TÓPICO 02: REVISÃO DE PYTHON

Estrutura de Dados - Professor Ramon Venson - SATC 2026.2

Declaração de Variáveis

O Python não possui tipos de dados explícitos, ou seja, não é necessário declarar o tipo de uma variável antes de utilizá-la. O tipo da variável é inferido automaticamente pelo interpretador com base no valor atribuído a ela.

```
# Exemplo de declaração de variáveis em Python
idade = 25 # inteiro
nome = "Estrutura de Dados"  # string
altura = 1.75  # float
esta_estudando = True  # booleano
```

```python
# Imprime a idade
idade = 25
print("Idade:", idade)

# Imprime o nome e idade
nome = "Estrutura de Dados"
print(nome + " tem " + str(idade) + " anos.")

# Imprime o resultado da soma
print("Soma:", 10 + 5)
```

Imprimindo Valores

Para imprimir valores no console, utilizamos a função `print()`. Podemos imprimir variáveis, strings e até mesmo realizar operações matemáticas dentro do `print()`.

> A "soma" de dois valores do tipo string resulta na concatenação das strings.

Lendo Valores do Usuário

Além de atribuir valores diretamente às variáveis, podemos ler valores do usuário utilizando a função `input()`. O valor lido é sempre do tipo string, então é necessário convertê-lo para o tipo desejado, se necessário.

```
# Lendo valores do usuário
nome = input("Digite seu nome: ")

# Convertendo para inteiro
idade = int(input("Digite sua idade: "))

# Imprimindo os valores lidos
print("Nome:", nome, "Idade:", idade)
```

Operadores Aritiméticos

Operadores aritméticos são utilizados para realizar operações matemáticas básicas:

Operador	Descrição	Exemplo
`+`	Adição	`5 + 3` resulta em `8`
`-`	Subtração	`5 - 3` resulta em `2`
`*`	Multiplicação	`5 * 3` resulta em `15`
`/`	Divisão	`5 / 3` resulta em `1.666...`

Outros operadores aritiméticos

Operador	Descrição	Exemplo
`%`	Módulo	`5 % 3` resulta em `2`
`//`	Divisão inteira	`5 // 3` resulta em `1`
`**`	Exponenciação	`5 ** 3` resulta em `125`

Operadores de Comparação

Os operadores de comparação comparam valores e retornam um valor booleano:

Operador	Descrição	Exemplo
`==`	Igualdade	`5 == 5` resulta em `True`
`!=`	Desigualdade	`5 != 3` resulta em `True`
`>`	Maior que	`5 > 3` resulta em `True`
`>=`	Maior ou igual a	`5 >= 5` resulta em `True`
`<`	Menor que	`3 < 5` resulta em `True`
`<=`	Menor ou igual a	`3 <= 5` resulta em `True`

Operadores Lógicos

Os operadores lógicos são utilizados para combinar expressões booleanas:

Operador	Descrição	Exemplo
`and`	E lógico	`True and False` resulta em `False`
`or`	OU lógico	`True or False` resulta em `True`
`not`	Negação lógica	`not True` resulta em `False`

Estruturas de Decisão

Estruturas de Decisão permitem que o programa tome decisões com base em condições. Existem quatro tipos de estruturas de decisão principais em Python: `if`, `elif`, `else` e `match`.

`if/elif/else`

O `if` é utilizado para executar um bloco de código se uma condição for verdadeira. O `elif` (else if) é utilizado para verificar múltiplas condições, e o `else` é utilizado para executar um bloco de código caso todas as condições anteriores sejam falsas.

```
idade = 18
if idade < 18:
    print("Menor de idade")
elif idade == 18:
    print("Tem 18 anos")
else:
    print("Maior de idade")
```

`match`

O `match` é uma estrutura de decisão que permite comparar um valor com diferentes padrões. É semelhante a uma estrutura de `switch/case` encontrada em outras linguagens de programação.

```
idade = 18
match idade:
    case 0:
        print("Recém nascido")
    case 18:
        print("Tem 18 anos")
    case _:
        print("Outra idade")
```

> O `case _` representa qualquer valor que não corresponda aos casos anteriores.

Estruturas de Repetição

Estruturas de repetição permitem que um bloco de código seja executado várias vezes. Existem duas estruturas de repetição principais em Python: `for` e `while`.

`for`

O `for` é utilizado para iterar sobre uma sequência (como uma lista, tupla, dicionário, conjunto ou string) ou contador:

```
# Iterando sobre uma lista
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)

# Iterando sobre um intervalo de números
for i in range(5):
    print(i)

# Iterando com um contador
for i in range(5):
    print(i)
```

`while`

O `while` é utilizado é geralmente utilizado quando não sabemos o número exato de iterações que serão necessárias. Ele continua executando o bloco de código enquanto a condição for verdadeira.

```
numero_sorteado = 8
numero_usuario = 0

while numero_usuario != numero_sorteado:
    numero_usuario = int(input("Digite um número entre 1 e 10: "))

print("Parabéns! Você acertou o número sorteado.")
```

Funções

Funções são blocos de código reutilizáveis que realizam uma tarefa específica. Elas podem receber parâmetros e retornar valores.

```
# Definindo uma função
def saudacao(nome):
    print(f"Olá, {nome}!")

# Chamando a função
saudacao("Fulano")  # Imprime "Olá, Fulano"
```

Retorno

Um valor pode ser retornado de uma função utilizando a palavra-chave `return`. Se nenhuma instrução `return` for especificada, a função retornará `None`.

```
# Função com retorno
def soma(a, b):
    return a + b

# Chamando a função e armazenando o resultado
resultado = soma(5, 3)
print(resultado)  # Imprime 8
```

Retorno múltiplo

Uma função pode retornar múltiplos valores utilizando uma tupla. Os valores retornados podem ser desembrulhados em variáveis separadas.

```
# Função com retorno múltiplo
def operacoes(a, b):
    soma = a + b
    subtracao = a - b
    return soma, subtracao

# Chamando a função e desembrulhando os valores retornados
resultado_soma, resultado_subtracao = operacoes(5, 3)
print("Soma:", resultado_soma)  # Imprime 8

---

### Parâmetros e Argumentos

Parâmetros são valores que podem ser passados para uma função quando ela é chamada. Eles permitem que a função opere com diferentes valores de entrada.

```python
# Função com parâmetros
def saudacao(nome, idade):
    print(f"Olá, {nome}! Você tem {idade} anos.")

# Chamando a função com diferentes argumentos
saudacao("Fulano", 25)  # Imprime "Olá, Fulano! Você tem 25 anos."
saudacao("Ciclano", 30)  # Imprime "Olá, Ciclano! Você tem 30 anos."
```

> O valor que é passado para a função é chamado de **argumento**, enquanto o valor que é definido na função é chamado de **parâmetro**.
```

Vetores

Vetores são estruturas de dados que armazenam uma coleção de elementos do mesmo tipo. Em Python, os vetores são representados por listas.

```
# Criando um vetor (lista) de números
numeros = [1, 2, 3, 4, 5]

# Acessando elementos do vetor
print(numeros[0])  # Imprime 1
print(numeros[2])  # Imprime 3

# Modificando elementos do vetor
numeros[0] = 10
print(numeros)  # Imprime [10, 2, 3, 4, 5]
```

Funções Úteis

Função	Descrição	Exemplo
`len()`	Retorna o tamanho de uma lista	`len([1, 2, 3])` resulta em `3`
`max()`	Retorna o maior valor de uma lista	`max([1, 2, 3])` resulta em `3`
`min()`	Retorna o menor valor de uma lista	`min([1, 2, 3])` resulta em `1`
`sum()`	Retorna a soma dos valores de uma lista	`sum([1, 2, 3])` resulta em `6`
`sorted()`	Retorna uma lista ordenada	`sorted([3, 1, 2])` resulta em `[1, 2, 3]`