Apostila de Lógica de Programação com Exemplos de Código

Sumário

---

1. Introdução à Lógica de Programação

A lógica de programação envolve passos lógicos para resolver problemas computacionais, sendo essencial para o desenvolvimento de software. O **algoritmo** é uma sequência ordenada de instruções, representada em **pseudocódigo** ou **fluxogramas**. Esses métodos ajudam a estruturar a solução de forma clara, sendo úteis antes mesmo da codificação em uma linguagem de programação.

Por exemplo, para calcular o dobro de um número, o pseudocódigo ficaria assim:

objectivec
Copiar código
```
INÍCIO
  LER número
  dobro ← número * 2
  IMPRIMIR dobro
FIM
```

Este pseudocódigo mostra o básico de um algoritmo em uma sequência lógica, permitindo identificar passos importantes antes de transformá-lo em código. Veja como esse pseudocódigo se traduz para Python:

python
Copiar código
```
# Código em Python para calcular o dobro de um número
numero = int(input("Digite um número: "))
dobro = numero * 2
print("O dobro do número é:", dobro)
```

Esse exemplo ilustra a ideia de algoritmos sequenciais em uma linguagem de programação. No decorrer dos tópicos, exploraremos formas de estruturar melhor esses algoritmos, além de técnicas para que os códigos respondam de acordo com diferentes condições.

---

2. Estruturas Sequenciais

As **estruturas sequenciais** executam comandos de forma linear, na ordem em que foram escritos. Este tipo de estrutura é ideal para problemas simples, como cálculos matemáticos e exibições de informações básicas. Como exemplo, vamos calcular a média de duas notas informadas pelo usuário.

No pseudocódigo, seria algo assim:

objectivec
Copiar código
```
INÍCIO
  LER nota1, nota2
  média ← (nota1 + nota2) / 2
  IMPRIMIR média
FIM
```

Em Python, o algoritmo ficaria assim:

python
Copiar código
```
# Código em Python para calcular a média de duas notas
nota1 = float(input("Digite a primeira nota: "))
nota2 = float(input("Digite a segunda nota: "))
media = (nota1 + nota2) / 2
print("A média é:", media)
```

Esse exemplo ilustra como a sequência de instruções leva a um cálculo simples. Esse conceito é básico, mas crucial para entender a ordem de execução e desenvolver algoritmos mais complexos posteriormente.

---

3. Estruturas Condicionais

As **estruturas condicionais** permitem que o programa tome decisões com base em condições específicas. Em programação, as instruções condicionais mais comuns são **if-else** e **elif**, que possibilitam ao programa escolher um caminho entre várias opções. Isso torna os programas interativos e adaptáveis a diferentes entradas de dados.

Um exemplo prático de estrutura condicional é a classificação de um número em par ou ímpar:

python
Copiar código
```
# Código em Python para verificar se um número é par ou ímpar
numero = int(input("Digite um número: "))
if numero % 2 == 0:
   print("O número é par.")
else:
   print("O número é ímpar.")
```

Neste código, usamos uma condicional **if** para verificar se o resto da divisão do número por 2 é zero (par) ou não (ímpar). Outro exemplo comum é o uso de várias condições com **elif**, como em um sistema de notas:

python
Copiar código
```
# Código para classificação de uma nota
nota = float(input("Digite a nota: "))
if nota >= 90:
   print("Conceito A")
```

```
elif nota >= 80:
   print("Conceito B")
elif nota >= 70:
   print("Conceito C")
else:
   print("Conceito D")
```

Esses exemplos mostram como as condicionais tornam o código flexível e permitem respostas personalizadas para diferentes situações.

---

4. Estruturas de Repetição

As **estruturas de repetição** permitem que um bloco de código seja executado várias vezes, o que é útil para lidar com grandes conjuntos de dados ou quando precisamos repetir uma ação até que uma condição seja atendida. As repetições mais comuns são **for** e **while**.

Um exemplo de **for** é exibir uma contagem de 1 a 5:

python
Copiar código
```
# Contagem de 1 a 5 usando o for
for i in range(1, 6):
   print(i)
```

Com o **while**, o programa pode repetir uma ação até que uma condição seja atendida, como pedir uma senha até que o usuário insira a correta:

python
Copiar código
```
# Exemplo de uso do while para validar senha
senha = ""
while senha != "12345":
   senha = input("Digite a senha: ")
print("Acesso permitido!")
```

Esse exemplo mostra como as estruturas de repetição tornam o código mais compacto e eficiente, permitindo que o programa realize ações repetitivas sem a necessidade de escrever cada comando individualmente.

---

5. Funções e Procedimentos

As **funções** e **procedimentos** são blocos de código que executam tarefas específicas e podem ser reutilizados, o que organiza o código e evita repetição. Uma função geralmente retorna um valor e permite que o programa realize cálculos ou operações de forma modular.

Por exemplo, uma função que calcula o quadrado de um número:

python
Copiar código
```
# Função para calcular o quadrado de um número
def quadrado(numero):
   return numero * numero

# Chamando a função
num = int(input("Digite um número: "))
```

```
print("O quadrado do número é:", quadrado(num))
```

Além de funções com retorno, podemos criar procedimentos que apenas executam uma tarefa sem retornar um valor, como exibir uma mensagem de boas-vindas:

```
python
Copiar código
# Procedimento para exibir uma mensagem
def boas_vindas():
    print("Bem-vindo ao sistema!")

# Chamando o procedimento
boas_vindas()
```

Esses exemplos mostram como as funções e procedimentos simplificam o código, permitindo reutilização e manutenção fácil, especialmente em projetos maiores onde há muitas partes repetitivas.

---

6. Arrays (Vetores e Matrizes)

Os **arrays** são estruturas que armazenam múltiplos valores em uma única variável, permitindo o armazenamento e acesso a grandes conjuntos de dados de maneira eficiente. Arrays unidimensionais, ou vetores, guardam uma sequência de valores, enquanto arrays bidimensionais, ou matrizes, organizam os dados em linhas e colunas.

Para armazenar as notas de um aluno em um array e calcular a média, o pseudocódigo seria assim:

```
python
Copiar código
# Exemplo de array unidimensional para armazenar notas
notas = [7.5, 8.0, 9.2]
media = sum(notas) / len(notas)
print("A média das notas é:", media)
```

Para matrizes, podemos armazenar dados como em uma tabela, por exemplo, uma matriz para armazenar a tabela de multiplicação de 1 a 3:

```
python
Copiar código
# Exemplo de matriz para tabela de multiplicação
tabela = [[i * j for j in range(1, 4)] for i in range(1, 4)]
for linha in tabela:
    print(linha)
```

Esses exemplos mostram como os arrays simplificam o manuseio de dados e organizam informações em estruturas ordenadas, tornando-se ferramentas valiosas para programadores.

Esses tópicos e exemplos fornecem uma introdução sólida para quem deseja aprender lógica de programação, abordando desde conceitos básicos até estruturas mais avançadas com uso prático de código.