Lógica de Programação

Material Complementar

#PraCegoVer: Fotografia do autor Vinicius

Vinicius Oliveira

Desenvolvedor Fullstack apaixonado por tecnologia, música e um bom café. Com a missão de espalhar a palavra da era digital por esse mundão. Bora codar 🚀

linkedin.com/in/vinnydeveloper

github.com/vinnydeveloper

Índice

Introdução

Ao ouvir sobre lógica de programação talvez logo de cara você possa pensar: *Ixiii, tenho que saber muito sobre matemática e ter um cérebro de um génio*. E apesar de comum esse tipo de pensamento, a primeira coisa que você deve saber é: **Não é verdade!**

Não precisamos saber tudo sobre matemática (mas se souber é de boa ajuda) e muito menos ser um gênio, pois a lógica de programação segue princípios humanos comuns ao dia a dia. Na maior parte das vezes, precisamos apenas adaptá los para um contexto que um computador entenda.

E esse será o nosso objetivo no conteúdo a seguir, ter um acesso rápido e didático sobre o conceito por trás de lógica de programação.

01. Algoritmos

Os algoritmos são muito comuns na matemática para descrever as regras para as equações. Mas eles podem ser aplicados a qualquer sequência de ações que tendem a ter uma resolução final.

As ações que executamos no dia a dia podem ser transformadas em algoritmos, afinal elas seguem uma ordem de sequências para que seja executada.

Se fizéssemos um algoritmo para o preparo de um suco de limão, por exemplo, poderíamos ter as seguintes etapas:

1. Pegar um copo;
2. Pegar os limões e cortar em duas metades;
3. Espremer os limões no copo;
4. Completar com água;
5. Adicionar a açúcar

01. Algoritmos

Então quando falamos de desenvolvimento, o algoritimos serão a sequencia a ser executada pelo computador ou dispositivo para realizar determinada tarefa.

Com base nisso os algoritmos são compostos por dados de entrada (input), processamento e dados de saída (output). Essa estrutura pode ser facilmente entendida com o exemplo de uma calculadora:

Dados de entrada: valores e operações a serem realizadas
Processamento: cálculos realizados pela máquina
Dados de saída: resultado da operação

01. Algoritmos

Geralmente para criarmos algum algoritmo iremos usar um recurso que chamamos de linguagem de programação, pois serão por meio da sintaxe delas que iremos definir cada ação necessária pelo nosso computador.

Hoje temos inúmeras linguagens como: **Javascript, Python, Java, C# e etc.** Porém para ficar mais fácil seu entendimento veja um algoritmo escrito em uma linguagem chamada portugol que usa a sintaxe da nossa língua portuguesa

```
programa {
    funcao inicio() {
        escreva("Oi, meu primeiro programa!\n")
}}
```

Veja que estamos dando algumas instruções para o computador como a **programa e início** para indicar que estamos criando um algoritmo, e de fato a sequência de apenas um passo: **Escrever a seguinte mensagem:** "Oi, meu primeiro programa!\n"

01. Algoritmos

Você pode testar alguns desses algoritmos usando uma plataforma específica para portugol:

Acesso a plataforma

É sempre uma boa colocar a mão na massa e ir testando alguns conceitos, então deixamos aqui um mini-desafio:

Crie um algoritmo usando portugol na plataforma acima, que apresenta duas frases:

“ Ser dev é incrível “
“ A tecnologia muda o mundo”

02. Variáveis

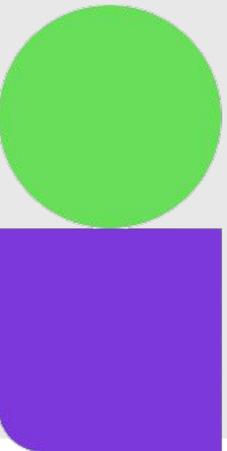
" Armazenar informações você deve! " - Dev Yoda

02. Variáveis

De uma forma simple e rápida: **É uma forma de armazenar valores e identificar esses valores para a reutilização.**

As variáveis tem nome para que possamos identificar e se referir a esses valores sempre que necessário. Veja um algoritmo que irá armazenar um nome e exibir o seu valor.

```
programa {
    funcao inicio() {
        cadeia nomeCompleto = "Vinicius Oliveira"
        escreva(nomeCompleto)
    }
}
```



02. Variáveis

Perceba que geralmente informamos que tipo informação a variável irá armazenar, alguns tipos são bem comuns:

- Cadeia: Texto
- Inteiro: Números inteiros

Dessa forma conseguimos organizar melhor cada informação tanto para nós quanto para o computador.

02. Variáveis

Uma das característica principal de uma variável é que ela pode ter seu valor mudado a qualquer momento no código após sua declaração.

Veja esse exemplo:

```
programa {
    funcao inicio() {
        cadeia nomeCompleto = "Vinicius Oliveira"
        nomeCompleto = "Teste Oliveira"
        escreva(nomeCompleto) // Irá imprimir "Teste Oliveira"
    }
}
```

02. Variáveis

As variáveis nos permite fazer operações matemáticas com cada valor em si! Temos os seguintes operadores:

***** = Multiplicação
+ = Adição
- = Subtração
/ = Divisão

```
programa {
    funcao inicio() {
        inteiro numeroA = 10
        inteiro numeroB = 10
        escreva( numeroA + numeroB) // Irá imprimir 20
    }
}
```

03. Decisões

03. Decisões

O processo de decisão é algo natural de qualquer pessoa. “*Se estiver chovendo, vou levar um guarda chuva, se não nem mochila vou levar*”, “*Se tiver transito vou pegar um atalho*”.

E como de costume, essa possibilidade também existe dentro da lógica de programação e a chamamos de **Decisões** ou (**Se** e **Se não).** Com esse recurso podemos criar instruções para determinadas situações. E para isso usamos sempre um valor lógico baseado em **Verdadeiro ou Falso.** Que chamamos de **condição**

SE verdade
    faça isso
SE NÃO
    faça aquilo

03. Decisões

Com esse recurso, é possível criar algoritmos que se adaptam facilmente. Veja esse exemplo onde temos uma **condição** para caso esteja calor ou não:

```
programa {
    funcao inicio() {
        logico estaCalor = falso
        se (estaCalor){
            escreva("Vou comprar um sorvete")
        } senao {
            escreva("Vou vestir uma blusa de frio")
        }
    }
```

Esse código irá imprimir "Vou vestir uma blusa de frio" pois a variável **estaCalor** está com um valor falso, caindo assim no fluxo do **senao**

03. Decisões

É possível usar os operadores relacionais para gerar valores lógicos por meio de comparações. São esses:

>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
==	igualdade

Com eles podemos comparar valores diretos ou variáveis para termos os valores lógicos: **Verdadeiro ou Falso.**

03. Decisões

Veja esse exemplo utilizando operadores relacionais:

```
programa {
    funcao inicio() {
        logico idade = 18
        se (idade >= 18){
            escreva("Pode dirigir")
        } senao {
            escreva("Não é permitido dirigir")
        }
    }
}
```

Veja que estamos comparando se a **idade é maior ou igual a 18** e definindo uma instrução para cada caso. Aqui irá imprimir "Pode dirigir")

03. Decisões

E se precisamos avaliar duas **condições** para uma mesma instrução? Exemplo: Para dirigir é necessário ser maior ou ter 18 anos e ter uma habilitação para dirigir, caso contrário mesmo que tenha 18 anos não poderá dirigir.

Para resolver esse tipo de problema temos os operadores lógicos:

e (and) com o símbolo && - Verdadeira se condição 1 e condição 2 forem verdadeiras.
ou (or) com o símbolo || - Verdadeira se condição 1 ou condição 2 for verdadeira.

Com isso podemos ter ainda mais recursos na hora de criar condições dentro das estrutura de condições. Veja a seguir o exemplo proposto acima usando novamente o portugol:

03. Decisões

Veja esse exemplo utilizando operadores lógicos:

```
programa {
    funcao inicio() {
        logico idade = 18
        logico habilitacaoParaDirigir = falso
        se (idade >= 18 e habilitacaoParaDirigir){
            escreva("Pode dirigir")
        } senao {
            escreva("Não é permitido dirigir")
        }
    }
}
```

Veja que estamos comparando se a **idade é maior ou igual a 18 e se há habilitação para dirigir.** Esse código irá imprimir "Não é permitido dirigir".

Pois mesmo que a comparação da idade seja verdadeira a segunda condição é falsa, não cumprindo o requisito do operador e

03. Decisões

Mas e se usarmos o operador **ou** o que aconteceria?

```
programa {
    funcao inicio() {
        logico idade = 18
        logico habilitacaoParaDirigir = falso
        se (idade >= 18 ou habilitacaoParaDirigir){
            escreva("Pode dirigir")
        } senao {
            escreva("Não é permitido dirigir")
        }
    }
}
```

Agora estamos comparando se **idade é maior ou igual a 18 *ou* se há habilitação para dirigir.** Esse código irá imprimir "Pode dirigir".

Pois caso qualquer uma das duas condições seja verdadeira irá entrar no fluxo do se como verdade

04. Repetições

"Todo dia ela faz tudo sempre igual" - Cotidiano, Chico Buarque

04. Decisões

Hoje em dia, vemos automações incríveis dentro de fábricas. Que usam robôs que fazem a mesma rotina de instruções por dezenas de vezes.

E dentro da lógica de programação temos estruturas que nos permitem fazer um **loop** de instruções, garantindo que não seja preciso ficar criando diversas instruções iguais dentro do nosso programa.

Chamamos elas de Repetições (Loops) e são essas:

enquanto (While)

```
enquanto ( condicao ){
    instrução a ser repetida
}
```

para (for)

```
para(contador; condicao; incremento){
    instrução a ser repetida
}
```

04. Enquanto

A tarefas de casa pode ser monótonas certo? Se pudéssemos criar um programa que passe o pano na casa várias vezes para garantir uma boa limpeza seria incrível certo? Bom para começar, podemos pelo menos criar algo que represente essa ação.

Usando o laço de repetição Enquanto o código seria assim:

```
programa {
    funcao inicio() {
        inteiro contador = 0
        enquanto( contador < 3 ) {
            escreva("Passando o pano\n")
            contador = contador + 1
        }
    }
}
```

Esse código irá repetir por 3 vezes, executando a instrução de **escreva**.

Enquanto o nosso contador for menor que 3 ele irá continuar a repetição. No momento que esse valor for **falso** a repetição irá parar.

Por isso é importante garantir que temos algo que torne a **condição da repetição falsa, que no caso é o contador.**

04. Para

A tarefas de casa pode ser monótonas certo? Se pudéssemos criar um programa que passe o pano na casa várias vezes para garantir uma boa limpeza seria incrível certo? Bom para começar, podemos pelo menos criar algo que represente essa ação.

Usando o laço de repetição Enquanto o código seria assim:

```
programa {
    funcao inicio() {
        para(inteiro contador= 0; contador < 3; contador++ ){
           escreva("Passando o pano\n")
        }
    }
}
```

Dica: contador++ é a mesma coisa que:
contador = contador + 1

Veja que nesse caso, as condições que vão no **para** são descritas diretamente nele, e podemos dividir em 3 partes:

1. Criamos um contador
2. Definimos quantos ciclos será feitos criando uma condição com o nosso contador.
3. Incrementamos o nosso contador para que dada a condição correta ele pare.

04. Decisões

E quando devemos usar cada um desses recursos?

No geral eles conseguem ter o mesmo resultado porém de forma diferente. O laço de repetição **enquanto** não demonstra quantas repetições serão feitas, além de ter um cuidado em garantir que tenha uma forma da condição se tornar falsa.

Enquanto o laço **para** permite explicitamente dizer quantas repetições serão feitas. Tornando assim a interpretação por parte dos desenvolvedores mais clara.

Então no geral a utilização do **para** é mais comum.

Fechamento

A lógica de programação é essencial no desenvolvimento da nossa habilidade de codar.

Ela será a base para desenvolvermos soluções em todas as linguagens de programação logo é um dos principais itens que todo dev deve priorizar.

Sempre procure treinar sua lógica, sites como https://www.codewars.com tem muitos exercícios para que vocês possam praticar usando diversas linguagens a lógica de programação.

Lembre-se codar, é igual andar a aprender a andar de bicicleta. Não conseguimos aprender apenas lendo e olhando artigos, vídeos e etc. É preciso prática, dedique algum tempo dos seus estudos para praticar e assim absorver melhor o conteúdo.