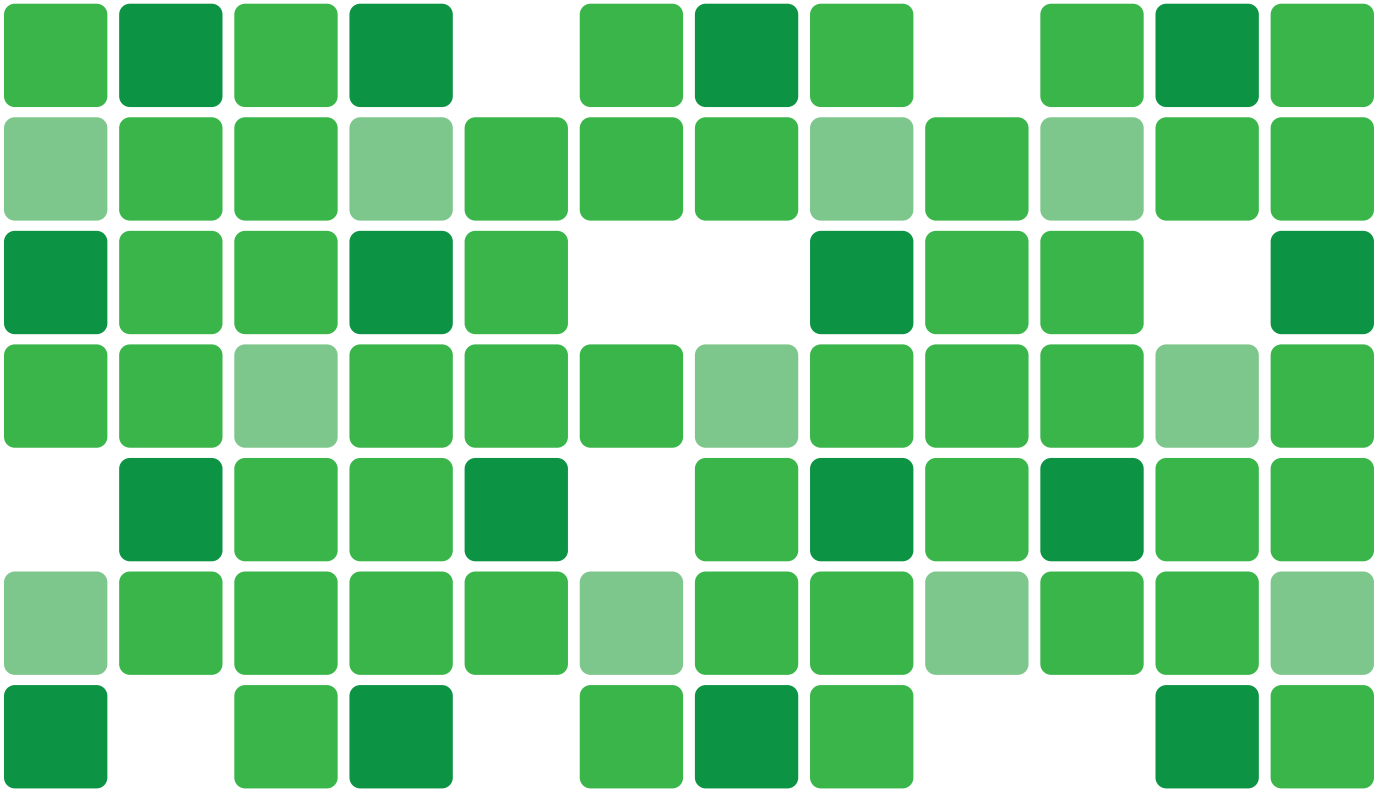
Programação para a Web

Curso Técnico em Informática

Prof. Dr. Ricardo Maroquio



Instituto Federal do Espírito Santo
Campus Cachoeiro

Disponível para download em:
http://maroquio.com

Fevereiro de 2023

Sumário

Lista de Exemplos de Código

Parte I: Python

1. Fundamentos da Linguagem

O Python é uma linguagem de programação poderosa e versátil amplamente utilizada em aplicações de computação científica, análise de dados, desenvolvimento de aplicações e muito mais. Neste primeiro capítulo, revisaremos os conceitos básicos da programação em Python e veremos como aplicá-los para resolver problemas comuns do mundo da programação.

Vale ressaltar que este capítulo tem como proposta a realização de uma rápida revisão da linguagem Python, ou seja, pressupõe-se que você já tenha tido alguma experiência com programação em Python. Ao longo do capítulo, você aprenderá sobre tipos de dados, variáveis, estruturas de controle de fluxo, funções e muito mais. Também veremos muitos exemplos práticos que irão ajudá-lo a aplicar os conceitos abordados.

Ao concluir este capítulo, você terá revisto todos os recursos necessários para escrever programas simples em Python e estará preparado para se aprofundar em recursos adicionais de programação em Python com mais confiança.

1.1 Variáveis

Variáveis são consideradas componentes fundamentais da programação. Elas permitem armazenar valores que podem ser usados posteriormente ao longo do programa. Em Python, as variáveis são criadas usando o sinal de igual (=). O exemplo 1.1 mostra a criação de duas variáveis simples.

```
1 nome = "João"
2 idade = 30
```

Exemplo de Código 1.1: Atribuição de valores a variáveis

Neste exemplo, criamos duas variáveis: `nome` e `idade`. O nome da variável fica à esquerda do sinal de igual e o valor da variável fica à direita. O tipo da variável é determinado automaticamente com base no valor atribuído. Uma variável pode ter seu valor modificado a qualquer momento. O exemplo 1.2

```
nome = 'João'
idade = 30
print("Meu nome é", nome, "e tenho", idade, "anos.")

nome = "Maria"
idade = 35
print('Meu nome é', nome, "e tenho", idade, "anos.")
```

Exemplo de Código 1.2: Reatribuição de valores a variáveis

Neste exemplo, alteramos o valor da variável `nome` de "João"para "Maria"e o valor da variável `idade` de 30 para 35. Quando o programa é executado, o texto "Meu nome é Maria e tenho 35 anos."é impresso na tela.

Lembre-se de que as variáveis em Python não precisam ser declaradas com antecedência. Basta atribuir um valor a uma variável para criá-la automaticamente. Além disso, é importante escolher nomes de variáveis descritivos e significativos para facilitar a leitura e manutenção do código. As regras para nomeação de variáveis em Python são as seguintes:

1. Nomes de variáveis devem começar com uma letra ou um sublinhado (_);
2. Após o primeiro caractere, podem conter letras, números e sublinhados;
3. Nomes de variáveis não podem ser uma palavra reservada do Python, como `if`, `else`, `for` etc.;
4. Nomes de variáveis devem ser descritivos e significativos;
5. Nomes de variáveis não podem conter espaços. Em vez disso, usa-se o sublinhado para separar palavras;
6. Nomes de variáveis devem ser escritos em minúsculas, exceto em casos de convenções de nomenclatura como `camelCase`, `PascalCase` ou `snake_case`.

Agora que você já sabe como declarar variáveis em Python, vamos quais são os possíveis tipos de valores que podemos atribuir a uma variável.

1.2 Tipos de Dados

Em programação, é importante compreender os diferentes tipos de dados que você pode trabalhar. Em Python, existem vários tipos de dados básicos que você precisa conhecer, incluindo números, strings, listas, dicionários, conjuntos, entre outros. Vamos explorar cada um desses tipos em detalhes.

1.2.1 Números

Existem dois tipos numéricos em Python: inteiros (*int*) e de ponto flutuante (*float*). No exemplo 1.3, veja que a variável `x` recebe um valor inteiro e que a variável `y` recebe um valor real.

```
1 x = 10  # int
2 y = 3.14  # float
```

Exemplo de Código 1.3: Tipos numéricos

1.2.2 Textos

Textos são uma sequência de caracteres. Em Python, são representados pelo tipo *string* seus valores podem ser delimitados por aspas simples ou duplas. O exemplo 1.4 mostra como criar strings usando aspas simples, aspas duplas e 3 aspas duplas ou simples para strings de múltiplas linhas.

```
1 frase1 = "Olá, Python!"  # string delimitado por aspas duplas
2 frase2 = 'Olá, Python!'  # string delimitado por aspas simples
3 frase3 = """Este é um texto com
4     múltiplas linhas. Você deve
5     usar 3 aspas para delimitar
6     o início e o fim da string""" # string delimitado pro 3 aspas duplas
```

Exemplo de Código 1.4: Tipo Texto

1.2.3 Listas

As listas são uma estrutura de dados em Python que permitem armazenar uma coleção de itens tipos iguais ou diferentes. As listas são delimitadas por colchetes [] e os itens são separados por vírgulas. O exemplo 1.5 mostra a criação de 3 listas diferentes.

Você pode acessar, alterar ou remover itens em uma lista usando seus índices (posições) e também pode realizar operações comuns em listas, como concatenação, repetição e pesquisa. Além disso, as listas são objetos mutáveis, o que significa que você pode modificar seus itens

```python
# criando uma lista de números inteiros
numeros = [1, 2, 3, 4, 5]

# criando uma lista de strings
frutas = ['maçã', 'banana', 'laranja']

# criando uma lista mista de diferentes tipos de dados
mista = [1, 'dois', 3.0, ['a', 'b', 'c']]
```

Exemplo de Código 1.5: Tipo Lista

depois de criá-los. Veremos listas novamente mais adiante de forma mais aprofundada.

1.2.4 Dicionários

Em Python, um dicionário é uma estrutura de dados que permite armazenar uma coleção de pares chave-valor. Ao contrário de uma lista, que armazena itens em uma ordem específica baseada em índices, um dicionário armazena itens com base nas suas chaves. As chaves são usadas para acessar seus valores correspondentes.

Os dicionários são delimitados por chaves e cada par chave-valor é separado por vírgulas. As chaves podem ser de diferentes tipos, como números, strings ou tuplas, mas precisam ser imutáveis. Os valores podem ser de qualquer tipo. O exemplo 1.6 mostra a criação e algumas operações básicas sobre um dicionário.

```python
# criando um dicionário com pares chave-valor
pessoa = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}

# acessando um valor pelo seu nome de chave
print(pessoa['nome']) # imprime "João"

# alterando o valor de uma chave
pessoa['idade'] = 31
print(pessoa['idade']) # imprime 31

# adicionando um novo par chave-valor
pessoa['pais'] = 'Brasil'
print(pessoa) # imprime {'nome': 'João', 'idade': 31, 'cidade': 'São Paulo', 'pais': 'Brasil'}

# removendo um par chave-valor
del pessoa['cidade']
print(pessoa) # imprime {'nome': 'João', 'idade': 31, 'pais': 'Brasil'}
```

Exemplo de Código 1.6: Tipo Dicionário

1.2.5 Conjuntos

Em Python, conjuntos são coleções não ordenadas e não indexadas de elementos únicos. Eles são definidos usando chaves ou a função set(). O exemplo 1.7 mostra as duas formas de criação de conjuntos.

```
conjunto = {1, 2, 3, 4}
print(conjunto) # output: {1, 2, 3, 4}

conjunto = set([1, 2, 3, 4])
print(conjunto) # output: {1, 2, 3, 4}
```

Exemplo de Código 1.7: Tipo Conjunto

Os elementos em um conjunto não podem ser repetidos e não possuem uma ordem específica. Isso significa que você não pode acessar um elemento específico de um conjunto usando um índice, mas você pode verificar se um elemento está presente no conjunto usando o operador `in`. Você pode realizar operações comuns com conjuntos, como união, interseção e diferença. O exemplo 1.8 mostra a realização de algumas dessas operações.

```
conjunto1 = {1, 2, 3, 4}
conjunto2 = {3, 4, 5, 6}

# União
print(conjunto1 | conjunto2) # output: {1, 2, 3, 4, 5, 6}

# Interseção
print(conjunto1 & conjunto2) # output: {3, 4}

# Diferença
print(conjunto1 - conjunto2) # output: {1, 2}
```

Exemplo de Código 1.8: Operações Sobre Conjuntos

1.2.6 Tuplas

As tuplas são uma das estruturas de dados básicas em Python e são semelhantes a listas. A principal diferença é que as tuplas são imutáveis, ou seja, uma vez criadas, seus elementos não podem ser alterados. As tuplas constumam ser bastante usadas em aplicações que acessam banco de dados, pois é uma boa forma de se representar um registro ou um objeto de um banco de dados. A sintaxe para criar uma tupla é colocar vários elementos separados por vírgulas, envolvidos por parênteses. O exemplo 1.9 mostra a criação de algumas tuplas.

```
# Exemplo 1: Criando uma tupla vazia
tupla_vazia = ()
print(tupla_vazia)  # Saída: ()

# Exemplo 2: Criando uma tupla com elementos
tupla = 1, 2, 3, 4
print(tupla)  # Saída: (1, 2, 3, 4)

# Exemplo 3: Criando uma tupla com elementos e parênteses
tupla = (1, 2, 3, 4)
print(tupla)  # Saída: (1, 2, 3, 4)

# Exemplo 4: Acessando elementos em uma tupla
tupla = (1, 2, 3, 4)
print(tupla[0])  # Saída: 1
print(tupla[-1])  # Saída: 4

# Exemplo 5: Tuplas com elementos de diferentes tipos
tupla = (1, 2, "três", 4.0)
print(tupla)  # Saída: (1, 2, "três", 4.0)
```

Exemplo de Código 1.9: Operações Sobre Conjuntos

Como as tuplas são imutáveis, não é possível adicionar, remover ou alterar seus elementos após sua criação. No entanto, é possível concatenar tuplas, criar novas tuplas a partir de tuplas existentes, entre outras operações. Vamos ver mais algumas informações sobre tuplas:

- Índices: assim como nas listas, é possível acessar elementos específicos de uma tupla usando seus índices. Os índices começam a partir de 0 e você também pode usar índices negativos, o que significa contar a partir do final da tupla.
- Fatiamento: é possível fazer o "fatiamento" (slicing) de tuplas da mesma forma que se faz com listas. Isso significa que você pode selecionar uma sub-tupla de uma tupla existente, especificando o índice inicial e o final da seleção.
- Desempacotamento: é possível "desempacotar" os elementos de uma tupla e atribuí-los a várias variáveis. Isso é útil quando você precisa extrair vários valores de uma tupla e tratá-los separadamente.
- Funções nativas: existem várias funções nativas em Python que são úteis para trabalhar com tuplas, como len(), min(), max(), sum(), entre outras.

O código 1.10 mostra mais alguns exemplos do uso de tuplas com os operadores citados.

```python
# Exemplo 6: Fatiamento de tuplas
tupla = (1, 2, 3, 4, 5)
print(tupla[1:3])  # Saída: (2, 3)
print(tupla[:-2])  # Saída: (1, 2, 3)

# Exemplo 7: Desempacotamento de tuplas
tupla = (1, 2, 3)
a, b, c = tupla
print(a)  # Saída: 1
print(b)  # Saída: 2
print(c)  # Saída: 3

# Exemplo 8: Concatenação de tuplas
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)
tupla_concatenada = tupla1 + tupla2
print(tupla_concatenada)  # Saída: (1, 2, 3, 4, 5, 6)

# Exemplo 9: Funções built-in
tupla = (1, 2, 3, 4, 5)
print(len(tupla))  # Saída: 5
print(min(tupla))  # Saída: 1
print(max(tupla))  # Saída: 5
print(sum(tupla))  # Saída: 15
```

Exemplo de Código 1.10: Operações Sobre Tuplas

1.3 Operadores Matemáticos

Em Python, existem vários operadores matemáticos que permitem realizar operações matemáticas básicas com números, como adição, subtração, multiplicação, divisão, exponenciação, etc. As subseções a seguir apresenta os operadores matemáticos nativos mais comuns da linguagem Python, juntamente com exemplos uso.

1.3.1 Adição

O operador de adição, representado pelo caractere "+", adiciona dois números ou concatena duas strings. Veja o exemplo 1.11 a seguir.

1.3.2 Subtração

O operador de subtração, representado pelo caractere "-", subtrai dois números. Veja o exemplo 1.12 a seguir.

```
a = 5
b = 3
c = a + b
print(c) # imprime 8

s1 = "Olá"
s2 = " mundo!"
s3 = s1 + s2
print(s3) # imprime "Olá mundo!"
```

Exemplo de Código 1.11: Usos do operador de adição

```
a = 5
b = 3
c = a - b
print(c) # imprime 2
```

Exemplo de Código 1.12: Operador de subtração

1.3.3 Multiplicação

O operador de multiplicação, representado pelo caractere "*", multiplica dois números ou repetidamente concatena uma string. Veja o exemplo 1.13 a seguir.

```
a = 5
b = 3
c = a * b
print(c) # imprime 15

s = "Oi"
r = s * 3
print(r) # imprime "OiOiOi"
```

Exemplo de Código 1.13: Operador de multiplicação

1.3.4 Divisão

O operador de divisão, representado pelo caractere "/", divide um número por outro. Veja o exemplo 1.15 a seguir.

1.3.5 Divisão Inteira

O operador de divisão inteira, representado pelos caracteres "//", divide um número por outro e retorna somente a parte inteira do resultado. Veja o exemplo 1.15 a seguir.

```
a = 6
b = 3
c = a / b
print(c) # imprime 2.0
```

Exemplo de Código 1.14: Operador de divisão

```
a = 7
b = 3
c = a // b
print(c) # imprime 2
```

Exemplo de Código 1.15: Operador de divisão inteira

1.3.6 Resto da Divisão Inteira

O operador de resto da divisão inteira, representado pelo caractere "%", retorna o resto da divisão inteira de um número por outro. Veja o exemplo 1.16 a seguir.

```
a = 7
b = 3
c = a % b
print(c) # imprime 1
```

Exemplo de Código 1.16: Operador de resto da divisão inteira

1.3.7 Exponenciação

O operador de exponenciação, representado pelos caracteres "**", eleva um número a uma potência. Veja o exemplo 1.17 a seguir.

```
a = 2
b = 3
c = a ** b
print(c) # imprime 8
```

Exemplo de Código 1.17: Operador de multiplicação

1.3.8 Operações Trigonométricos

As funções trigonométricas são funções matemáticas que estudam a relação entre os ângulos de um triângulo retângulo e suas medidas. Em Python, você pode usar as funções trigonométricas da biblioteca nativa `math`. O código 1.18 mostra exemplos de uso das funções trigonométricas mais comuns.

```python
import math

# Exemplo 1: seno
angulo = 30
seno = math.sin(math.radians(angulo))
print("Seno de", angulo, "graus:", seno)

# Exemplo 2: cosseno
angulo = 60
cosseno = math.cos(math.radians(angulo))
print("Cosseno de", angulo, "graus:", cosseno)

# Exemplo 3: tangente
angulo = 45
tangente = math.tan(math.radians(angulo))
print("Tangente de", angulo, "graus:", tangente)

# Exemplo 4: arco seno
seno = 0.5
angulo = math.degrees(math.asin(seno))
print("Arco seno de", seno, ":", angulo, "graus")

# Exemplo 5: arco cosseno
cosseno = 0.5
angulo = math.degrees(math.acos(cosseno))
print("Arco cosseno de", cosseno, ":", angulo, "graus")

# Exemplo 6: arco tangente
tangente = 1.0
angulo = math.degrees(math.atan(tangente))
print("Arco tangente de", tangente, ":", angulo, "graus")
```

Exemplo de Código 1.18: Operador de resto da divisão inteira

Lembre-se de que, em todos os exemplos acima, a entrada para as funções trigonométricas deve ser em radianos, então é necessário converter o ângulo em graus para radianos usando a função math.radians() antes de chamar a função trigonométrica desejada.

1.3.9 Conversão de Tipos Primitivos

A conversão de tipos primitivos em Python é a operação de mudar o tipo de uma variável de sua forma original para outro tipo. Veja o exemplo 1.19 a seguir, que mostra alguns exemplos de conversão entre tipos primitivos.

Observe que, na conversão de `float` para `int`, o valor é truncado, ou seja, a parte decimal é

```python
# Exemplo 1: inteiro para string
numero = 42
string = str(numero)
print("Número:", numero, "tipo:", type(numero))
print("String:", string, "tipo:", type(string))

# Exemplo 2: string para inteiro
string = "42"
numero = int(string)
print("String:", string, "tipo:", type(string))
print("Número:", numero, "tipo:", type(numero))

# Exemplo 3: float para string
ponto_flutuante = 3.14
string = str(ponto_flutuante)
print("Ponto flutuante:", ponto_flutuante, "tipo:", type(ponto_flutuante))
print("String:", string, "tipo:", type(string))

# Exemplo 4: string para float
string = "3.14"
ponto_flutuante = float(string)
print("String:", string, "tipo:", type(string))
print("Ponto flutuante:", ponto_flutuante, "tipo:", type(ponto_flutuante))

# Exemplo 5: inteiro para float
numero = 42
ponto_flutuante = float(numero)
print("Número:", numero, "tipo:", type(numero))
print("Ponto flutuante:", ponto_flutuante, "tipo:", type(ponto_flutuante))

# Exemplo 6: float para inteiro
ponto_flutuante = 3.14
numero = int(ponto_flutuante)
print("Ponto flutuante:", ponto_flutuante, "tipo:", type(ponto_flutuante))
print("Número:", numero, "tipo:", type(numero))
```

Exemplo de Código 1.19: Operador de multiplicação

descartada. Além disso, na conversão de *string* para número, é necessário garantir que a *string* represente realmente um número válido, caso contrário uma exceção do tipo *ValueError* será gerada.

1.4 Entrada e Saída de Dados

A linguagem Python possui comandos nativos para entrada e saída de dados em programas console. Através desses comandos, você consegue permitir que o usuário interaja com seu programa, digitando valores para gerar resultados. Ainda, é possível criar saídas formatadas de diferentes maneiras. As subseções a seguir tratam desses tópicos.

1.4.1 Entrada de Dados

Em uma aplicação console em Python, a entrada de dados geralmente é feita usando a função `input()` e a saída de dados é feita usando a função `print()`. A função `input()` permite que o usuário forneça dados para o programa. Quando a função é chamada, ela exibe uma mensagem na tela e aguarda a entrada do usuário. O valor digitado pelo usuário é armazenado como uma *string*. O exemplo 1.20 ilustra o uso da função input().

```
1 nome = input("Digite seu nome: ")
2 print("Olá, " + nome + "!")
```

Exemplo de Código 1.20: Entrada de dados em console

1.4.2 Saída de Dados

A função print() é usada para exibir resultados ou mensagens na tela console. Ela pode exibir uma *string* ou o resultado de uma expressão. O exemplo 1.21 ilustra o uso da função print().

```
1 nome = "Lucas"
2 print("Olá, " + nome + "!") # imprime Olá, Lucas!
```

Exemplo de Código 1.21: Saída de dados em console

1.4.3 Saída Formatada

Para imprimir uma saída formatada em Python, você pode usar o método `format()` ou o operador de formatação de *string* %. O método `format()` é uma forma fácil de formatar strings e incorporar valores de variáveis nelas. O exemplo 1.22 mostra como usá-lo.

```
1 nome = "Lucas"
2 idade = 30
3 print("Meu nome é {} e tenho {} anos".format(nome, idade))
4 # imprime Meu nome é Lucas e tenho 30 anos
```

Exemplo de Código 1.22: O método format()

O operador de formatação de *string* % é uma forma mais antiga de formatar strings. O exemplo 1.23 mostra como usá-lo.

```
nome = "Lucas"
idade = 30
print("Meu nome é %s e tenho %d anos" % (nome, idade))
# imprime Meu nome é Lucas e tenho 30 anos
```

Exemplo de Código 1.23: O operador de formatação %

O operador "`f`" é uma forma mais recente e conveniente de formatar strings em Python. Ele permite que você insira valores de variáveis diretamente na *string*, usando chaves e o prefixo "`f`". O exemplo 1.24 mostra como usá-lo.

```
nome = "Lucas"
idade = 30
print(f"Meu nome é {nome} e tenho {idade} anos")
# imprime Meu nome é Lucas e tenho 30 anos
```

Exemplo de Código 1.24: O operador de formatação "f"

O operador "`f`" é uma forma mais legível e concisa de formatar strings em comparação com outros métodos, e é amplamente utilizado na comunidade Python. Ele também permite a execução de expressões dentro das chaves e a formatação avançada de números, entre outras coisas.

1.5 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo. É importante que, além de resolver o problema, você faça uso dos recursos de formatação de strings para apresentar uma saída textual bem apresentável ao usuário.

Exercício 1.1 Armazene seu nome e idade em variáveis separadas e imprima uma saída formatada com elas. ■

Exercício 1.2 Armazene dois números em variáveis e imprima a soma, subtração, multiplicação e divisão deles. ■

Exercício 1.3 Peça ao usuário para digitar três números, armazene-os em variáveis e imprima a média aritmética dos três números. ■

Exercício 1.4 Peça ao usuário para digitar seu peso e altura, calcule o índice de massa corporal (IMC) e imprima o resultado. ■

Exercício 1.5 Peça ao usuário para digitar três números e imprima a soma deles. ■

Exercício 1.6 Peça ao usuário para digitar um número e calcule o seu dobro. ■

Exercício 1.7 Peça ao usuário para digitar o nome, o preço de custo, o preço de venda e a quantidade em estoque de determinado produto e mostre o lucro que esse estoque pode gerar se todos os produtos forem vendidos. ■

Exercício 1.8 Peça ao usuário para digitar um número e calcule a raiz quadrada desse número. ■

Exercício 1.9 Peça ao usuário para digitar um número e calcule o seno, cosseno e tangente desse número. ■

Exercício 1.10 Peça ao usuário para digitar dois números e calcule a potência do primeiro número elevado ao segundo. ■

Exercício 1.11 Peça ao usuário para digitar três números e calcule a fórmula de Bhaskara para esses números. ■

Exercício 1.12 Peça ao usuário para digitar o raio de um círculo e calcule a área e o comprimento do círculo. ■

Exercício 1.13 Peça ao usuário para digitar as dimensões de um retângulo (largura e altura) e calcule a área e o perímetro desse retângulo. ■

Exercício 1.14 Peça ao usuário para digitar a base e a altura de um triângulo e calcule a área desse triângulo. ■

Exercício 1.15 Peça ao usuário para digitar a distância e a velocidade inicial de um objeto em queda livre e calcule o tempo que leva para atingir o solo, desconsiderando a resistência do ar. ■

Exercício 1.16 Peça ao usuário para digitar o valor inicial de um investimento, a taxa de juros e o número de anos e calcule o valor final do investimento. ■

Exercício 1.17 Peça ao usuário para digitar o preço de uma mercadoria, o desconto e o imposto e calcule o preço final da mercadoria. ■

Exercício 1.18 Peça ao usuário para digitar a massa e a aceleração de um objeto e calcule a força resultante. ■

Exercício 1.19 Peça ao usuário para digitar a velocidade final, a velocidade inicial e o tempo e calcule a aceleração. ■

Exercício 1.20 Peça ao usuário para digitar o valor da medida de um ângulo em radianos e calcule o valor desse ângulo em graus. ■

Exercício 1.21 Peça ao usuário para digitar o comprimento de dois lados de um triângulo retângulo e calcule o comprimento da hipotenusa. ■

Exercício 1.22 Peça ao usuário para digitar a distância percorrida por um objeto e o tempo gasto e calcule a velocidade média do objeto. ■

Exercício 1.23 Peça ao usuário para digitar a distância percorrida, o tempo gasto e aceleração e calcule a velocidade inicial e final do objeto. ■

Exercício 1.24 Calcular o perímetro de um círculo dado o seu raio como entrada. ■

Exercício 1.25 Calcular o volume de uma esfera dado o seu raio como entrada. ■

Exercício 1.26 Calcular a área de um triângulo retângulo dado as medidas dos seus catetos como entrada. ■

Exercício 1.27 Receber o nome, o salário e o valor do imposto de uma pessoa como entrada e imprimir o salário líquido. ■

Exercício 1.28 Crie uma lista com os nomes de 5 membros da sua família. ■

Exercício 1.29 Crie um dicionário com os nomes, idades e cor dos olhos de 5 (cinco) membros da sua família. ■

Exercício 1.30 Crie dois conjuntos e combine-os usando as operações de união, interseção e diferença, apresentando os resultados de cada operação. ■

1.6 Considerações Sobre o Capítulo

Este capítulo apresentou os fundamentos da programação na linguagem Python. Foi mostrado como criar variáveis de diferentes tipos, como utilizar os operadores matemáticos básicos e como usar os comandos de entrada e saída de dados para se criar programas mais interativos. Por fim, foram propostos diversos exercícios para você colocar em prática aquilo que aprendeu ao longo do capítulo. No próximo capítulo, você verá como executar códigos baseados em condições através das estruturas condicionais.

2. Estruturas Condicionais

As estruturas condicionais são parte fundamental da programação, pois permitem que o programa tome decisões e execute ações diferentes de acordo com as condições especificadas. Isso é importante porque o programa pode se adaptar a diferentes situações e entrada de dados, tornando-se mais flexível e capaz de lidar com problemas complexos.

Por exemplo, imagine que você esteja escrevendo um programa para calcular o salário de um funcionário. Sem a utilização de estruturas condicionais, o programa sempre calcularia o salário de acordo com as mesmas regras, independentemente do funcionário em questão. No entanto, se você utilizar estruturas condicionais, poderá incluir regras específicas para cada funcionário, levando em consideração fatores como horas extras, férias etc.

De maneira geral, as estruturas condicionais tornam os programas mais inteligentes e capazes de lidar com situações variadas, ajudando a garantir que as decisões tomadas pelo programa sejam as mais adequadas para cada caso específico.

2.1 Sintaxe Básica

Seguindo essa linha de pensamento, as estruturas condicionais em Python permitem que você execute ações diferentes de acordo com a verificação de determinadas condições. A sintaxe de uso de uma estrutura condicional básica em Python é apresentada no exemplo 2.2.

Vale ressaltar que os blocos `elif` e `else` não são de uso obrigatório. Portanto, uma estrutura condicional simples poderia ter somente o bloco correspondente ao `if`, que só seria executado se a condição fosse verdadeira. O código 2.2 ilustra essa situação.

```
if condição1:
    # Execute alguma ação se a condição 1 for verdadeira
elif condição2:
    # Execute alguma ação se a condição 1 for falsa a condição 2 for verdadeira
else:
    # Execute alguma outra ação se a condição for falsa
```

Exemplo de Código 2.1: A estrutura condicional `if`

```
# Exemplo 1: verificação de idade
idade = int(input("Digite sua idade: "))
if idade >= 18:
    print("Você é maior de idade.")

# Exemplo 2: verificação de número par
numero = int(input("Digite um número: "))
if numero % 2 == 0:
    print("O número é par.")

#Exemplo 3: verificação de nota
nota = float(input("Digite sua nota: "))
if nota >= 7:
    print("Você foi aprovado.")
```

Exemplo de Código 2.2: Estruturas condicionais simples

2.1.1 Estrutura Condicional Composta

A estrutura condicional composta é uma estrutura de controle de fluxo que permite a execução de diferentes trechos de código com base em diferentes condições. Ela é composta pelo comando `if`, `elif` (abreviação de `else if`) e `else`. A sintaxe básica de uma estrutura condicional composta é mostrada nos exemplos do código 2.3.

Em síntese, quando se tem mais de uma condição a ser avaliada, como no exemplo do código 2.3, você pode utilizar a estrutura `elif` (uma ou várias) para verificar cada uma das condições, uma a uma. Se a primeira condição não for verdadeira, o programa passa para a próxima condição verificada pelo `elif`, e assim por diante, até que uma das condições seja verificada como verdadeira ou até que seja atingido o `else` (caso nenhuma das condições seja verdadeira).

2.2 Combinação de Condições com Operadores Lógicos

É possível combinar várias condições em uma única estrutura condicional. Existem várias maneiras de combinar condições, dependendo do que você deseja verificar. Esta seção abora

```python
# Exemplo 1: verificação de idade
idade = int(input("Digite sua idade: "))
if idade < 18:
    print("Você é menor de idade.")
elif idade >= 18 and idade < 60:
    print("Você é maior de idade.")
else:
    print("Você é idoso.")

# Exemplo 2: verificação de nota
nota = float(input("Digite sua nota: "))
if nota >= 9:
    print("Você foi aprovado com louvor.")
elif nota >= 7:
    print("Você foi aprovado.")
else:
    print("Você foi reprovado.")
```

Exemplo de Código 2.3: Exemplos de uso da estrutura condicional composta

os operadores lógicos usados para combinação de condições.

2.2.1 O Operador `AND`

O operador lógico AND é utilizado para combinar duas ou mais condições lógicas e verificar se todas elas são verdadeiras ao mesmo tempo. O operador AND é representado pela palavra reservada `and` na sintaxe do Python. O código 2.4 mostra alguns exemplos de uso do operador AND em Python.

2.2.2 O Operador `OR`

O operador lógico OR é utilizado para combinar duas ou mais condições lógicas e verificar se pelo menos uma delas é verdadeira. O operador OR é representado pela palavra reservada `or` na sintaxe do Python. O código 2.5 mostra alguns exemplos de uso do operador OR em Python.

Estes são alguns exemplos de como o operador lógico OR pode ser usado em Python. Em cada exemplo, o operador OR é usado para combinar duas ou mais condições lógicas e verificar se pelo menos uma delas é verdadeira, permitindo uma tomada de decisão mais avançada.

2.2.3 O Operador Lógico NOT

O operador lógico NOT em Python é usado para inverter o valor de uma condição lógica. Em outras palavras, se uma condição é verdadeira, o operador NOT fará com que ela se torne falsa

```python
#Exemplo 1: autenticação de usuário
usuario = input("Digite seu nome de usuário: ")
senha = input("Digite sua senha: ")
if usuario == "admin" and senha == "123":
    print("Você tem permissão de acesso.")
else:
    print("Você não tem permissão de acesso.")

#Exemplo 2: verificação de horário de trabalho
hora = int(input("Digite a hora atual: "))
dia = input("Digite o dia da semana: ")
if hora >= 9 and hora <= 18 and dia != "sábado" and dia != "domingo":
    print("Você está no horário de trabalho.")
else:
    print("Você não está no horário de trabalho.")
```

Exemplo de Código 2.4: Exemplos de uso do operador AND

```python
# Exemplo 1: verificação do horário de funcionamento
dia_da_semana = input("Digite o dia da semana: ")
hora = int(input("Digite a hora atual: "))
if dia_da_semana == "sábado" or dia_da_semana == "domingo" or hora < 9 or hora >= 17:
    print("Loja fechada.")
else:
    print("Loja aberta.")

# Exemplo 2: verificação de feriado
dia = int(input("Digite o dia: "))
mes = int(input("Digite o mês: "))
if dia == 1 and mes == 1 or dia == 25 and mes == 12:
    print("Hoje é feriado.")
else:
    print("Hoje não é feriado.")
```

Exemplo de Código 2.5: Exemplos de uso do operador OR

e vice-versa. O operador NOT é representado por `not` na sintaxe do Python. O código 2.6 mostra alguns exemplos de uso do operador NOT.

Estes são apenas alguns exemplos de como o operador NOT pode ser usado em Python. Em cada exemplo, o operador NOT é usado para inverter o valor de uma condição lógica, permitindo uma tomada de decisão mais avançada.

```python
# Exemplo 1: verificação de maioridade
idade = int(input("Digite sua idade: "))
if not idade >= 18:
    print("Você não tem mais de 18 anos.")
else:
    print("Você tem mais de 18 anos.")

# Exemplo 2: verificação de horário de trabalho
hora = int(input("Digite a hora atual: "))
if not (hora >= 9 and hora <= 18):
    print("Você não está no horário de trabalho.")
else:
    print("Você está no horário de trabalho.")
```

Exemplo de Código 2.6: Exemplos de uso do operador NOT

2.3 Comparação de Valores em Condições

Os operadores de comparação são utilizados para comparar valores e determinar se uma determinada condição é verdadeira ou falsa. Aqui estão alguns dos principais operadores de comparação em Python:

- == (*igual a*) : Verifica se os valores são iguais;
- != (*diferente de*) : Verifica se os valores são diferentes;
- < (*menor que*) : Verifica se o valor à esquerda é menor que o valor à direita;
- > (*maior que*) : Verifica se o valor à esquerda é maior que o valor à direita;
- <= (*menor ou igual a*) : Verifica se o valor à esquerda é menor ou igual ao valor à direita;
- >= (*maior ou igual a*) : Verifica se o valor à esquerda é maior ou igual ao valor à direita.

No código 2.7 estão alguns exemplos de como esses operadores podem ser usados.

Em geral, os operadores de comparação são uma parte fundamental da programação, pois permitem que o programa tome decisões baseadas nas condições especificadas, tornando-o mais flexível e capaz de lidar com situações variadas.

2.4 Estruturas Condicionais Aninhadas

As estruturas condicionais aninhadas são estruturas condicionais dentro de outras estruturas condicionais. Em outras palavras, você pode colocar uma estrutura condicional dentro de outra para verificar condições mais complexas. Isso pode ser útil quando você precisa verificar várias

```python
# Exemplo 1: Verificação de igualdade
a = 5
b = 5
if a == b:
    print("a é igual a b")
else:
    print("a é diferente de b")
# Saída: a é igual a b

# Exemplo 2: Verificação de desigualdade
a = 5
b = 3
if a != b:
    print("a é diferente de b")
else:
    print("a é igual a b")
# Saída: a é diferente de b

# Exemplo 3: Verificação de número par ou ímpar
numero = 4
if numero % 2 == 0:
    print(numero, "é par")
else:
    print(numero, "é ímpar")
# Saída: 4 é par
```

Exemplo de Código 2.7: Exemplos de uso dos operadores de comparação

condições em sequência ou quando precisa tomar decisões com base em várias condições diferentes. O código 2.8 mostra um exemplo de estrutura condicional aninhada.

```python
idade = 25
pais = "Brasil"
if idade >= 18:
    if pais == "Brasil":
        print("Você pode votar no Brasil.")
    else:
        print("Você pode votar em outro país.")
else:
    print("Você não pode votar.")
# Saída: Você pode votar no Brasil.
```

Exemplo de Código 2.8: Exemplo de estrutura condicional aninhada.

Nesse exemplo, a primeira estrutura condicional verifica se a idade é maior ou igual a 18. Se for, a segunda estrutura condicional verifica se o país é o Brasil. Se for, a mensagem “Você pode

votar no Brasil." é exibida. Se não for, a mensagem "Você pode votar em outro país." é exibida. Se a primeira condição não for verdadeira, a mensagem "Você não pode votar." é exibida.

É importante lembrar que as estruturas condicionais aninhadas podem tornar o código mais complexo e difícil de entender, então é importante usá-las com moderação e cuidado. Se o código estiver ficando muito complexo, pode ser uma boa ideia refatorá-lo em funções ou em classes para torná-lo mais legível e fácil de manter.

2.5 Operador de Atribuição Condicional Ternário

Existe um operador de atribuição ternário em Python. É uma forma concisa de escrever uma estrutura condicional que retorna um valor diretamente para uma atribuição de variável. O operador ternário é escrito da forma mostrada no código **??**.

```
valor_verdadeiro if condição else valor_falso
```

Exemplo de Código 2.9: O operador ternário de atribuição condicional

O código 2.10 mostra um exemplo de como o operador ternário pode ser usado.

```
idade = 25
status = "Maior de idade" if idade >= 18 else "Menor de idade"
print(status)
# Saída: Maior de idade
```

Exemplo de Código 2.10: Exemplo de uso do operador ternário

Nesse exemplo, o operador ternário verifica se a idade é maior ou igual a 18. Se for, a *string* "Maior de idade" é atribuída à variável `status`. Se não for, a *string* "Menor de idade" é atribuída à variável `status`.

O operador ternário é uma forma concisa e eficiente de escrever estruturas condicionais simples, mas é importante lembrar que ele pode tornar o código mais difícil de entender se for usado de forma excessiva ou em casos mais complexos. Em geral, é uma boa ideia usar o operador ternário com moderação e optar por estruturas condicionais mais explícitas quando o código se torna mais complexo.

2.6 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo. É importante que, além de resolver o problema, você tente fazer uso de todos os recursos estudados ao longo do capítulo.

Exercício 2.1 Escreva um programa que verifique se um número é par ou ímpar. ■

Exercício 2.2 Escreva um programa que verifique se um número digitado pelo usuário é positivo, negativo ou zero. ■

Exercício 2.3 Escreva um programa que verifique se uma letra é uma vogal ou consoante. ■

Exercício 2.4 Escreva um programa que verifique se um ano é bissexto ou não. ■

Exercício 2.5 Escreva um programa que verifique se uma pessoa é maior de idade ou não. ■

Exercício 2.6 Escreva um programa que verifique se uma *string* é um palíndromo ou não. ■

Exercício 2.7 Escreva um programa que verifique se uma *string* é um número inteiro ou não e mostre uma mensagem de acordo. ■

Exercício 2.8 Escreva um programa que verifique se uma *string* é um número flutuante ou não. ■

Exercício 2.9 Escreva um programa que verifique se uma *string* é uma data válida ou não. ■

Exercício 2.10 Escreva um programa que calcule a média de três números e exiba uma mensagem de “Aprovado” se a média for maior ou igual a 6, ou “Reprovado” caso contrário. Se a nota for 10, exiba também a mensagem “Parabéns”. ■

Exercício 2.11 Escreva um programa que verifique se uma temperatura está acima, abaixo ou dentro da faixa normal (36°C a 37°C). ■

Exercício 2.12 Escreva um programa que verifique se uma pessoa pode votar ou não (se é maior de 18 anos e se é brasileira). ■

Exercício 2.13 Escreva um programa que verifique se uma pessoa é elegível para aposentadoria (se é maior de 60 anos para mulheres e 65 anos para homens). ■

Exercício 2.14 Escreva um programa que verifique se um número é divisível por outro número ou não. ■

Exercício 2.15 Escreva um programa que verifique se uma *string* é uma data de nascimento digitada pelo usuário é válida (dd/mm/aaaa). ■

Exercício 2.16 Escreva um programa que pergunte ao usuário seu salário e exiba uma mensagem de "Alto salário" se o salário for maior do que R$10.000,00, ou "Baixo salário" caso contrário. ■

Exercício 2.17 Escreva um programa que pergunte ao usuário seu gênero (M para masculino, F para feminino) e exiba uma mensagem de "Gênero masculino" ou "Gênero feminino". ■

Exercício 2.18 Escreva um programa que pergunte ao usuário seu peso e altura e exiba uma mensagem de "Você está abaixo do peso" se o IMC (índice de massa corporal) for menor do que 18,5, "Você está com o peso normal" se o IMC estiver entre 18,5 e 24,9, "Você está com sobrepeso" se o IMC estiver entre 25 e 29,9, ou "Você está com obesidade" caso contrário. ■

Exercício 2.19 Escreva um programa que pergunte ao usuário sua idade e exiba uma mensagem de "Você é jovem" se a idade for menor do que 30, "Você é adulto" se a idade estiver entre 30 e 60, ou "Você é idoso" caso contrário. ■

Exercício 2.20 Escreva um programa que peça ao usuário para digitar 5 números inteiros. O programa deve exibir uma mensagem informando se todos os números digitados são pares ou se há pelo menos um número ímpar. ■

2.7 Considerações Sobre o Capítulo

Este capítulo apresentou os conceitos e formas de uso da estrutura condicional `if` na linguagem Python. Foi mostrado como usar a estrutura `if`, quais são os operadores lógicos, quais são os operadores de comparação, como usar estruturas `if` aninhadas, como usar o operador de atribuição condicional ternário e, concluindo, foram apresentados várias propostas de exercícios. No próximo capítulo, você verá como repetir a execução de um trecho de código usando as

estruturas condicionais.

3. Estruturas de Repetição

As estruturas de repetição são uma das ferramentas mais importantes na programação, pois permitem que o código seja executado várias vezes, o que é útil em muitos casos. Elas são utilizadas para repetir uma sequência de instruções até que uma determinada condição seja atendida. Algumas das características das estruturas de repetição incluem:

- Controle de repetição: As estruturas de repetição permitem que o programador controle quantas vezes o código será executado, seja por meio de uma contagem fixa ou através de uma condição que deve ser atendida;
- Flexibilidade: As estruturas de repetição são muito flexíveis e permitem que o programador escolha o número de vezes que o código será executado, ou até mesmo deixar que o código seja executado indefinidamente;
- Automatização de tarefas repetitivas: As estruturas de repetição permitem que tarefas repetitivas sejam automatizadas, economizando tempo e esforço para o programador;
- Processamento de dados em massa: As estruturas de repetição permitem processar grandes quantidades de dados de uma só vez, o que é útil em aplicações como processamento de dados financeiros, análise de dados de saúde, etc.

Em síntese, as estruturas de repetição são importantes para a programação porque permitem automatizar tarefas repetitivas e a processar grandes quantidades de dados, além de fornecer flexibilidade e controle de repetição para o programador.

3.1 Sintaxe Básica

Em Python, existem duas estruturas de repetição principais: o laço `for` e o laço `while`. O laço `for` é usado para repetir uma ação um número fixo de vezes. Por exemplo, você pode usar um

laço for para imprimir todos os elementos de uma lista, como mostra o código 3.1.

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

Exemplo de Código 3.1: Exemplos de uso do laço `for`

O laço `while` é usado para repetir uma ação enquanto uma determinada condição for verdadeira. Por exemplo, você pode usar um laço `while` para ler números digitados pelo usuário até que ele digite um número negativo, como mostra o código 3.2.

```
numero = int(input("Digite um número: "))
while numero >= 0:
    numero = int(input("Digite outro número: "))
print("Você digitou um número negativo.")
```

Exemplo de Código 3.2: Exemplos de uso do laço `while`

Estas são as duas principais estruturas de repetição em Python. Cada uma delas tem suas próprias vantagens e usos, e é importante conhecer ambas para escolher a mais adequada para a tarefa em questão. As seções seguintes apresentam maiores detalhes sobre cada uma dessas duas estruturas.

3.2 A Estrutura de Repetição *for*

Como introduzido na seção anterior, a estrutura de repetição `for` em Python é usada para repetir uma ação um número fixo de vezes. Ela funciona percorrendo uma sequência de elementos, como uma lista ou uma *strings*, e executando uma ação para cada elemento da sequência. A sintaxe geral da estrutura de repetição for é apresentada no código 3.3.

```
for elemento in sequencia:
    # código a ser executado para cada elemento da sequência
```

Exemplo de Código 3.3: Sintaxe geral da estrutura `for`

No código 3.3, `elemento` é uma variável que representa cada elemento da sequencia na iteração atual e `sequencia` é uma sequência de elementos, como uma lista, *string*, *range*, entre outros. O código 3.4 mostra exemplos mais específicos de uso do `for`.

Estes são apenas alguns exemplos de como a estrutura de repetição `for` pode ser usada em Python. Em cada exemplo, a estrutura `for` é usada para repetir uma ação para cada elemento

```python
# Exemplo 1: impressão dos elementos de uma lista
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)

# Exemplo 2: soma dos elementos de uma lista
numeros = [1, 2, 3, 4, 5]
soma = 0
for numero in numeros:
    soma += numero
print("A soma dos números é", soma)

# Exemplo 3: impressão de caracteres de uma string
nome = "João"
for letra in nome:
    print(letra)

# Exemplo 4: cálculo de fatorial
numero = 5
fatorial = 1
for i in range(1, numero + 1):
    fatorial *= i
print("O fatorial de", numero, "é", fatorial)
```

Exemplo de Código 3.4: Exemplos de uso da estrutura `for`

de uma sequência, permitindo a realização de tarefas mais avançadas e eficientes.

3.3 A Estrutura de Repetição *while*

A estrutura de repetição `while` em Python é usada para repetir uma ação enquanto uma determinada condição é verdadeira. A estrutura `while` funciona verificando a condição no início de cada iteração e, se a condição for verdadeira, a ação é executada. A sintaxe geral da estrutura de repetição `while` é mostrada na figura 3.5.

```python
while condicao:
    # código a ser executado enquanto a condição for verdadeira
```

Exemplo de Código 3.5: Sintaxe geral da estrutura `while`

No código 3.5, `condicao` é uma expressão lógica que é avaliada antes de cada iteração do laço. Se a condição for verdadeira, o código dentro do laço será executado, caso contrário, o laço será encerrado. É importante lembrar de incluir código dentro do laço para alterar a condição de forma que, eventualmente, ela se torne falsa, caso contrário, o laço se tornará infinito, travando

a execução do programa. O código 3.6 mostra exemplos mais específicos de uso da estrutura `while` em Python.

```
# Exemplo 1: leitura de números positivos
numero = int(input("Digite um número: "))
while numero >= 0:
    print("Você digitou o número", numero)
    numero = int(input("Digite outro número: "))
print("Você digitou um número negativo.")

# Exemplo 2: impressão de números pares
numero = 0
while numero <= 10:
    print(numero)
    numero += 2

# Exemplo 3: cálculo da média para vários números
soma = 0
contador = 0
media = 0
numero = float(input("Digite um número: "))
while numero >= 0:
    soma += numero
    contador += 1
    media = soma / contador
    numero = float(input("Digite outro número: "))
print("A média dos números é", media)
```

Exemplo de Código 3.6: Exemplos de uso da estrutura `while`

Estes são apenas alguns exemplos de como a estrutura de repetição `while` pode ser usada em Python. Em cada exemplo, a estrutura `while` é usada para repetir uma ação enquanto uma determinada condição é verdadeira, permitindo a realização de tarefas mais avançadas e eficientes.

3.4 Controle de Fluxo com *break* e *continue*

Os comandos break e continue são usados para controlar o fluxo de execução dentro de laços em Python. Eles permitem interromper ou pular iterações de um laço, respectivamente. O comando `break` é usado para interromper um laço prematuramente. Quando o comando `break` é executado dentro de um laço, o laço é imediatamente encerrado, independentemente da condição de repetição. O código 3.7 mostra um exemplo de uso do comando `break` que pede ao usuário que digite um número até que ele digite um valor negativo.

```
numero = int(input("Digite um número: "))
while True:
    if numero < 0:
        break
    print("Você digitou o número", numero)
    numero = int(input("Digite outro número: "))
print("Você digitou um número negativo.")
```

Exemplo de Código 3.7: Exemplo de uso do comando `break`

O outro comando de controle de fluxo é o `continue`, que é usado para pular uma iteração de um laço. Quando o comando continue é executado dentro de um laço, a iteração atual é imediatamente encerrada e a próxima iteração é iniciada. O código 3.8 mostra um exemplo de uso do comando `continue` que imprime os números ímpares de 0 a 9.

```
for numero in range(10):
    if numero % 2 == 0:
        continue
    print(numero)
```

Exemplo de Código 3.8: Exemplo de uso do comando `continue`

Estes são os conceitos básicos sobre como usar os comandos `break` e `continue` para controlar o fluxo de execução dentro de laços em Python. É importante lembrar que, embora estes comandos possam ser úteis em muitas situações, eles também podem tornar o código mais difícil de ler e manter, portanto, é importante usá-los com moderação.

3.5 Estruturas de Repetição Aninhadas

A utilização de laços dentro de outros laços é uma técnica importante na programação que permite realizar tarefas mais complexas. Isso é conhecido como aninhamento de laços. Quando você aninha laços, você pode repetir ações dentro de outras ações repetidas, permitindo a realização de tarefas mais complexas. A sintaxe geral para aninhar laços em Python é mostrada no código 3.9.

```
for elemento_externo in sequencia_externa:
    # código a ser executado para cada elemento da sequência externa
    for elemento_interno in sequencia_interna:
        # código a ser executado para cada elemento da sequência interna
```

Exemplo de Código 3.9: Sintaxe geral do uso de laços aninhados

No código 3.9, as variáveis `elemento_externo` e `elemento_interno` representam os elementos de `sequencia_externa` e `sequencia_interna` nas iterações atuais, respectivamente. O código 3.10 mostra exemplos de uso de laços aninhados em Python.

```
# Exemplo 1: impressão de tabuada
for i in range(1, 11):
    for j in range(1, 11):
        print(i, "x", j, "=", i * j)

# Exemplo 2: verificação de número primo
numero = 17
e_primo = True
for i in range(2, numero):
    if numero % i == 0:
        e_primo = False
        break
if e_primo:
    print(numero, "é primo.")
else:
    print(numero, "não é primo.")
```

Exemplo de Código 3.10: Exemplos de uso de laços aninhados

Estes são apenas alguns exemplos de como laços aninhados podem ser usados em Python para realizar tarefas mais complexas. É importante lembrar que, embora laços aninhados possam ser úteis em muitas situações, eles também podem tornar o código mais difícil de ler e manter, portanto, é importante usá-los com moderação.

3.6 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo. É importante que, além de resolver o problema, você tente fazer uso de todos os recursos estudados ao longo do capítulo.

Exercício 3.1 Imprima todos os números de 1 a 100. ■

Exercício 3.2 Imprima todos os números pares de 1 a 100. ■

Exercício 3.3 Imprima todos os números ímpares de 1 a 100. ■

Exercício 3.4 Soma todos os números de 1 a 100. ■

Exercício 3.5 Soma todos os números pares de 1 a 100. ■

Exercício 3.6 Soma todos os números ímpares de 1 a 100. ■

Exercício 3.7 Calcule a média dos números digitados pelo usuário. O usuário deve digitar números até digitar um número negativo. ■

Exercício 3.8 Calcule o fatorial de um número digitado pelo usuário. ■

Exercício 3.9 Imprima todos os números divisíveis por 3 ou 5 de 1 a 100. ■

Exercício 3.10 Imprima todos os números primos de 1 a 100. ■

Exercício 3.11 Soma todos os números divisíveis por 3 ou 5 de 1 a 100. ■

Exercício 3.12 Soma todos os números primos de 1 a 100. ■

Exercício 3.13 Imprima a tabuada de um número digitado pelo usuário. ■

Exercício 3.14 Verifique se um número digitado pelo usuário é primo. ■

Exercício 3.15 Verifique se um número digitado pelo usuário é perfeito. ■

Exercício 3.16 Encontre o maior e o menor número em uma lista de 10 números digitada pelo usuário. ■

Exercício 3.17 Encontre o segundo maior e o segundo menor número em uma lista de 10 números digitada pelo usuário. ■

Exercício 3.18 Imprimir a sequência de Fibonacci até o n-ésimo termo, onde n é digitado pelo usuário. ■

Exercício 3.19 Verificar se uma palavra digitada pelo usuário é um palíndromo. Se for, imprimir, ao final, “A palavra é um palíndromo”. Se não for, imprimir “A palavra não é um palíndromo”. ■

Exercício 3.20 Calcular o fatorial de todos os números de 1 a n, onde n é digitado pelo usuário. Imprima o resultado para cada número. ■

3.7 Considerações Sobre o Capítulo

Este capítulo apresentou os conceitos e formas de uso das estruturas de repetição `for` e `while` na linguagem Python. Foi mostrado como usar cada uma das estruturas, bem como os comandos de controle de fluxo `break` e `continue` e as estruturas de repetição aninhadas, concluindo, com uma série de exercícios propostos. No próximo capítulo, você vera como manipular números, textos e listas de forma mais avançada.

4. Números, Textos e Coleções

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5. Funcões

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6. Tratamento de Exceções

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

7. Arquivos e Módulos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

8. Classes e Objetos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

9. Manipulação de Bancos de Dados

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

10. Manipulação de Bancos de Dados NoSQL

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Bibliografia

Artigos

Livros