Python

Python

Nome: _____

Sobre o curso

Este curso oferece uma introdução abrangente e prática à linguagem de programação Python, especialmente projetado para iniciantes. Durante a jornada de aprendizado, você construirá uma base sólida em Python, abordando conceitos essenciais de forma acessível e envolvente.

Quantidade de Aulas
16 aulas

Carga horária
24 horas

Programas utilizados
Python Versão 3.11.4 e Pycharm-community

O que aprender com este curso?

Ao longo deste curso, você desenvolverá habilidades fundamentais que o capacitarão a criar programas simples, resolver problemas de maneira eficaz e dar início à sua emocionante jornada na programação. Com exemplos interativos e exercícios práticos, este curso não apenas fornecerá conhecimento, mas também a confiança necessária para explorar projetos mais complexos e enfrentar desafios futuros com segurança. Venha conosco nesta emocionante jornada de aprendizado em Python!

Sumário

1.1. Python

Python é uma linguagem de programação de alto nível, interpretada e versátil que se destaca pela sua simplicidade e legibilidade. Criada por Guido van Rossum nos anos 90, Python foi projetada para ser fácil de aprender e utilizar, tornando-se uma das linguagens mais populares entre programadores de diferentes níveis de habilidade. Com uma sintaxe clara e concisa, Python é conhecida por sua abordagem orientada a objetos e sua ampla biblioteca padrão, que oferece uma vasta gama de funcionalidades prontas para uso. Essa linguagem suporta múltiplos paradigmas de programação, como programação procedural, funcional e orientada a objetos, permitindo que os desenvolvedores a utilizem em diversas aplicações, desde a criação de scripts simples até o desenvolvimento de aplicativos web, ciência de dados, automação de tarefas e muito mais. Python é amplamente valorizado por sua legibilidade e facilidade de manutenção, tornando-se uma escolha popular para iniciantes e profissionais em todo o mundo.

1.2. Um pouco de história

A história do Python começa no final dos anos 80, quando Guido van Rossum, um programador holandês, começou a trabalhar em uma nova linguagem de programação como um hobby durante o Natal de 1989. Ele deu ao projeto o nome de "Python" em homenagem ao programa de televisão britânico "Monty Python's Flying Circus".

A primeira versão pública do Python, a 0.9.0, foi lançada em fevereiro de 1991. Desde o início, Guido enfatizou a legibilidade e a simplicidade da linguagem, o que a tornava ideal para iniciantes. Em 2000, foi lançada a versão 2.0, que trouxe melhorias significativas, consolidando ainda mais a popularidade da linguagem.

Ao longo dos anos, Python se desenvolveu e cresceu, ganhando destaque em diversos campos, como desenvolvimento web, ciência de dados, automação, inteligência artificial e muito mais. A comunidade Python se expandiu rapidamente, e a linguagem tornou-se uma das mais populares no mundo da programação.

Uma grande mudança ocorreu em 2008, quando o Python 3 foi lançado, trazendo melhorias significativas, mas também quebrando

a compatibilidade com versões anteriores, o que gerou um período de transição para a adoção da nova versão.

Com sua filosofia "The Zen of Python" e uma comunidade ativa, Python continuou a evoluir e a ganhar mais adeptos, tornando-se uma das principais escolhas para projetos e aplicações em diversos setores, graças à sua facilidade de uso, legibilidade e versatilidade. Atualmente, Python continua a ser uma das linguagens de programação mais influentes e populares do mundo.

Momento curiosidade:

O logotipo do Python é conhecido como "Python logo" ou "Pythonic logo" e retrata uma cobra enrolada em torno de um "P". A escolha da cobra como símbolo do Python foi feita para representar a flexibilidade, poder e elegância da linguagem. Curiosamente, ao longo dos anos, surgiram várias versões e interpretações do logotipo Python, incluindo variações com diferentes estilos de cobra, cores e até mesmo adaptações criativas. Esse elemento visual contribui para a identidade única e reconhecível da linguagem Python.

1.3. IDE

IDE (Integrated Development Environment) é um ambiente de desenvolvimento integrado que oferece uma série de ferramentas e recursos para auxiliar os desenvolvedores na criação, edição, depuração e gerenciamento de software.

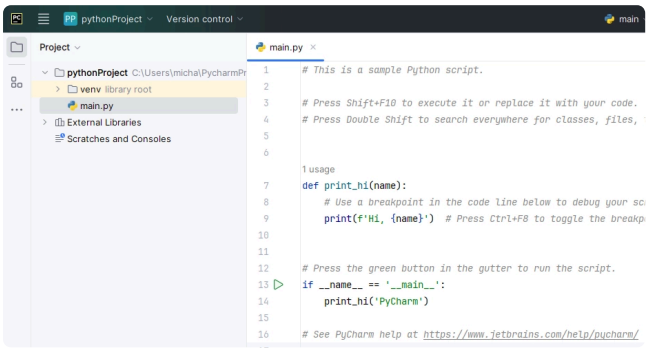
Uma IDE geralmente é composta por um editor de código, que permite escrever e editar programas, fornecendo recursos como realce de sintaxe, sugestões de código e formatação automática. Além disso, ela inclui um compilador ou interpretador que traduz o código-fonte em um formato executável ou interpreta o código em tempo real.

Existem várias IDEs disponíveis para diferentes linguagens de programação, como PyCharm para Python, Visual Studio para C#, Eclipse para Java, entre muitas outras. Essas ferramentas proporcionam um ambiente de trabalho completo e eficiente para os desenvolvedores, aumentando a produtividade e facilitando o desenvolvimento de software.

1.4. PyCharm

O PyCharm é uma IDE (Integrated Development Environment) desenvolvida pela JetBrains e voltada especificamente para programação em Python. Ele oferece uma ampla gama de recursos e ferramentas para facilitar o desenvolvimento de projetos Python.

O PyCharm possui um editor de código poderoso, com recursos como realce de sintaxe, autocompletar, refatoração de código e formatação automática. Ele oferece suporte a diferentes versões do Python e possui integração com virtualenv e Anaconda, permitindo criar e gerenciar ambientes virtuais para isolar projetos e suas dependências.



python

```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace
it with your code.
# Press Double Shift to search everywhere
for classes, files, tool windows, actions,
and settings.


def print_hi(name):
    # Use a breakpoint in the code line
below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to
toggle the breakpoint.


# Press the green button in the gutter to
run the script.
if __name__ == '__main__':
    print_hi('PyCharm')
```

Em resumo, o PyCharm é uma poderosa IDE Python que oferece um conjunto abrangente de recursos para facilitar o desenvolvimento de software em Python, desde a edição de código até a depuração e o gerenciamento de projetos.

1.5. Exercícios opcionais

Exercício 1 - Complete as frases com a palavra correta

1. O Python foi criado por _____.

2. Python é uma linguagem de programação de _____ nível.

3. A sigla IDE significa _____ _____ _____.

4. Um _____ é uma instância de uma classe.

5. O programa de TV que inspirou o nome Python foi o _____ _____'s Flying Circus.

Exercício 2 - Responda as questões abaixo

1. Qual foi a inspiração para o nome da linguagem Python?

A) Um tipo de cobra venenosa

B) Um jogo de computador

C) Um programa de televisão britânico

D) Um livro de ficção científica

2. Python pode ser usada para:

A) Apenas desenvolvimento de jogos

B) Somente automação de tarefas simples

C) Apenas para iniciantes em programação

D) Aplicações variadas como ciência de dados, automação e web

3. A versão 2.0 do Python foi lançada no ano:

A) 1989

B) 1991

C) 2000

D) 2008

4. Responda com Verdadeiro ou Falso.

(} Python é uma linguagem compilada que exige transformação para código binário antes de rodar.

5. Um dos principais motivos para a popularidade do Python é:

A) Sua complexidade sintática

B) A dificuldade de manutenção

C) Sua simplicidade e legibilidade

D) A ausência de bibliotecas

6. Responda com Verdadeiro ou Falso.

( ) O lançamento do Python 3 não causou mudanças importantes em relação às versões anteriores.

7. Qual das alternativas representa um paradigma de programação suportado por Python?

A) Programação visual

B) Programação orientada a objetos

C) Programação mecânica

D) Programação randômica

8. O logotipo do Python representa:

A) Um computador antigo

B) Duas engrenagens entrelaçadas

C) Uma cobra enrolada em torno da letra “P”

D) Um robô com uma serpente

9. Responda com Verdadeiro ou Falso.

( ) IDE é uma ferramenta que permite apenas escrever o código, sem oferecer recursos de depuração.

10. Sobre o PyCharm, assinale a correta:

A) É uma linguagem de programação concorrente ao Python

B) Foi desenvolvido pela Microsoft

C) Permite trabalhar com ambientes virtuais como o Anaconda

D) Só funciona com códigos escritos em C++

Anotações

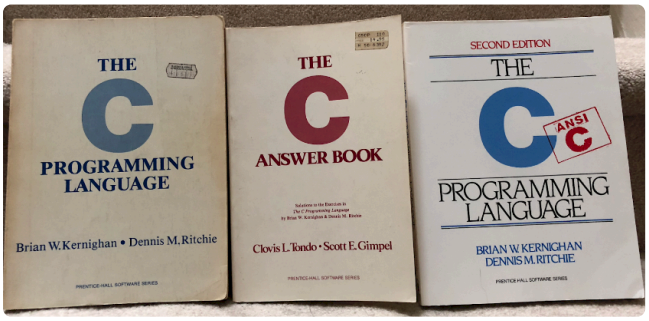
Python
2. Primeiros passo com Python

2.1. Hello, world!

A frase "Hello, world!" é uma das mais famosas e icônicas no mundo da programação. Ela tem sido tradicionalmente usada como um exemplo básico de código para ilustrar o funcionamento de uma nova linguagem de programação ou ambiente de desenvolvimento.

```
1  print("Hello, World!")

Hello, World!
[Finished in 0.2s]
```

A origem do "Hello, world!" remonta ao início dos anos 1970, quando o livro "The C Programming Language", escrito por Brian Kernighan e Dennis Ritchie, foi publicado. O livro, conhecido como "K&R C", foi um marco importante na popularização da linguagem C.



Dentro do livro, um exemplo de programa em C foi apresentado com a simples mensagem "hello, world" sendo exibida na tela. Esse programa básico serviu para demonstrar como escrever um programa simples em C e mostrar a sintaxe básica da linguagem.

Desde então, a tradição de usar "Hello, world!" como o primeiro programa em uma nova linguagem ou ambiente de programação se espalhou amplamente. Programadores em todo o mundo adotaram essa prática como uma maneira simples de verificar se a configuração do ambiente de desenvolvimento está correta e de começar a explorar os conceitos fundamentais da linguagem.

Momento curiosidade:

Python é uma linguagem conhecida por sua legibilidade e ênfase na escrita de um código limpo e legível. Essa filosofia é refletida no documento chamado "The Zen of Python" (O Zen do Python), que é uma coleção de princípios orientadores para escrever código Python de forma clara e eficiente. Uma curiosidade interessante é que você pode visualizar o "The Zen of Python" diretamente no interpretador Python. Basta digitar o comando "import this" no interpretador para exibir esses princípios na tela.

2.2. Variáveis e dados

Em Python, as variáveis são espaços de memória reservados para armazenar dados. Elas servem para atribuir um nome a um valor específico, permitindo que você se refira a esse valor posteriormente no programa.

Ao usar uma variável em Python, você pode armazenar diferentes tipos de dados, como números, strings, listas, dicionários e outros

objetos. As variáveis são flexíveis e não possuem um tipo fixo, ou seja, podem ser retribuídas para armazenar diferentes tipos de dados ao longo do programa.

python
```
nome = "Maria"
idade = 25
altura = 1.65
```

Nesse exemplo, criamos três variáveis: nome, idade e altura. A variável nome armazena uma string, a variável idade armazena um número inteiro e a variável altura armazena um número de ponto flutuante.

Os dados armazenados em variáveis podem ser utilizados em expressões e operações dentro do programa. Você pode fazer cálculos, concatenar strings, acessar elementos de uma lista, entre outras operações utilizando as variáveis que contêm os dados necessários.

Tipos de variáveis no Python

1. Números inteiros (int): Variáveis do tipo inteiro são usadas para armazenar números inteiros, como 1, 10, -5, etc.

python
```
idade = 25
```

2. Números de ponto flutuante (float): Variáveis do tipo float são usadas para armazenar números com casas decimais, como 3.14, 2.5, -0.75, etc.

python
```
altura = 1.65
```

3. Strings (str): Variáveis do tipo string são usadas para armazenar sequências de caracteres, como "Olá", "Python", "123", etc

python
```
nome = "Maria"
```

4. Booleanos (bool): Variáveis do tipo booleano podem ter apenas dois valores: True (verdadeiro) ou False (falso). Elas são frequentemente usadas para expressar condições lógicas.

python
```
temperatura_alta = True
```

5. Listas (list): Variáveis do tipo lista são usadas para armazenar uma coleção ordenada de elementos. Os elementos podem ser de diferentes tipos e podem ser acessados por meio de índices.

python
```
numeros = [1, 2, 3, 4, 5]
```

6. Tuplas (tuple): Variáveis do tipo tupla são semelhantes às listas, mas são imutáveis, ou seja, não podem ser modificadas depois de criadas.

python
```
coordenadas = (10, 20)
```

7. Dicionários (dict): Variáveis do tipo dicionário são usadas para armazenar pares de chave-valor, permitindo que você acesse os valores por meio de suas chaves.

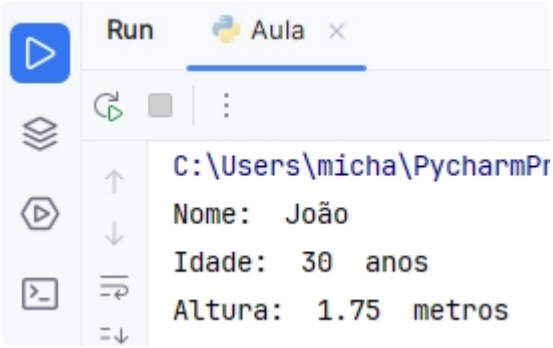
python
```
pessoa = {"nome": "João", "idade": 30}
```

Esses são apenas alguns exemplos dos tipos de variáveis mais comuns em Python. A linguagem Python é conhecida por sua flexibilidade em lidar com diferentes tipos de dados e permite que você crie seus próprios tipos de variáveis personalizados, caso necessário.

2.3. Exercícios opcionais

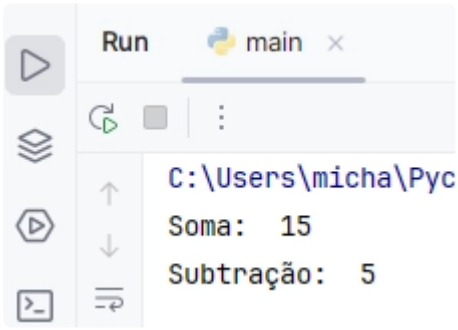
Exercício 1 - Armazenando e exibindo informações

1. Crie um novo arquivo em Python.
2. Na primeira linha do código, digite o seguinte: nome = "João" e pressione "Enter" para continuar.
3. Na segunda linha do código, digite o seguinte: idade = 30 e pressione "Enter" para continuar.
4. Na terceira linha do código, digite o seguinte: altura = 1.75 e pressione "Enter" para continuar.
5. Na quarta linha do código, digite o seguinte: print("Nome:", nome) e pressione "Enter" para continuar.
6. Na quinta linha do código, digite o seguinte: print("Idade:", idade, "anos") e pressione "Enter" para continuar.
7. Na sexta linha do código, digite o seguinte: print("Altura:", altura, "metros") e pressione "Enter" para continuar.
8. Execute o código, as informações armazenadas devem aparecer no console.
9. Confira o resultado abaixo.



Exercício 2 - Realizando cálculo com variáveis

1. Crie um novo arquivo em Python.
2. Na primeira linha do código, digite o seguinte: numero1 = 10 e pressione "Enter" para continuar.
3. Na segunda linha do código, digite o seguinte: numero2 = 5 e pressione "Enter" para continuar.
4. Na terceira linha do código, digite o seguinte: soma = numero1 + numero2 e pressione "Enter" para continuar.
5. Na quarta linha do código, digite o seguinte: subtracao = numero1 - numero2 e pressione "Enter" para continuar.
6. Na quinta linha do código, digite o seguinte: print("Soma:", soma) e pressione "Enter" para continuar.
7. Na sexta linha do código, digite o seguinte: print("Subtração:", subtracao) e pressione "Enter" para continuar.
8. Execute o código, os valores calculados devem aparecer no console.
9. Confira o resultado abaixo.



Anotações

3.1. Estrutura condicional

A estrutura condicional é uma construção fundamental em programação que permite ao programador controlar o fluxo do programa com base em condições lógicas. Ela permite que diferentes blocos de código sejam executados ou ignorados dependendo do resultado da avaliação dessas condições.

Em Python, a estrutura condicional é geralmente implementada usando as palavras-chave "if", "else" e "elif". A sintaxe básica da estrutura condicional em Python é a seguinte:

```python
if condição:
    # bloco de código a ser executado se a condição for verdadeira
else:
    # bloco de código a ser executado se a condição for falsa
```

A condição é uma expressão lógica que é avaliada como verdadeira (True) ou falsa (False). Se a condição for verdadeira, o bloco de código indentado abaixo do "if" será executado. Caso contrário, o bloco de código abaixo do "else" será executado.

3.2. If, else e elif

Em Python, "if", "else" e "elif" são palavras-chave usadas para controlar o fluxo do programa com base em condições lógicas. Essas estruturas condicionais permitem que o programa tome decisões e execute diferentes blocos de código com base nos resultados dessas condições.

If

A estrutura "if" permite que você execute um bloco de código se uma condição for avaliada como verdadeira (True). Se a condição fornecida após o "if" for verdadeira, o bloco de código indentado abaixo do "if" será executado. Caso contrário, o bloco de código será ignorado.

```python
idade = 18
if idade >= 18:
    print("Você é maior de idade!")
```

Nesse exemplo, o bloco de código dentro do "if" será executado porque a condição idade >= 18 é verdadeira.

Else

A estrutura "else" é usada em conjunto com o "if" e permite que você execute um bloco de código alternativo caso a condição do "if" seja avaliada como falsa (False). O "else" não possui uma condição separada, pois é o caminho alternativo quando a condição do "if" não é verdadeira.

```python
idade = 16
if idade >= 18:
    print("Você é maior de idade!")
else:
    print("Você é menor de idade.")
```

Nesse exemplo, se a condição idade >= 18 for falsa, o bloco de código após o "else" será executado.

Elif

A palavra-chave "elif" é usada para adicionar mais condições às estruturas condicionais. Ela permite que você verifique múltiplas condições sequencialmente. Se a condição do "if" for falsa, o "elif" verificará a próxima condição e, se for verdadeira, executará o bloco de código associado a essa condição.

```python
idade = 20
if idade < 18:
    print("Você é menor de idade.")
elif idade >= 18 and idade < 21:
    print("Você é maior de idade, mas ainda não pode beber nos Estados Unidos.")
else:
    print("Você é maior de idade e pode beber nos Estados Unidos.")
```

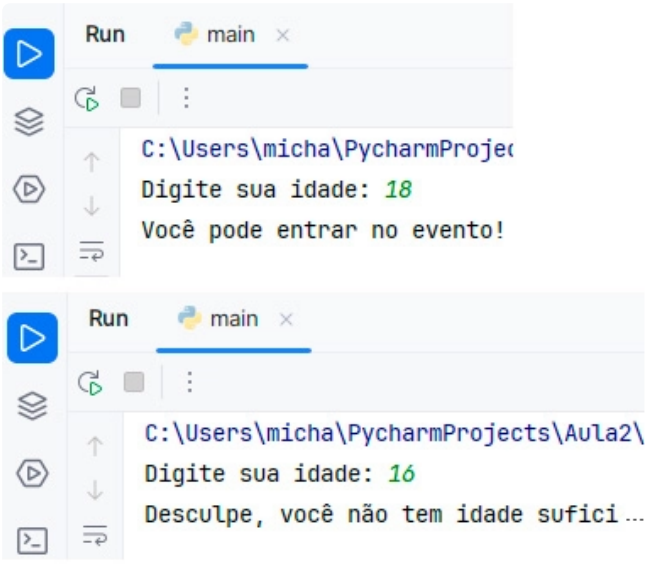
Nesse exemplo, o programa verifica diferentes faixas etárias e exibe uma mensagem de acordo com a idade fornecida.

Essas estruturas condicionais ("if", "else" e "elif") são fundamentais para controlar o fluxo do programa com base em condições lógicas e permitem que você tome decisões e execute diferentes blocos de código de acordo com essas condições

3.3. Exercícios opcionais

Exercício 1 - Verificando idade para entrada em um evento

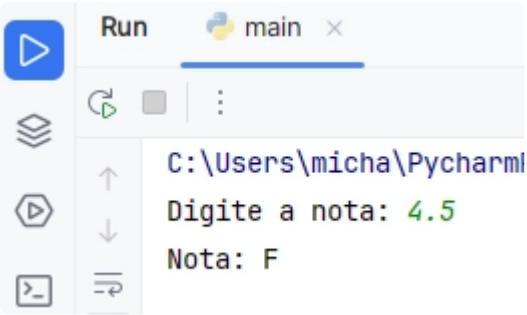
1. Crie um novo arquivo em Python.
2. Na primeira linha do código, digite o seguinte: "idade = int(input("Digite sua idade: "))" e pressione "Enter" para continuar.
3. Na segunda linha do código, digite o seguinte: "if idade >= 18:" e pressione "Enter" para continuar.
4. Na terceira linha do código, digite o seguinte: "print("Você pode entrar no evento!")" e pressione "Enter" para continuar.
5. Na quarta linha do código, digite o seguinte: "else:" e pressione "Enter" para continuar.
6. Na quinta linha do código, digite o seguinte: "print("Desculpe, você não tem idade suficiente para entrar no evento.")".
7. Execute o código e informe as duas situações, uma idade acima de 18 e outra abaixo de 18 para verificar se o código funciona.
8. Confira o resultado abaixo.



Exercício 2 - Classificação de notas escolares

1. Crie um novo arquivo em Python.
2. Na primeira linha do código, digite o seguinte: "nota = float(input("Digite a nota: "))" e pressione "Enter" para continuar.
3. Na segunda linha do código, digite o seguinte: "if nota >= 9:" e pressione "Enter" para continuar.
4. Na terceira linha do código, digite o seguinte: "print("Nota: A")" e pressione "Enter" para continuar.
5. Na quarta linha do código, digite o seguinte: "elif nota >= 7:" e pressione "Enter" para continuar.
6. Na quinta linha do código, digite o

seguinte: "print("Nota: B")" e pressione "Enter" para continuar.
7. Na sexta linha do código, digite o seguinte: "elif nota >= 5:" e pressione "Enter" para continuar.
8. Na sétima linha do código, digite o seguinte: "print("Nota: C")" e pressione "Enter" para continuar.
9. Na oitava linha do código, digite o seguinte: "else:" e pressione "Enter" para continuar.
10. Na nona linha do código, digite o seguinte: "print("Nota: F")" e pressione "Enter" para continuar.
11. Execute o código e teste todas as opções para verificar se tudo está funcionando.
12. Confira o resultado abaixo.



Anotações

4.1. O que são "loops"?

"Loops", também conhecidos como estruturas de repetição, são construções fundamentais em programação que permitem que um bloco de código seja executado repetidamente com base em uma condição específica. Os "loops" são usados para automatizar a repetição de tarefas e para processar conjuntos de dados de forma iterativa.

Existem dois tipos principais de "loops" em Python: o "loop" "while" e o "loop" "for".

Exemplo de um "loop" "while":

```python
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Nesse exemplo, o bloco de código será executado cinco vezes, pois o contador aumenta a cada iteração até que a condição contador < 5 seja falsa.

4.2. "Loop" "while"

O "loop" "while" executa um bloco de código repetidamente enquanto uma condição específica for avaliada como verdadeira (True). A estrutura básica de um "loop" "while" é a seguinte:

```python
while condição:
    # bloco de código a ser repetido enquanto a condição for verdadeira
```

O bloco de código dentro do "loop" é executado repetidamente até que a condição seja avaliada como falsa (False). É importante garantir que a condição mude eventualmente, para evitar "loops" infinitos.

4.3. "Loop" "for"

O "loop" "for" é usado para iterar sobre uma sequência de elementos, como uma lista, uma string, um dicionário, entre outros. Ele executa um bloco de código para cada elemento da sequência. A estrutura básica de um "loop" "for" é a seguinte:

```python
for elemento in sequência:
    # bloco de código a ser executado para cada elemento da sequência
```

Exemplo de um "loop" "for":

python

```
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)
```

Nesse exemplo, o bloco de código será executado três vezes, uma vez para cada elemento da lista "frutas". A variável "fruta" assume o valor de cada elemento da lista em cada iteração do "loop".

Os "loops" permitem automatizar a execução de tarefas repetitivas e processar conjuntos de dados de maneira eficiente. Eles são fundamentais para lidar com iterações, processamento de listas, leitura de arquivos, entre muitas outras situações em programação.

Como podemos usar os "loops"?

Os "loops" são amplamente utilizados em programação para automatizar a execução de tarefas repetitivas e para processar conjuntos de dados de maneira eficiente. Aqui estão alguns exemplos de como podemos usar os "loops" em Python:

1. Iterar sobre uma lista: Podemos usar um "loop" "for" para percorrer todos os elementos de uma lista e executar um bloco de código para cada elemento. Isso é útil quando queremos realizar uma operação em cada item da lista

python

```
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)
```

2. Contagem com "loop" "for": Podemos usar um "loop" "for" para executar um bloco de código um número específico de vezes. Isso é útil quando queremos repetir uma tarefa um determinado número de vezes. Por exemplo:

python

```
for i in range(5):
    print(i)
```

Nesse exemplo, o bloco de código será executado 5 vezes, imprimindo os números de 0 a 4.

3. "Loop" "while" com condição de parada: Podemos usar um "loop" "while" para executar um bloco de código enquanto uma condição específica for verdadeira. Isso é útil quando queremos repetir uma tarefa até que uma determinada condição seja atendida. Por exemplo:

python

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Nesse exemplo, o bloco de código será executado enquanto a variável "contador" for menor que 5. O valor de "contador" é incrementado a cada iteração.

4. "Loop" com controle de interrupção: Podemos usar a palavra-chave "break" para interromper a execução de um "loop" antecipadamente, com base em uma determinada condição. Isso é útil quando queremos sair de um "loop" antes de atingir a condição de parada normal. Por exemplo:

```python
for i in range(10):
    if i == 5:
        break
    print(i)
```

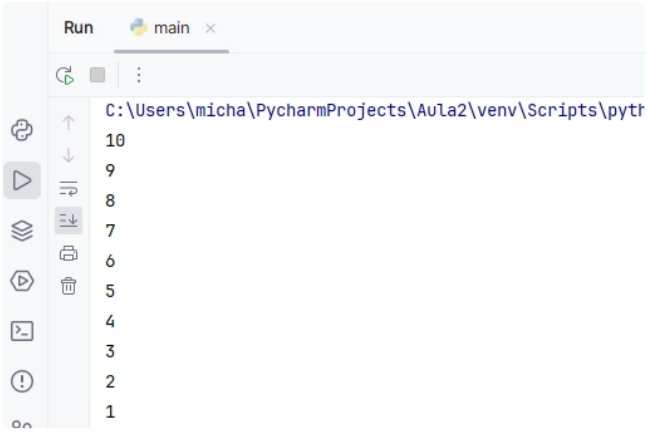
Nesse exemplo, o "loop" "for" será interrompido quando o valor de "i" for igual a 5, e a execução será encerrada.

Esses são apenas alguns exemplos de como podemos usar os loops em Python. Os "loops" fornecem um poderoso mecanismo de controle de fluxo que nos permite automatizar tarefas repetitivas e processar dados de forma eficiente, tornando nossos programas mais eficazes e flexíveis.

4.4. Exercícios opcionais

Exercício 1 - Contagem regressiva com o "loop" "while"

1. Crie um novo arquivo em Python.
2. Na primeira linha do código, digite o seguinte: "contador = 10" e pressione "Enter" para continuar.
3. Na segunda linha do código, digite o seguinte: "while contador>=0:" e pressione "Enter" para continuar.
4. Na terceira linha do código, digite o seguinte: "print(contador)" e pressione "Enter" para continuar.
5. Na quarta linha do código, digite o seguinte: "contador -= 1" e pressione "Enter" para continuar.
6. Execute o código, o programa deve iniciar uma contagem regressiva de 10 a 0.
7. Confira o resultado abaixo.



Exercício 2 - Tabuada com "loop" "for"

1. Crie um novo arquivo em Python.
2. Na primeira linha do código, digite o seguinte: "numero = int(input("Digite um número: "))" e pressione "Enter" para continuar.
3. Na segunda linha do código, digite o seguinte: "for i in range (1, 11):" e pressione "Enter" para continuar.
4. Na terceira linha do código, digite o seguinte: "resultado = numero * i" e pressione "Enter" para continuar.
5. Na quarta linha do código, digite o seguinte: "print(f"{numero} x {i} = {resultado}")" e pressione "Enter" para continuar.
6. Execute o código, o programa deve exibir a tabuada do número informado.
7. Confira o resultado abaixo.

5.1. Listas

Em Python, uma lista é uma coleção ordenada e mutável de elementos. Ela permite armazenar diversos valores em uma única variável. As listas são uma das estruturas de dados mais versáteis e fundamentais na linguagem Python e são amplamente utilizadas para manipular e organizar dados. Exemplo:

```python
frutas = ["maçã", "banana", "laranja", "uva", "manga"]
```

Para criar uma lista em Python, você pode utilizar colchetes ("[]") e separar os elementos por vírgula. Por exemplo:

```python
minha_lista = [1, 2, 3, 4, 5]
```

As listas em Python podem conter diferentes tipos de dados, como números inteiros, números de ponto flutuante, "strings", "booleanos" e até outras listas. Por exemplo:

```python
lista_mista = [1, "Python", True, 3.14, [6, 7, 8]]
```

A principal característica das listas é a possibilidade de modificar seus elementos após a criação. Isso significa que você pode adicionar, remover ou modificar elementos dentro da lista a qualquer momento.

Operações com listas

Aqui estão algumas das operações e funcionalidades básicas que você pode realizar com listas em Python:

1. Acessar elementos da lista por meio de índices:

```python
minha_lista = [10, 20, 30, 40, 50]
primeiro_elemento = minha_lista[0]  # Retorna 10
segundo_elemento = minha_lista[1]   # Retorna 20
```

2. Modificar elementos da lista:

```python
minha_lista[2] = 35  # Agora a lista será [10, 20, 35, 40, 50]
```

3. Adicionar elementos no final da lista:

```python
minha_lista.append(60)  # Agora a lista será [10, 20, 35, 40, 50, 60]
```

4. Inserir elementos em uma posição específica:

```python
minha_lista.insert(2, 25)  # Agora a lista será [10, 20, 25, 35, 40, 50, 60]
```

5. Remover elementos da lista:

```python
minha_lista.remove(35)  # Agora a lista será [10, 20, 25, 40, 50, 60]
```

6. Verificar a quantidade de elementos na lista:

```python
tamanho_da_lista = len(minha_lista)  # Retorna 6
```

7. Verificar se um elemento está presente na lista:

```python
if 40 in minha_lista:
    print("40 está na lista.")
```

8. Percorrer a lista usando "loops":

```python
for elemento in minha_lista:
    print(elemento)
```

As listas em Python são extremamente úteis para armazenar coleções de dados, como listas de tarefas, resultados de uma pesquisa, informações de clientes, entre outras aplicações. A capacidade de serem modificadas, combinadas com outras estruturas de dados e operações torna-as uma escolha poderosa para diversas tarefas de programação.

5.1.1. Curiosidade do dia:

Uma curiosidade interessante sobre listas em Python é que elas suportam indexação negativa. Isso significa que você pode acessar os elementos da lista a partir do final, utilizando índices negativos.

A indexação negativa facilita a obtenção dos elementos no final da lista, tornando o código mais conciso e legível em alguns casos.

Além disso, essa característica pode ser muito útil quando você trabalha com listas cujo tamanho pode variar ou quando precisa acessar os elementos em ordem inversa, por exemplo, percorrendo a lista de trás para frente. A indexação negativa é apenas mais um recurso que torna as listas em Python uma estrutura de dados poderosa e versátil.

5.1.2. ::Dicas::

Python também permite acessar os elementos da lista de trás para frente, usando índices negativos.

O último elemento está no índice -1

O penúltimo no -2

O antepenúltimo no -3

E assim por diante...

Essa técnica se chama indexação negativa.

O que isso significa na prática?

Significa que você não precisa saber o tamanho da lista para acessar os últimos itens.

Você pode fazer isso de forma direta e prática.

Por exemplo:

Quer o último item? Use lista[-1]

Quer os dois últimos? Use lista[-2:]

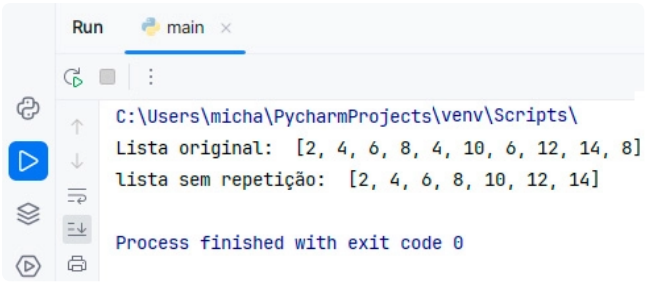
Quer inverter a lista? Use lista[::-1]

5.2. Exercícios opcionais

Exercício 1 - Remover elementos iguais:

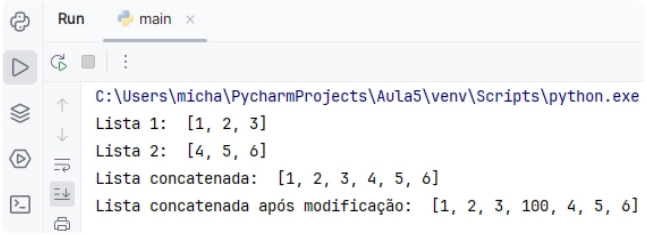
1. Crie uma lista chamada "numeros" contendo alguns números repetidos.
2. Exiba a lista original para verificar quais números estão presentes.
3. Crie uma nova lista vazia chamada "numeros_sem_repeticao".
4. Percorra cada elemento da lista numeros usando um "loop for".
5. Verifique se o elemento atual não está presente na lista "numeros_sem_repeticao".
6. Se o elemento não estiver na lista, adicione-o usando o método "append".

7. Exiba a lista "numeros_sem_repeticao" para mostrar os números únicos.
8. Confira o resultado abaixo.



Exercício 2 - Concatenar listas:

1. Crie duas listas: "lista1" e "lista2", cada uma contendo alguns elementos.
2. Exiba as duas listas para verificar os elementos que elas possuem.
3. Use o operador de adição "+" para concatenar as duas listas em uma nova lista chamada "lista_concatenada".
4. Exiba a lista "lista_concatenada" para mostrar os elementos das duas listas combinadas em uma única lista.
5. Modifique uma das listas originais, adicionando ou removendo elementos.
6. Exiba novamente a lista "lista_concatenada" para confirmar que ela não foi afetada pelas modificações na lista original.
7. Confira o resultado abaixo.



Anotações

6.1. Strings

Em Python, uma string é uma sequência de caracteres que representa texto. Uma string pode conter letras, números, espaços em branco, pontuação, símbolos e até mesmo caracteres especiais. As strings são uma das estruturas de dados fundamentais em Python e são usadas para manipular e armazenar informações textuais.

Segue um exemplo de string em Python:

```python
"Python é uma linguagem de programação poderosa."
```

Para criar uma string em Python, você pode usar aspas simples ' ' ou aspas duplas " ". Veja alguns exemplos:

```python
mensagem1 = 'Olá, mundo!'
mensagem2 = "Python é uma linguagem de programação poderosa."
```

As strings são imutáveis em Python, o que significa que, depois de criada, não é possível alterar seus caracteres individualmente. No entanto, você pode criar novas strings a partir de operações em strings existentes.

6.2. Operações com Strings

Algumas operações e funcionalidades comuns relacionadas a strings em Python incluem:

1. Concatenação de strings: Você pode combinar duas ou mais strings usando o operador de adição +.

```python
nome = "João"
sobrenome = "Silva"
nome_completo = nome + " " + sobrenome  # Resultado: "João Silva"
```

2. Indexação de caracteres: É possível acessar caracteres individuais de uma string usando índices.

```python
mensagem = "Python"
primeiro_caractere = mensagem[0]  # Resultado: "P"
segundo_caractere = mensagem[1]   # Resultado: "y"
```

3. Fatias de strings (slicing): Você pode extrair uma parte específica de uma string usando a notação de fatiamento.

```python
mensagem = "Python é ótimo"
parte_da_mensagem = mensagem[7:10]  # Resultado: "é"
```

4. Funções de manipulação de strings: Python oferece várias funções embutidas para manipular strings, como len(), upper(), lower(), strip(), split(), entre outras.

```python
texto = "   Olá, mundo!   "
tamanho_do_texto = len(texto)          # Resultado: 19
texto_em_maiusculas = texto.upper()    # Resultado: "   OLÁ, MUNDO!   "
texto_sem_espacos = texto.strip()      # Resultado: "Olá, mundo!"
palavras = texto.split(",")            # Resultado: ['   Olá', ' mundo!   ']
```

As strings em Python são amplamente utilizadas em programação para trabalhar com texto, manipular dados, formatar saídas de informações e muito mais. Sua versatilidade e recursos facilitam a escrita de códigos para uma variedade de aplicações, desde simples scripts até aplicações mais complexas.

6.3. Utilidade das strings

As strings em Python, ou em qualquer outra linguagem de programação, têm uma ampla gama de aplicações e são essenciais para trabalhar com texto e informações baseadas em caracteres. Algumas das principais utilizações das strings incluem:

1. Armazenamento e manipulação de texto: As strings são usadas para armazenar e manipular informações em formato de texto. Elas permitem que os programadores trabalhem com palavras, frases, sentenças e qualquer outra forma de dados baseada em caracteres.
2. Entrada e saída de dados: As strings são frequentemente usadas para interagir com os usuários, permitindo que eles insiram informações por meio do teclado e exibindo resultados em formato de texto na tela.
3. Processamento de arquivos: Ao ler e escrever arquivos de texto, as strings são usadas para representar o conteúdo lido de um arquivo ou para escrever informações em um arquivo de texto.Processamento de arquivos:
4. Manipulação de dados em bancos de dados: Muitas vezes, os dados armazenados em bancos de dados são representados em formato de texto, e as strings são usadas para armazenar e consultar informações de forma eficiente.
5. Manipulação de URL e dados da web: Ao trabalhar com solicitações da web e URLs, as strings são usadas para representar endereços da web, parâmetros de consulta e outras informações relacionadas à web.
6. Formatação de saída: As strings são frequentemente usadas para formatar a saída em programas, permitindo que os resultados sejam apresentados de maneira adequada e legível para os usuários.
7. Manipulação de linguagem natural: Em tarefas de processamento de linguagem natural, as strings são usadas para representar texto de documentos, mensagens, análise de sentimentos, tradução, entre outras aplicações.
8. Criação de mensagens e logs: Strings são usadas para criar mensagens informativas ou de erro para os usuários ou para registrar eventos e informações em logs durante a execução do programa.

Em resumo, as strings são fundamentais para trabalhar com texto e informações baseadas em caracteres em Python e em muitas outras linguagens de programação. Elas são amplamente usadas em praticamente todos os aspectos do desenvolvimento de software, desde aplicações simples até projetos mais complexos. A capacidade de manipular e processar strings é uma das habilidades mais importantes para um programador, independentemente da área de atuação.

6.3.1. Curiosidades:

Uma curiosidade interessante sobre strings em Python é que você pode criar strings multilinhas usando três pares de aspas (simples ou duplas).

No entanto, há situações em que você pode querer criar uma string que abranja várias linhas, como para escrever um parágrafo longo ou um bloco de texto. Nesses casos, você pode usar aspas triplas para criar uma string multilinha sem precisar inserir sequências de escape (\n) para indicar novas linhas.

Por exemplo:

texto_multilinha = """Esta é uma string

que se estende por várias linhas

sem a necessidade de usar \n para quebra de linha.

Você pode escrever textos mais longos

de forma mais legível e organizada."""

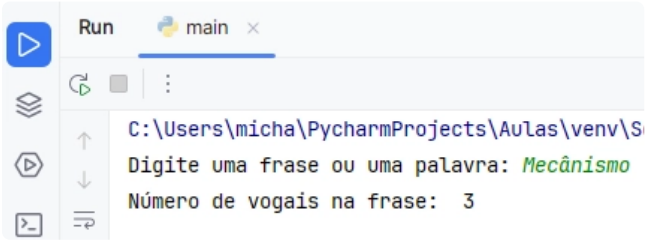
Essa funcionalidade é especialmente útil quando você deseja criar strings com várias linhas de texto, como para documentação ou para armazenar informações de texto formatado em seu código.

Vale ressaltar que, ao usar aspas triplas para criar uma string multilinha, as quebras de linha serão mantidas na string, ou seja, a string manterá a formatação exata que você escreveu.

6.4. Exercícios opcionais

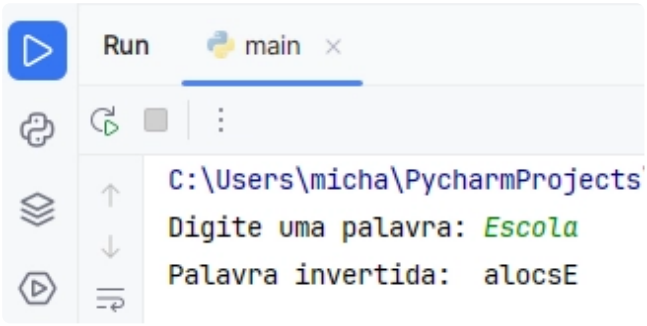
Exercício 1 - Contar Vogais:

1. Peça ao usuário para digitar uma frase ou uma palavra.
2. Crie uma variável chamada vogais e atribua uma string contendo todas as vogais em minúsculas, por exemplo: 'aeiou'.
3. Inicialize uma variável chamada contador com o valor zero para contar a quantidade de vogais na frase.
4. Percorra cada caractere da frase usando um loop for.
5. Verifique se o caractere é uma vogal (ou seja, se está presente na string vogais).
6. Se o caractere for uma vogal, incremente o valor do contador.
7. Exiba o valor do contador, que representará o número total de vogais na frase.
8. Confira o resultado abaixo.



Exercício 2 - Palavra Invertida:

1. Peça ao usuário para digitar uma palavra.
2. Use a indexação de strings para inverter a palavra digitada pelo usuário.
3. Crie uma nova variável chamada palavra_invertida e atribua a palavra invertida a essa variável.
4. Exiba a palavra invertida na tela para o usuário ver o resultado.
5. Confira o resultado abaixo.



Anotações

7.1. Funções

Em Python, uma função é um bloco de código nomeado que realiza uma tarefa específica ou executa um conjunto de operações. Ela é uma maneira de encapsular um conjunto de instruções para que possam ser reutilizadas facilmente em diferentes partes do programa. As funções são um dos conceitos mais fundamentais da programação e são usadas para organizar o código, torná-lo mais legível, modular e facilitar a manutenção.

As funções em Python têm a seguinte sintaxe geral:

```python
def nome_da_funcao(parametros):
    # Corpo da função - conjunto de instruções a serem executadas
    # Pode haver um retorno de valor, usando a palavra-chave 'return', se necessário
```

Estrutura das funções

A estrutura de uma função em Python é composta por alguns elementos essenciais que definem a função e sua funcionalidade. A seguir, apresento a estrutura básica de uma função em Python:

1. Definição da função: A definição da função começa com a palavra-chave def, seguida pelo nome da função e, opcionalmente, os parâmetros entre parênteses.
2. Parâmetros: São valores que a função pode receber como entrada. Eles são opcionais, mas, se a função espera receber algum valor, você deve declará-los entre os parênteses. Os parâmetros são usados como variáveis dentro da função para manipular os dados recebidos.
3. Corpo da função: O corpo da função é o bloco de código que contém as instruções a serem executadas quando a função é chamada. O corpo da função é definido pela indentação, ou seja, um recuo de espaços em relação à margem esquerda do código.
4. Retorno (opcional): A função pode ter uma instrução return, que é usada para retornar um valor da função. Se a função não tiver um return, ela ainda pode ser usada para executar ações, mas não retornará explicitamente nenhum valor.

Aqui está a estrutura completa de uma função em Python:

```python
def nome_da_funcao(parametros):
    # Corpo da função
    # Pode haver uma instrução return, se necessário
```

Exemplo de uma função que recebe dois números e retorna a soma deles:

```python
def somar(a, b):
    return a + b
```

Nesse exemplo, temos:

- nome_da_funcao: somar
- parametros: a e b
- corpo da função: return a + b

Essa função "somar" pode ser usada em qualquer lugar do código, chamando-a pelo nome e passando os valores necessários:

```python
resultado = somar(5, 3)
print(resultado)  # Saída: 8
```

A estrutura de uma função é fundamental para a organização e reutilização de código em Python, tornando o desenvolvimento mais eficiente e legível.

7.2. Tipos de funções

Python possui uma ampla biblioteca padrão que oferece diversas funções prontas para uso. Essas funções são agrupadas em módulos, e cada módulo é projetado para resolver um conjunto específico de problemas ou fornecer um conjunto de funcionalidades relacionadas.

Além das funções padrão, é possível importar bibliotecas externas que fornecem ainda mais funcionalidades para atender a diversas necessidades específicas.

A seguir, menciono algumas das funções disponíveis em Python a partir da biblioteca padrão, mas é importante ressaltar que a lista é extensa e não é possível listar todas as funções em um único espaço:

1. Funções matemáticas: abs(), round(), pow(), min(), max(), sum(), sqrt(), ceil(), floor(), etc.
2. Funções de conversão de tipos de dados: int(), float(), str(), bool(), list(), tuple(), set(), dict(), etc.
3. Funções de manipulação de strings: len(), upper(), lower(), split(), strip(), join(), etc.
4. Funções para trabalhar com listas, tuplas e conjuntos: append(), pop(), count(), index(), sort(), reverse(), tuple(), set(), etc.
5. Funções para trabalhar com dicionários: keys(), values(), items(), get(), update(), pop(), clear(), etc.
6. Funções para interação com o sistema: print(), input(), open(), write(), read(), os(), sys(), time(), datetime(), etc.
7. Funções para manipulação de arquivos e diretórios: open(), read(), write(), close(), os.path(), os.listdir(), os.makedirs(), etc.
8. Funções para tratamento de exceções: try, except, raise, assert, etc.
9. Funções relacionadas a expressões regulares: re.match(), re.search(), re.findall(), re.sub(), etc.
10. Funções para gerar números aleatórios: random.random(), random.randint(), random.choice(), etc.

Essas são apenas algumas das muitas funções disponíveis na biblioteca padrão do Python. Além disso, como mencionado anteriormente, você pode importar módulos externos para ter acesso a mais funções especializadas em diversas áreas, como ciência de dados, aprendizado de máquina, processamento de imagens, entre outras.

Curiosidade do dia:

Uma curiosidade legal sobre as funções em Python, é que através da instrução return podemos retornar múltiplos valores em forma de uma tupla. Normalmente, em muitas linguagens de programação, uma função pode retornar apenas um único valor. No entanto, em Python, é possível retornar vários valores agrupados em uma tupla, mesmo que a função não especifique explicitamente que irá retornar uma tupla.

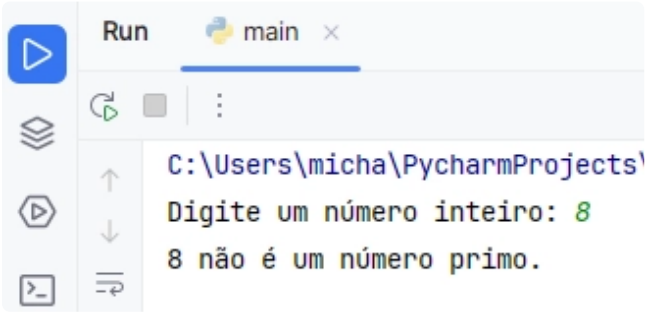
Essa característica é muito útil quando você deseja retornar vários resultados relacionados de uma função sem a necessidade de criar uma estrutura de dados complexa. A capacidade de retornar múltiplos valores em uma única chamada de função torna Python uma linguagem ainda mais versátil e eficiente para resolver problemas diversos. Essa característica é muito útil quando você deseja retornar vários resultados relacionados de uma função sem a

necessidade de criar uma estrutura de dados complexa. A capacidade de retornar múltiplos valores em uma única chamada de função torna Python uma linguagem ainda mais versátil e eficiente para resolver problemas diversos.

7.3. Exercícios opcionais

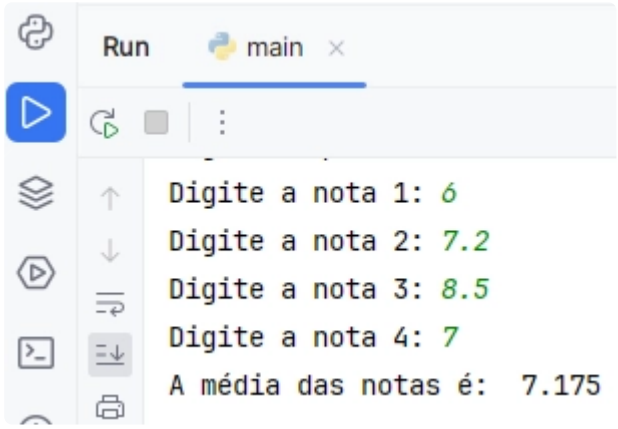
Exercício 1 - Verificar Número Primo:

1. Crie uma função chamada eh_primo que recebe um número inteiro como parâmetro.
2. Dentro da função, verifique se o número é maior que 1, pois números primos são maiores que 1.
3. Use um loop for para iterar de 2 até a raiz quadrada do número (use a função math.sqrt() para calcular a raiz quadrada).
4. Verifique se o número é divisível por qualquer um dos valores do loop. Se for, retorne False porque não é um número primo.
5. Se o loop não encontrar divisores, retorne True, pois o número é primo.
6. Fora da função, solicite ao usuário que insira um número inteiro.
7. Chame a função eh_primo com o número inserido pelo usuário e exiba uma mensagem indicando se o número é primo ou não.
8. Confira o resultado abaixo:
9.



Exercício 2 - Média de Notas:

1. Crie uma função chamada calcular_media que recebe uma lista de notas (números reais) como parâmetro.
2. Dentro da função, calcule a média das notas somando todos os valores da lista e dividindo pelo número total de notas.
3. Retorne a média calculada.
4. Fora da função, peça ao usuário que insira a quantidade de notas que deseja calcular.
5. Em seguida, solicite que o usuário digite cada nota individualmente e armazene-as em uma lista.
6. Chame a função calcular_media com a lista de notas inseridas e exiba a média calculada.
7. Confira o resultado abaixo:



Anotações

8.1. Erros em Python

Em Python, erros são ocorrências indesejadas que acontecem durante a execução de um programa. Quando o interpretador de Python encontra um erro, ele interrompe a execução e exibe uma mensagem de erro, conhecida como "traceback" ou "rastreamento", indicando qual o tipo de erro ocorreu e onde ele foi encontrado no código.

8.2. Tipos de erros

Os erros em Python podem ser divididos em três categorias principais:

Erros de Sintaxe (Syntax Errors): Esses erros ocorrem quando há um erro de digitação ou uma estrutura incorreta no código, violando as regras de sintaxe da linguagem. Python exibirá uma mensagem de erro apontando para a linha específica onde o erro foi detectado.

Exemplo de erro de sintaxe:

python
```
print("Olá, mundo!)  # Erro de sintaxe: falta uma aspa no final da string
```

Exceções (Exceptions): As exceções ocorrem durante a execução do código, quando algo inesperado acontece e impede a conclusão normal da tarefa.

As exceções são frequentemente causadas por erros lógicos, como divisão por zero, tentativa de acessar um índice inválido em uma lista, entre outros.

Exemplo de exceção:

python
```
lista = [1, 2, 3]
print(lista[5])  # Erro: tentativa de acessar um índice que não existe na lista
```

Erros Lógicos (Logical Errors): Esses erros ocorrem quando o programa não produz o resultado esperado, mas não geram mensagens de erro. Os erros lógicos geralmente são causados por falhas no raciocínio ou na lógica do código, resultando em um comportamento incorreto.

Exemplo de erro lógico:

python
```
def calcular_area_retangulo(largura, altura):
    return largura * altura  # Deveria ser altura * largura

area = calcular_area_retangulo(5, 10)
print(area)  # Saída: 50 (incorreta) em vez de 50 (correta)
```

É importante lembrar que os erros são parte natural do processo de desenvolvimento, e programadores enfrentam erros o tempo todo. Eles são uma maneira de identificar problemas no código e corrigi-los para torná-lo mais robusto e funcional. Ao encontrar erros, o Python fornece informações valiosas no rastreamento para ajudar a entender a causa do problema, facilitando a depuração e resolução dos mesmos.

Tratando erros em Python

Em Python, você pode tratar erros usando blocos "try", "except" para capturar exceções e lidar com elas de forma controlada. O tratamento de erros permite que você previna a interrupção abrupta do programa e forneça um tratamento adequado para erros específicos, o que ajuda a tornar o código mais robusto e a evitar que o programa pare de funcionar inesperadamente.

A estrutura básica para tratar erros em Python é a seguinte:

python

```
try:
    # Código que pode gerar um erro
except TipoDoErro:
    # Código para lidar com o erro do tipo especificado
```

Explicação passo a passo:

1. Coloque o código que pode gerar um erro dentro do bloco try.
2. Se ocorrer um erro do tipo especificado após a palavra-chave except, o código dentro do bloco except será executado.
3. O bloco except é usado para lidar com exceções específicas. Você pode especificar o tipo de erro após a palavra-chave except, como ZeroDivisionError, ValueError, IndexError, etc. Ou simplesmente usar except sem nenhum tipo para tratar qualquer exceção.
4. Se ocorrer um erro que não corresponda a nenhum tipo especificado no bloco except, o programa exibirá um traceback como de costume, indicando que não foi tratado.

Exemplo de tratamento de erro:

python

```
try:
    x = int(input("Digite um número: "))
    y = int(input("Digite outro número: "))
    resultado = x / y
except ValueError:
    print("Digite apenas números inteiros.")
except ZeroDivisionError:
    print("Divisão por zero não é permitida.")
else:
    print("O resultado da divisão é:", resultado)
finally:
    print("Fim do programa.")
```

Neste exemplo, o código dentro do bloco try tentará obter dois números inteiros do usuário e realizar uma divisão. Se ocorrer um erro do tipo ValueError (por exemplo, se o usuário digitar um valor não inteiro) ou um erro ZeroDivisionError (se o usuário digitar 0 como segundo número), o bloco except correspondente será executado para lidar com o erro específico. Se nenhuma exceção for gerada, o bloco else será executado. O bloco finally será sempre executado, independentemente de ocorrer ou não uma exceção. O tratamento de erros é uma prática importante na programação, pois ajuda a garantir que o programa continue funcionando mesmo quando situações inesperadas ocorrem.

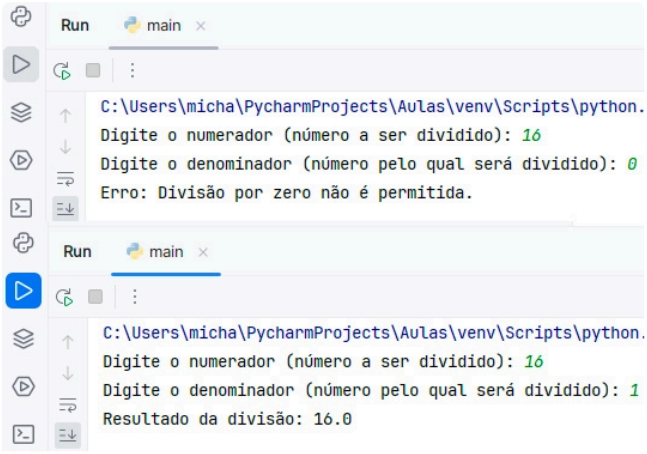
Curiosidade do dia:

Uma curiosidade legal sobre erros em Python é que você pode criar suas próprias exceções personalizadas, chamadas de "exceções personalizadas" ou "exceções customizadas". Isso permite que você crie tipos de erros específicos para o seu programa, tornando a depuração mais fácil e fornecendo mensagens de erro mais claras e informativas para os usuários.

8.3. Exercícios opcionais

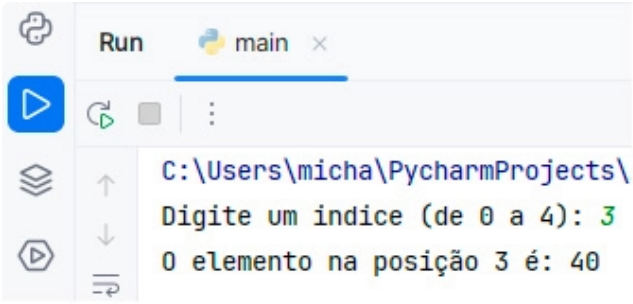
Exercício 1 - Divisão de Números:

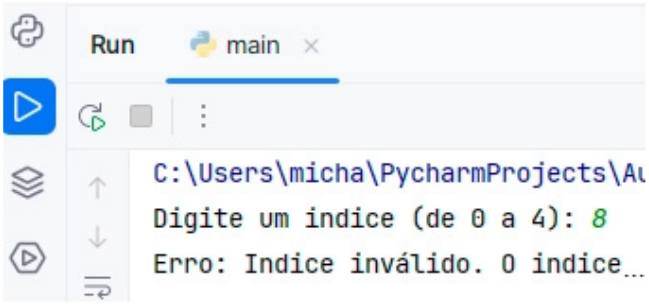
1. Peça ao usuário que digite dois números inteiros.
2. Use um bloco try/except para tratar possíveis erros durante a divisão dos números.
3. Se a divisão for possível, exiba o resultado da divisão.
4. Se ocorrer um erro de divisão por zero, capture o erro e exiba uma mensagem indicando que a divisão não é possível.
5. Confira o resultado abaixo.



Exercício 2 - Índice em Lista:

1. Crie uma lista de números.
2. Peça ao usuário que digite um número inteiro como índice para acessar um elemento da lista.
3. Use um bloco try/except para tratar possíveis erros ao tentar acessar o elemento da lista com o índice fornecido.
4. Se o índice estiver dentro dos limites válidos da lista, exiba o elemento correspondente.
5. Se ocorrer um erro de índice inválido (por exemplo, índice maior que o tamanho da lista), capture o erro e exiba uma mensagem indicando que o índice é inválido.
6. Confira o resultado.





Anotações

9.1. Módulos

Em Python, um módulo é um arquivo contendo definições de funções, classes e variáveis que podem ser reutilizadas em outros programas. Ele é uma unidade de organização de código que permite que você divida o seu código em partes menores, tornando-o mais modular e fácil de gerenciar.

9.2. Características dos módulos

Os módulos em Python têm as seguintes características:

1. Reutilização de código: Você pode escrever funções, classes e outras definições em um módulo e, em seguida, importar esse módulo em outros programas para reutilizar esse código. Isso evita a repetição de código e promove a organização.
2. Encapsulamento: As definições no módulo são encapsuladas, o que significa que o código fora do módulo não pode acessar essas definições diretamente, a menos que sejam explicitamente importadas.
3. Organização: Módulos ajudam a organizar o código em unidades lógicas. Isso facilita a manutenção, pois alterações em um módulo não afetarão diretamente outros módulos.
4. Biblioteca Padrão: Python possui uma biblioteca padrão extensa, que é um conjunto de módulos que acompanham a instalação do Python. Esses módulos fornecem funcionalidades para tarefas comuns, como manipulação de strings, operações matemáticas, acesso a arquivos, etc.
5. Módulos Externos: Além da biblioteca padrão, você também pode instalar módulos externos através do gerenciador de pacotes do Python (como o pip). Esses módulos adicionais fornecem recursos adicionais para tarefas específicas, como ciência de dados, desenvolvimento web, aprendizado de máquina, entre outros.

9.3. Utilização de módulos

Para usar um módulo em Python, você precisa importá-lo no seu programa. Isso é feito usando a palavra-chave "import", seguida do nome do módulo. Por exemplo, para usar o módulo de matemática padrão em Python, você pode fazer:

```python
import math

# Agora, você pode usar as funções e
constantes do módulo 'math', como
math.sqrt(), math.sin(), math.pi, etc.
```

Os módulos em Python são uma maneira poderosa de organizar e reutilizar código, permitindo que você aproveite a funcionalidade existente e crie programas mais eficientes e organizados.

9.4. Tipos de módulos

O Python possui uma ampla variedade de módulos que abrangem diversos domínios e funcionalidades. Abaixo estão alguns exemplos de módulos populares e úteis em Python:

Módulo math: Fornecer funções matemáticas comuns, como trigonometria, logaritmo, exponencial, entre outros.

```python
import math

print(math.sqrt(25))   # Saída: 5.0
print(math.sin(math.pi/2))   # Saída: 1.0
print(math.log(10))   # Saída:
2.302585092994046
```

Módulo random: Oferecer recursos para gerar números aleatórios.

```python
import random

print(random.random())   # Saída: Número
aleatório entre 0 e 1
print(random.randint(1, 10))   # Saída:
Número aleatório inteiro entre 1 e 10
```

Módulo datetime: Permitir trabalhar com datas e horários.

```python
import datetime

data_atual = datetime.date.today()
print(data_atual)   # Saída: 2023-07-31

hora_atual = datetime.datetime.now()
print(hora_atual)   # Saída: 2023-07-31
14:30:45.123456
```

Módulo os: Fornecer funcionalidades para interação com o sistema operacional.

```python
import os

print(os.getcwd())   # Saída: Caminho do
diretório de trabalho atual
print(os.listdir())   # Saída: Lista de
arquivos e pastas no diretório atual
```

Módulo json: Oferecer suporte para codificação e decodificação de dados JSON.

```python
import json

dados = {'nome': 'João', 'idade': 30}
json_string = json.dumps(dados)
print(json_string)   # Saída: '{"nome":
"João", "idade": 30}'

dados_decodificados =
json.loads(json_string)
print(dados_decodificados)   # Saída:
{'nome': 'João', 'idade': 30}
```

Módulo requests: Possibilitar a realização de requisições HTTP para acessar APIs e páginas web.

```python
import requests

resposta =
requests.get('https://api.exemplo.com/dados'
)
dados = resposta.json()
print(dados)
```

Esses são apenas alguns exemplos dos muitos módulos disponíveis em Python. Além desses, existem módulos específicos para ciência de dados (como numpy, pandas), desenvolvimento web (como flask, django), aprendizado de máquina (como scikit-learn, tensorflow), entre muitos outros. Cada módulo tem suas próprias funcionalidades e pode ser útil em diferentes contextos de desenvolvimento.

9.5. Pacotes

Em Python, um pacote é uma forma de organizar e estruturar módulos relacionados em diretórios para facilitar a criação e manutenção de projetos maiores e mais complexos. Os pacotes são uma maneira de organizar e modularizar o código em uma hierarquia, permitindo que você agrupe módulos relacionados em um único diretório.

Um pacote Python é simplesmente um diretório que contém um arquivo especial chamado __init__.py. Esse arquivo é executado

quando o pacote é importado e pode conter código de inicialização do pacote. O pacote pode conter outros módulos Python, bem como outros subpacotes, que são subdiretórios contendo também arquivos __init__.py.

9.6. Hierarquia dos Pacotes

A hierarquia de pacotes e subpacotes é indicada pelo uso de pontos (.) no nome dos pacotes. Por exemplo, se você tiver um pacote chamado meu_pacote que contém um subpacote chamado subpacote, a estrutura de diretórios será a seguinte:

python

```
meu_pacote/
    __init__.py
    modulo1.py
    modulo2.py
    subpacote/
        __init__.py
        modulo3.py
        modulo4.py
```

9.7. Acessando Pacotes

Para acessar os módulos dentro de um pacote ou subpacote, você pode usar a sintaxe de importação com os pontos (.). Por exemplo:

python

```
# Importando um módulo dentro do pacote
from meu_pacote import modulo1

# Importando um módulo dentro do subpacote
from meu_pacote.subpacote import modulo3
```

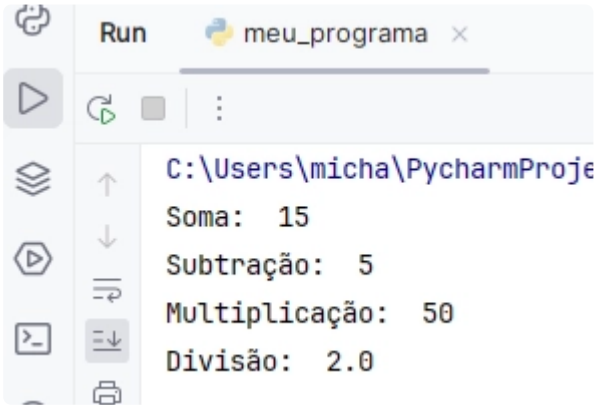
Quando você importa um pacote, o arquivo __init__.py do pacote é executado, permitindo que você faça qualquer configuração ou inicialização necessária para o pacote.

Pacotes são uma maneira eficaz de organizar projetos maiores em partes menores e relacionadas, permitindo que você trabalhe com código modular e reutilizável. Eles são amplamente utilizados em projetos Python mais complexos e em bibliotecas que você deseja compartilhar com outras pessoas.

9.8. Exercícios opcionais

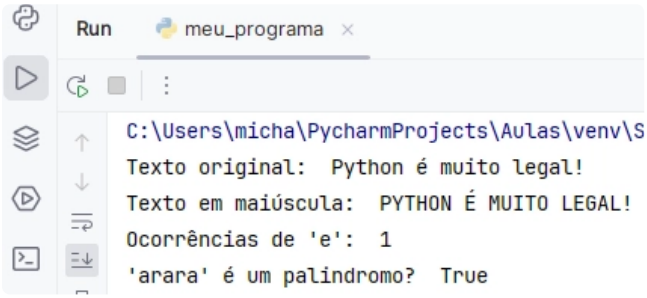
Exercício 1 - Criando um Módulo de Cálculos Matemáticos:

1. Crie um arquivo chamado calculadora.py.
2. Defina funções para as operações matemáticas básicas (soma, subtração, multiplicação e divisão) no arquivo calculadora.py.
3. Crie um novo arquivo Python chamado meu_programa.py.
4. Importe o módulo calculadora criado no passo 2 no arquivo meu_programa.py.
5. Utilize as funções do módulo calculadora para realizar algumas operações matemáticas.
6. Confira o resultado abaixo:



Exercício 2 - Criando um Módulo de Manipulação de Strings:

1. Crie um arquivo chamado string_utils.py.
2. Defina funções para manipulação de strings, como converter uma string para maiúsculas, contar a ocorrência de um caractere específico ou verificar se uma string é um palíndromo.
3. Crie um novo arquivo Python chamado meu_programa.py.
4. Importe o módulo string_utils criado no passo 2 no arquivo meu_programa.py.
5. Utilize as funções do módulo string_utils para manipular algumas strings.
6. Confira o resultado abaixo.



Anotações

10.1. Objetos

Em Python, objetos são entidades fundamentais que representam dados e possuem comportamentos associados a eles. Python é uma linguagem de programação orientada a objetos (POO), o que significa que a maioria das estruturas de dados e funcionalidades são implementadas como objetos ou são baseadas em objetos.

Um objeto em Python é uma instância de uma classe, que é uma espécie de "modelo" que define as características e comportamentos que o objeto terá. Uma classe serve como um plano ou projeto para criar objetos com atributos e métodos específicos.

10.2. Conceito de objetos

Aqui estão alguns conceitos importantes relacionados a objetos em Python:

1. Classe: Uma classe é uma estrutura que define o comportamento e as características de um objeto. Ela funciona como um molde para criar objetos com atributos e métodos específicos.
2. Objeto: Um objeto é uma instância de uma classe. Quando uma classe é instanciada, ela cria um objeto com os atributos e métodos definidos pela classe.
3. Atributo: Um atributo é uma variável que está associada a um objeto e armazena um valor específico. Cada objeto de uma classe pode ter seus próprios valores de atributos.
4. Método: Um método é uma função definida dentro de uma classe que define o comportamento dos objetos dessa classe. Os métodos permitem que os objetos executem ações específicas.
5. Herança: A herança é um conceito importante na programação orientada a objetos que permite criar uma nova classe (chamada classe derivada ou subclasse) a partir de uma classe existente (chamada classe base ou superclasse). A classe derivada herda os atributos e métodos da classe base e pode adicionar ou modificar comportamentos.
6. Encapsulamento: O encapsulamento é o princípio de ocultar detalhes internos de um objeto e expor apenas uma interface pública. Em Python, isso é geralmente alcançado através do uso de métodos para acessar e modificar os atributos do objeto.
7. Polimorfismo: O polimorfismo é a capacidade de diferentes classes compartilharem uma mesma interface (métodos com o mesmo nome), mas cada classe pode implementar esse método de maneira diferente. Isso permite que objetos de diferentes classes sejam tratados de forma uniforme quando eles têm um comportamento comum.

Em resumo, objetos em Python são entidades que combinam dados e comportamentos relacionados em uma única unidade. Eles permitem que você modele e manipule dados complexos de maneira mais estruturada e organizada, tornando a programação mais eficiente e elegante.

10.3. Exemplo de Objetos

Objeto Pessoa:

python

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def apresentar(self):
        print(f"Olá, meu nome é {self.nome}
e tenho {self.idade} anos.")

# Criando objetos da classe Pessoa
pessoa1 = Pessoa("João", 30)
pessoa2 = Pessoa("Maria", 25)

# Chamando o método apresentar para cada
objeto
pessoa1.apresentar()
pessoa2.apresentar()
```

Objeto Carro:

python

```
class Carro:
    def __init__(self, marca, modelo, ano):
        self.marca = marca
        self.modelo = modelo
        self.ano = ano

    def info(self):
        print(f"Marca: {self.marca}, Modelo:
{self.modelo}, Ano: {self.ano}")

# Criando objetos da classe Carro
carro1 = Carro("Toyota", "Corolla", 2020)
carro2 = Carro("Honda", "Civic", 2019)

# Chamando o método info para cada objeto
carro1.info()
carro2.info()
```

Objeto Círculo:

python

```
import math

class Circulo:
    def __init__(self, raio):
        self.raio = raio

    def calcular_area(self):
        return math.pi * self.raio ** 2

    def calcular_perimetro(self):
        return 2 * math.pi * self.raio

# Criando objetos da classe Circulo
circulo1 = Circulo(5)
circulo2 = Circulo(7)

# Chamando os métodos para cada objeto
print("Área do círculo1:",
circulo1.calcular_area())
print("Perímetro do círculo1:",
circulo1.calcular_perimetro())

print("Área do círculo2:",
circulo2.calcular_area())
print("Perímetro do círculo2:",
circulo2.calcular_perimetro())
```

Esses são exemplos simples de objetos em Python. Em projetos mais complexos, você pode criar classes com mais atributos e métodos para modelar entidades e funcionalidades mais avançadas. Os objetos permitem que você mantenha o código organizado, reutilize funcionalidades e aplique conceitos importantes da programação orientada a objetos, como encapsulamento e herança.

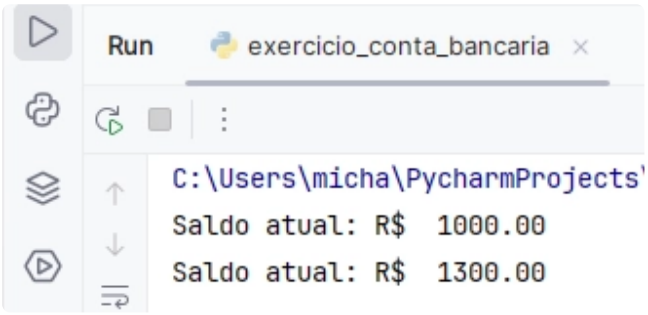
Curiosidade do dia:

Uma curiosidade interessante sobre objetos em Python é que, na realidade, tudo em Python é um objeto. Isso mesmo, absolutamente tudo! Até mesmo os tipos de dados básicos, como números, strings e listas, são objetos em Python. Isso acontece porque Python é uma linguagem de programação altamente orientada a objetos. Quando você cria uma variável, ela é realmente uma referência para um objeto na memória, não apenas um valor primitivo. Isso significa que até mesmo números e operações matemáticas são realizados através de objetos.

10.4. Exercícios opcionais

Exercício 1 - Simulação de uma conta Bancária:

1. Crie um novo arquivo Python em seu editor de código favorito e nomeie-o como "exercicio_conta_bancaria.py".
2. Escreva a seguinte linha de código para criar a classe ContaBancaria, que possui atributos para o titular e saldo inicial: class ContaBancaria:class ContaBancaria:
3. Passe para a próxima linha e adicione o método __init__ à classe ContaBancaria, que receberá o titular e o saldo inicial como parâmetros e inicializará os atributos da classe:
4. Passe para a próxima linha e adicione o método depositar à classe ContaBancaria, que receberá um valor como parâmetro e adicionará esse valor ao saldo:
5. Passe para a próxima linha e adicione o método sacar à classe ContaBancaria, que receberá um valor como parâmetro e verificará se há saldo suficiente para o saque. Se houver, o valor será subtraído do saldo; caso contrário, uma mensagem de saldo insuficiente será exibida:
6. Passe para a próxima linha e adicione o método consultar_saldo à classe ContaBancaria, que imprimirá o saldo atual da conta:
7. Passe para a próxima linha e crie um objeto da classe ContaBancaria, fornecendo um titular e saldo inicial:
8. Passe para a próxima linha e realize operações com a conta bancária, como consultar o saldo, depositar e sacar:
9. Salve o arquivo "exercicio_conta_bancaria.py" e execute-o. Observe a saída que exibe o saldo antes e após as operações de depósito e saque.
10. Confira o resultado abaixo:

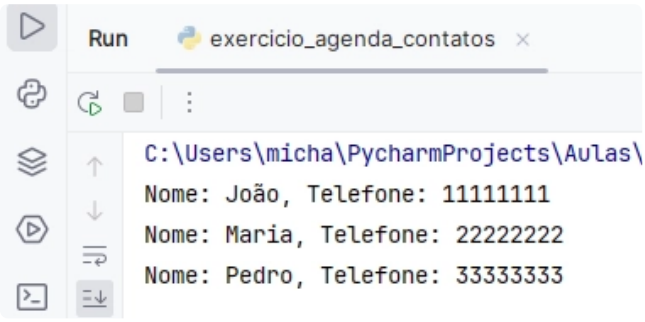


Exercício 2: Simulação de uma Agenda de Contatos

1. Crie um novo arquivo Python em seu editor de código favorito e nomeie-o como "exercicio_agenda_contatos.py".
2. Escreva a seguinte linha de código para criar a classe Contato, que possui atributos para o nome e telefone do contato: class Contato:class Contato:
3. Passe para a próxima linha e adicione o método __init__ à classe Contato, que receberá o nome e o telefone do contato como parâmetros e inicializará os atributos da classe:
4. Passe para a próxima linha e adicione o método exibir_info à classe Contato, que imprimirá o nome e o telefone do contato:
5. Passe para a próxima linha e crie uma lista vazia para armazenar os contatos:
6. Passe para a próxima linha e crie objetos da

classe Contato e adicione-os à lista de contatos:

7. Passe para a próxima linha e exiba as informações dos contatos armazenados na lista:
8. Salve o arquivo "exercicio_agenda_contatos.py" e execute-o. Verifique a saída que exibe as informações dos contatos adicionados à lista.
9. Confira o resultado abaixo:



Anotações

11.1. Dicionários

Em Python, um dicionário é uma estrutura de dados que permite armazenar elementos em pares de chave-valor. Cada valor é associado a uma chave única, e os elementos no dicionário não possuem uma ordem específica. Dicionários são conhecidos por outros nomes em diferentes linguagens de programação, como "hash maps" ou "associações".

Exemplo de um dicionário:

python
```
# Criando um dicionário de informações de
uma pessoa
pessoa = {
    'nome': 'João',
    'idade': 30,
    'cidade': 'São Paulo'
}
```

11.2. Características de um dicionário

As principais características dos dicionários em Python são:

1. Chave-Valor: Cada elemento no dicionário é representado por um par chave-valor, onde a chave é a única usada para acessar o valor correspondente.
2. Chaves Únicas: As chaves em um dicionário devem ser únicas, o que significa que não pode haver chaves duplicadas.
3. Mutabilidade: Os dicionários são mutáveis, o que significa que você pode adicionar, modificar ou remover pares chave-valor após a criação do dicionário.
4. Não-Ordenação: Os elementos em um dicionário não possuem uma ordem específica, o que significa que eles não são acessados por meio de índices numéricos, mas sim pelas chaves.

11.3. Criação de dicionários

Para criar um dicionário em Python, você utiliza chaves {} e coloca os pares chave-valor separados por dois-pontos :. Veja um exemplo:

python
```
# Criando um dicionário de informações de
uma pessoa
pessoa = {
    'nome': 'João',
    'idade': 30,
    'cidade': 'São Paulo'
}
```

Para acessar os valores do dicionário, você utiliza a chave entre colchetes []:

python
```
print(pessoa['nome'])  # Saída: João
print(pessoa['idade'])  # Saída: 30
```

Você pode modificar os valores associados a uma chave existente ou adicionar novos pares chave-valor da seguinte forma:

python
```
pessoa['idade'] = 31
pessoa['profissao'] = 'Engenheiro'
```

Também é possível verificar se uma chave específica está presente no dicionário usando o operador in:

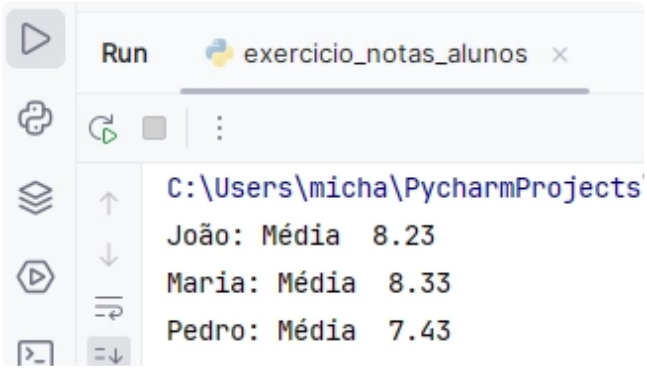
```python
if 'cidade' in pessoa:
    print("Chave 'cidade' encontrada no dicionário.")
```

Os dicionários em Python são amplamente utilizados para mapear dados em diversas aplicações e oferecem uma forma eficiente de acessar, adicionar e manipular informações associadas a chaves específicas.

11.4. Exercícios opcionais

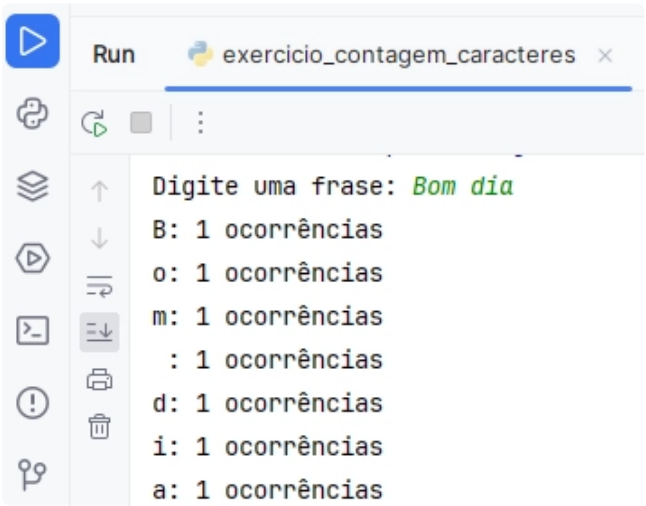
Exercício 1 - Gerenciando Notas de Alunos

1. Crie um novo arquivo Python em seu editor de código favorito e nomeie-o como "exercicio_notas_alunos.py".
2. Escreva a seguinte linha de código para criar um dicionário vazio chamado "notas_alunos": notas_alunos = {}notas_alunos = {}
3. Passe para a próxima linha e adicione as notas de três alunos ao dicionário. Utilize o nome do aluno como chave e uma lista com as notas, como valor:
4. Passe para a próxima linha e escreva um loop para calcular a média de cada aluno e armazenar em um novo dicionário chamado "medias_alunos":
5. Passe para a próxima linha e exiba as médias de cada aluno:
6. Salve o arquivo "exercicio_notas_alunos.py" e execute-o. Verifique a saída que exibe as médias de cada aluno.
7. Observe o resultado abaixo:



Exercício 2: Contando Caracteres em uma String

1. Crie um novo arquivo Python em seu editor de código favorito e nomeie-o como "exercicio_contagem_caracteres.py".
2. Escreva a seguinte linha de código para receber uma string de entrada do usuário: entrada = input("Digite uma frase: ")
3. Passe para a próxima linha e crie um dicionário vazio chamado "contagem_caracteres":
4. Passe para a próxima linha e use um loop para contar a ocorrência de cada caractere na string e armazenar no dicionário "contagem_caracteres":
5. Passe para a próxima linha e exiba a contagem de cada caractere na string:
6. Salve o arquivo "exercicio_contagem_caracteres.py" e execute-o. Digite uma frase de entrada e verifique a saída que exibe a contagem de cada caractere na frase. Salve o arquivo "exercicio_contagem_caracteres.py" e execute-o.
7. Observe o resultado abaixo:



12.1. Arquivos

Em Python, arquivos são recursos usados para armazenar dados em dispositivos de armazenamento permanentes, como discos rígidos ou SSDs. Eles permitem que você leia informações de um arquivo ou escreva dados em um arquivo para persistência ou compartilhamento de dados entre diferentes execuções de um programa.

Os arquivos podem ser de diferentes tipos, como arquivos de texto (.txt), arquivos binários, arquivos CSV (.csv), arquivos JSON (.json), entre outros. Cada tipo de arquivo possui sua própria estrutura e forma de manipulação.

12.2. Manipulação de Arquivos

Em Python, a manipulação de arquivos é feita usando funções e métodos específicos que permitem abrir, ler e escrever em arquivos. As principais operações em arquivos são:

1. Abrir um arquivo: Para começar a trabalhar com um arquivo, você precisa abri-lo usando a função open(). Essa função recebe o caminho do arquivo e o modo de abertura, que pode ser "r" para leitura, "w" para escrita, "a" para anexar conteúdo no final do arquivo ou "x" para criar um novo arquivo para escrita.
2. Ler um arquivo: Para ler o conteúdo de um arquivo, você pode usar o método read() após abrir o arquivo em modo de leitura ("r"). Ele retorna o conteúdo completo do arquivo como uma string.
3. Escrever em um arquivo: Para escrever em um arquivo, você pode usar o método write() após abrir o arquivo em modo de escrita ("w" ou "a"). Esse método permite gravar strings no arquivo.
4. Fechar um arquivo: Após a manipulação do arquivo, é importante fechá-lo usando o método close() para liberar os recursos do sistema operacional associados ao arquivo.

Exemplo de leitura de um arquivo de texto:

```python
# Abrir um arquivo em modo de leitura
arquivo = open("arquivo.txt", "r")

# Ler o conteúdo do arquivo
conteudo = arquivo.read()

# Fechar o arquivo após a leitura
arquivo.close()

# Imprimir o conteúdo lido
print(conteudo)
```

Exemplo de escrita em um arquivo de texto:

```python
# Abrir um arquivo em modo de escrita
arquivo = open("novo_arquivo.txt", "w")

# Escrever no arquivo
arquivo.write("Este é um novo arquivo criado por Python!")

# Fechar o arquivo após a escrita
arquivo.close()
```

Os arquivos em Python são amplamente usados para armazenar e acessar dados persistentes, como configurações, registros de eventos, arquivos de texto, imagens, e muito mais.

12.3. Trabalhando com arquivos

Trabalhar com arquivos em Python envolve algumas etapas básicas: abrir, ler ou escrever,

manipular o conteúdo e, finalmente, fechar o arquivo. Vamos explorar essas etapas em mais detalhes:

Abrir um arquivo: Para abrir um arquivo, você usa a função open() que recebe o caminho do arquivo e o modo de abertura como parâmetros. O modo de abertura pode ser "r" para leitura, "w" para escrita, "a" para anexar conteúdo no final do arquivo ou "x" para criar um novo arquivo para escrita.

python
```
# Exemplo: Abrir um arquivo para leitura
arquivo_leitura = open("arquivo.txt", "r")

# Exemplo: Abrir um arquivo para escrita
arquivo_escrita = open("novo_arquivo.txt",
"w")
```

Ler o conteúdo do arquivo: Para ler o conteúdo de um arquivo aberto em modo de leitura, você pode usar o método read(). Esse método retornará o conteúdo completo do arquivo como uma string.

python
```
conteudo = arquivo_leitura.read()
print(conteudo)
```

Escrever em um arquivo: Para escrever em um arquivo aberto em modo de escrita ou anexo, você pode usar o método write(). Esse método permite gravar strings no arquivo.

python
```
arquivo_escrita.write("Este é um novo
arquivo criado por Python!")
```

Fechar o arquivo: Sempre que terminar de trabalhar com um arquivo, é importante fechá-lo usando o método close() para liberar os recursos do sistema operacional associados ao arquivo.

python
```
arquivo_leitura.close()
arquivo_escrita.close()
```

Lembre-se de que a manipulação de arquivos pode levantar exceções, como quando o arquivo não é encontrado ou quando há problemas de permissão. Certifique-se de incluir tratamento de exceções para garantir que o programa lide adequadamente com essas situações.

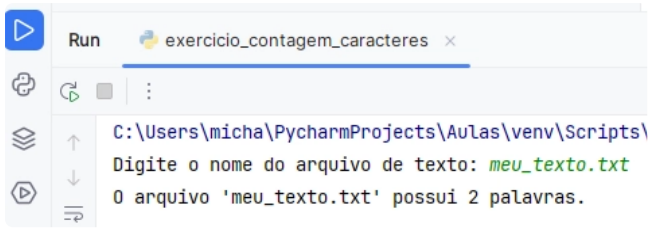
12.4. Exercícios opcionais

Exercício 1 - Contador de Palavras em um Arquivo de Texto

Obs.: Lembre-se de criar um documento de texto na mesma pasta onde os exercícios estão sendo salvos. Exemplo: C:\Users\meuPC\PycharmProjects\Aulas\meu_te

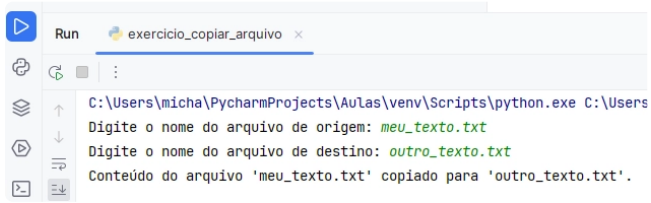
1. Crie um novo arquivo Python em seu editor de código favorito e nomeie-o como "exercicio_contador_palavras.py".
2. Escreva a seguinte linha de código para receber o nome do arquivo de texto a ser lido como entrada do usuário: nome_arquivo = input("Digite o nome do arquivo de texto: ")
3. Passe para a próxima linha e abra o arquivo em modo de leitura, usando o bloco with:
4. Passe para a próxima linha e leia o conteúdo completo do arquivo usando o método read():
5. Passe para a próxima linha e conte o número de palavras no conteúdo do arquivo usando o método split() para dividir o conteúdo em palavras:
6. Passe para a próxima linha e exiba o resultado do contador de palavras:
7. Salve o arquivo "exercicio_contador_palavras.py" e execute-o. Digite o nome de um arquivo de

texto e verifique o número de palavras no arquivo.
8. Observe o resultado abaixo:



Exercício 2: Copiar Conteúdo de um Arquivo para Outro

1. Crie um novo arquivo Python em seu editor de código favorito e nomeie-o como "exercicio_copiar_arquivos.py".
2. Escreva a seguinte linha de código para receber os nomes dos arquivos de origem e destino como entrada do usuário:
3. Passe para a próxima linha e abra o arquivo de origem em modo de leitura e o arquivo de destino em modo de escrita, usando o bloco with:
4. Passe para a próxima linha e leia o conteúdo completo do arquivo de origem usando o método read():
5. Passe para a próxima linha e escreva o conteúdo lido no arquivo de destino usando o método write():
6. Passe para a próxima linha e exiba uma mensagem informando que a cópia foi realizada com sucesso:
7. Salve o arquivo "exercicio_copiar_arquivos.py" e execute-o. Digite os nomes de um arquivo de origem e um arquivo de destino e verifique se o conteúdo foi copiado corretamente.
8. Confira o resultado abaixo:



Anotações

13.1. O que são Bibliotecas Externas?

As bibliotecas externas são conjuntos de código pré-escrito que estendem a funcionalidade do Python. Elas contêm funções, classes e métodos que podem ser importados e utilizados para tarefas específicas.

13.2. Por que usar Bibliotecas Externas?

As bibliotecas externas economizam tempo e esforço, permitindo que você reutilize soluções desenvolvidas por outros desenvolvedores. Elas fornecem funcionalidades avançadas sem que você precise escrever todo o código do zero.

13.3. Como instalar Bibliotecas Externas

Para utilizar bibliotecas externas em seus projetos Python, é necessário instalá-las no ambiente em que você está trabalhando. A ferramenta padrão para fazer isso é o pip, o gerenciador de pacotes do Python. Siga os passos abaixo para instalar bibliotecas externas:

1. Verificando a Instalação do Python e do Pip.

Antes de tudo, verifique se você tem o Python e o Pip instalados em seu sistema. Abra o terminal e digite os seguintes comandos:

```python
python --version
pip --version
```

Certifique-se de que não há erros e de que a versão do Python e do Pip é exibida corretamente. Caso contrário, você precisará instalar o Python e configurar o Pip corretamente.

2. Encontrando o Nome da Biblioteca.

Você geralmente encontrará o nome da biblioteca que deseja instalar em sua documentação oficial ou em repositórios online, como o PyPI (Python Package Index).

3. Instalando a Biblioteca.

No terminal, utilize o comando pip install seguido do nome da biblioteca para instalá-la. Por exemplo, para instalar a biblioteca NumPy:

```python
pip install numpy
```

Repita esse passo a passo para qualquer outra biblioteca que você deseja instalar.

13.4. Principais Bibliotecas Externas em Python

Abaixo, vamos verificar as bibliotecas mais populares em Python.

1. NumPy

O NumPy é uma biblioteca fundamental para computação científica em Python. Ela oferece suporte a arrays multidimensionais e funções matemáticas de alto desempenho.

Exemplo de uso:

python
```python
import numpy as np

array = np.array([1, 2, 3, 4, 5])
mean = np.mean(array)
print("Média:", mean)
```

2. Pandas

O Pandas é amplamente usado para análise e manipulação de dados. Ele fornece as estruturas de dados DataFrame e Series, essenciais para lidar com dados tabulares.

Exemplo de uso:

python
```python
import pandas as pd

data = {'Nome': ['Alice', 'Bob', 'Carol'],
        'Idade': [25, 30, 22]}
df = pd.DataFrame(data)
print(df)
```

3. Matpoltbit

O Matplotlib é uma biblioteca de visualização que permite criar gráficos e visualizações de dados de forma fácil e flexível.

Exemplo de uso:

python
```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 9]
plt.plot(x, y)
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')
plt.title('Gráfico de Linha')
plt.show()
```

4. Requests

A biblioteca Requests é usada para fazer requisições HTTP, permitindo a interação com APIs da web e a obtenção de conteúdo online.

Exemplo de uso:

python
```python
import requests

response =
requests.get('https://api.exemplo.com/dados'
)
data = response.json()
print(data)
```

Curiosidade do dia:

O arquivo requirements.txt é usado para listar todas as bibliotecas externas que seu projeto utiliza. Isso facilita a replicação do ambiente em outros sistemas.

13.5. Exercícios opcionais

Exercício 1 - Instalando e utilizando a biblioteca NumPy

1. Abra seu ambiente Python (IDE ou terminal);

2. No terminal, digite o comando "pip install numpy" para instalar a biblioteca;

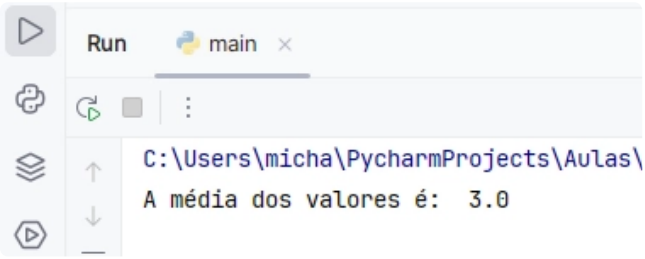
3. Crie um novo arquivo Python;

4. Insira o seguinte código

```python
import numpy as np

# Crie um array NumPy com os números de 1 a 5
array = np.array([1, 2, 3, 4, 5])

# Calcule a média dos valores do array
mean = np.mean(array)

# Imprima a média
print("A média dos valores é:", mean)
```

5. Execute o código, você deve ver a média dos números impressos na saída;

6. Confira o resultado abaixo.



Exercício 2 - Criando um gráfico em linha com Matplotlib

1. Abra seu ambiente Python (IDE ou terminal);

2. No terminal, digite o comando "pip install matplotlib" para instalar a biblioteca;

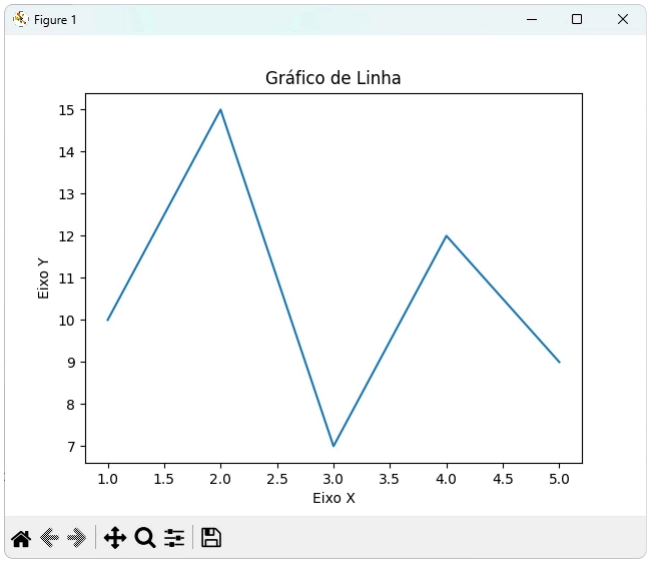
3. Crie um novo arquivo Python;

4. Insira o seguinte código:

```python
import matplotlib.pyplot as plt

# Dados para o gráfico
x = [1, 2, 3, 4, 5]
y = [10, 15, 7, 12, 9]

# Crie um gráfico de linha
plt.plot(x, y)

# Adicione rótulos aos eixos
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# Adicione um título ao gráfico
plt.title('Gráfico de Linha')

# Exiba o gráfico
plt.show()
```

5. Execute o código, você deve ver a média dos números impressos na saída;

6. Confira o resultado abaixo.



14.1. O que é Data e Hora?

Em nosso mundo, a noção de data e hora é essencial para organizar e compreender o tempo que passa. Da mesma forma, em programação, a capacidade de trabalhar com datas e horas é crucial para muitos tipos de aplicativos e tarefas. Em Python, o módulo datetime oferece ferramentas poderosas para representar, manipular e formatar informações relacionadas a data e hora. Abaixo temos o conceito de cada um:

Data é uma representação numérica de um dia específico em um calendário. Normalmente, é composta por dia, mês e ano. Por exemplo, 15 de agosto de 2023 é uma data.

Hora refere-se a um ponto específico no tempo dentro de um dia. Ela é composta por horas, minutos e segundos. Por exemplo, 14 horas, 30 minutos e 0 segundos representam um horário específico durante o dia.

A combinação de data e hora permite que representemos momentos precisos no tempo. Isso é vital para agendamento, registro de eventos, cálculos de duração e muitos outros cenários.

14.2. O módulo "datetime" em Python

Python oferece o módulo **datetime** para trabalhar com data e hora de forma eficiente. Ele fornece classes que permitem a criação, manipulação, formatação e conversão de objetos de data e hora. Com o módulo **datetime**, você pode realizar cálculos temporais, lidar com fusos horários e muito mais.

14.3. Manipulação de Data e Hora

A manipulação de data e hora é crucial em várias áreas:

1. Agendamento e Calendários: Aplicações que envolvem agendamento de eventos, reservas, lembretes e calendários precisam manipular datas e horários.
2. Análise de Dados Temporais: Muitas vezes, precisamos analisar e visualizar dados que evoluem ao longo do tempo, como vendas diárias, tráfego de site por hora, entre outros.
3. Sistemas Financeiros: Aplicações financeiras dependem do cálculo preciso de prazos de vencimento, taxas de juros e outras operações relacionadas ao tempo.
4. Computação Científica: Em áreas como meteorologia, física e biologia, a manipulação de data e hora é crucial para análises e modelagem. **Exemplo na prática:**

1. Representando Data e Hora

Exemplo de como criar objetos de data e hora utilizando as classes fornecidas pelo módulo "datetime".

python
```python
from datetime import datetime, date, time

# Data e hora atuais
data_hora_atual = datetime.now()

# Somente a data
data_atual = date.today()

# Somente o horário
hora_atual = datetime.time(datetime.now())

print("Data e Hora Atuais:",
data_hora_atual)
print("Data Atual:", data_atual)
print("Horário Atual:", hora_atual)
```

2. Manipulando Data e Hora

Exemplo de como realizar operações de cálculo e manipulação com objetos de data e hora.

```python
from datetime import datetime, timedelta

data_atual = datetime.now()
data_futura = data_atual + timedelta(days=7)

diferenca = data_futura - data_atual
print("Diferença:", diferenca)
```

3. Formatando Data e Hora

Exemplo de como formatar objetos de data e hora em strings legíveis.

```python
from datetime import datetime

data_atual = datetime.now()

# Formato padrão
formato_padrao = data_atual.strftime('%Y-%m-%d %H:%M:%S')
print("Formato Padrão:", formato_padrao)

# Formato customizado
formato_customizado = data_atual.strftime('%d/%m/%Y %I:%M %p')
print("Formato Customizado:", formato_customizado)
```

4. Convertendo strings para data e hora

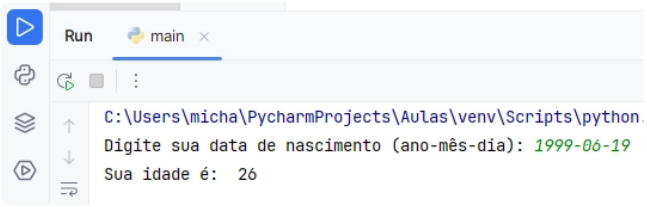
Exemplo de como converter strings em objetos de data e hora.

```python
from datetime import datetime

data_string = "2023-08-15 14:30:00"
data_objeto = datetime.strptime(data_string, '%Y-%m-%d %H:%M:%S')

print("Data e Hora como Objeto:", data_objeto)
```

14.4. Exercícios opcionais

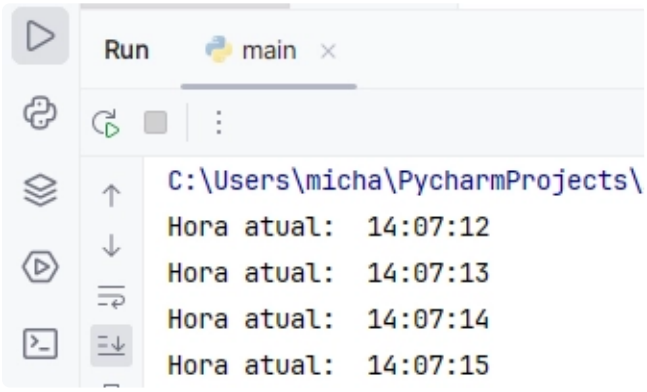
Exercício 1 - Calculando Idade a partir da Data de Nascimento

1. Abra seu ambiente Python (IDE ou terminal).
2. Crie um novo arquivo Python;
3. Comece a digitar o seguinte código, na primeira linha: from datetime import datetime;
4. Na segunda linha, o seguinte: data_nascimento_str = input("Digite sua data de nascimento (ano-mês-dia): ")
5. Na terceira linha: data_nascimento = datetime.strptime(data_nascimento_str, '%Y-%m-%d')
6. Na quarta linha: data_atual = datetime.now()
7. Na quinta linha: idade = data_atual.year - data_nascimento.year - ((data_atual.month, data_atual.day) < (data_nascimento.month, data_nascimento.day))
8. Na sexta linha do código, digite o seguinte: print("Sua idade é:", idade)
9. Execute o código e insira a data de nascimento quando solicitado;
10. O programa calculará a idade com base na data atual.
11. Confira o resultado abaixo.



Exercício 2 - Criando um relógio digital

1. Abra seu ambiente Python (IDE ou terminal);
2. Crie um novo arquivo Python;
3. Comece a digitar o seguinte código, na primeira linha: import time
4. Na segunda linha, digite o seguinte: whileTrue:
5. Na terceira linha: hora_atual = time.strftime('%H:%M:%S')
6. Na quarta linha: print("Hora atual:", hora_atual)
7. Na quinta linha, digite o seguinte: time.sleep(1)
8. Execute o código.
9. O programa exibirá a hora atual em formato de relógio digital, atualizando a cada segundo.
10. Confira o resultado abaixo.



Anotações

15.1. O que são Expressões Regulares?

Expressões regulares (regex ou regexp) são sequências de caracteres que definem padrões de busca. Elas são usadas para encontrar, substituir ou manipular partes específicas de strings com base nesses padrões.

15.2. Aplicações das Expressões Regulares

As expressões regulares são amplamente usadas para:

- Validar formatos de entrada (e-mails, números de telefone, etc.).
- Extrair informações específicas de um texto.
- Substituir ou formatar partes do texto.
- Realizar buscas complexas em grandes volumes de dados.

15.3. Sintaxe básica das Expressões Regulares

Caracteres Literais

Os caracteres literais são os próprios caracteres que você deseja encontrar.

Exemplo

```python
Padrão: apple
```

Metacaracteres

Metacaracteres são caracteres com significados especiais nas expressões regulares.

- .: Corresponde a qualquer caractere, exceto nova linha.
- ^: Corresponde ao início de uma linha.
- $: Corresponde ao final de uma linha.
- *: Corresponde a zero ou mais ocorrências do caractere anterior.
- +: Corresponde a uma ou mais ocorrências do caractere anterior.
- ?: Corresponde a zero ou uma ocorrência do caractere anterior.
- \: Escapa o próximo caractere (usado para tratar metacaracteres como literais).

15.4. Utilizando Expressões Regulares em Python

Funções Principais do Módulo re

- re.search(padrao, texto): Procura por um padrão no texto.
- re.match(padrao, texto): Verifica se o padrão corresponde no início do texto.
- re.findall(padrao, texto): Retorna todas as ocorrências do padrão no texto.
- re.sub(padrao, substituicao, texto): Substitui ocorrências do padrão pelo texto de substituição.

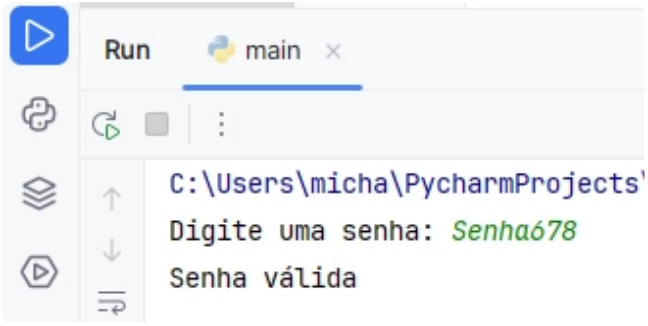
Exemplo:

```python
import re

texto = "Python é uma linguagem poderosa e
Python é divertido de aprender."

padrao = r"Python"
encontrado = re.search(padrao, texto)
print("Padrão encontrado:",
encontrado.group())

ocorrencias = re.findall(padrao, texto)
print("Ocorrências:", ocorrencias)

substituido = re.sub(padrao, "Java", texto)
print("Substituído:", substituido)
```

15.5. Exercício opcionais

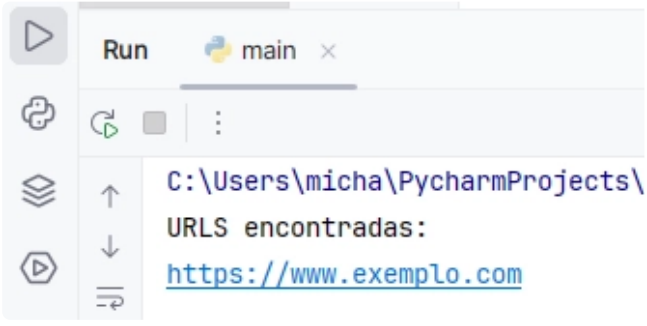
Exercício 1 - Validação de senhas

1. Abra seu ambiente Python (IDE ou terminal).
2. Crie um novo arquivo Python;
3. Comece a digitar o seguinte código, na primeira linha: import re
4. Na segunda linha, digite o seguinte: defvalidar_senha(senha)::
5. Na terceira linha, digite o seguinte: padrao = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$"
6. Na quarta linha, digite o seguinte: if re.match(padrao, senha):
7. Na sétima linha, digite o seguinte: returnTrue
8. Na oitava linha, digite o seguinte: returnFalse
9. Na nona linha, digite o seguinte: senha = input("Digite uma senha: ")
10. Na décima linha, digite o seguinte: if validar_senha(senha):
11. Na décima primeira linha, digite o seguinte: print("Senha válida.")
12. Na décima segunda linha, digite o seguinte: else:
13. Na décima quarta linha, digite o seguinte: print("Senha inválida.")
14. Execute o código e insira a senha quando solicitado.
15. O programa verificará se a senha atende aos critérios especificados na expressão regular. Confira o resultado abaixo.



Exercício 2 - Extração de URLs de um texto

1. Abra seu ambiente Python (IDE ou terminal).
2. Crie um novo arquivo Python;
3. Comece a digitar o seguinte código, na primeira linha: import reimport re;
4. Na segunda linha, digite o seguinte: texto = "Confira nossos produtos em https://www.exemplo.com/produtos e faça suas compras online."
5. Na terceira linha, digite o seguinte: padrao = r"https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+""
6. Na quarta linha, digite o seguinte: urls = re.findall(padrao, texto)
7. Na quinta linha, digite o seguinte: print("URLs encontradas:")
8. Na sexta linha, digite o seguinte: for url in urls:
9. Na sétima linha, digite o seguinte: print(url)
10. Execute o código. O programa extrairá e exibirá as URLs encontradas no texto de acordo com a expressão regular.
11. Confira o resultado abaixo.



16.1. Guia do Projeto Final

O projeto consiste em um programa que permite ao usuário registrar despesas e receitas, calcular o saldo, gerar relatórios por categoria e exportar dados para um arquivo CSV. O projeto foi dividido em partes para facilitar o entendimento e o desenvolvimento gradual.

16.2. Parte 1 - Registro de despesas e Receitas

Nesta parte, utilizamos o seguinte conhecimento:

- Uso de listas para armazenar dados.
- Criação de funções para adicionar despesas e receitas.
- Interagindo com o usuário para obter valores e registrar transações.

16.3. Parte 2 - Cálculo do Saldo e Categorias

Nesta parte, exploramos:

- Cálculo de saldo através da diferença entre receitas e despesas.
- Utilização de categorias para classificar transações.
- Melhoria na interface com o usuário.

16.4. Parte 3 - Geração de Relatórios

Nesta parte, abordamos:

- Criação de relatórios por categoria de despesas e receitas.
- Uso de dicionários para agrupar dados relacionados.
- Apresentação de informações ao usuário de forma organizada.

16.5. Parte 4 - Finalização do Projeto

Nesta parte final, discutimos:

- Exportação de dados para um arquivo CSV.
- Encerramento do programa de forma controlada.
- Revisão geral do projeto e funcionalidades.

Conceitos abordados

Durante o desenvolvimento deste projeto, exploramos os seguintes conceitos chave:

- Listas: Armazenamento e manipulação de conjuntos de dados.
- Funções: Criação de blocos reutilizáveis de código.
- Estruturas de Controle: Utilização de loops e condicionais para controlar o fluxo do programa.
- Interatividade: Interação com o usuário através de entradas e saídas.
- Dicionários: Organização de dados em pares chave-valor para categorização.
- Manipulação de Arquivos: Escrita de dados em um arquivo CSV para exportação.
- Modularização: Divisão do código em partes para facilitar o desenvolvimento e manutenção.

Conclusão

Este projeto de Controle Financeiro em Python oferece uma base sólida para entender conceitos de programação e aplicá-los em um cenário real. Com as habilidades adquiridas, você pode personalizar e expandir o projeto para incluir mais funcionalidades, melhorias na interface e até mesmo uma análise mais avançada dos dados financeiros.

O conhecimento obtido ao longo deste projeto pode ser aplicado em várias outras áreas da programação, desde o desenvolvimento de aplicativos até a automação de tarefas. Esperamos que este guia tenha sido útil para a compreensão dos conceitos abordados no projeto e inspire você a explorar ainda mais o mundo da programação em Python.

16.6. Exercício opcional

O Nosso exercício opcional de hoje, é na verdade o desenvolvimento de um novo projeto, neste caso, este projeto será voltado para o controle de estoque de uma empresa.

1. Crie um arquivo chamado controle_estoque.py para o projeto.
2. Defina uma lista vazia chamada estoque para armazenar os produtos. Cada produto será um dicionário contendo informações como nome, quantidade e preço.
3. Crie a função adicionar_produto(nome, quantidade, preco) para adicionar um novo produto ao estoque. Esta função deve criar um dicionário para o produto e adicioná-lo à lista estoque.
4. Crie a função listar_produtos() para exibir a lista de produtos do estoque.
5. No corpo do programa, crie um loop infinito usando while True.
6. Dentro do loop, exiba um menu com opções para o usuário: adicionar produto, listar produtos ou sair.
7. Se o usuário escolher "adicionar produto", solicite o nome, quantidade e preço do produto usando a função input().
8. Chame a função adicionar_produto(nome, quantidade, preco) para adicionar o novo produto ao estoque.
9. Se o usuário escolher "listar produtos", chame a função listar_produtos() para exibir os produtos e suas informações.
10. Adicione uma opção no menu para vender produtos.
11. Solicite ao usuário o nome do produto a ser vendido e a quantidade.
12. Implemente uma função vender_produto(nome, quantidade) que atualize a quantidade em estoque do produto escolhido.
13. Adicione uma opção no menu para gerar um relatório de estoque.
14. Implemente uma função gerar_relatorio() que exiba a lista de produtos, suas quantidades e preços.
15. Se o usuário escolher "sair", encerre o loop infinito usando break.
16. Execute o programa e teste as diferentes opções do menu.
17. Confira o resultado abaixo:

```
Controle de Estoque

1 - Adicionar Produto
2 - Listar Produto
3 - Vender Produto
4 - Gerar Relatório
5 - Sair

Escolha uma opção: 1
Digite o nome do produto: Arroz
Digite a quantidade: 50
Digite o preço: R$ 20.50
Produto 'Arroz' adicionado ao estoque.
```