Aula 06

Modularização de programas com uso de funções

Tópicos

- Introdução à Modularização
- Funções em Python
- Módulos em Python
- Módulos Built-in
- Importação Específica

Introdução à Modularização

Introdução à Modularização

- O que é modularização?
- Benefícios da modularização: Reutilização de código, manutenção facilitada, organização.
- Diferença entre programas monolíticos e modulares.

O que é Modularização

- É a prática de dividir um programa grande e complexo em partes menores e mais gerenciáveis chamadas módulos.
- Cada módulo tem uma função específica e é responsável por uma parte do problema.

```
[ ]  # Módulo de gestão de clientes
     def adicionar_cliente(nome, email):
         print(f"Cliente {nome} adicionado com sucesso!")

     # Módulo de processamento de pagamentos
     def processar_pagamento(valor):
         print(f"Pagamento de R${valor:.2f} processado com sucesso!")
```

Vantagens

- Reutilização de código
- Manutenção facilitada
- Organização de Leitura

Diferença entre Programas Monolíticos e Modulares

- Programas Monolíticos:
  - Descrição: Todo o código é escrito em um único arquivo, o que pode levar a um código confuso e difícil de manter.
  - Desvantagens: Dificuldade na manutenção, baixa reutilização de código, alta propensão a erros quando o código cresce.

- Programas Modulares:
  - Descrição: O código é dividido em módulos independentes, cada um com uma responsabilidade clara.
  - Vantagens: Melhor organização, manutenção facilitada, maior reutilização de código.

Funções em Python

Funções em Python

- Definição e criação de funções (`def` keyword)
- Parâmetros e argumentos
- Retorno de valores (`return` keyword)
- Escopo de variáveis (local e global)
- Funções com múltiplos retornos

Definição e Criação de Funções

- Uma função é um bloco de código reutilizável que realiza uma tarefa específica. As funções ajudam a estruturar o código, evitando a repetição e tornando-o mais organizado e legível.
- Criação de Funções:
  - Para definir uma função em Python, utilizamos a palavra-chave def, seguida pelo nome da função e parênteses. Dentro dos parênteses, podemos passar parâmetros (opcional).

```
[ ]  # Definindo uma função que imprime uma saudação
     def saudacao():
         print("Olá, bem-vindo!")

     # Chamando a função
     saudacao()
```

Parâmetros, Argumentos e retorno de Valores

- Parâmetros e Argumentos:
  - Parâmetros: São variáveis listadas dentro dos parênteses na definição da função. Eles funcionam como "espaços reservados" para os valores que a função espera receber.
  - Argumentos: São os valores reais passados para a função quando ela é chamada.
- Retorno de Valores:
  - Retorno (return keyword): A palavra-chave return é usada para retornar um valor da função para o código que a chamou.

Exemplos

```python
# Função que soma dois números
def soma(a, b):  # a e b são parâmetros
    return a + b

# Chamando a função com argumentos
resultado = soma(3, 5)
print(resultado)  # Saída: 8
```

```
8
```

```python
# Função que retorna o quadrado de um número
def quadrado(x):
    return x * x

valor = quadrado(4)
print(valor)  # Saída: 16
```

```
16
```

Escopo de variáveis

- Escopo Local: Variáveis definidas dentro de uma função só existem enquanto a função está sendo executada e não são acessíveis fora dela.
- Escopo Global: Variáveis definidas fora de qualquer função são globais e podem ser acessadas por qualquer parte do código.

```python
x = 10  # Variável global

def minha_funcao():
    x = 5  # Variável local
    print(x)  # Saída: 5

minha_funcao()
print(x)  # Saída: 10
```

```
5
10
```

Funções com múltiplos retornos

- Uma função pode retornar mais de um valor ao mesmo tempo usando a palavra-chave return seguida de uma tupla.

```
[4]  # Função que retorna a soma e a diferença de dois números
     def operacoes(a, b):
         soma = a + b
         diferenca = a - b
         return soma, diferenca

     # Capturando múltiplos retornos
     resultado_soma, resultado_diferenca = operacoes(10, 4)
     print(resultado_soma)  # Saída: 14
     print(resultado_diferenca)  # Saída: 6
```

```
14
6
```

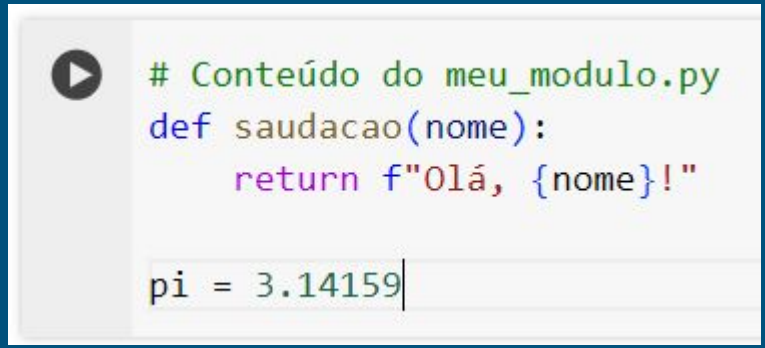
Módulos em Python

Módulos em Python

- O que são módulos?
- Como criar um módulo em Python (arquivos `.py`)
- Importando módulos com `import`
- Usando funções e variáveis de um módulo
- Renomeando módulos e funções durante a importação (`as` keyword)

O que são e como criar um Módulo

- Um módulo em Python é um arquivo que contém definições e implementações de funções, classes, e variáveis. Módulos ajudam a organizar o código em partes menores e reutilizáveis.
- Um módulo é simplesmente um arquivo com a extensão .py que contém código Python. Este código pode incluir funções, classes, e variáveis.



Importando Módulos

- Uso do import: Para usar um módulo em outro script, utilizamos a palavra-chave import seguida do nome do módulo.

```
[ ]  # Importando o módulo
     import meu_modulo

     # Usando a função e variável do módulo
     mensagem = meu_modulo.saudacao("Alice")
     print(mensagem)  # Saída: "Olá, Alice!"

     print(meu_modulo.pi)  # Saída: 3.14159
```

Usando funções e variáveis de um módulo

- Acessando componentes do módulo: Após importar, as funções e variáveis do módulo podem ser acessadas usando a notação de ponto (.).
- Boas práticas: Organizar funcionalidades relacionadas em módulos faz o código mais limpo e estruturado.

```
[ ]  # Importando o módulo
     import meu_modulo

     # Usando a função e variável do módulo
     mensagem = meu_modulo.saudacao("Alice")
     print(mensagem)  # Saída: "Olá, Alice!"

     print(meu_modulo.pi)  # Saída: 3.14159
```

Renomeando Módulo e Funções

- Uso do ‘as’: Às vezes, podemos desejar renomear um módulo para simplificar o acesso ou evitar conflitos de nomes. Usamos a palavra-chave ‘as’ para atribuir um alias ao módulo.

```
[ ]  # Importando e renomeando o módulo
     import meu_modulo as mm

     # Usando o módulo com o alias
     print(mm.saudacao("Bob"))  # Saída: "Olá, Bob!"
     print(mm.pi)  # Saída: 3.14159
```

Renomeando Módulo e Funções

- Renomeando componentes específicos: Também podemos renomear funções ou variáveis individuais de um módulo durante a importação.

```
[ ]  # Importando e renomeando uma função
     from meu_modulo import saudacao as greet

     # Usando a função renomeada
     print(greet("Charlie"))  # Saída: "Olá, Charlie!"
```

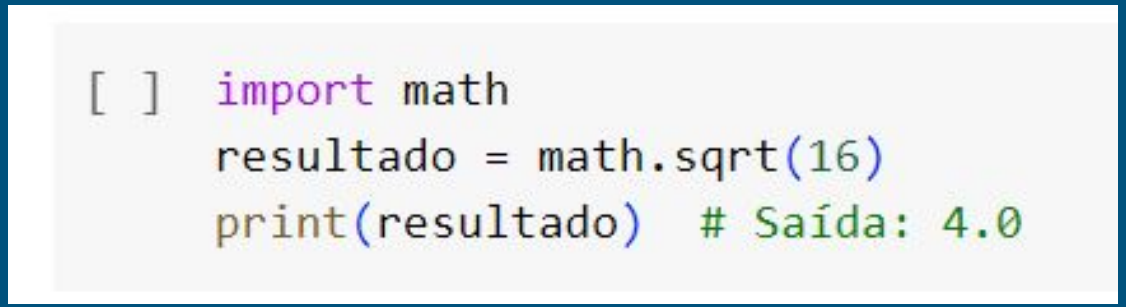
Módulos Built-in

Módulos Built-in

- Introdução aos módulos padrão de Python
- Exemplos práticos:
  - `math` (operações matemáticas)
  - `random` (geração de números aleatórios)
  - `datetime` (trabalhando com datas e horas)
  - `os` (interação com o sistema operacional)
  - `sys` (argumentos de linha de comando e interação com o interpretador)
  - `re` (expressões regulares)

Módulos Built-in

- São bibliotecas padrão que vêm incluídas com a instalação do Python. Eles fornecem funcionalidades prontas para uso, eliminando a necessidade de reescrever código comum.
  - Vantagem: Esses módulos são otimizados e testados, oferecendo uma maneira eficiente de realizar operações comuns, desde cálculos matemáticos até manipulação de arquivos e datas.



Módulos mais populares

- ‘math’ para operações matemáticas avançadas.
- ‘random’ para geração de números aleatórios.
- ‘datetime’ para manipulação de datas e horas.
- ‘os’ para interação com o sistema operacional.
- ‘sys’ para manipulação de argumentos da linha de comando e interação com o interpretador.
- ‘re’ para operações com expressões regulares.

Exemplos

```python
import math

# Calculando a raiz quadrada de um número
print(math.sqrt(25))  # Saída: 5.0

# Calculando o seno de um ângulo em radianos
print(math.sin(math.pi/2))  # Saída: 1.0
```

```
5.0
1.0
```

```python
[6] import random

    # Gerando um número aleatório entre 1 e 10
    numero_aleatorio = random.randint(1, 10)
    print(numero_aleatorio)

    # Embaralhando uma lista
    lista = [1, 2, 3, 4, 5]
    random.shuffle(lista)
    print(lista)
```

```
8
[2, 4, 5, 3, 1]
```

Exemplos

```
[7]  from datetime import datetime

     # Obtendo a data e hora atual
     agora = datetime.now()
     print(agora)

     # Formatando a data
     print(agora.strftime("%d/%m/%Y %H:%M:%S"))
```

```
2024-09-01 22:28:57.839108
01/09/2024 22:28:57
```

```
[9]  import re

     # Verificando se uma string contém um padrão específico
     padrao = r'\d+'  # Padrão para um ou mais dígitos
     texto = 'Meu número é 1234'
     resultado = re.search(padrao, texto)

     if resultado:
         print(f"Encontrado: {resultado.group()}")
     else:
         print("Nenhum número encontrado")
```

```
Encontrado: 1234
```

Importação Específica

Importação Específica

- Importação de funções e variáveis específicas (`from ... import ...`)
- Importação de tudo de um módulo (`from ... import *`)
- Cuidados ao usar `import *` (poluição do namespace)

Importação específica em Python

- Em vez de importar todo um módulo, você pode importar apenas funções ou variáveis específicas que deseja usar.

```
[10] from math import sqrt, pi

     # Usando a função sqrt e a constante pi sem precisar do prefixo "math."
     raiz = sqrt(25)
     print(f"Raiz quadrada de 25: {raiz}")  # Saída: 5.0

     print(f"Valor de pi: {pi}")  # Saída: 3.141592653589793
```

```
Raiz quadrada de 25: 5.0
Valor de pi: 3.141592653589793
```

Boa prática:

- Use ‘from ... import ...’ quando precisar de apenas algumas funções ou variáveis de um módulo. Isso mantém seu código limpo e fácil de entender.

```
[12] from random import randint

     numero = randint(1, 10)
     print(f"Número aleatório entre 1 e 10: {numero}")
```

```
Número aleatório entre 1 e 10: 1
```

Importação de todo o módulo

- Em vez de importar todo um módulo, você pode importar apenas funções ou variáveis específicas que deseja usar.

```
[11] from math import *

     # Agora você pode usar qualquer função ou variável do módulo
     # math sem precisar do prefixo "math."
     resultado = sin(pi/2)
     print(f"Seno de pi/2: {resultado}")  # Saída: 1.0
```

```
Seno de pi/2: 1.0
```

Cuidados

- Poluição do Namespace: Todas as funções e variáveis são importadas para o namespace atual, o que pode causar conflitos de nomes e tornar o código difícil de ler e depurar.
- Perda de Clareza: Não fica claro de onde vêm as funções e variáveis, o que pode dificultar a manutenção do código.

    - O Namespace (ou "espaço de nomes") em Python é um sistema que permite organizar e controlar o escopo e a acessibilidade dos nomes usados no programa, como nomes de variáveis, funções, classes, e módulos. Ele funciona como um contêiner que mapeia os nomes para os objetos correspondentes, garantindo que não haja conflitos entre nomes que podem aparecer em diferentes partes do código.

Vamos praticar!
Link

Vamos exercitar!
Link