Lógica de programação Python - Nível intermediário

Introdução

A lógica de programação é o alicerce para todo desenvolvimento de software. Ao dominar a lógica, você se torna capaz de resolver problemas, criar soluções e automatizar tarefas. Neste módulo intermediário, vamos aprofundar o uso do Python para aplicar estruturas de dados, funções, estruturas de controle e programação orientada a objetos (POO).

Python é uma linguagem de alto nível, com sintaxe simples e recursos poderosos. É amplamente utilizada em áreas como ciência de dados, automação, desenvolvimento web e inteligência artificial.

O que você vai aprender ?

- **Introdução à lógica e ao Python:** por que aprender e como a linguagem se destaca.
- **Estruturas de dados:** listas, dicionários e conjuntos.
- **Controle de fluxo:** laços `for`/`while` e condicionais.
- **Funções:** criação, parâmetros, retorno e escopo de variáveis.
- **Recursão:** conceito e exemplo com fatorial.
- **Orientação a objetos:** classes, métodos, atributos e encapsulamento.
- **Encerramento:** revisão dos aprendizados e próximos passos.

1. Estruturas de Dados

1.1 Listas

Listas são coleções ordenadas que podem armazenar vários tipos de dados. Você pode adicionar, remover e acessar elementos facilmente.

Exemplo:

```
frutas = ["maçã", "banana", "uva"]
frutas.append("laranja")  # Adiciona no fim
print(frutas[0])  # Saída: maçã
```

1.2 Dicionários

Dicionários armazenam dados em pares chave:valor, ótimos para representar informações organizadas.

```
aluno = {"nome": "Carlos", "idade": 20}
print(aluno["nome"])  # Saída: Carlos
```

1.3 Conjuntos

Conjuntos (sets) não permitem valores repetidos e não têm ordem definida.

```
numeros = {1, 2, 2, 3}
print(numeros)  # Saída: {1, 2, 3}
```

2. Controle de Fluxo

2.1 Laços (for e while)

Use for quando souber o número de repetições e while quando depender de uma condição.

for

for i in range(3): print(i)

while

n = 0 while n < 3:

print(n)

n += 1

2.2 Condicionais

Permitem executar diferentes blocos de código com base em condições.

nota = 7

```
if nota >= 7:
    print("Aprovado")
elif nota >= 5:
        print("Recuperação")
else:
    print("Reprovado")
```

3. Funções

Funções ajudam a organizar o código e evitar repetição. Podem receber parâmetros e retornar resultados.

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

print(saudacao("Maria"))
```

3.1 Parâmetros e Retorno

Você pode definir valores padrão e retornar qualquer tipo de dado.

```
def soma(a, b=0):
    return a + b

print(soma(5))      # 5
print(soma(5, 3))   # 8
```

3. 2 Escopo

O escopo define onde as variáveis existem. Variáveis criadas dentro da função só existem ali.

```
def exemplo():
    x = 10
    print(x)
```

```
exemplo()

# print(x)  # Isso daria erro
```

4. Recursão

Recursão é quando uma função chama a si mesma. Muito usada para problemas que se repetem de forma menor.

```
def fatorial(n):
    if n == 0:
        return 1
    return n * fatorial(n - 1)


print(fatorial(5))  # 120
```

Atenção: toda função recursiva precisa de um **caso base**, ou ela nunca vai parar.

5. Programação Orientada a Objetos (POO)

POO permite organizar o código em torno de objetos, facilitando o reaproveitamento e a manutenção.

5. 1 Classes e Objetos

class Pessoa: def **init**(self, nome): self.nome = nome

```
def falar(self):
    print(f"Olá, eu sou {self.nome}")


p1 = Pessoa("João") p1.falar() # Olá, eu sou João
```

5.2 Encapsulamento

É possível proteger dados sensíveis de uma classe usando dois underlines (__):

class Conta: def **init**(self): self.__saldo = 0

def depositar(self, valor):
    if valor > 0:
        self.__saldo += valor

def ver_saldo(self):
    return self.__saldo

Conclusão

Neste curso intermediário, vimos como a lógica de programação aliada ao Python pode ajudar a construir códigos mais claros, reutilizáveis e poderosos. Você aprendeu a trabalhar com estruturas de dados, controle de fluxo, funções, recursão e deu os primeiros passos na programação orientada a objetos.