Revisão de Python

Professora Carolina Ribeiro Xavier
CCOMP - UFSJ

Objetivos da Aula

Nesta aula de revisão de Python, abordaremos:

- Tipos de dados fundamentais
- Funções e argumentos opcionais
- Estruturas de dados: Listas e Dicionários
- Cópia por valor e por referência
- Manipulação de arquivos
- Classes e Objetos

Tipos de Dados Fundamentais

Em Python, os principais tipos de dados são:

- `int`: números inteiros
- `float`: números de ponto flutuante
- `str`: cadeias de caracteres (strings)
- `bool`: valores booleanos (`True` ou `False`)

Exemplo de Código: Tipos de Dados

Veja como podemos declarar variáveis com diferentes tipos de dados:

```
x = 10       # int
y = 3.14     # float
nome = "Python"  # str
ativo = True # bool
```

Operações Básicas

Podemos realizar operações matemáticas e manipulações com tipos de dados básicos:

- `int` e `float`: operações aritméticas (`+`, `-`, `*`, `/`)
- `str`: concatenação (`+`), repetição (`*`)
- `bool`: operadores lógicos (`and`, `or`, `not`)

```
a = 10 + 5        # Soma
texto = "Olá" + " Mundo"  # Concatenação de strings
ativo = True and False    # Operador lógico
```

Funções em Python

As funções permitem a reutilização e a organização do código.
Usamos a palavra-chave `def` para definir uma função:

```
def saudacao(nome):
    print(f"Olá, {nome}!")
```

Chamando Funções

Depois de definir uma função, podemos chamá-la em qualquer lugar do nosso código:

```
saudacao("Ana")    # Saída: Olá, Ana!
saudacao("Carlos")  # Saída: Olá, Carlos!
```

Argumentos Opcionais

Funções podem ter argumentos opcionais com valores padrão:

```
def saudacao(nome, saudacao="Olá"):
    print(f"{saudacao}, {nome}!")

saudacao("Ana")                 # Usa valor padrão
saudacao("Carlos", "Oi")        # Sobrescreve valor padrã
```

Retorno de Valores

Funções podem retornar valores usando a palavra-chave `return`:

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

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

Exercícios sobre Funções

Pratique criando suas próprias funções:

- Crie uma função que recebe uma lista de números e retorna a soma deles.
- Crie uma função que verifica se um número é par ou ímpar.

Listas

Listas são coleções ordenadas e mutáveis em Python. São definidas usando colchetes:

```
lista = [1, 2, 3, "Python", True]
print(lista[0])  # Acessa o primeiro elemento
lista.append(4)  # Adiciona um elemento no final
```

Manipulação de Listas

Podemos realizar várias operações em listas:

- Adicionar elementos: `append()`
- Remover elementos: `remove()`, `pop()`
- Acessar por índice: `lista[índice]`
- Fatiamento: `lista[início:fim]`

Exemplo de Fatiamento de Lista

```
lista = [1, 2, 3, 4, 5]
sublista = lista[1:3]  # Retorna [2, 3]
print(sublista)
```

Iterando sobre Listas

Podemos usar loops para iterar sobre listas:

```
lista = [1, 2, 3, 4, 5]
for elemento in lista:
    print(elemento)
```

Exercícios sobre Listas

- Crie uma lista de números e imprima apenas os números pares.
- Implemente uma função que receba uma lista e retorne o maior valor.

Dicionários

Dicionários são coleções não ordenadas de pares chave-valor. São definidos usando chaves:

```
dicionario = {"nome": "Ana", "idade": 25}
print(dicionario["nome"])  # Acessa o valor da chave "n
```

Manipulação de Dicionários

Operações comuns em dicionários:

- Adicionar ou atualizar valores: `dicionario[chave] = valor`
- Remover elementos: `del dicionario[chave]`
- Iterar sobre chaves ou valores

Exemplo de Iteração sobre Dicionários

```
dicionario = {"nome": "Ana", "idade": 25}
for chave, valor in dicionario.items():
    print(f"{chave}: {valor}")
```

Exercícios sobre Dicionários

- Crie um dicionário com informações de um aluno (nome, idade, notas).
- Implemente uma função que receba um dicionário e retorne a média das notas.

Cópia por Valor e por Referência

Em Python, objetos mutáveis (listas, dicionários) são passados por referência, enquanto objetos imutáveis (int, float, str) são passados por valor.

Cópia por Valor

Para tipos imutáveis como `int`, `float`, `str`, uma nova cópia do valor é criada quando atribuída a outra variável:

```
a = 10
b = a   # Cópia por valor
b = 20
print(a)  # Saída: 10
```

O valor de a permanece o mesmo, pois é uma cópia.

Cópia por Referência

Para objetos mutáveis como listas e dicionários, o valor não é copiado, mas sim referenciado:

```
lista1 = [1, 2, 3]
lista2 = lista1  # Cópia por referência
lista2.append(4)
print(lista1)    # Saída: [1, 2, 3, 4]
```

A modificação feita em `lista2` também afeta `lista1`.

Como Fazer Cópia Profunda

Para criar cópias independentes de objetos mutáveis, usamos o módulo `copy` com a função `deepcopy()`:

```
import copy

lista1 = [1, 2, 3]
lista2 = copy.deepcopy(lista1)
lista2.append(4)
print(lista1)  # Saída: [1, 2, 3]
```

Neste caso, `lista1` não foi modificada.

Exercícios sobre Cópias

- Crie um dicionário e faça uma cópia superficial usando `copy()`.
- Modifique a cópia e observe se o original é alterado.

Manipulação de Arquivos em Python

Python permite trabalhar com arquivos de maneira simples. Para abrir um arquivo, usamos a função `open()`.

Leitura de Arquivos

Para ler o conteúdo de um arquivo:

```
with open('arquivo.txt', 'r') as f:
    conteudo = f.read()
    print(conteudo)
```

O arquivo é aberto em modo de leitura (`'r'`). O `with` garante que o arquivo seja fechado após a leitura.

Escrita em Arquivos

Para escrever em um arquivo:

```
with open('arquivo.txt', 'w') as f:
    f.write('Hello, World!')
```

O arquivo é aberto em modo de escrita (`'w'`). Se o arquivo não existir, ele será criado.

Modos de Abertura de Arquivos

Diferentes modos para abrir arquivos:

- `'r'`: leitura
- `'w'`: escrita (sobrescreve o arquivo)
- `'a'`: escrita (anexa ao final do arquivo)
- `'b'`: modo binário

Exercícios sobre Manipulação de Arquivos

- Abra um arquivo de texto e conte o número de linhas.
- Crie um programa que lê um arquivo e grava o conteúdo em outro arquivo.

Classes e Objetos

Python suporta Programação Orientada a Objetos (POO). Classes definem a estrutura de objetos, que são instâncias de classes.

Definição de Classe

Uma classe é definida usando a palavra-chave `class`:

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

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

Criando Objetos

Para criar um objeto de uma classe, chamamos a classe como se fosse uma função:

```
p1 = Pessoa("Ana", 30)
p1.saudacao()  # Saída: Olá, meu nome \\
é Ana e tenho 30 anos.
```

O método `__init__()` é o construtor da classe e é chamado automaticamente.

Herança

Uma classe pode herdar atributos e métodos de outra classe:

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

    def saudacao(self):
        super().saudacao()
        print(f"Eu estudo {self.curso}.")
```

Exemplo de Herança

```
est = Estudante("Carlos", 22, "Engenharia")
est.saudacao()
```

Saída: `Olá, meu nome é Carlos e tenho 22 anos.`
`Eu estudo Engenharia.`

Exercícios sobre Classes

- Crie uma classe `Carro` com atributos como `modelo`, `ano` e métodos como `acelerar()`.
- Crie uma classe derivada de `Carro`, chamada `CarroEletrico`, que adicione um atributo `bateria`.

Funções Lambda

Funções lambda são funções anônimas definidas em uma única linha. Elas são úteis para operações simples.

```
soma = lambda a, b: a + b
print(soma(5, 3))  # Saída: 8
```

Funções de Alta Ordem em Python

Python possui várias funções de alta ordem embutidas, como:

- `map()`: Aplica uma função a cada item de um iterável.
- `filter()`: Filtra itens de um iterável com base em uma função que retorna `True` ou `False`.
- `reduce()` (no módulo `functools`): Aplica uma função acumuladora a uma sequência, reduzindo-a a um valor único.

Exemplo com `filter()`

Exemplo com a função `filter()`:

```
numeros = [1, 2, 3, 4, 5, 6]
pares = filter(lambda x: x % 2 == 0, numeros)
print(list(pares))  # Saída: [2, 4, 6]
```

Aqui, a função `filter()` usa uma função `lambda` (anônima) para selecionar apenas os números pares da lista.
As funções de alta ordem permitem maior flexibilidade e reuso de código ao manipular funções de forma dinâmica.

Exemplo: Uso de Lambda em Funções de Alta Ordem

Funções como `map()`, `filter()` e `sorted()` aceitam funções como argumento, tornando-as ideais para lambdas:

```
numeros = [1, 2, 3, 4]
dobrados = list(map(lambda x: x * 2, numeros))
print(dobrados)  # Saída: [2, 4, 6, 8]
```

Decoradores

Decoradores são usados para modificar o comportamento de funções ou métodos. Um decorador é uma função que recebe outra função como argumento e retorna uma nova função.

```
def meu_decorador(func):
    def wrapper():
        print("Executando algo antes da função")
        func()
        print("Executando algo depois da função")
    return wrapper

@meu_decorador
def diga_ola():
    print("Olá!")

diga_ola()
```

Exercícios sobre Lambda e Decoradores

- Crie uma função lambda que receba um número e retorne o triplo dele.
- Crie um decorador que registra o tempo de execução de uma função.

Revisão: Principais Conceitos

Vamos revisar os principais tópicos abordados:

- Tipos de dados
- Funções e argumentos opcionais
- Listas e dicionários
- Cópia por valor e referência
- Manipulação de arquivos
- Classes, objetos e herança
- Funções lambda e decoradores

Discussão de Casos Práticos

Aplicações práticas dos conceitos:

- Crie um programa que leia um arquivo de notas de alunos e calcule a média.
- Implemente um decorador que limite o número de execuções de uma função.

Próxima aula

Nesta aula de revisão, cobrimos uma ampla gama de tópicos importantes em Python. Agora é hora de praticar! Continue explorando e criando seus próprios projetos.
Na próxima aula veremos sobre o gerenciamento de ambientes e pacotes com conda.