de operações como argumentos de palavra-chave, usando **kwargs. As operações podem incluir "contar_palavras", "contar_letras", "inverter_texto" e "substituir_palavra".

- Use funções lambda para realizar as operações de acordo com as palavras-chave especificadas nos argumentos de palavra-chave.

DESAFIO PRÁTICO

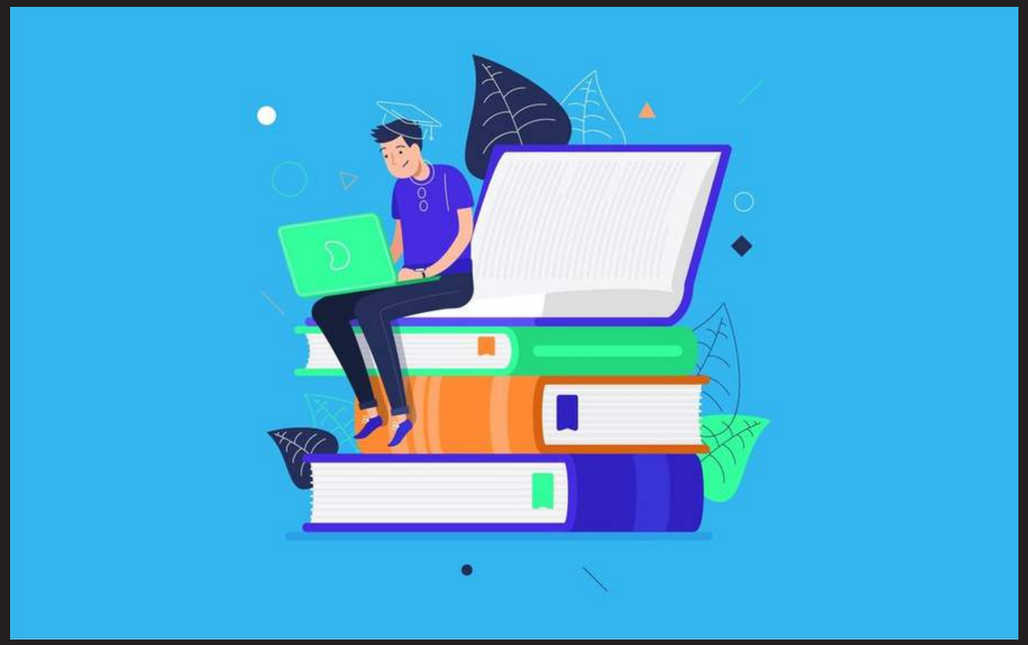
Processador de Texto - passo 3

- Se a operação "substituir_palavra" for especificada, a função deve aceitar uma palavra-chave adicional, como "substituir_palavra" e "nova_palavra", para realizar a substituição em todo o texto.

- A função deve retornar o texto resultante após todas as operações.

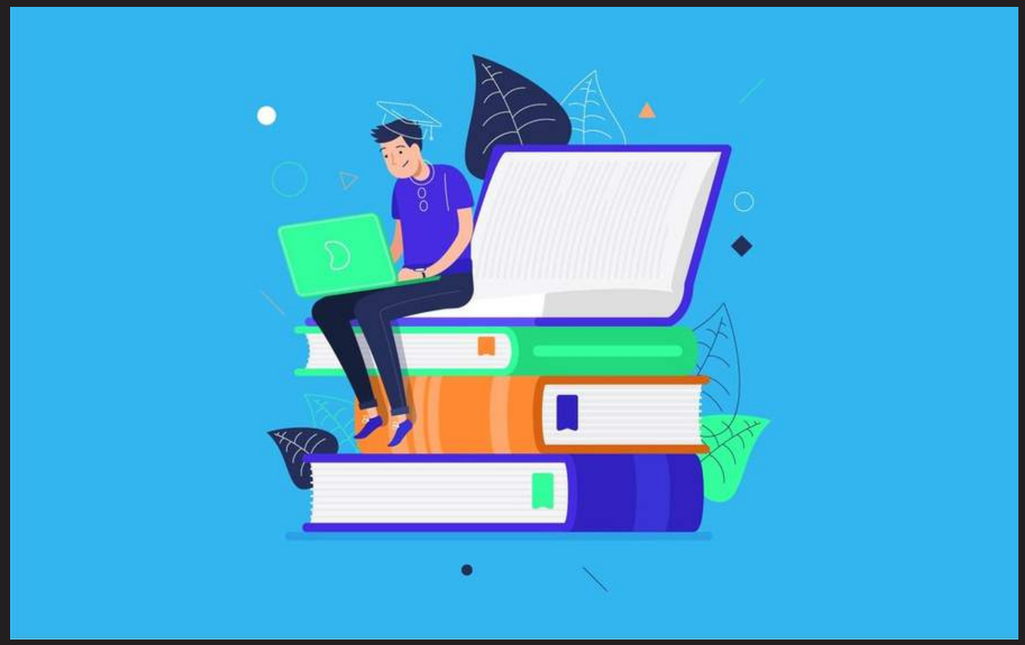
AMBIENTES VIRTUAIS

Ambientes virtuais são uma ferramenta fundamental em Python que permitem isolar e gerenciar de maneira eficaz as dependências de projetos diferentes. Eles criam um ambiente separado em que as bibliotecas e pacotes Python podem ser instalados, garantindo que um projeto não afete o ambiente global do sistema ou outros projetos. Isso é particularmente útil quando você trabalha em vários projetos Python que têm diferentes requisitos de biblioteca ou versões.



AMBIENTES VIRTUAIS

Para criar um ambiente virtual em Python, você pode usar a biblioteca padrão venv (para Python 3.3 e versões posteriores) ou ferramentas de terceiros, como virtualenv ou conda, dependendo das suas necessidades. O processo geralmente envolve a criação de um diretório que contém uma cópia isolada do interpretador Python e um diretório lib onde as bibliotecas podem ser instaladas.



AMBIENTES VIRTUAIS

Passo a passo para criar e gerenciar um ambiente virtual em Python usando o módulo venv

Passo 1: Abra um terminal ou prompt de comando.

Passo 2: Navegue até o diretório onde você deseja criar o ambiente virtual. Você pode usar os comandos cd (Change Directory) no terminal para navegar até o diretório desejado.

AMBIENTES VIRTUAIS

Passo 3: Para criar o ambiente virtual, use o seguinte comando:

→ python -m venv myenv ←

Substitua "myenv" pelo nome que você deseja dar ao seu ambiente virtual.

Passo 4: Para ativar o ambiente virtual, use o seguinte comando, dependendo do seu sistema operacional:

No Windows: myenv/scripts/activate

No macOS e Linux: source myenv/bin/activate

Após ativar o ambiente virtual, você verá o nome do ambiente no seu prompt de comando, indicando que o ambiente está ativo.

AMBIENTES VIRTUAIS

Passo 5: Agora que o ambiente virtual está ativo, você pode instalar bibliotecas e pacotes Python nele usando o pip.
Por exemplo: pip install nome_da_biblioteca

Dica: Para listar as bibliotecas instaladas em um ambiente virtual, você pode usar o comando pip list.

Passo 6: Quando terminar de trabalhar no seu projeto e quiser sair do ambiente virtual, você pode desativá-lo usando o seguinte comando: deactivate

Se você deseja reativar o ambiente virtual posteriormente, basta repetir o Passo 4.

PROJETO

Desenvolver um programa de linha de comando que permite aos usuários gerenciar suas tarefas diárias, atribuindo-lhes prioridades e categorias. O projeto será organizado em várias partes e usará funções, listas, tuplas, dicionários, conjuntos e um ambiente virtual. Passos do projeto:

Configuração do Ambiente Virtual:
Crie um ambiente virtual usando o módulo venv

PROJETO

Definição de Dados:

- Defina estruturas de dados para representar tarefas. Cada tarefa pode incluir informações como nome, descrição, prioridade e categoria. Você pode usar dicionários para representar as tarefas.

Funções:

- Crie funções para adicionar tarefas, listar tarefas, marcar tarefas como concluídas, exibir tarefas por prioridade ou categoria, e outras funcionalidades que desejar.

Menu de Comandos:

- Crie um menu de comandos de linha de comando que permita ao usuário interagir com o programa.

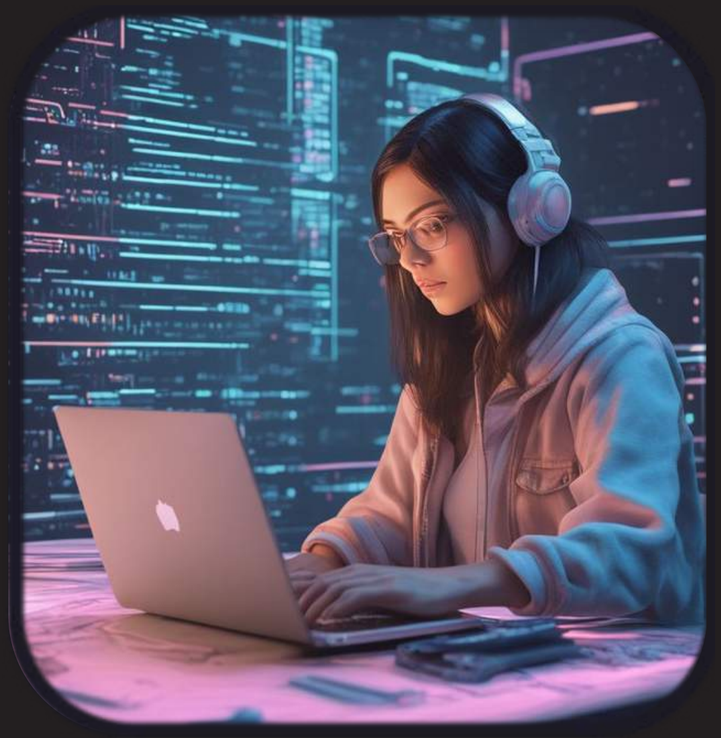
Material Complementar

- 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 06 DE PYTHON:
MÓDULOS E BIBLIOTECAS.

IN

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

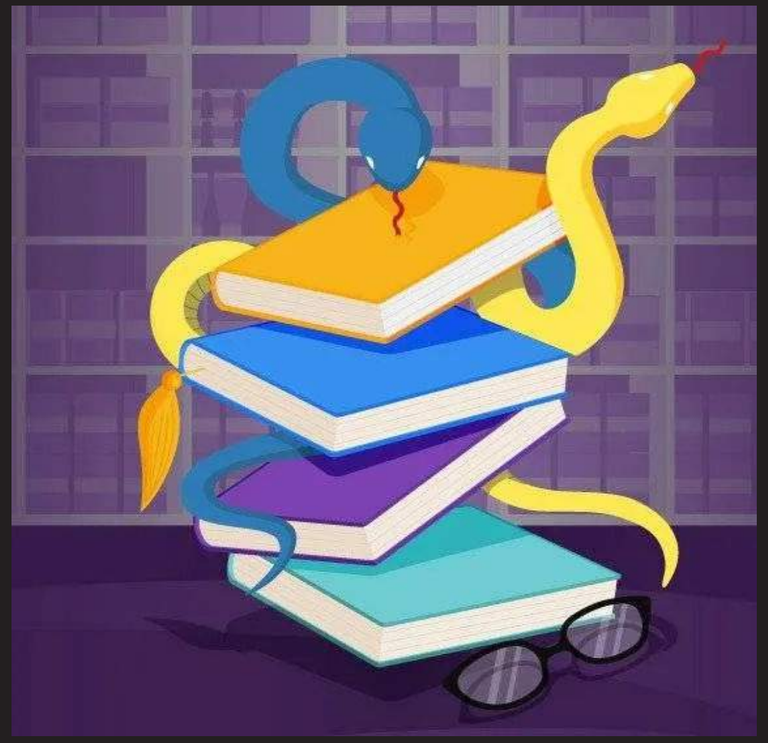
Oque são Módulos

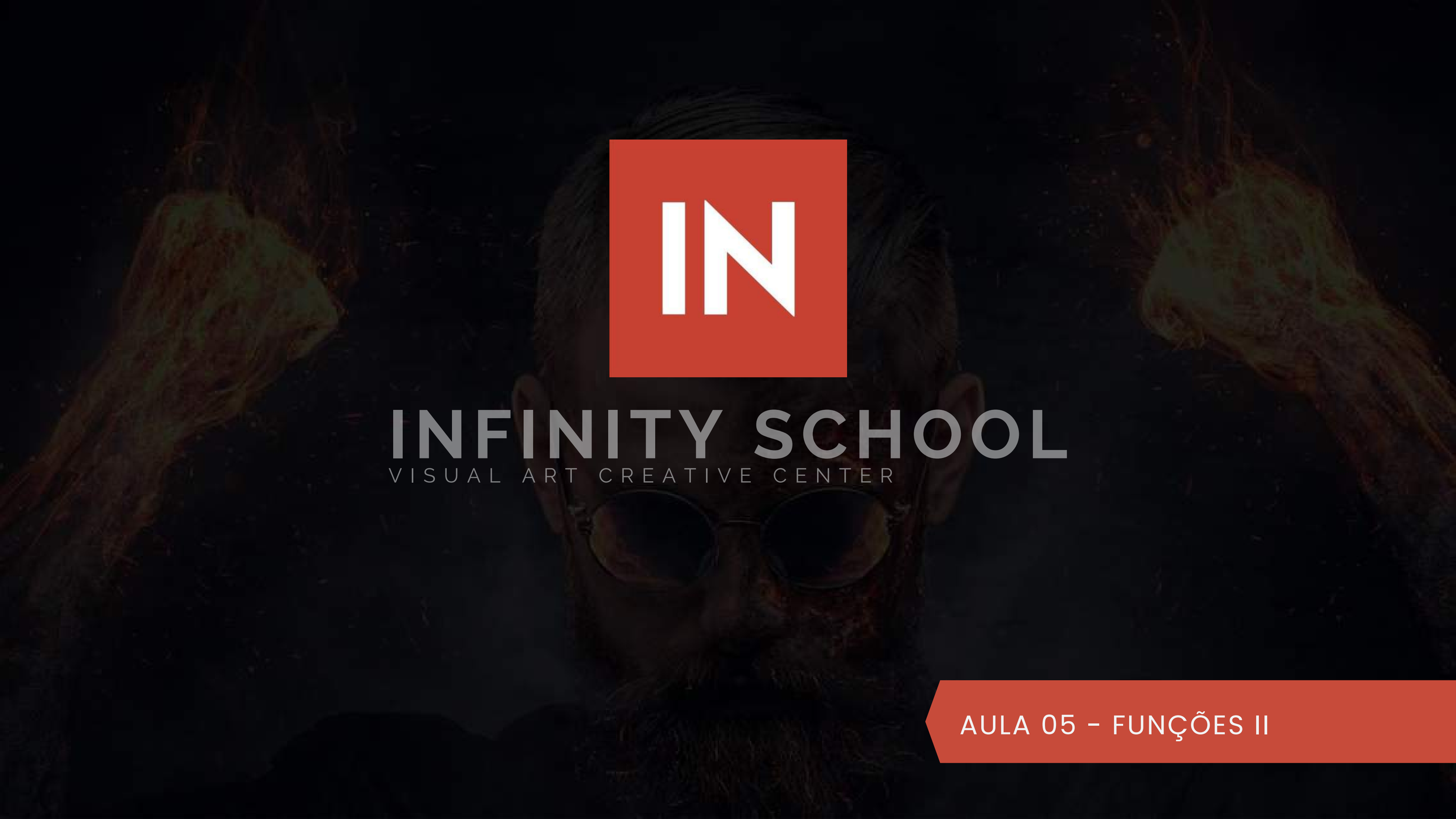
Ao programar, é importante dividir o código em diferentes arquivos .py, ou módulos, para evitar problemas como dificuldade de legibilidade e manutenção. Isso permite que cada arquivo contenha um pedaço de código, tornando-o mais organizado.

- 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.

Oque são 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.





INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 05 – FUNÇÕES II