

Nikael Enzo Horn Martins

**Algoritmos e Lógica de Programação**  
Definições, Lógica e Exemplos

Cuiabá  
2024

Nikael Enzo Horn Martins

**Algoritmos e Lógica de Programação**  
Definições, Lógica e Exemplos

Trabalho apresentado no Sesc Escola, curso Jogos Digitais, Senac

Orientador: Wanderson

Cuiabá  
2024

**SUMARIO:**

1. **ALGORITMOS ..................................................................................................................4**
   1. **Definição de algoritmos e sua importância na programação .................................4**
   2. **Lógica de programação e sua relação com algoritmos ............................................4**
2. **ESTRUTURAS DE CONTROLE ....................................................................................4**
   1. **Pesquisa sobre estruturas de controle .......................................................................4**
   2. **Exemplos de algoritmos que utilizam estruturas de controle .................................4**
3. **TIPOS DE DADOS E VARIAVEIS .................................................................................5**
   1. **Definição de tipos de dados básicos ....5**
      1. Inteiros
      2. Flutuante
      3. Booleanos
   2. **Exemplos de declaração e utilização de variáveis em algoritmos ...........................5**
4. **FUNÇÕES E MODULARIZAÇÃO .................................................................................5**
   1. **Pesquisa sobre o conceito de função na programação .............................................5**
   2. **A importância da modularização da escrita de algoritmos ....................................5**
   3. **Exemplos de funções simples e sua utilização em algoritmos mais complexos .....6**

**1.1** Um algoritmo é uma sequência finita e bem definida de instruções ou regras que são seguidas para resolver um problema ou executar uma tarefa específica. Os algoritmos são usados na programação para resolver uma variedade de problemas computacionais, desde a manipulação de dados até a tomada de decisões inteligentes em sistemas complexos. Eles são uma ferramenta fundamental para desenvolver softwares eficiente, escalável e funcional. Um exemplo seria o Alien do jogo Alien: Isolation, onde ele tem duas inteligências artificias que aprendem a lidar com o jogador.

**1.2** Lógica de programação é a organização coesa de uma sequência de instruções voltadas à resolução de um problema, ou à criação de um software ou aplicação. Ela e baseada na logica matemática e na teoria da computação. A lógica de programação permite que os programadores organizem suas ideias de forma coerente e desenvolvam algoritmos eficazes para resolver problemas, envolvendo a compreensão de conceitos como sequência, seleção e repetição.

**2.1** Estruturas de controle são fundamentais para o desenvolvimento de programas, pois permitem que os programadores controlem o fluxo de execução do código, tomando decisões e repetindo tarefas conforme necessário. Algumas estruturas de controle são;  
Sequência: A estrutura de controle de sequência refere-se à execução de instruções em ordem sequencial, sendo executadas em linha reta, uma após a outra, sem desvios ou condições especiais.Seleção (ou decisão): A estrutura de controle de seleção permite que o programa escolha entre dois ou mais caminhos diferentes, com base em uma condição ou conjunto de condições. Isso é feito usando instruções condicionais, como "if", "else" e "else if” verificando se uma condição seja verdadeira ou falsa e, em seguida, um específico ou pode verificar outra condição.Repetição (ou iteração): A estrutura de controle de repetição permite que o programa execute um bloco de código várias vezes, com base em uma condição ou conjunto de condições. Isso é útil quando você precisa executar uma tarefa repetidamente. As estruturas de repetição mais comuns são "for", "while" e "do-while".

**2.2** Uns exemplos seria o Arduino, com a linguagem C++, onde os códigos são dispostos em uma sequência, onde um sempre acontecera um após o outro, seleção, tendo os "if" e "else" para fazer verificações, e podendo conter repetições, onde ele repetir tudo que estiver em “void loop”

**3.1.1** Inteiros (int): Inteiros são números inteiros sem parte fracionária. Eles podem ser positivos, negativos ou zero. Em muitas linguagens de programação, os inteiros são representados usando um número finito de bits, o que determina o intervalo de valores que podem ser armazenados, "int" é usado para declarar variáveis que armazenam números inteiros. Por exemplo, em C, um int geralmente ocupa 4 bytes de memória, permitindo representar valores entre -2,147,483,648 e 2,147,483,647.

**3.1.2** Ponto Flutuante (float ou double): Ponto flutuante é um tipo de dado usado para representar números reais (números que podem ter uma parte fracionária). Eles são usados quando a precisão é necessária, como em cálculos matemáticos envolvendo números reais. Geralmente, existem dois tipos de ponto flutuante: float, que geralmente ocupa 4 bytes de memória e tem uma precisão limitada, e double, que geralmente ocupa 8 bytes e oferece uma precisão maior.

**3.1.3** Booleanos (bool): Booleanos representam valores lógicos, que podem ser verdadeiros (true) ou falsoS (false). Eles são usados em operações lógicas e de controle de fluxo para determinar o resultado de uma condição. Em muitas linguagens de programação, os booleanos ocupam um byte de memória, embora o tamanho exato possa variar.

**3.2** “numero\_inteiro = 10” é uma declaração que diz que toda vez que eu escrever “numero\_inteiro” ele irá ler como “10”, e para a variável “if numero\_inteiro = 10:” “print("verdadeiro.")” “else:” “print("falso.")”

**4.1** Uma função em programação é um bloco de código que executa uma tarefa específica ou calcula um valor, e pode ser reutilizado em diferentes partes de um programa. Ela permite que você agrupe um conjunto de instruções sob um único nome, tornando o código mais organizado, modular e fácil de entender. As funções também facilitam a manutenção do código, uma vez que você pode fazer alterações em uma função sem afetar outras partes do programa.

**4.2** Cada função ou módulo pode ser dividida e concentrar em uma única tarefa, o que facilita a compreensão do que está acontecendo em cada parte do algoritmo.  
Ao modularizar um algoritmo, você pode reutilizar funções ou módulos em diferentes partes do programa ou mesmo em projetos diferentes.  
Quando um algoritmo é dividido em módulos ou funções pequenas e coesas, qualquer alteração ou correção necessária pode ser feita em um local específico, sem afetar outras partes do código.  
Módulos ou funções pequenas são mais fáceis de testar individualmente.

**4.3 “**def calcular\_media(lista):” “total = sum(lista)” “return total / len(lista)” Utilização em um algoritmo mais complexo “notas = [8, 7, 9, 6, 8]” “media = calcular\_media(notas)” “print("A média das notas é:", media)”