

AULA 1

O que são Algoritmos?

Algoritmos são sequências lógicas que tem o objetivo de solucionar algo de forma eficiente.

O que é Linguagem de Máquina?

Linguagem de máquina (ou linguagem binária) é a forma própria de linguagem que a máquina suporta. Ela é formada de "zeros" e "uns", ou seja é binária. Porém é muito complexa, não facilmente entendida por humanos, então para isso existe a linguagem de programação, que é uma *linguagem intermediária*.

O que é Linguagem de Programação?

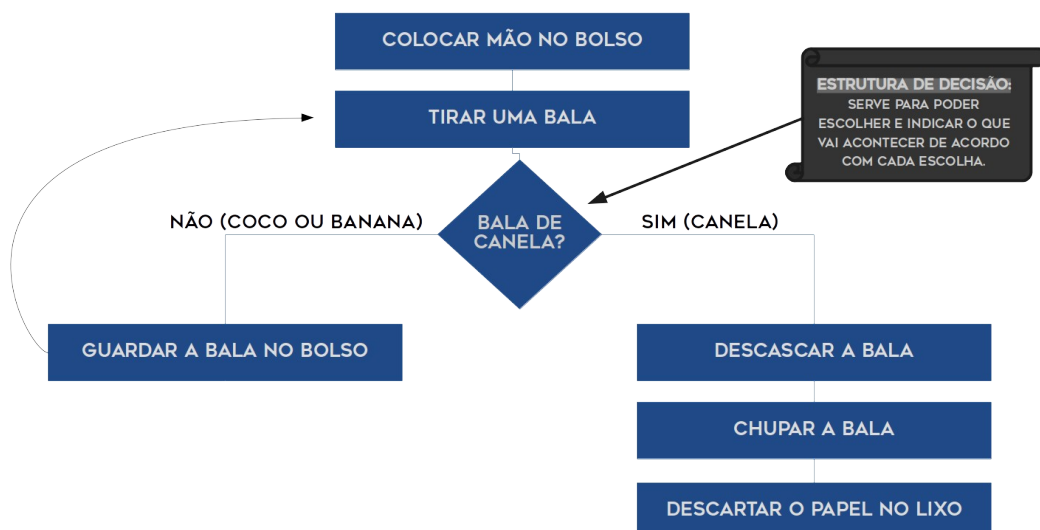
Como a linguagem de máquina era muito complexa para o homem, então criou-se a linguagem de programação. Ela é considerada uma linguagem intermediária, pois é de fácil entendimento do homem e consegue conversar com a máquina. Como exemplo temos as linguagens Pascal, C, C++, C#, Java, Python, Lua, Elixir, JavaScript, PHP, TypeScript, Ruby, Swift, entre outras tantas.

Algoritmos no dia a dia

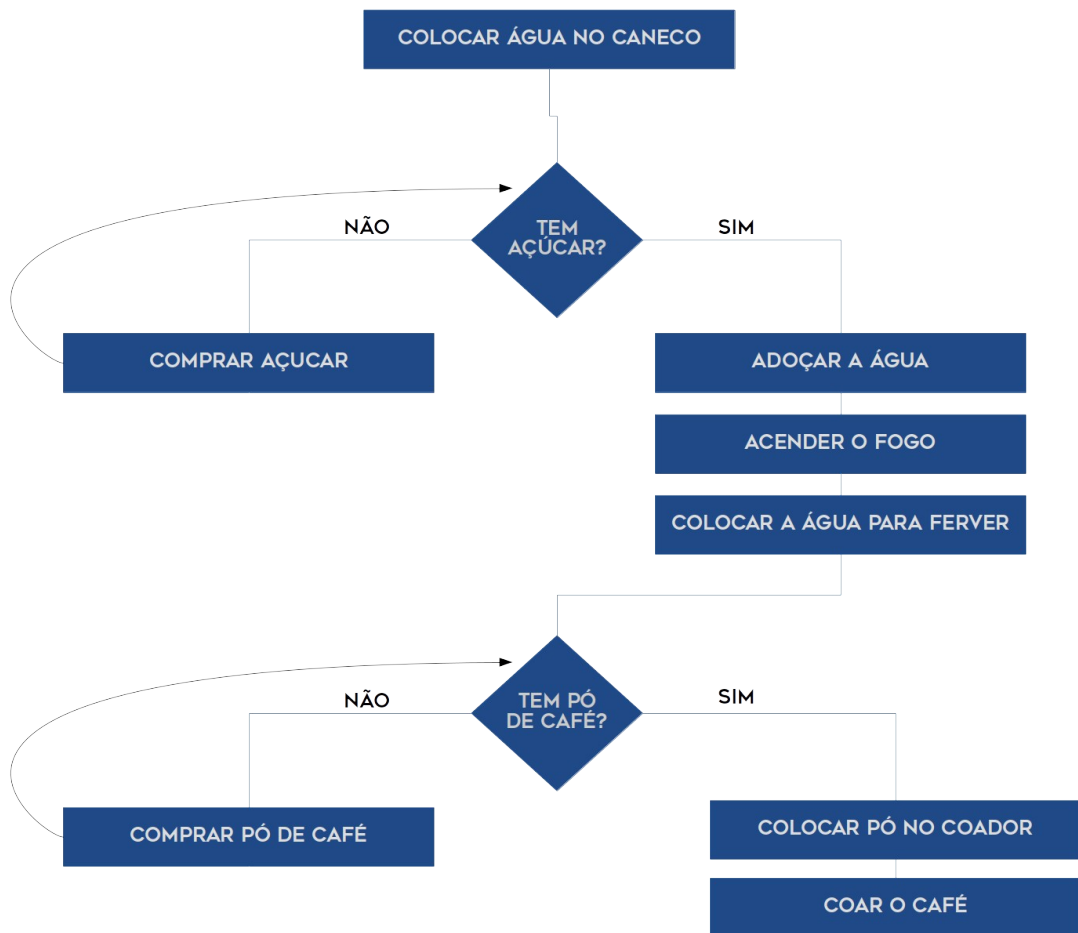
Imagine que no dia a dia exista rotinas, essas rotinas seguem um passo a passo, ou seja, são sequências, sabendo disso podemos considerá-las como algoritmos.

EXEMPLOS

Imagine que você precise pegar uma bala no bolso. Seu bolso está cheio delas, com três sabores diferentes: coco, banana e canela. Você quer chupar uma bala de canela. Faça um algoritmo disso, supondo que você poderá pegar apenas uma bala por vez.



Agora vamos fazer a mesma coisa, porém com uma receita de café. Supondo que o açúcar e o pó de café podem estar em falta, logo sabemos que novamente iremos usar a estrutura de decisão.



Nos dois exemplos acima foram usadas formas geométricas para simbolizar os processos, etapas dos algoritmos. Porém são meramente para exemplificar e não estão seguindo ao pé da letra as regras para criação de fluxogramas, até por que seria muito confuso neste momento estudarmos precocemente fluxogramas. Então não se preocupe ainda com isso.

CURIOSIDADES

As 20 linguagens de programação mais usadas em 2020 (até março) pela lista Tiobe:

Mar 2020	Mar 2019	Change	Programming Language
1	1		Java
2	2		C
3	3		Python
4	4		C++
5	6	⬆	C#
6	5	⬇	Visual Basic .NET
7	7		JavaScript
8	8		PHP
9	9		SQL
10	18	⬆	Go
11	14	⬆	R
12	12		Assembly language
13	17	⬆	Swift
14	15	⬆	Ruby
15	11	⬇	MATLAB
16	22	⬆	PL/SQL
17	13	⬇	Perl
18	20	⬆	Visual Basic
19	10	⬇	Objective-C
20	19	⬇	Delphi/Object Pascal

Tabela 1: Acessível em: <https://www.tiobe.com/tiobe-index/>

AULA 2

O que é Pseudocódigo?

Um pseudocódigo é uma forma mais simples de converter algoritmos em resultados, utilizado em grande escala para estudo. Aqui usaremos o *Visualg* com o *portugol* como programa de pseudocódigo. Ou seja, não chega a ser uma linguagem de programação, mas é necessária para uma melhor aprendizagem.

O que é Visualg?

Visualg é um programa desenvolvido pelo professor Antônio Carlos Nicolodi, que tem como função trabalhar com pseudocódigos, no caso o popular *portugol*, que simula de forma traduzida para o português o que seria possível fazer em linguagens de programação, como o Pascal ou C.

O que são Variáveis?

Variável é o nome dedicado ao espaço que vai armazenar um dado dentro de um programa, por exemplo um nome ou um valor. No Visualg elas podem ser classificadas como *inteiro*, *real*, *caractere* e *lógico*.*

TIPO	DEFINIÇÃO	EXEMPLOS		
INTEIRO	GUARDA VALORES NUMÉRICOS, SEM CASAS DECIMAIS	8	17	500
REAL	GUARDA VALORES QUEBRADOS, OU SEJA, COM CASAS DECIMAIS	3,5	18,9	10,5
CARACTERE	GUARDA PALAVRAS, OU APENAS LETRAS	A	TIGRE	JUNIOR CRISTE
LÓGICO	GUARDA UM VALOR QUE VERDADEIRO OU FALSO.	VERDADEIRO		FALSO

**Existem mais tipos de variáveis, mas no Visualg são apenas essas. Você verá mais tipos quando começar a estudar alguma linguagem de programação.*

O que são Operadores?

Um operador é um símbolo que na matemática indica uma operação. No Visualg temos operadores aritméticos, relacionais, lógicos e de caractere.

OPERADORES ARITMÉTICOS	
+,-	Operadores unários, isto é, são aplicados a um único operando. São os operadores aritméticos de maior precedência. Exemplos: -3, +x. Enquanto o operador unário - inverte o sinal do seu operando, o operador + não altera o valor em nada o seu valor.
\	Operador de divisão inteira. Por exemplo, $8 \setminus 3 = 2$.
+, -, *, /	Operadores aritméticos tradicionais de adição, subtração, multiplicação e divisão. Por convenção, * e / têm precedência sobre + e -. Para modificar a ordem de avaliação das operações, é necessário usar parênteses como em qualquer expressão aritmética.
MOD ou %	Operador de módulo (isto é, resto da divisão inteira). Por exemplo, $10 \text{ MOD } 3 = 1$.
^	Operador de potenciação. Por exemplo, $5 ^ 2 = 25$.

OPERADORES RELACIONAIS	
=	Igual a
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a
<>	Diferente de

OPERADORES LÓGICOS	
nao	Operador de negação. Uma coisa que é "nao VERDADEIRO" é "FALSO".
ou	Operador que resulta VERDADEIRO quando um dos seus dois valores for verdadeiro.
e	Operador que resulta VERDADEIRO somente se seus dois valores forem verdadeiros.
xou	Operador que resulta VERDADEIRO se seus dois operandos lógicos forem diferentes, e FALSO se forem iguais.

OPERADOR DE CARACTERES	
+	Operador de concatenação do tipo "caractere". Por exemplo: "Rio " + " de Janeiro" = "Rio de Janeiro".

AULA 3

Estrutura de um Pseudocódigo

O pseudocódigo, diferente de um algoritmo, tem a obrigação de seguir uma estrutura básica, com início, meio e fim.

O **início** é onde temos nome e declaração das variáveis;

O **meio** é onde acontece todo o processamento, onde acontece todo o programa em si;

E o **fim** é o fechamento apenas.

Observe abaixo a estrutura padrão, que já vem quando abrimos o Visualg.

```
1 Algoritmo "se nome"
2 //
3 //
4 // Descrição : Aqui você descreve o que o programa faz!
5 // Autor(a) : Nome do(a) aluno(a)
6 // Data atual : 12/3/2020
7 Var
8 // Seção de Declarações das variáveis
9
10
11 Início
12 // Seção de Comandos, procedimento, funções, operadores, etc...
13
14
15 Fimalgoritmo
```

Agora observe a mesma estrutura com as cores rosa, vermelha e amarela, que separam o início (rosa), meio (vermelho) e fim (amarelo).

```
1 Algoritmo "se nome"
2 //
3 //
4 // Descrição : Aqui você descreve o que o programa faz!
5 // Autor(a) : Nome do(a) aluno(a)
6 // Data atual : 12/3/2020
7 Var
8 // Seção de Declarações das variáveis
9
10
11 Início
12 // Seção de Comandos, procedimento, funções, operadores, etc...
13
14
15 Fimalgoritmo
```

Como você pode observar a estrutura tem algumas *palavras reservadas*.

Palavras Reservadas é o nome que damos a comandos de uma determinada linguagem. Essas palavras não podem ser usadas em nomes de variáveis, apenas serão usadas em suas respectivas funções.

A parte importante do programa começa realmente na declaração das variáveis. Primeiro declara-se o nome da variável, depois o tipo dela, por exemplo: "NOME: CARACTERE", "IDADE: INTEIRO", "NOTA1: REAL" ou "APROVADO: LOGICO".

```
7 Var
8 // Seção de Declarações das variáveis
9     nome: caractere
10    idade: inteiro
11    nota1: real
12    aprovado: logico
13
```

Uma regra é que nunca um nome de variável se comece com número.

O texto em verde recebe o nome de comentário, ele não será executado, serve apenas para comentar algo dentro do código, ou seja o usuário não tem acesso a isso. Para

comentar basta escrever duas barras e em seguida o que se deseja, por exemplo: "// esse é um comentário".

O visualg aceita tanto letras maiúsculas, quanto minúsculas. Já outras linguagens tem suas próprias regras quanto a isso.

Entrada, Processamento e Saída

Já falamos de início, meio e fim. Agora seremos ainda mais específico, a entrada, o meio e a saída ocorrem no meio do programa.

Entrada é onde recebemos informações, por exemplo quando alguém digita um nome, nota ou qualquer outro tipo de dado.

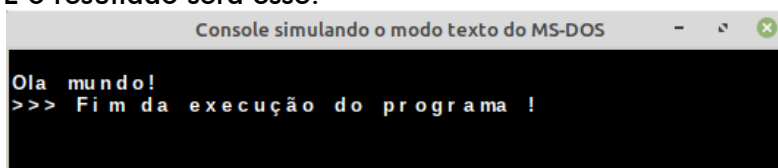
Processamento é a parte em que vamos trabalhar com esses dados, por exemplo, se eu for fazer uma conta, é no processamento que ficarão os comandos dessa operação matemática. Mas não se limita apenas a isso. Processamento é qualquer alteração que eu faça em valores de variáveis, independente do tipo dela.

E por fim temos a **Saída**, essa é responsável para apresentar a solução do problema, no caso da soma é o resultado. Qualquer solução apresentada é uma saída.

Para escrever alguma coisa na tela para que o programa possa se comunicar com o usuário é necessário usar o comando "ESCREVA" seguido de parênteses, e dentro desses parênteses existem aspas duplas, dentro dessas abas o texto que vai aparecer na tela. Observe:

```
13
14 Início
15 // Seção de Comandos, procedimento, funções, operadores, etc...
16 escreva("Ola mundo!")
17
18 Fim algoritmo
```

E o resultado será esse:



```
Console simulando o modo texto do MS-DOS
Ola mundo!
>>> Fim da execução do programa !
```

Agora para que haja um retorno, por exemplo, quando você solicitar que o usuário digite o nome dele é preciso salvar isso em uma variável, para isso usamos o comando "LEIA" seguido de parênteses e o nome da variável dentro. Observe:

```
13 escreva("Digite seu nome ")
14 leia (nome)
```

Acabamos de fazer um processo de **entrada**!

EXEMPLOS

Agora vamos solicitar o nome e sobrenome dessa pessoa. Salvaremos isso em variáveis. Também será necessário outra variável para concatenar (juntar) os dois valores (nome e sobrenome) em apenas um valor.

Para fazermos isso, será necessário então três variáveis, vamos chamá-las de "nome", "snome" e "ncompleto", sendo referentes ao nome do usuário, sobrenome e o nome junto com o sobrenome.

Após capturar o *nome* e o *snome* iremos atribuir isso ao *ncompleto*.

Mas como faremos isso?

Usaremos a atribuição. Atribuir é dizer o que será feito com aqueles dados, ou seja, entramos na parte de **processamento**.

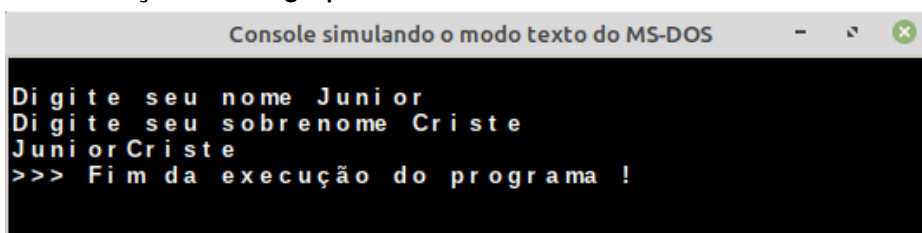
Para atribuir o valor de *ncompleto* devemos seguir uma estrutura. Primeiro coloca-se o nome da variável, depois o símbolo de atribuído que é " \leftarrow ", basicamente desenhar uma seta voltada a variável que vai receber o valor. E em seguida colocamos o valor. No caso vai ser a concatenação de nome com *snome*, ficará assim: "*ncompleto* \leftarrow nome + *snome*". O que acabamos de fazer foi o processamento, onde indicamos que o nome completo do usuário é a junção do nome com o sobrenome.

Agora que já temos a solução, precisamos de mostrá-la ao usuário, isso é o processo de **saída**. Para mostrar um valor na tela é preciso usar novamente o comando "ESCREVA" seguido dos parênteses, porém dessa vez SEM as aspas duplas e com o nome da variável na tela. As aspas duplas não são usadas nesse caso, pois não está sendo apresentado um texto fixo, e sim um valor de variável. Lembrando que a variável apresentada aqui será a solução, ou seja, a *ncompleto* que acabou de receber um valor na etapa de processamento.

Vejamos como isso fica na prática:

```
1 Algoritmo "Nome Completo"
2 //
3 //
4 // Descrição : Aqui você descreve o que o programa faz!
5 // Autor(a) : Nome do(a) aluno(a)
6 // Data atual : 12/3/2020
7 Var
8 // Seção de Declarações das variáveis
9 nome, snome, ncompleto: caractere
10
11 Início
12 // Seção de Comandos, procedimento, funções, operadores, etc..
13 escreva("Digite seu nome ")
14 leia (nome)
15
16 escreva("Digite seu sobrenome ")
17 leia (snome)
18
19 ncompleto  $\leftarrow$  nome + snome
20
21 escreva(ncompleto)
22
23 Fimalgoritmo
```

E a execução será algo parecido com isso:



```
Console simulando o modo texto do MS-DOS

Digite seu nome Junior
Digite seu sobrenome Criste
JuniorCriste
>>> Fim da execução do programa !
```

Observe que na declaração das três variáveis, todas ficaram em apenas uma linha, separadas por vírgulas. Isso devido o fato de serem do mesmo tipo, então podem ficar juntas, porém poderiam estar separadas também, é apenas uma questão de prática e estética. Elas foram declaradas do tipo "CARACTERE", pois são nomes, e no Visualg nomes são considerados caracteres, em outras linguagens usamos o tipo "STRING" para isso.

AULA 4

Salvar projetos online

Esse capítulo é dedicado ao, e somente ao GitHub.

O que é GitHub?

“O GitHub é mundialmente usado e chega a ter mais de 36 milhões de usuários ativos mundialmente contribuindo em projetos comerciais ou pessoais. Hoje o GitHub abriga mais de 100 milhões de projetos, alguns deles que são conhecidos mundialmente. WordPress, GNU/Linux, Atom, Electron. GitHub também oferece suporte ao recurso de organização que é amplamente utilizado por aqueles que querem uma escala maior para seus projetos. Na maioria das vezes, o recurso é usado por empresas já existentes como a Google, Microsoft e WordPress.”
Por Wikipédia.

Em resumo: O GitHub é um site onde você pode hospedar seu código, assim, você poderá acessá-lo em qualquer máquina, apenas bastando baixá-lo para editar. Além disso, outras pessoas poderão se basearem em seu código, copiarem e até mesmo criarem melhorias. Como se fosse uma rede social de códigos.

Por que usaremos?

Simple, assim será mais fácil identificar erros, basta criar seu código, postá-lo e compartilhar o link com outros, podendo pedir ajuda em busca de erros ou até mesmo solucionando-os.

Nome do usuário

O email

Senha

Verifique se há pelo menos 15 caracteres OU pelo menos 8 caracteres, incluindo um número e uma letra minúscula . [Saiba mais](#) .

Inscreva-se no GitHub

Ao clicar em "Inscreva-se no GitHub", você concorda com nossos [Termos de Serviço](#) e [Política de Privacidade](#) . Ocasionalmente, enviaremos e-mails relacionados à sua conta.

Siga o tutorial da aula 4 em vídeo ensinando a criar seu repositório – gratuitamente – de códigos.

AULA 5

O que é Estrutura de Decisão?

Muitas vezes precisamos de fazer escolhas. Entre ir ou vir, doce ou amargo, aceso ou apagado, dormir ou trabalhar, enfim.

Na programação não é diferente! Escolhas dão vida e real função a uma aplicação. Para ter essas escolhas, ou melhor, estruturas de decisão, nós usaremos três recursos.

Se...FimSe

Essa é a estrutura básica para que o usuário consiga escolher entre duas ou mais opções. Ou então não escolher, mas receber "um resultado". Por exemplo, ao final do ano o aluno recebe a nota final, e se ele tirar 60 ou mais ele está aprovado, caso contrário ele reprova. Nesse pequeno exemplo já se tem ideia de como funciona o Se...Fimse.

No próximo exemplo você verá um programa que pede que o usuário digite a sua idade e a aplicação informará se ele atingiu ou não a maioridade.

```
Área dos algoritmos ( Edição do código fonte ) -> Nome do arquivo: [semnome] -
1 Algoritmo "se...fimse"
2
3 Var
4 // Seção de Declarações das variáveis
5 idade: inteiro
6
7
8 Inicio
9 // Seção de Comandos, procedimento, funções, o
10 escreva("Por favor, digite sua idade: ")
11 leia(idade)
12
13 se idade >= 18 entao
14     escreva("Voce e maior de idade! ")
15     fimse
16
17 se idade < 18 entao
18     escreva("Voce e menor de idade! ")
19     fimse
20
21 Fimalgoritmo
```

Se...Senao

Essa estrutura é uma versão mais sofisticada e ágil do Se...Fimse.

Se compararmos o mesmo programa acima com o abaixo, você notará que não houve a repetição de um novo Fim...Se caso o usuário tenha menos de 18 anos. O código economizou simplesmente dizendo que, se não for maior ou igual a 18 é menor (logo, se não for verdadeiro é falso). Observe abaixo e compare com a versão anterior:

Área dos algoritmos (Edição do código fonte) -> Nome do arquivo: [semnome] -

```
1 Algoritmo "se...senao"
2
3 Var
4 // Seção de Declarações das variáveis
5   idade: inteiro
6
7
8 Início
9 // Seção de Comandos, procedimento, funções, o
10  escreva("Por favor, digite sua idade: ")
11  leia(idade)
12
13  se idade >= 18 entao
14    escreva("Voce e maior de idade! ")
15    senao
16    escreva("Voce e menor de idade! ")
17  fimse
```

Escolha...Caso

Essa estrutura é interessante para ser utilizada quando existem muitas opções de entradas pré-definidas, como por exemplo uma sigla que significa alguma coisa e dependendo de qual seja essa sigla o programa retornará uma resposta diferente pra cada. Mas, o Escolha...Caso não se limita apenas a caracteres.

O próximo exemplo é simples e fácil de ser compreendido. Se trata de uma aplicação que recebe uma entrada, que poderá ser "m", "h", ou "o", se ele digitar outra opção ele informará erro.

Observe também que para cada caso a aplicação retorna uma resposta diferente.

Área dos algoritmos (Edição do código fonte) -> Nome do arquivo: [semnome] -

```
1 Algoritmo "escolha...caso"
2
3 Var
4 // Seção de Declarações das variáveis
5   sexo: caractere
6
7 Início
8 // Seção de Comandos, procedimento, funções, operad
9   escreva("M = mulher | H = homem | O = outros")
10  escreva("Digite uma sigla para seu sexo: ")
11  leia(sexo)
12
13  sexo <- maiusc(sexo)
14
15  escolha sexo
16    caso "m"
17      escreva("voce e mulher")
18    caso "h"
19      escreva("voce e homem")
20    caso "o"
21      escreva("voce e outro")
22  outrocaso
23    escreva("Entrada invalida")
24  fimsecolha
25
26 Fimalgoritmo
```

AULA 6

O que é Estrutura de Repetição?

Repetir, repetir, repetir... A repetição pode ser até fácil, mas cansa, é como correr ao redor de um poste, você pode rapidamente dar uma volta, duas voltas, três... mas 100 voltas são cansativas.

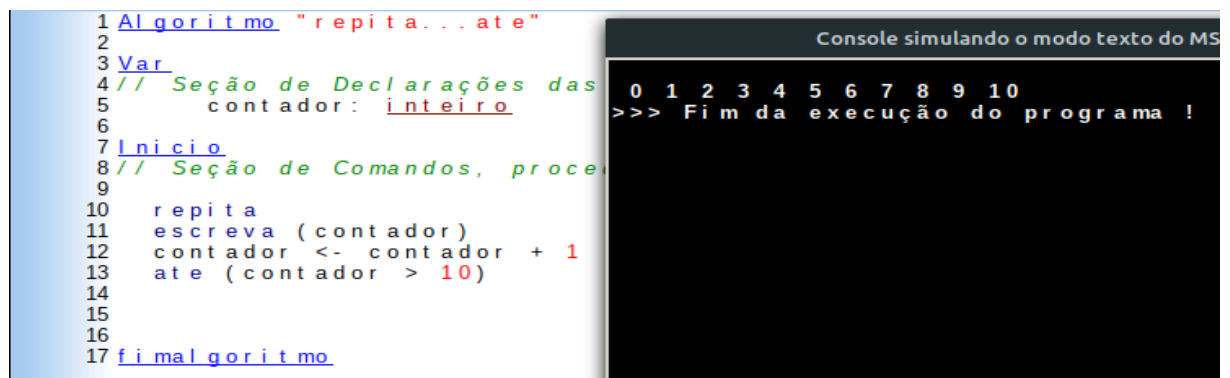
Pra isso que nós utilizamos automações, você deixaria de correr com os próprios pés. Utilizando essas estruturas nós temos uma redução absurda em linhas de códigos, deixamos nossa aplicação mais bonita, ágil e legível.

Repita...Ate

O Repita...Ate é uma estrutura de fácil aplicação, você informa apenas que aquele loop (nome que chamamos uma repetição em programação) irá se repetir até que uma variável receba um certo valor. Para que esse valor seja alcançado é necessário a existência de um contador. Esse contador é uma variável que começa com um valor e toda vez que o loop repete ela pega o valor dela e soma mais 1.

Observe o código a seguir, ele faz com que o loop seja executado até que a variável contador seja maior que 10. Essa variável não teve nenhum valor atribuído anteriormente, sendo assim ela começa do 0.

Resumindo, ele vai começar a contar do 0 e enquanto não for maior que 10 (enquanto não for 11) ele vai imprimir na tela o valor do contador, que a cada repetição recebe o próprio valor mais 1.



```
1 Algoritmo "repita...ate"
2
3 Var
4 // Seção de Declarações das
5     contador: inteiro
6
7 Inicio
8 // Seção de Comandos, processamento
9
10     repita
11         escreva (contador)
12         contador <- contador + 1
13     ate (contador > 10)
14
15
16
17 fim algoritmo
```

Console simulando o modo texto do MS

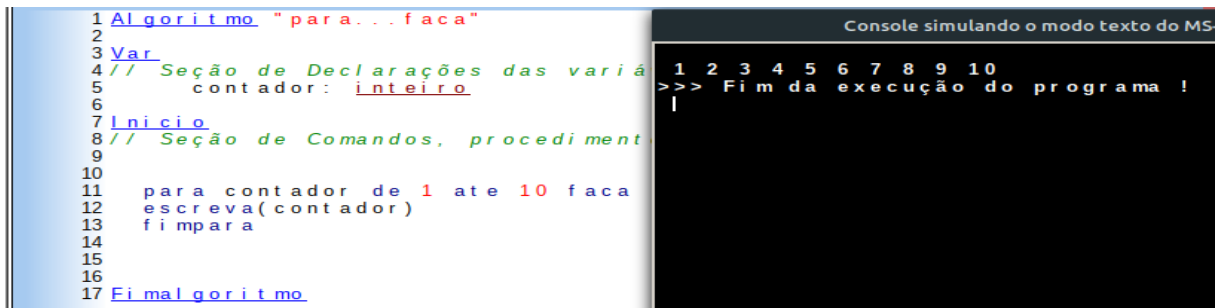
```
0 1 2 3 4 5 6 7 8 9 10
>>> Fim da execução do programa !
```

Para...Faca

O Para...Faca funciona do mesmo jeito, porém ele já tem o contador atribuído dentro do topo dele, então não necessita de colocar um "contador ← contador + 1", pois ele já fará isso automaticamente de acordo com o que for expressado no topo do loop.

O mais interessante desse modelo é que você informa de que número ele começa a contar e até quanto vai.

Observe o exemplo a seguir e compare com o anterior e note que a mesma repetição pode ser expressada de diferentes formas, sendo que no exemplo abaixo solicitamos que o programa comece a contar de 1 e não de 0:



```
1 Algoritmo "para...faca"
2
3 Var
4 // Seção de Declarações das variáveis
5     contador: inteiro
6
7 Inicio
8 // Seção de Comandos, procedimentos
9
10
11     para contador de 1 ate 10 faca
12         escreva(contador)
13     fimpara
14
15
16
17 Fimalgoritmo
```

Console simulando o modo texto do MS-DOS

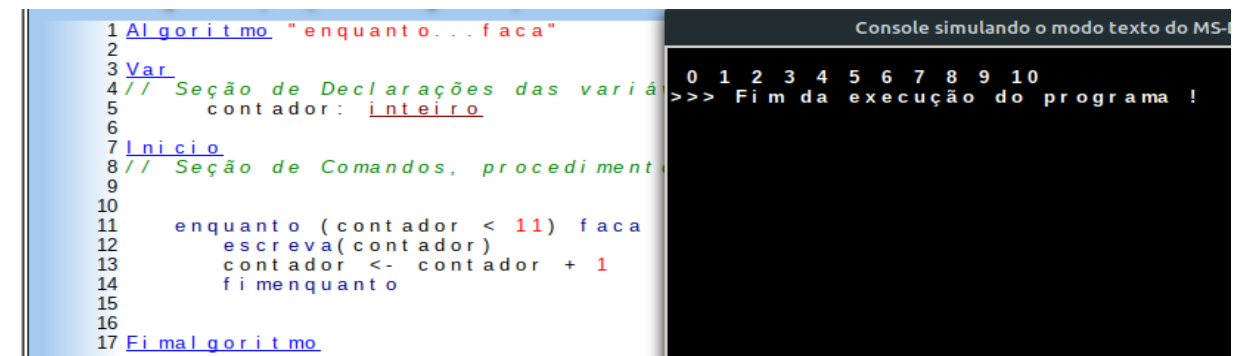
```
1 2 3 4 5 6 7 8 9 10
>>> Fim da execução do programa !
```

Enquanto...Faca

O Enquanto...Faca é semelhante ao Para...Faca, mas não recebe a expressão que soma ao contador mais 1 no seu topo como o exemplo anterior. Sendo assim você volta a usar o "contador ← contador + 1" dentro do loop.

Nele você também não informa no loop o valor que ele começa o contar. Para que isso seja possível é necessário atribuir um valor a variável pelo lado de fora do loop, mais precisamente antes de entrar no loop.

No exemplo a seguir nós não atribuímos nenhum valor ao contador antes de entrar no loop, então ele automaticamente começa a contar do 0, note:



```
1 Algoritmo "enquanto...faca"
2
3 Var
4 // Seção de Declarações das variáveis
5     contador: inteiro
6
7 Inicio
8 // Seção de Comandos, procedimentos
9
10
11     enquanto (contador < 11) faca
12         escreva(contador)
13         contador <- contador + 1
14     fimenquanto
15
16
17 Fimalgoritmo
```

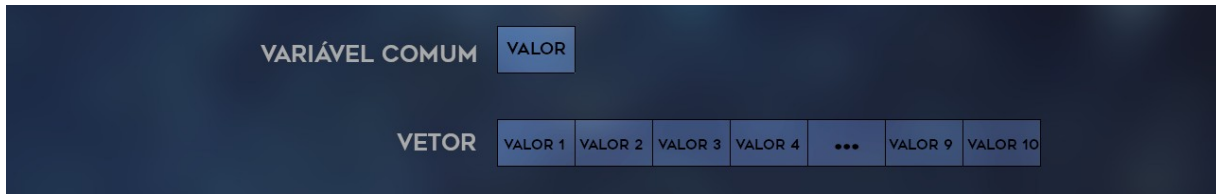
Console simulando o modo texto do MS-DOS

```
0 1 2 3 4 5 6 7 8 9 10
>>> Fim da execução do programa !
```

AULA 7

O que é um Vetor?

Um vetor é uma coleção de dados. E em algoritmos essa coleção tem itens do mesmo tipo. Pode ser uma coleção de inteiros, por exemplo.



Diferentemente de uma variável comum, onde se guarda apenas 1 valor, o vetor pode ser de qualquer tamanho, por exemplo, um vetor de 5 posições tem a mesma capacidade que 5 variáveis, sendo todas do mesmo formato.

A criação do vetor é na mesma parte de declaração das variáveis. Primeiro informa o nome, depois que é um vetor, após o número que começa até o número de posições que vai esse vetor e ao final avisamos o formato dos dados que ali serão guardados:

Var

// Seção de Declarações das variáveis

numero: vetor[1..5] de inteiro

Para atribuir um valor em uma das posições é necessário identificar o índice, que é o número da posição, no restante, a atribuição acontece da mesma forma que a atribuição de variáveis.

Início

// Seção de Comandos, procedimento,

numero[1] <- 9

numero[2] <- 6

numero[3] <- 2

numero[4] <- 7

numero[5] <- 8

Comumente usamos nessas aplicações com vetores o recurso de repetição. Assim é possível atribuir ou imprimir todos os valores de forma mais eficiente. A cada volta do loop uma casa do vetor é andada por intermédio de um contador.

```
23 Var
24
25 numero: vetor[1..5] de inteiro
26 i: inteiro
27
28 Início
29
30 numero[1] <- 9
31 numero[2] <- 6
32 numero[3] <- 2
33 numero[4] <- 7
34 numero[5] <- 8
35
36 para i de 1 ate 5 faça
37 escreva(numero[i])
38 fimpara
39
40 Fim algoritmo
```

The screenshot shows a console window titled 'Console simulando o modo texto do N'. It displays the output of the program: the numbers 9, 6, 2, 7, 8 on one line, followed by a prompt '>>>' and the text 'Fim da execução do programa !' on the next line.

AULA 8

O que é uma Matriz?

Matrizes, diferentemente dos vetores podem ter várias dimensões, enquanto o vetor tem várias posições, porém uma linha de dimensão.

Imagine uma matriz como uma tabela, ela pode ter várias colunas e várias linhas. Sendo assim, para percorrer ela em um processo de repetição, é necessário ao menos duas variáveis de contadores, uma responsável por percorrer o eixo X e a outro o eixo Y.

A criação de uma matriz é parecida com a criação de vetores. Primeiro informa o nome, depois que é um vetor (sim! matriz é um vetor, porém com várias dimensões), após o número que começa até o número de posições que vai ter no primeiro eixo, adiciona a virgula e coloca o número que começa até o número de posições que vai o outro eixo e ao final avisamos o formato dos dados que ali serão guardados.

Note abaixo um exemplo visual de vetor e matriz, observe suas diferenças:

numero: vetor[0..3] de inteiro

VETOR

0	1	2	3
5	3	0	1

numero: vetor[0..3, 0..3] de inteiro

MATRIZ

	0	1	2	3
0	5			
1			9	
2		7		
3				

Para atribuir um valor na matriz precisamos indicar duas posições, uma para cada eixo. Veja no exemplo a seguir como ocorre essa atribuição e impressão dos valores:

```
3 Var
4
5 numero: vetor[1..9, 1..5] de inteiro
6
7 Inicio
8
9   numero[2, 5] <- 39
10
11 escreva(numero[2, 5])
12
```

Console simulando o modo texto do MS
39
>>> Fim da execução do programa !

As matrizes são usadas em conjunto com estruturas de repetição para atribuição, entrada ou saída de dados.

Essas estruturas necessitam de dois contadores, um primeiro para correr um eixo, e um segundo para o outro eixo.

Normalmente chamamos esses contadores de "i" e "j", mas isso não é uma regra, o nome da variável de contador não faz diferença quanto a execução do programa.

Para ser possível essa atribuição de valores em uma matriz precisamos de entrar em um loop para o primeiro eixo, e dentro desse loop haverá outro loop para correr o segundo eixo. Ou seja, ele entra em uma posição do eixo X, corre todas posições do eixo Y, volta, faz mais uma posição do eixo X, e corre novamente todas as posições do eixo Y.

Abaixo, uma aplicação que sorteia números de 0 até 10 e guarda isso em uma matriz de 9x5, observe:

```
1 Algoritmo "Matriz Sorteio"
2 Var
3 numero: vetor[1..9, 1..5] de inteiro
4 i, j: inteiro
5
6 Inicio
7
8     para i de 1 ate 9 faca
9         escreval() //apenas para pular linha
10
11         para j de 1 ate 5 faca
12
13             numero[i,j] <- randi(10)
14             escreva(numero[i,j])
15
16         fimpara
17     fimpara
18
19 Fimalgoritmo
```

```
8 0 2 7 2
7 4 8 5 0
5 9 9 6 2
5 9 8 4 2
8 7 7 6 9
7 7 8 2 7
4 3 5 5 0
3 7 0 3 2
3 0 4 8 8
>>> Fim da
```

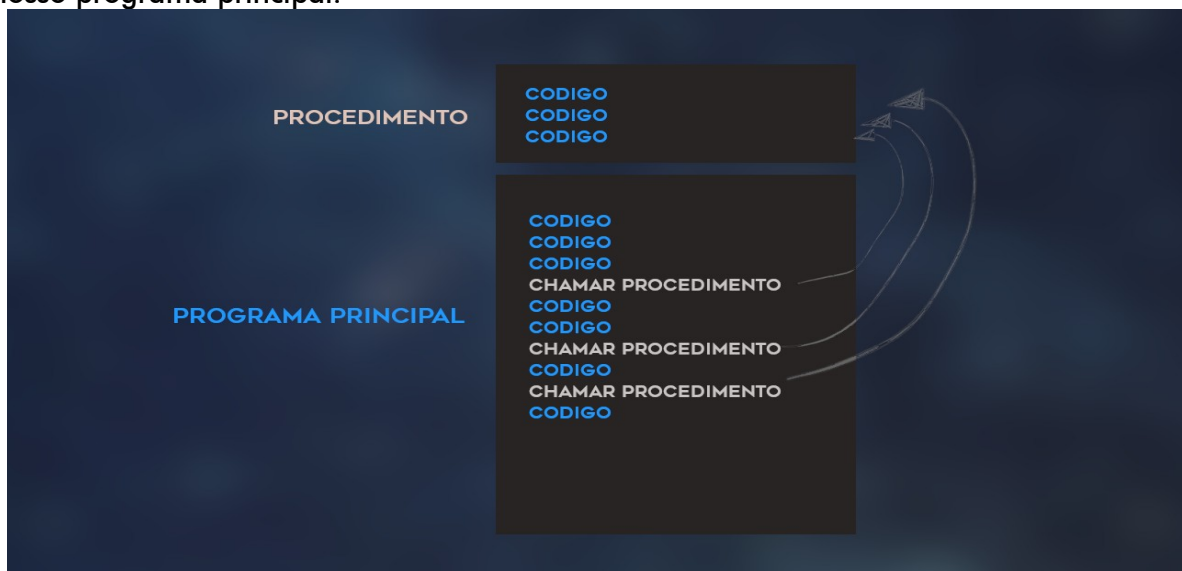

AULA 9

O que é um Procedimento?

O procedimento é uma forma de rotina. Rotinas são coisas que fazemos sempre, são repetitivas e desgastantes. Então pra facilitar isso usaremos procedimentos e funções.

Imagine que toda vez que um valor entrar em seu programa você precisará fazer um gigante calculo, gastando muitas linhas de código. Esse procedimento terá de ser repetido muitas vezes, mas você não quer escrever o mesmo código o tempo todo. Então você usará um procedimento.

O procedimento é um "programa menor" que ficará declarado abaixo das variáveis. Esse programa menor servirá para colocarmos processos que se repetem inúmeras vezes em nosso programa principal.



Mesmo estando no topo do código, o Visualg não lê esse procedimento, até que você o chame dentro de sua aplicação principal.

Todo procedimento recebe um "Inicio" e "FimProcedimento", para mostrar onde começa e acaba. Além disso eles podem ter variáveis declaradas em seu próprio corpo, mas deixaremos esse papel para as funções.

Note que o código abaixo solicita um inteiro para dobrar esse valor, porém o cálculo só acontece fora do programa principal, dentro de um procedimento:

```
1 Algoritmo "Procedimentos"
2 Var
3 valor, dobro: inteiro
4
5 procedimento calc() // procedimento
6 inicio
7
8     dobro <- valor * 2
9
10 fimprocedimento
11
12 inicio // programa principal
13
14     escreva("Digite um valor inteiro: ")
15
16     leia(valor)
17     calc() //chamando o procedimento
18
19     escreva("O dobro de ", valor, " e ", dobro)
20
21 Fimalgoritmo
```

```
Console simulando o mod

Digite um valor inteiro:
5
O dobro de 5 e 10
>>> Fim da execução do pro
```

AULA 10

O que é uma Função?

A função (também chamada de sub-rotina) tem o mesmo papel dos procedimentos, mas ela tem um poder maior, que é a entrada de um dado, tratamento e retorno de um outro dado, tudo isso por meio de parâmetros.

O que são Parâmetros?

Parâmetros são valores passados a uma função quando ela for chamada. Acontecendo isso, esses valores serão atribuídos a outras variáveis que existem somente dentro da função, são as variáveis locais. Dentro dessa função essas variáveis fazem todo o processo necessário e retornam um valor que pode ou não ser com um formato diferente do que entrou, esse valor retornado volta para o programa principal e as variáveis locais deixam de existir, até que sejam chamadas novamente e tudo se repetir.

As **variáveis locais** só existem dentro dessas funções que são declaradas, isso faz com que não ocupem espaço na memória enquanto não forem chamadas, diferentemente das variáveis globais (variáveis do programa principal), que já ocupam espaço logo no início da execução. Vendo isso podemos supor uma melhora significativa do desempenho de uma aplicação que tenha muitos processos e que goze do uso das funções e variáveis locais.

Ao chamar uma função passo um parâmetro:

```
Inicio           // programa principal  
  
    escreva("D i g i t e   s u a   n o t a: ")  
    leia(nota)  
  
    escreva(resultado(nota)) // invocação da função
```

Note acima que estou passando o parâmetro "nota" para a função "resultado" dentro de um escreva.

Isso é possível, pois o escreva recebe um valor para mostrar a usuário, esse valor é fruto da "nota" enviada para "resultado" que foi trabalhada lá dentro da função. Veja:

Algoritmo "Funcoes"

Var

nota: inteiro

funcao resultado(x: inteiro): caractere

var

resposta: caractere

inicio

se nota < 60 entao

resposta <- "Nao foi aprovado!"

senao

resposta <- "Foi aprovado!"

fimse

retorne resposta

fimfuncao // fimdafuncao

Note que na função anterior havia uma variável de nome "x" de tipo inteiro declarada no cabeçalho. Isso quer dizer que dentro da função "nota" não existe, mas sim "x", que passa a ter temporariamente o valor de "nota". Ao final do cabeçalho existe uma definição de "caractere", essa parte está dizendo que apesar de entrar um valor inteiro na função, será retornado um valor caractere. Ou seja, entra um inteiro, é tratado, retorna outro valor e tipo.

Dentro dessa função ainda existe uma variável local declarada como "resposta" do tipo "caractere", e é ela quem vai ser retornada, não "x", por isso da definição "caractere" no cabeçalho da função.

Em resumo, a função funciona como uma máquina, ela recebe em "x" um valor do programa principal, faz todos os procedimentos necessários, e guarda o que é importante dentro de "resposta", ao final retornando esse valor ao programa principal, que por sua vez, imprime na tela. Veja o código completo na próxima página:

Algoritmo "Funcoes"

Var

nota: inteiro

funcao resultado(x: inteiro): caractere

var

resposta: caractere

inicio

se nota < 60 entao

 resposta <- "Nao foi aprovado!"

senao

 resposta <- "Foi aprovado!"

fimse

retorne resposta

fimfuncao *// fimdafunção*

Inicio *// programa principal*

 escreva("Digite sua nota: ")

 leia(nota)

 escreva(resultado(nota)) *// invocação da função*

Fimalgoritmo