



**GOVERNO DO
ESTADO DO CEARÁ**

Secretaria da Educação

**ESCOLA ESTADUAL DE
EDUCAÇÃO PROFISSIONAL - EEEP**
ENSINO MÉDIO INTEGRADO À EDUCAÇÃO PROFISSIONAL

CURSO TÉCNICO EM INFORMÁTICA

PROGRAMAÇÃO WEB



GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

GOVERNADOR
Camilo Santana

VICE-GOVERNADORA
Maria Izolda Cela de Arruda Coelho

SECRETÁRIO DA EDUCAÇÃO
Rogers Vasconcelos Mendes

SECRETÁRIA EXECUTIVA DA EDUCAÇÃO
Rita de Cássia Tavares Colares

ASSESSORIA INSTITUCIONAL
Danielle Taumaturgo

COORDENADORIA DA EDUCAÇÃO PROFISSIONAL
Jussara de Luna Batista



Disciplina:
Programação Web (JavaScript/PHP/MySQL)

*Apostila destinada ao Curso Técnico de Nível Médio em Informática das Escolas Estaduais de
Educação Profissional – EEEP*

Equipe de Elaboração - 2012

*Adriano Gomes da Silva
Cíntia Reis de Oliveira
Evandilce do Carmo Pereira
Fernanda Vieira Ribeiro
Francisco Aislân da Silva Freitas
João Paulo de Oliveira Lima
Juliana Maria Jales Barbosa
Liane Coe Girão Cartaxo
Mirna Geyla Lopes Brandão
Moribe Gomes de Alcântara
Niltemberg Oliveira Carvalho
Paulo Ricardo do Nascimento Lima
Renanh Gonçalves de Araújo
Renato William Rodrigues de Souza
Valbert Oliveira Costa*

Colaboradores

*Maria Analice de Araújo Albuquerque
Maria Danielle Araújo Mota
Sara Maria Rodrigues Ferreira Feitosa*

Atualização – 2018

Paulo Ricardo do Nascimento Lima

SUMÁRIO

Apresentação	7
1.0 Introdução à JavaScript	8
1.1 Introdução ao JavaScript	8
1.2 Java é diferente de JavaScript	9
1.3 Entendendo o JavaScript	9
1.4 Usando JavaScript	11
2.0 Variáveis	12
2.1 Mas o que são variáveis em programação?	12
2.2 Mas o que seria o tipo de uma variável?	12
2.3 Nomeando variáveis	13
2.4 Declarando ou criando uma variável	14
2.5 Linguagens de Programação Fracamente e Fortemente Tipadas	14
2.6 Variáveis em JavaScript	15
2.6.1. Declarando ou criando uma variável JavaScript	15
2.7 Tipos de variáveis JavaScript	16
2.7.1 Trabalhando com String e Números em JavaScript	16
2.7.2 Trabalhando com outros tipos de dados JavaScript	17
2.8 Variáveis Globais e Locais	18
3.0 Operadores em JavaScript	20
3.1 Operadores de strings	20
3.2 Operadores Matemáticos	20
3.2.1 Aritméticos	20
3.2.2 Atribuição	21
3.3 Operadores de decremento e incremento	23
3.4 Operadores relacionais	23
3.5 Operadores lógicos ou booleanos	25
3.6 Operação de Negação:	25
3.7 Operação de conjunção:	26
3.8 Operação de disjunção	26
3.9 Precedência de Operadores	29
Exercícios Propostos	30
4.0 Caixa de Diálogo	32
4.1 Alert	32
4.2 Prompt	32
4.3 Confirm	33
Exercícios Propostos	33
5.0 Estruturas de controle e repetição	35
5.1 Blocos de controle	35
5.2 IF e ELSE	36
5.3 Atribuição condicional (ternário)	38
5.4 SWITCH	38
5.5 SWITCH com BREAK	39
5.6 WHILE	41
5.7 DO...WHILE	42
5.8 FOR	43
5.9 FOREACH	44
5.10 BREAK	45
5.11 CONTINUE	46
Exercícios Propostos	46
6.0 Objetos em JavaScript	48
6.1 Objeto String, Acessando atributos e métodos	48

6.2 Acessando atributos.....	49
6.3 Acessando Métodos	49
6.4 Objeto Number.....	50
6.4.1 Atributos objeto Number - CONSTANTES	51
6.4.2 Representação de um número em Hexadecimal, Octal e notação científica.....	51
6.5 Objeto Math.....	52
6.5.1 Constantes	52
6.5.2 Objeto Math e seus métodos	54
6.5.3 Arredondamentos	55
6.5.4 Trigonometria.....	55
6.5.5 Maior e Menor.....	56
6.5.6 Número Randômico	56
6.5.7 Objeto Boolean.....	57
6.5.8 Objeto Date	57
6.5.9 Criando um objeto Date	57
6.5.10 Definindo uma data específica	58
6.5.11 Métodos do Objeto Date.....	59
Exercícios Propostos:	60
7.0 Array.....	61
7.1 Criando um Array	61
7.2 Arrays associativos.....	62
7.3 Interações	63
7.4 Acessando um Array.....	64
7.5 Alterando um Array	65
7.6 Arrays multidimensionais.....	66
7.6 Funções com Arrays	70
7.7 Junção e Concatenação de Arrays (Vetores)	70
7.8 Método join()	70
7.8.1 Método concat()	71
7.8.2 Método push()	72
7.8.3 Método unshift().....	72
7.8.4 Método pop().....	73
7.8.5 Método shift().....	73
7.8.6 Método reverse()	74
7.8.6 Método sort().....	74
8.0 Manipulação de Funções.....	78
8.1 Declarando uma Função.....	78
8.2 Escopo de Variáveis em Funções	79
8.3 Valor de Retorno.....	79
8.4 Recursão.....	80
Exercícios Propostos	80
9.0 EventosJavaScript	82
9.1 Relação de eventos	82
9.2 Evento onclick.....	84
9.3 Evento onsubmit.....	85
Evento onmouseover e onmouseout.....	87
9.4 Eventos: onfocus e onblur	88
9.5 Evento: onchange	89
9.6 Eventos: onload e onunload	90
10.0 Document Object Model – DOM.....	93
Exercícios Propostos	98
11.0 Introdução ajQuery.....	99

11.1 Instalação e configuração	99
11.2 Primeiros Passos.....	100
11.2.1 Métodos de acesso a Elementos HTML.....	102
12.0 BibliotecajQuery	105
12.1 Funções de Estilo	105
12.2 Função .css()	105
12.3 Função .add().....	106
13.1.1. Função .addClass() e removeClass().....	108
12.4 Função toggleClass().....	109
12.5 Funções de visibilidade	109
12.6 Funções show() e hide()	110
12.6.1 Função Toggle().....	112
12.6.2 Função delay().....	113
12.7 Funções de Opacidade.....	114
12.7.1 Funções FadeIn() e Fadeout()	115
12.7.2 FadeTo();	116
12.7.3 FadeToggle()	117
13.0 Biblioteca jQueryUI	119
13.1 Instalação e configuração	119
13.2 Datepicker.....	119
13.3 Accordion	120
13.4 Tabs	121
13.5 Menu	123
13.6 Tooltip	124
1.0 PHP	126
1.1 O que é PHP	126
1.1.1 Um pouco da História do PHP.	127
1.2 Instalação do Servidor PHP	128
1.2.1 Instalação Apache.....	128
1.2.2 Instalação Php5.	129
1.3 Características de um programa PHP	131
1.3.1 Imprimindo algo no navegador WEB via PHP.....	132
1.3.2 Comentários PHP	132
2.0 Iniciando um projeto com netbeans.	134
2.1 Tipos Primitivos	136
2.1.1 Tipos de Primitivos de Dados	136
3.0 Atribuições em PHP	139
3.1 Variáveis.....	139
3.2 Tipos de Variáveis	141
3.3 Constantes	144
3.3.1 Constantes pré-definidas	144
3.3.2 Definindo constantes	144
3.4 Conversão de variável	145
4.0 Operadores em PHP	148
4.1 Operadores de strings	148
4.2 Operadores Matemáticos	148
4.2.1 Aritméticos	148
4.2.2 Atribuição	149
4.2.3 Operadores de decremento e incremento	151
4.3 Operadores relacionais.....	152
4.4 Operadores lógicos ou booleanos	153
4.5 Precedência de Operadores	156

5.0 Interações PHP com HTML.....	159
5.1 Formulários	159
5.2 Exemplo de formulário.	161
5.3 Métodos Post e Get	164
5.4 Método Get.....	164
5.5 Método Post	166
6.0 Estruturas de controle e repetição	168
6.1 Blocos de controle.....	168
6.2 If e else	169
6.3 Atribuição condicional (ternário)	170
6.4 Switch	171
6.4.1 Switch com break	172
6.5 While	173
6.6 Do...while	175
6.7 For	175
6.8 Foreach	177
6.9 Break	178
6.10 Continue	179
7.0 Manipulação de arrays	184
7.1 Criando um Array.....	184
7.2 Arrays Associativos.....	185
7.3 Interações	186
7.4 Acessando um Array	187
7.5 Alterando um Array.....	188
7.6 Arrays multidimensionais.....	189
7.7 Funções com Arrays	192
<i>Exercício resolvido com array.....</i>	196
8.0 Manipulação de funções.....	198
8.1 Declarando uma Função.....	198
8.2 Escopo de Variáveis em Funções	199
8.3 Passagem de Parâmetro.....	200
8.4 Valor de Retorno.....	201
8.5 Recursão.....	202
<i>EXERCÍCIO RESOLVIDO COM FUNÇÕES.</i>	203
9.0 Manipulação de arquivos e diretórios	205
9.1 Criando e Abrindo um Arquivo.....	205
9.2 Gravando em um arquivo.....	206
9.3 Fechando um arquivo.....	207
9.4 Lendo um arquivo.....	208
9.5 Copiando,Renomeando e Apagando um Arquivo	209
9.6 Manipulando Diretório.....	210
9.7 Interações com o Browser.....	211
9.10 Cookies.....	212
9.11 Sessão	216
9.12 Requisição de Arquivos.....	218
9.13 Tratamentos de erro.....	219
1.0 Introdução a Banco de Dados	223
1.1 Banco de dados? ou SGBD/SGDB? Ou SQL?	223
1.2 Entendendo o que é Banco de Dados ou Base de Dados.....	224
1.3 Sistema Gerenciador de Banco de Dados (SGBD)	225
1.4 Banco de dados X SGBD	225
1.5 Linguagem SQL - Structure Query Language	226

1.6 Anatomia de um Banco de Dados	227
Exercícios Propostos	228
2.0 Introdução ao MySQL.....	230
2.1 Um pouco da história do MySQL	230
2.2 Características	231
2.2.1 Portabilidade	231
2.2.2 Capacidades.....	231
2.2.3 Multithreads	231
2.2.4 Licença de uso	231
2.2.5 Formas de armazenamento.....	232
2.2.6 Suporta triggers	232
2.2.7 Suporta cursors (Non-Scrollable e Non-Updatable);	232
2.2.8 Suporta stored procedures e functions;	232
2.2.9 Suporte a replicação	232
2.2.10 Suporte a clusterização.....	233
2.2.11 Visões	233
Exercícios Propostos.....	233
3.0 Instalando o Mysql Server	234
3.1 Instalação no Linux	234
3.2 Instalação no Windows	239
4.0 Bases de dados MySQL	252
4.1 Criando uma base de dados.....	252
4.2 Listando base de dados.....	253
4.3 Ativando/Selecionando uma base de dados	253
4.4 Deletando uma base de dado	254
4.5 Alterando uma base de dados	254
4.6 Conjunto de caracteres	254
4.6.1 Conjuntos de caracteres e collations em Geral	254
4.6.2 Conjunto de caracteres e collations no MySQL.....	255
4.6.3 Conjunto de caracteres e collation de banco de dados	256
Exercícios Propostos.....	257
5.0 Manipulando tabelas	258
5.1 Criando tabelas no MySQL Server	258
5.2 Listando tabelas no MySQL Server	259
5.3 Alterando uma tabela no MySQL Server	260
5.3.1 Renomeando uma tabela com o comando ALTER TABLE.....	260
5.3.2 Adicionando uma nova coluna com o comando ALTER TABLE.....	260
5.3.3 Removendo uma coluna com o comando ALTER TABLE.....	261
5.3.4 Renomeando um campo com o comando ALTER TABLE.....	261
5.4 Deletando uma tabela no MySQL Server.....	261
5.5 Saiba mais...	262
5.5.1 Tipos de dados do MySQL.....	262
5.5.2 Tipo de tabelas/ Storage Engine/ Motor de Armazenamento do MySQL.....	265
Exercícios Propostos	267
6.0 CRUD.....	269
6.1 CRUD (Create, Read, Update e Delete).....	269
6.2 INSERT	269
6.3 SELECT	270
6.4 UPDATE	270
6.5 DELETE.....	271
Referências	273

Apresentação

O manual apresenta aulas práticas e conceitos importantes para o entendimento na prática, está distribuído em três partes.

Elaborado no intuito de qualificar o processo de formação, este Manual é um instrumento pedagógico que se constitui como um mediador para facilitar o processo de ensino-aprendizagem em sala de aula.

1^a Parte

Apresenta alguns fundamentos de JavaScript uma linguagem de programação para navegadores, onde será muito importante os conhecimentos adquirido em lógica de programação para facilitar o entendimento, neste capítulo veremos operadores da linguagem, estruturas de controle, funções, eventos e objetos array, bem como exercícios e mais exemplos inclusive de orientação a objetos usando a linguagem javascript.

2^a Parte

Introdução a linguagem de programação PHP, conhecendo seus operadores, estruturas de controle, definição de funções e arrays, onde logo podemos fazer a integração da linguagem com o banco de dados mysql. Framework Jquery uma biblioteca JavaScript cross-browser desenvolvida para simplificar os scripts client side que interagem com o HTML, neste capítulo também abordaremos JqueryUI e Jquery.

Formulários web, neste capítulo iremos desenvolver alguns formulários web, para aplicarmos os conhecimentos que estamos adquirindo no decorrer do processo de aprendizagem.

3^a Parte

Neste capítulo vamos trabalhar entender um pouco sobre o que é um banco de dados e logo iremos introduzir os trabalhos utilizando o SGBD MySQL que irá ser melhor abordado em sua continuação no próximo semestre.

Este material teve como grande contribuição e referência para alguns capítulos o material do projeto MEDIOTEC – SEDUC, como apostilas do módulo II e III, em uma verificação e adaptação do material, por ser um material muito importante e de altíssima qualidade, aproximando bastante a experiência à prática do técnico em informática na área de desenvolvimento para web.

Outras referências de pesquisa e adaptação, foram de matérias de sites e artigos confiáveis da internet.

O material em questão já havia sido desenvolvido pela antiga turma de produção dos materiais, onde há sempre a necessidade de atualizações e ajustes para a demanda e o novo perfil de profissional que o mercado necessita.

Por isso faça um bom proveito dos conhecimentos aqui destacados, e que você possa ir muito mais além, com práticas e vivências no dia a dia para a sua preparação no seu processo de aprendizagem.

1.0 Introdução à JavaScript

JavaScript é uma linguagem para auxílio na criação de Home-Pages, as funções escritas em JavaScript podem ser embutidas dentro de seu documento HTML, possibilitando o incremento das funcionalidades do seu documento HTML com elementos interessantes. Sendo possível: responder facilmente a eventos iniciados pelo usuário, incluir efeitos que tornem sua página dinâmica. Logo, podemos criar sofisticadas páginas com a ajuda desta linguagem.

1.1 Introdução ao JavaScript



JavaScript é uma linguagem de programação voltada para ambiente web que tem como objetivo "interagir" com a página HTML.

As páginas HTML foram construídas após a criação da internet com o objetivo de divulgar notas científicas e artigos técnicos entre as universidades norte-americanas.

A metodologia aplicada no desenvolvimento e criação da linguagem HTML informava que o conteúdo das páginas HTML seria estático. Eles não tinham a visão que a internet teria um papel tão grande no amadurecimento da sociedade.

Inicialmente JavaScript não era JavaScript, mas sim LiveScript, uma tecnologia criada pela Netscape, quando ela estava desenvolvendo a versão

2.0 de seu navegador. Ela se destacou principalmente pelo fato de não precisar ser compilada (interpretada), sendo adicionada no documento HTML e de seus scripts não serem lentos, uma vez que eles não precisavam ser executados no servidor, mas sim no PC do usuário que estava acessando a página.

Na época, não houve muito interesse pelo LiveScript, pois havia um grande marketing em relação a linguagem Java, criada pela Sun Microsystems há 3 anos e que oferecia um potencial muito maior que o LiveScript.

Para aproveitar deste marketing, a Netscape tornou o Netscape 2.0 compatível com a linguagem Java e deu apoio a Sun Microsystems para reestruturar o LiveScript de acordo com a linguagem Java. Naquele momento, o LiveScript se tornou JavaScript.

A consequência desta mudança foi em criar uma linguagem de script que conseguiu unir os dois

universos: um simples como LiveScript e outro robusto, poderoso e "conhecido" como Java. A partir da versão 2.0 do Netscape e da versão 3.0 do Internet Explorer, todo navegador web suporta JavaScript e desde então o JavaScript tem ganhado grandes evoluções. Hoje o JavaScript é um padrão aberto e empresas de todo o mundo o suportam.

1.2 Java é diferente de JavaScript

Apesar dos nomes bem parecidos, Java não é o mesmo que JavaScript. Estas são duas técnicas diferentes de programação na Internet. Java é uma linguagem de programação. JavaScript é uma linguagem de hipertexto.

A linguagem de programação JavaScript, desenvolvida pela Netscape, Inc., não faz parte da plataforma Java.

JavaScript não cria applets ou aplicativos independentes. Em sua forma mais comum hoje, JavaScript reside dentro de documentos HTML e pode fornecer níveis de interatividade com páginas da Web que não podem ser conseguidos com HTML simples.

As diferenças principais entre Java e JavaScript estão listadas.

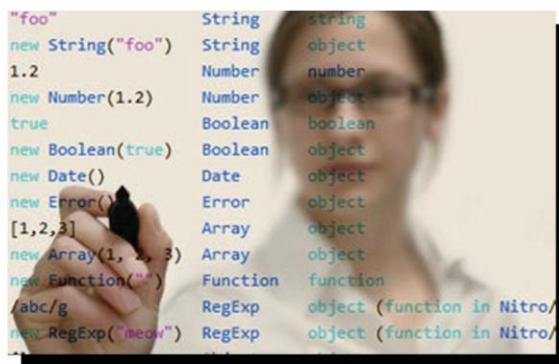
Java é uma linguagem de programação OOP, ao passo que Java Script é uma linguagem de scripts OOP.

Java cria aplicativos executados em uma máquina virtual ou navegador, ao passo que o código JavaScript é executado apenas em um navegador.

O código Java precisa ser compilado, ao passo que os códigos JavaScript estão totalmente em texto, apenas interpretado.

Eles requerem plug-ins diferentes.

1.3 Entendendo o JavaScript

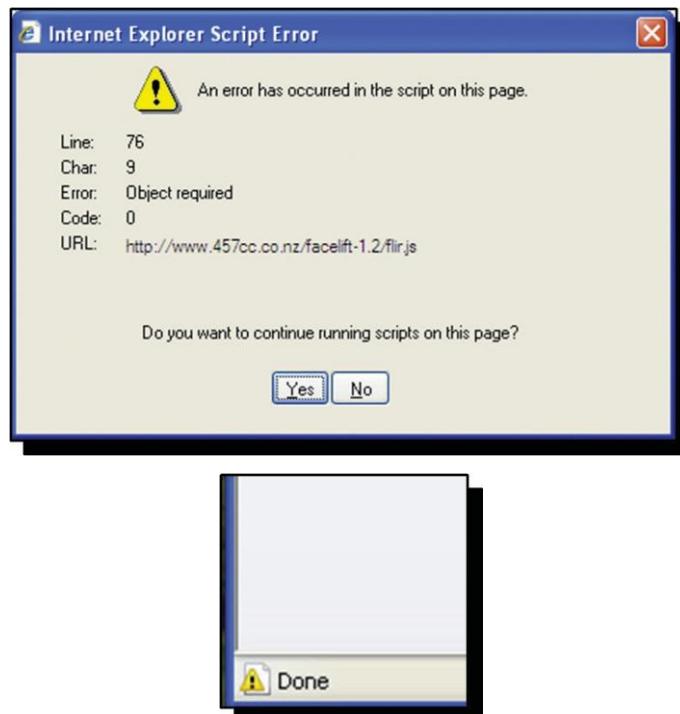


Por que o JavaScript é tão rápido? Acredito que esta seja uma das perguntas que a maioria dos desenvolvedores fazem. Para toda uma pergunta simples há uma resposta simples: Ela não é processada no servidor!

JavaScript não é processada no servidor por que ela não é uma linguagem compilada, mas sim interpretada. Quem a interpreta não é o servidor, mas sim o navegador do usuário que acessa a página

HTML. Então, todo o processamento dos scripts em JavaScript fica por conta do PC do usuário. Quanto mais rápido ele for, mais rápido será o processamento.

Para ter certeza disso, execute uma página HTML (que execute uma função em JavaScript) que está no seu próprio PC e acesse esta mesma página em um servidor web. Você irá perceber que a diferença de performance é muito pequena, pois o tempo que o navegador irá perder será o de baixar a página HTML, juntamente com suas dependências (em JavaScript ou não) para depois executá-la.



Já são embutidas no navegador do PC do usuário as bibliotecas de run-time para que ele entenda os scripts em JavaScript. Caso você use um navegador anterior ao Netscape 2.0 e ao Internet Explorer 3.0, provavelmente você não irá conseguir executar os scripts escritos em JavaScript, tampouco em qualquer outra linguagem. Erros no JavaScript

Como o JavaScript é uma linguagem interpretada, todo e qualquer erro será detectado na hora em que o navegador estiver lendo e/ou executando os scripts em JavaScript. Para verificar se seu script está errado ou houve algum erro de execução, basta verificar a barra de status do seu navegador. Caso ela tenha um ícone de aviso, informando um texto parecido ou semelhante com "Erro na consulta", clique duas vezes no ícone de aviso para visualizar os erros.

1.4 Usando JavaScript

Um código JavaScript pode ser inserido em um documento HTML de duas formas:

Colocando o código JavaScript como filho de um elemento com a tag script;

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="utf-8">
    <title>
        Inserindo código JS em um documento HTML
    </title>
    <script type="text/javascript" src="codigo.js">
    </script>
</head>

<body>

</body>
</html>
```

Utilizando o atributo src de um elemento com a tag script no qual devemos passar o caminho relativo ou absoluto para um arquivo que contenha o código JavaScript.

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="utf-8">
    <title>
        Inserindo código JS em um documento HTML
    </title>
    <script type="text/javascript">
        window.onload=function() {
            document.getElementById("ola-mundo")
            innerHTML='Olá Mundo!';
        }
    </script>
</head>

<body>
    <p id="ola-mundo"> </p>
</body>
</html>
```

Exercícios Propostos

Como aconteceu a evolução da tecnologia JavaScript no decorrer dos anos?

Qual a diferença entre Java e JavaScript?

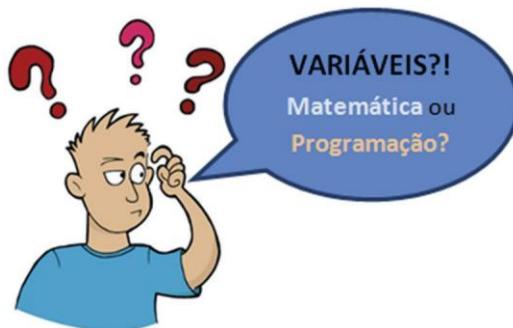
Quais as formas de se trabalhar com JavaScript junto com html. Dê exemplos.

Como funciona o processamento do JavaScript em uma página web?

Qual a importância do JavaScript em um projeto Web?

Pesquise e cite quais frameworks para JavaScript existem atualmente no mercado.

2.0 Variáveis



Sim, variáveis.

Na programação, a utilização da matemática é constante e é muito importante que você tenha um bom raciocínio lógico. Lembra daquela matéria de lógica matemática que você estudou no ensino fundamental e médio, achando que nunca iria utilizar o que estava aprendendo? Adivinha, vamos precisar daqueles conhecimentos agora!

2.1 Mas o que são variáveis em programação?



Vamos entender variável, como uma caixa, na qual você pode dar o nome que lhe achar conveniente, e guardar o conteúdo que desejar.

Ou seja, toda variável tem um nome, valor e tipo.

2.2 Mas o que seria o tipo de uma variável?

O tipo é uma classificação que damos a variável, essa classificação informa qual forma de dado se encontra ali armazenado.

As variáveis, podem ser classificadas com os tipos:

- Inteiro: Todo e qualquer dado numérico que pertença ao conjunto de números inteiros relativos (negativo, nulo ou positivo). Exemplos: {...-4,-3,-2,-1,0,1,2,3, 4...}.
- Real: Todo e qualquer dado numérico que pertença ao conjunto de números reais (negativo,

nulo ou positivo). Exemplos: {15.34; 123.08; 0.005 -12.0; 510.20}.

- Numérico: Trata-se de todo e qualquer número que pertença ao conjunto dos inteiros ou reais, também abrange números binários, octal e hexadecimal.
- Caracteres: são letras isoladas. Exemplo: {'a', 'd', 'A', 'h'}.
- Dados literais: Conhecido também como Conjunto de caracteres ou String, é todo e qualquer dado composto por um conjunto de caracteres alfanuméricos (números, letras e caracteres especiais). Exemplos: {"Aluno Aprovado", "10% de multa", "Confirma a exclusão ??", "S", "99-3000-2", "email", "123nm", "fd54fd"}.
- Lógico: A existência deste tipo de dado é, de certo modo, um reflexo da maneira como os computadores funcionam. Muitas vezes, estes tipos de dados são chamados de booleanos, devido à significativa contribuição de Boole a área da lógica matemática. A todo e qualquer dado que só pode assumir duas situações dados biestáveis, algo como por exemplo {0/ 1, verdadeiro/falso, sim/não, true/false}.

Então, o tipo de uma variável é determinado pelo conteúdo que ela guarda. Se uma variável armazena um número inteiro ela é do tipo inteiro, se é um conjunto de caracteres ela é do tipo String.

Dependendo da linguagem de programação que você estiver programando, os nomes dos tipos listados acima podem mudar e novos tipos também poderão surgir.

2.3 Nomeando variáveis

No tópico anterior vimos que as varáveis precisam de nome, tipo e valor. Já estudamos um pouco sobre tipos de variáveis, agora iremos discutir sobre como criar um nome ou **identificador** válido para uma variável.

As regras para se criar um identificador de variável podem sofrer algumas alterações dependendo da linguagem de programação adotada vamos listar as regras que geralmente são adotadas pela maioria:

Os identificadores são **case sensitive**, ou seja, faz distinção entre maiúsculas e minúsculas. Isso significa que um nome de variável, como **meuContador**, é diferente do nome de variável **MEUCountador**;

Nomes de variáveis podem ser de qualquer comprimento, em algumas linguagens existe limite no número de caracteres que compõe o identificador, por essa e pela questão de legibilidade tente não ultrapassar 15 caracteres.

Crie identificadores que informe explicitamente para que aquela variável irá servir na composição do seu programa, ou seja se uma variável vai armazenar um valor inteiro que irá aumentar de um em um até chegar 100, coloque o nome dessa variável de **contador**

O primeiro caractere deve ser uma letra ASCII (em maiúscula ou minúscula) ou um caractere de sublinhado (_). Observe que um número não pode ser usado como o primeiro caractere.

Os caracteres subsequentes devem ser letras, números ou sublinhados (_).

O nome da variável não deve ser uma **palavra reservada**. Existem palavras que são utilizadas pela linguagem de programação, que você está utilizando para implementar seus programas, para controlar e estruturar seu algoritmo. Essas palavras classificadas com o nome de **palavras reservadas**.

Questão: Quais dos identificadores de variáveis abaixo são válidos?

_pagecount

99Balloons

Number_Items

Alpha&Beta

Part9

2.4 Declarando ou criando uma variável

Declarar uma variável tecnicamente é pedir ao sistema operacional um espaço exclusivo na memória RAM do seu computador para armazenar uma informação.

Abaixo temos exemplos de declaração e inicialização de variáveis em linguagens de programação distintas:

```
String nome = new String("Jonas");
int idade = 25;
char opcao = 'S';
Inteiro quantidade = 100;
contador = 1;
modeloCarro = "Celta";
```

Perceba que dependendo da linguagem de programação processo de declaração é diferente.

2.5 Linguagens de Programação Fracamente e Fortemente Tipadas

Linguagem fortemente tipadas(LST), são linguagens de programação que cada variável do programa, representa um tipo bem definido, ou seja explicitamente você será obrigado a declarar o tipo da variável ao qual estará declarando e caso tente fazer operações com variáveis de tipos diferentes precisará fazer conversões, caso contrário irá acontecer erros em seu código.

Exemplo de declaração de variáveis fortemente tipadas:

Inteiro contador = 1;

Caracter sexo = 'F';

String nome = "Jonas";

Já nas linguagens de programação que são classificadas como Linguagem de Programação Fracamente

Tipada(LWT) ou de tipagem dinâmica, não precisamos colocar o tipo da variável antes de seu identificador, além do tipo da variável ser modificado em tempo de execução sem precisar fazer conversão, ou seja o tipo da linguagem é modificada automaticamente segundo o valor que você armazena na mesma.

idade = “18” - (tipo String);

idade = 18 - (agora a variável idade é do tipo inteiro);

2.6 Variáveis em JavaScript

Como a linguagem de programação escolhida para se aprender lógica de programação foi o JavaScript, vamos estudar nesse tópico como funciona a manipulação de variáveis nessa linguagem.

2.6.1. Declarando ou criando uma variável JavaScript

Diferente da maioria das linguagens o JavaScript define as variáveis dinamicamente, portanto ao atribuir uma variável ele escolhe o tipo conforme o valor passado para a variável, não sendo necessário especificar o mesmo.

Existem dois tipos de sintaxe de declaração de variável em JavaScript que são:

```
var nome-da-variável = valor-da-variável;      ou      nome-da-variável = valor-da-
```

Lembre-se que nome-da-variável precisa seguir todas as regras do tópico nomeando variáveis. Abaixo palavras reservadas da linguagem de programação JavaScript:

Palavras-chave em JavaScript				
break	case	catch	continue	default
delete	do	else	false	finally
for	function	if	in	instanceof
new	null	return	switch	this
throw	true	try	typeof	var
void	while	with		

Palavras-chave que são reservadas, mas não usadas pelo JavaScript				
abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
import	int	interface	long	native
package	private	protected	public	short
static	super	synchronized	throws	transient
volatile				

2.7 Tipos de variáveis JavaScript

Como já mencionado, no JavaScript não declaramos explicitamente o tipo de uma variável. Por exemplo, se a variável recebe um número inteiro declaramos, por exemplo, *int x*, ou se ela recebe um número real declaramos *float y*, mas no JavaScript isso não acontece. O tipo de uma variável é definido dinamicamente, não precisando o desenvolvedor especificar o seu tipo.

Veremos nos tópicos abaixo como trabalhar com os tipos de variáveis JavaScript.

2.7.1 Trabalhando com String e Números em JavaScript

Vamos ver como declaramos Número e String no JavaScript abaixo:

```
<script>
    var variavel = 10;           //Inteiro
    alert(variavel);

    var variavel2 = "João";      //String
    alert(variavel2);

    var variavel3 = 'Cristiane'; //String
    alert(variavel3);
</script>
```

Quando declaramos uma String utilizamos aspas duplas (") ou simples ('). Você pode declarar número utilizando aspas, mas não é obrigatório. Se utilizarmos no início de uma String a ("") temos que finalizarmos ela com a (""), isso vale para quando utilizamos (').

2.7.2 Trabalhando com outros tipos de dados JavaScript

Agora, vamos descrever outros tipos de dados ou variáveis suportadas no JavaScript.

Booleanos: suportam dois valores true/false (verdadeiro ou falso). Exemplo:

```
<script>
|   var variavel = false //Booleano
|   var variavel = true //Booleano
</script>
```

Constante: Valor que não se altera no decorrer do código. Exemplo:

```
<script>
|   const variavel = 7;
|   alert(variavel);
</script>
```

Undefined: quando você tenta acessar um atributo de um objeto que não existe ou uma variável que ainda não foi inicializada o JavaScript define o valor dela como undefined.

Null: é um tipo especial de valor que indica para o navegador que a variável está vazia. O valor null que significa ausência de valor, vazio ou nada quando atribuído a uma variável está definindo-a como vazia. A diferença do null para undefined é que se uma variável tem o valor null ela está definida.

Array: o array é um tipo composto de dado. O array é uma estrutura de dados que possibilita guardar vários dados e indexá-los utilizando índices. Ou seja, uma variável que pode conter vários valores diferentes.

```
<script>
|   var variosTipos = ["e-Jovem", 10, [1,2]];
</script>
```

Object: no JavaScript também podemos trabalhar com variáveis do tipo objeto, que são variáveis robustas onde podemos guarda desde um único valor até um programa completo. Iremos discutir sobre esse tipo em cursos posteriores. Veja abaixo um exemplo de criação de uma variável do tipo objeto no JavaScript.

```

<script>
    nome = new Array(3);
    nome[0] = "André ";
    nome[1] = "Thiago ";
    nome[2] = "Nardy";
    document.write("Meu nome é " + nome[0] + nome[
        1] + nome[2])
</script>

```

Exemplo:

2.8 Variáveis Globais e Locais

As variáveis são classificadas em dois tipos em relação a sua área de atuação, que são as globais e as locais. A diferença entre elas é:

Variável Global: Criada ou declarada fora de uma função, portanto podem ser utilizadas a qualquer momento no seu script;

Variável Local: Criada ou declarada dentro de uma função, portanto só podem ser utilizadas dentro da função criada.

Funções são bloco de códigos que são executados todas as vezes que invocados por um identificador.

Iremos discutir mais sobre funções em capítulos posteriores.

Exercícios Propostos:

Qual a principal finalidade de uma variável?

O que significa tipagem dinâmica.

Das variáveis abaixo, quais possuem nomenclaturas válidas na linguagem JavaScript.

a____b;	a_1_;	_início;	@nome;	val_!;
nome;	a_ _;	#valor;	palavra;	tele#;
123;	_ = ;	VALOR_MAIOR;	_____;	all;

Crie dez variáveis atribuindo valores diversos, logo após use o comando **document.write()** pra imprimir na tela do browser, exemplo:

```

<script>
var nome = "Maria Cavalcante"; document.write(nome);
</script>

```

Quais os tipos de variáveis que podemos citar em JavaScript.

Como podemos distinguir um tipo de variável de outro, uma vez que a tipagem é feita de forma dinâmica no JavaScript.

Qual a principal finalidade de um constante e como elas são definidas em JavaScript.

Em que momentos precisamos converter uma variável de um tipo em outro. Crie uma constante e imprima com o comando `document.write()`;

3.0 Operadores em JavaScript

Neste capítulo iremos estudar os tipos e quais os operadores; Falar do conceito de atribuição e concatenação de String; exemplificar os operadores, sua importância e funcionamento

Os operadores têm seu papel importante dentro de qualquer linguagem de programação. É através deles que podemos realizar diversos operações dentro de um programa, seja ela de atribuição, aritmética, relacional, lógico, dentre outros. Em JavaScript não é diferente, os operadores são utilizados constantemente, porém existem algumas regras que veremos mais adiante.

3.1 Operadores de strings

São operadores utilizados para unir o conteúdo de uma string a outra, com isso podemos dizer que há dois operadores de string. O primeiro é o operador de concatenação ('+') que já utilizamos em exemplos anteriores, ele retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação ('+='), que acrescenta o argumento do lado direito no argumento do lado esquerdo.

Observe o exemplo abaixo:

Nesse exemplo pode-se observar a declaração da variável **d**, logo após temos uma inicialização e atribuição de concatenação em uma mesma linha, isso é possível em JavaScript, deixando o código mais otimizado, porém menos legível.

3.2 Operadores Matemáticos

3.2.1 Aritméticos

Chamamos de **operadores aritméticos** o conjunto de símbolos que representa as operações básicas da matemática.

Operações	Operadores	Exemplo	Resposta
Adição	+	$3 + 5$	8
Subtração	-	$20 - 5$	15
Multiplicação	*	$7 * 8$	56
Divisão	/	$15 / 3$	5
Módulo (Resto da divisão)	%	$10 \% 3$	1
Negação (Número Posto)	-(valor)	Se for 15 a atribuição	-15

3.2.2 Atribuição

O operador básico de atribuição é "=" (igual). Com ele podemos atribuir valores as variáveis como foi visto em exemplos anteriores. Isto quer dizer que o operando da esquerda recebe o valor da expressão da direita (ou seja, "é configurado para"). Mas podemos usar algumas técnicas, observe o exemplo abaixo:

Resultado: a= 9, b=4

Além do operador básico de atribuição, há "operadores combinados" usados para array e string, eles permitem pegar um valor de uma expressão e então usar seu próprio valor para o resultado daquela expressão.

Por exemplo:

```
<script>
    //EXEMPLO 01
    a = 3;
    a *= 5; /* mesmo que: a = a*5, ou seja,
               a recebe seu valor anterior(3)
               e multiplica este valor por 5 */
    document.write("a = "+a);
    //EXEMPLO 02
    b= "Bom ";
    b+="Dia!";
    document.write(" b = "+b);
</script>
```

Resultado: a = 15, b = Bom Dia!

Observe a expressão: a = 3 e logo após a*=5. Isto significa a mesma coisa de a = a * 5, ou, a = 3 *5. A ideia pode ser usada para string, como foi feito com a variável b, onde b = "Bom", logo após usamos ponto(+) e igual(=) para concatenar os valores, ficando assim: b+="Dia!". Lembrando que isso significa a mesma coisa que b = b+"Dia". Observe mais um exemplo:

```
<script>
    a = "Dia ";
    b = "Bom ";
    b += a +="turma!"; //concatena 'a', depois 'b'
    document.write(" b = "+b);
</script>
```

Resultado: Bom Dia turma!

Podemos definir uma sequência com duas concatenações, onde `a = "Dia"+“turma!”` e logo após temos `b = “Bom”+“Dia turma!”`.

Os operadores de atribuição são usados para economizar linhas de código, deixando assim o código mais funcional e otimizado. A tabela abaixo mostra os principais operadores de atribuição:

Operadores	Descrição
<code>=</code>	<i>Atribuição simples.</i>
<code>+=</code>	<i>Soma, depois atribui. Quando usado com Strings concatena.</i>
<code>-=</code>	<i>Subtrai, depois atribui.</i>
<code>*=</code>	<i>Multiplica, depois atribui.</i>
<code>/=</code>	<i>Divide, depois atribui.</i>
<code>%=</code>	<i>Modulo(resto) da divisão, depois atribui.</i>

Observe um exemplo aplicando os operadores.

```
<script type="text/javascript">
    a = 8;
    // 'a' recebe seu valor anterior(8) e multiplica por 3;
    document.write( a*= 3);
    document.write( "<br/>");

    // 'a' recebe seu valor anterior(24) e divide por 2;
    document.write( a/= 2);
    document.write( "<br/>");

    // 'a' recebe o resto da divisão de 'a'(12) por 5;
    document.write( a%= 5);
    document.write( "<br/>");

    // 'a' recebe seu valor anterior(2) e soma com 3;
    document.write( a+= 3);
    document.write( "<br/>");

    // 'a' recebe seu valor anterior(5) e subtrai 1;
    document.write( a-= 1);
    document.write( "<br/>");
</script>
```

Resultado : 24 - 12 - 2 - 5 - 4

Vale ressaltar que a cada `document.write()`, o valor de `a` sofre modificações. Isso devido a atribuição feita após a operação. Usamos o operador ponto (+) para concatenar os valores obtidos com `
` código usado em HTML para quebra de linha.

3.3 Operadores de decremento e incremento

São operadores usados para atribuir em 1 ou -1 a variável, isso pode ser feito antes ou depois da execução de determinada variável. A tabela abaixo mostra tais operadores:

Operadores	Descrição
++a	<i>Pré-incremento. Incrementa a em um e, então, retorna a.</i>
a++	<i>Pós-incremento. Retorna a, então, incrementa a em um.</i>
--a	<i>Pré-decremento. Decrementa a em um e, então, retorna a</i>
a--	<i>Pós-decremento. Retorna a, então, decrementa a em um</i>

Exemplo:

```
<script type="text/javascript">
    a = 1;
    // 'a'(1) recebe o incremento de 1, e depois imprime seu valor(2);
    document.write( ++a);
    document.write( "<br/>");

    // 'a'(2) imprime primeiro seu valor(2), depois recebe o incremento de 1;
    document.write( a++);
    document.write( "<br/>");

    // 'a'(3) recebe o decremento de 1, e depois imprime seu valor(2);
    document.write( --a);
    document.write( "<br/>");

    // 'a'(2) imprime primeiro seu valor(2), depois recebe o decremento de 1;
    document.write( a--);
    document.write( "<br/>");

    // valor final de 'a'
    document.write(a);
</script>
```

Resultado : 2

2 2 2 1

Nesse exemplo temos uma forma aplicada do uso de decremento e incremento, lembrando que a variável a pode ter qualquer nome. Também podemos fazer um comparativo com o pré-incremento ou incremento-prefixado com operações que já conhecemos, observe:

Operador	Forma extensa.	Forma simplificada
++a	$a = a + 1$	$a += 1$
--a	$a = a - 1$	$a -= 1$

3.4 Operadores relacionais

Os operadores relacionais ou conhecidos também como operadores de comparação, são utilizados para

fazer determinadas comparações entre valores ou expressões, resultando sempre um valor booleano verdadeiro ou falso(TRUE ou FALSE). Para utilizarmos esses operadores usamos a seguinte sintaxe:

(valor ou expressão) + (comparador) + (segundo valor ou expressão)

Observe a tabela abaixo:

Comparadores	Descrição
==	Igual. Resulta em TRUE se as expressões forem iguais.
==	Idêntico. Resulta em TRUE se as iguais e do mesmo tipo de dados.
!=	Diferente. Resulta verdadeiro se as variáveis foram diferentes.
<	Menor ou menor que. Resulta TRUE se a primeira expressão for menor.
>	Maior ou maior que. Resulta TRUE se a primeira expressão for maior.
<=	Menor ou igual. Resulta TRUE se a primeira expressão for menor ou igual.
>=	Maior ou igual. Resulta TRUE se a primeira expressão for maior ou igual.

Veja um exemplo prático:

a <= b

Compara se a é menor ou igual a b, onde, retorna verdadeiro (TRUE), caso contrário retorna falso (FALSE).

Para testarmos essas comparações podemos utilizar o condicional “?:” (ou **ternário**), sua sintaxe é a seguinte:

(expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);

Agora podemos ver um exemplo envolvendo as sintaxes e empregabilidade dos comparadores:

```
<script type="text/javascript">
    var a = 15;
    var b = "42";
    var c = 42.0;

    document.writeln(b == c ? "verdadeiro" : "falso"); // VERDADEIRO
    document.writeln(b === c ? "verdadeiro" : "falso"); // FALSO
    document.writeln(b != c ? "verdadeiro" : "falso"); // FALSO
    document.writeln(a < c ? "verdadeiro" : "falso"); // VERDADEIRO
    document.writeln(a > c ? "verdadeiro" : "falso"); // FALSO
    document.writeln(a <= c ? "verdadeiro" : "falso"); // VERDADEIRO
    document.writeln(a >= c ? "verdadeiro" : "falso"); // FALSO
</script>
```

Nesse exemplo declaramos e iniciamos três variáveis. Usamos então o comando **document.write()** para imprimir o resultado, onde o condicional “?:” foi utilizado. Iniciamos as comparações de a, b e c, caso a comparação individual retorne TRUE, imprime verdadeiro, caso retorne FALSE, imprime falso. Observe que o comparador “==” compara o valor e o tipo, retornando FALSE por b se tratar de um tipo inteiro, e c um tipo ponto flutuante, já o comparador “==” compara somente os valores onde 45 é igual a 45.0 retornando verdadeiro. Também podemos usar o operador “!=” onde tem a função semelhantemente ao operador “!=”, mas retorna TRUE se os tipos forem diferentes. Se a variável for do tipo booleano, podemos compará-los assim:

a == TRUE, a == FALSE

3.5 Operadores lógicos ou booleanos

São utilizados para avaliar expressões lógicas. Estes operadores servem para avaliar expressões que resultam em valores lógico sendo verdadeiro ou falso:

E	&&
OU	
XOR	^
Não	!

Esses operadores tem a finalidade de novas proposições lógicas composta a partir de outras proposições lógicas simples. Observe:

Operador	Função
não	negação
e	conjunção
Ou exclusivo(xor)	disjunção exclusiva
ou	disjunção

Com isso podemos construir a Tabela verdade onde trabalhamos com todas as possibilidades combinatória entre os valores de diversas variáveis envolvidas, as quais se encontram em apenas duas situações (V e F), e um conjunto de operadores lógicos. Veja o comportamento dessas variáveis:

3.6 Operação de Negação:

A	(!) não A
---	-----------

F	V
V	F

Trazendo para o nosso cotidiano:

Considerando que A = “Está chovendo”, sua negação seria : “Não está chovendo”. Considerando que A = “Não está chovendo” sua negação seria : “Está chovendo”

3.7 Operação de conjunção:

A	B	A e B
F	F	F
F	V	F
V	F	F
V	V	V

Trazendo para o nosso cotidiano:

Imagine que acontecerá um casamento:

Considerando que A = “Noivo presente” e B = “Noiva presente”.

Sabemos que um casamento só pode se realizar, se os 2 estejam presente

3.8 Operação de disjunção

- **Não exclusiva**

A	B	A ou B
F	F	F
F	V	V
V	F	V
V	V	V

Trazendo para o nosso cotidiano:

Imagine que acontecerá uma prova e para realizá-la você precisará da sua Identidade ou título de eleitor no dia da prova.

Considerando que A = “Identidade” e B = “Título de eleitor”.

Sabemos que o candidato precisa de pelo menos 1 dos documentos para realizar a prova.

- Exclusiva**

A	B	A xor B
F	F	F
F	V	V
V	F	V
V	V	F

Trazendo para o nosso cotidiano:

Imagine que Cateriny nasceu em Fortaleza. Se for indagado para Cateriny se ela nasceu em Crato, Juazeiro do Norte, Sobral ou Fortaleza. Ela iria escolher apenas umas dessas opções que seria Fortaleza. Não há possibilidade de Cateriny nascer em mais de uma cidade.

Considerando que A = “Crato” e B = “Juazeiro do Norte” e C=“Sobral” e D=”Fortaleza”.

A expressão: A xor B xor C xor D só poderá ser verdadeira se tiver apenas uma resposta válida.

São chamados de operadores lógicos ou booleanos por se tratar de comparadores de duas ou mais expressões lógicas entre si, fazendo agrupamento de testes condicionais e tem como retorno um resultado booleano.

Na tabela abaixo temos os operadores e suas descrições:

Operador	Descrição
(a && b)	E : Verdadeiro se tanto a quanto b forem verdadeiros.
(a b)	OU : Verdadeiro se a ou b forem verdadeiros.
(a^b)	XOR: somente um pode ser verdade para que a expressão seja verdadeira(bit-a-bit)
(! a)	NOT : Verdadeiro se a for falso, usado para inverter o resultado da condição.

Exemplo:

```
<script type="text/javascript">
    var a = 10 > 2;    // 'a' recebe TRUE
    var b = 12 >= 12; // 'b' recebe TRUE
    var c = false;    // 'c' recebe FALSE

    document.writeln(b && a ? "SIM" : "NÃO"); // SIM
    document.writeln(b || c ? "SIM" : "NÃO"); // SIM
    document.writeln(b ^ a ? "SIM" : "NÃO"); // NÃO
    document.writeln((c!) ? "SIM" : "NÃO"); // SIM
</script>
```

3.9 Precedência de Operadores

Agora já conhecemos uma boa quantidade de operadores no JavaScript, falta agora conhecer a precedência de cada um deles, ou seja, quem é mais importante, qual operador é avaliado primeiro e qual

```
<script type="text/javascript">
    document.write(5+2*6); // Resultado: 17
</script>
```

é avaliado em seguida. Observe o seguinte exemplo:

O resultado será 17, pois o operador * tem maior precedência em relação ao operador +. Primeiro ocorre a multiplicação $2*6$, resultando em 12, em seguida a soma de $5 + 12$. Caso desejar realizar a operação com o operador + para só em seguida realizar a operação com o operador *, temos que fazer conforme o exemplo abaixo:

```
<script type="text/javascript">
    document.write((5+2)*6); // Resultado: 42
</script>
```

Observe que utilizamos os parênteses para determinarmos quem deve ser executado primeiro, assim alterando o resultado para 42. Os parênteses determinam qual bloco de código executa primeiro, e também serve para isolar determinadas operações. Veja mais um exemplo onde as operações são feitas separadamente. Primeiro executa a soma, em seguida a subtração e só então é executado a multiplicação, imprimindo um resultado final 21:

```
<script type="text/javascript">
    document.write((5+2)*(6-3)); // Resultado: 21
</script>
```

Exemplo:

A tabela a seguir lista os operadores JavaScript, ordenados da maior até a menor precedência. Operadores com a mesma precedência são avaliados da esquerda para a direita.

Operador	Descrição
. [] 0	Acesso a campos, indexação de matriz, chamadas de função e agrupamento de expressões
++ -- ~ ! delete new typeof void	Operadores unários, tipo de dados de retorno, criação de objetos, valores indefinidos
* / %	Multiplicação, divisão, divisão de módulo
+ - +	Adição, subtração, concatenação de cadeias de caracteres
<< >> >>	Deslocamento de bits
< <= > >= instanceof	Menor que, menor que ou igual a, maior que, maior que ou igual a, instância de
== != === !==	Igualdade, desigualdade, igualdade estrita e desigualdade estrita
&	AND bit a bit
^	XOR bit a bit
	OR bit a bit
&&	Lógico AND
	OU lógico
?:	Condicional
= OP=	Atribuição, atribuição com operação (como += e &=)
,	Avaliação múltipla

É importante lembrar que primeiro o JavaScript executará todas as operações que estiverem entre parênteses, se dentro dos parênteses houver diversas operações, a precedência dos operadores será utilizada para definir a ordem. Após resolver todas as operações dos parentes, o JavaScript volta a resolver o que está fora dos parênteses baseando-se na tabela de precedência de operadores. Havendo operadores de mesma prioridade o JavaScript resolverá a operação da esquerda para direita.

Também podemos trabalhar com procedência de parênteses, fazendo associações com um ou mais operadores, observe o seguinte exemplo:

```
<script type="text/javascript">
    document.write( (3+2)*(9-3) / ( 16-((5+2)*2) ) );
    // Resultado: 15
</script>
```

Seguindo a ordem de procedência temos:

(5) * (6) / (16 - ((7)*2)) >>> 5 * 6 / (16 - (14)) >>> 5 * 6 / 2 >>> 30 / 2

Resultado: 15

Observe que primeiro executa todos os parênteses, e só então temos as procedências das demais operações.

Exercícios Propostos

Qual a finalidade dos operadores de strings?

Quais os operadores de decremento e incremento? Cite alguns exemplos:

Qual a finalidade do operador aritmético %(modulo)?

Cite os operadores relacionais, mostre alguns exemplos.

Quais operadores lógicos ou booleanos?

Quais os operadores de atribuição?

Qual a sintaxe do uso de ternário e cite um exemplo?

Observe o código abaixo e diga quais das operações são executadas primeiro, coloque a resposta em ordem decrescente.

A = 8*5-3+4/2+19%5/2+1;

Faça testes com os operadores relacionais substituindo o operando > do código fonte abaixo:

<script>

var1 = 2.2564;

var2 = 2.2635;

document.write(var1 > var2 ? "sim" : "não");

</script>

2. Usando o operador de concatenação “+” para montar a seguinte frase abaixo:

<script>

a = “de”; b = “é um”; c = “comunicação”; d = “a”;

d = “internet”; e = “meio”;

document.write(.....);

</script>

4.0 Caixa de Diálogo

Um recurso interessante de JavaScript é a possibilidade de criar caixas de diálogo simples, que podem ser muito informativas aos usuários que a visualizam.

Essas caixas de diálogo podem ser de alerta, de confirmação ou de prompt de entrada. Todas elas são chamadas de forma simples e intuitiva por uma função.

4.1 Alert

As caixas de diálogo de alerta são simples e informativas. Elas, geralmente, são utilizadas em validação de formulários ou bloqueio de ações.

Sua função é mostrar apenas uma mensagem com um botão de confirmação para que esta seja fechada.

Para chamar esta caixa de diálogo usamos a função **alert()**. Esta função recebe como parâmetro uma string que será a mensagem a ser exibida. Vejamos o código abaixo:

```
<script type="text/javascript">
    alert ("Esta é uma caixa de diálogo ALERT do javascript!")
</script>
```

Em caixas de diálogo há a possibilidade de controlar o fluxo de texto usando \n para a quebra de linhas.

```
<script type="text/javascript">
    alert ("javascript \n Caixa de dialogo \n Alert")
</script>
```

4.2 Prompt

A caixa de diálogo de prompt nos possibilita requerer uma entrada ao usuário apesar de não ser tão útil, pois esse recurso pode facilmente ser substituído por um campo de texto feito em HTML.

Para chamarmos esta caixa de diálogo, usamos a função **prompt()** que recebe uma string como parâmetro. Esse parâmetro será a mensagem a ser exibida dentro da caixa de diálogo.

A caixa de diálogo de prompt possui três elementos: um campo input para texto, um botão OK e outro CANCELAR.

A função **prompt ()** sempre irá retornar um valor, ou seja, podemos gravar o resultado da função em uma variável ou algo assim. Se clicarmos no botão OK, o valor a ser retornado será o que estiver escrito no campo de texto, mesmo se ele estiver vazio. Se clicarmos em CANCELAR, o valor retornado será **null**.

Abaixo criamos um exemplo no qual exige que o usuário digite o nome dele. Para isso, colocamos o prompt dentro de uma estrutura de repetição *while* que tem a seguinte condição: se o resultado for null (ou seja, se o usuário clicar em cancelar), ou então, se o resultado for vazio (ou seja, se o usuário não digitar nada e clicar no OK), neste caso, deve-se executar a repetição.

Dessa forma nos asseguramos que o usuário sempre irá digitar alguma coisa dentro da caixa de diálogo.

```
<script type="text/javascript">
    var nome;
    do {
        nome = prompt ("Qual é o seu nome?");
    }while (nome == null || nome == "");
    alert ("Seu nome é: "+nome);
</script>
```

4.3 Confirm

A caixa de diálogo de confirmação é chamada pela função `confirm()` e tem apenas dois botões: um OK e outro CANCELAR. Assim como a função `prompt()`, a função `confirm()` também retorna um valor que pode ser true (verdadeiro) ou false (falso).

Como `confirm()` retorna um valor booleano, isso o torna ideal para ser usado com uma estrutura seletiva *if*. Por exemplo, podemos usar a caixa de diálogo de confirmação antes de redirecionarmos uma página para executar uma rotina para apagar algum registro do banco de dados.

No exemplo abaixo, não iremos tão profundamente quanto o cenário acima, pois envolve mais do que simples JavaScript. Aqui, apenas iremos demonstrar o resultado do clique em algum dos dois botões.

```
<script type="text/javascript">
    decisao = confirm("Clique em um botão!");
    if (decisao) {
        alert ("Você clicou no botão OK, \n"+
               "porque foi retornado o valor: "+decisao);
    } else{
        alert ("Você clicou no botão CANCELAR, \n"+
               "porque foi retornado o valor: "+decisao);
    };
</script>
```

Exercícios Propostos

Usando a instrução `alert`, faça um algoritmo que exibe para o usuário a mensagem: “Bom dia usuário.”.

Faça um algoritmo que solicite ao usuário que digite seu nome, após o usuário dar entrada da informação exiba a seguinte mensagem para o mesmo: “Bom dia nome_usuario.”. Onde há `nome_usuario` será

colocado o nome que o usuário digitou.

Faça um algoritmo que solicite o ano que o usuário nasceu e depois o ano atual, seu código irá pegar essas informações e calcular a idade do usuário e exibi-la na tela usando alert().

Faça um algoritmo que receba dois números inteiros, digitados pelo usuário, e exiba em um único alert() o resultado da soma, subtração, multiplicação e divisão entre os dois números digitados.

Faça um algoritmo que peça sua amado(a) em casamento ou namoro, utilize a instrução confirm(). Caso ele(a) aceite exiba a mensagem “Eu aceito”, caso contrário exiba a mensagem “Desculpa, não posso aceitar”.

Faça um algoritmo de validação de login. O usuário irá digitar o login e depois a senha. Se o login digitado é igual a “root” e a senha “qwe123”, então exiba a mensagem “Acesso permitido”, senão exiba “Login/Senha inválidos”;

Escreva um algoritmo que calcula quantos meses um funcionário trabalhou para uma empresa. Solicite o dia, mês e ano que um funcionário foi contratado e depois o dia, mês e ano que ele foi demitido. Calcule quantos meses esse funcionário trabalhou nessa empresa.

Escreva um algoritmo que receba a idade do usuário e se a idade for maior ou igual a 18 exiba a mensagem “Sou um adulto”, senão exiba a mensagem “Sou uma Criança/Adolescente” .

Escreva um algoritmo que solicite ao usuário uma lista de compras para se fazer no supermercado. Essa lista terá 10 produtos e após todos serem digitados exiba a lista em um só alert() utilizando essa formatação:

- 1 – arroz
- 2 – feijão
- 3 – macarrão
- 4 – farofa

Escreva um algoritmo que receba 3 valores inteiros que representam a tamanho dos lados de um triângulo. Usando o operador ternário exiba a mensagem “Sou equilátero” se todos os lados forem iguais, “Sou isósceles” se dois lados forem iguais e “Sou escaleno “, se todos os lados forem diferentes.

5.0 Estruturas de controle e repetição

Objetivos

Mostra estruturas de controle e sua aplicação prática em JavaScript; Definir qual a principal finalidade dessas estruturas; Mostrar exemplos em sua sintaxe;.

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando descrever a sintaxe de cada uma delas resumindo o funcionamento. Independente do JavaScript, boa parte das outras linguagens de programação tem estruturas bem semelhantes, mudando apenas algumas sintaxes.

5.1 Blocos de controle

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em JavaScript são delimitados pelos caracteres “{” e “}”. A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador. Outro detalhe importante: usar as estruturas de controle sem blocos delimitadores faz com que somente o próximo comando venha ter ligação com a estrutura. Observe os exemplos:

```
<script type="text/javascript">
    var a = 0;
    if (a>2)
        alert("comando 1");
        alert("comando 2");
</script>
```

Observe que temos um comando IF, onde é passado a ele uma expressão booleana que retorna verdadeiro ou falso.

O resultado da expressão é FALSE (falso), pois 0 não é maior que 2, fazendo com que o IF não execute o echo com “comando1”. Somente o segundo echo é executado, pois não pertence ao IF declarado.

Mas se quisermos que mais de um comando pertença a estrutura de controle, será usado blocos de comandos ({comando;}), onde através deles podemos delimitar e organizar os códigos.

```
<script type="text/javascript">
    var a = 0;
    if (a>2) {
        alert("comando 1");
        alert("comando 2");
    }
</script>
```

No código ao lado, temos um bloco onde inserimos dois comandos. Observe que eles não serão executados, pois a expressão booleana passada para o IF é falsa.

5.2 IF e ELSE

Essa estrutura condicional está entre as mais usadas na programação. Sua finalidade é induzir um desvio condicional, ou seja, um desvio na execução natural do programa. Caso a condição dada pela expressão seja satisfeita, então serão executadas a instruções do bloco de comando. Caso a condição não seja satisfeita, o bloco de comando será simplesmente ignorado. Em lógica de programação é o que usamos como “SE (expressão) ENTÃO {comando:}”.

–

```
if (expressão)
    comando;
if
(expressão) {
    comando1;
    comando2
```

Sintaxe:

```
<script type="text/javascript">
    var idade = 16;
    if (idade >18) {
        var texto = "Maior idade";
    };
    alert(texto);
</script>
```

Exemplo:

Caso a condição não seja satisfatória (FALSE), podemos atribuir outro comando pertencente ao IF chamado ELSE, como se fosse a estrutura SENÃO em lógica de programação.

Sintaxe:

```
if
  (expressão)
    comando;
else
  comando
; if
  (expressão)
{
  comando
  o1;
```

Exemplo:

```
<script type="text/javascript">
  var idade = 16;
  if (idade>18) {
    var texto = "Maior de idade";
  }else{
    var texto = "Menor de idade";
  }
  alert(texto);
</script>
```

Nesse exemplo temos uma expressão booleana onde retorna falso, com isso o IF não executa, passando a execução para o else, que por sua vez executa e atribui o valor “menor idade” a variável texto.

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Isso é o que podemos chamar de “If's encadeados”, onde usamos a estrutura **ELSE IF**. Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
  comando1;

  else
    if (expressao2)

      comando2;
    else

      if (expressao3)
        comando3;
      else comando4;
```

Sintaxe:**Exercício rápido:**

```
<script type="text/javascript">
  var idade = 16;
  if (idade>18) {
    var texto = "Maior de idade";
  }else if (idade < 12){
    var texto = "Criança";
  }else{
    var texto = "Adolescente";
  }
  alert(texto);
</script>
```

Exemplo:

- 1º) Faça um script em JavaScript que possua 4 notas de um aluno (cada uma em uma variável). Depois calcule e imprima a média aritmética das notas e a mensagem de aprovado para média superior ou igual a 7.0 ou a mensagem de reprovado para média inferior a 7.0.
- 2º) Faça um script em JavaScript que receba a idade de um nadador (representada por uma variável chamada “idade”) e imprima a sua categoria seguindo as regras:

Categoria	Idade
Infantil A	5 - 7 anos

Infantil B	8 - 10 anos
Juvenil A	11- 13 anos
Juvenil B	14- 17 anos

5.3 Atribuição condicional (ternário)

Como já vimos exemplos de atribuição condicionais (ternários), podemos defini-los usando a sintaxe: (Expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);

Isso se aplica quando queremos uma estrutura resumida, onde podemos ter um resultado mais direto, como por exemplo, atribuir um valor a uma variável dependendo de uma expressão. Observe o exemplo abaixo onde envolvemos uma variável do tipo string, porém o valor atribuído a essa variável deverá ser de acordo com o valor da idade:

```
<script type="text/javascript">
    var idade = 16;
    var texto = idade > 18 ? "Maior de idade" :"Menor de idade";
    alert(texto);
</script>
```

É uma estrutura parecida com IF e ELSE, onde dependendo da expressão booleana podemos executar um bloco ou não.

Exercício rápido:

1º) Faça uma script em JavaScript que receba um número representado por uma variável.

Verifique se este número é par ou ímpar e imprima a mensagem.

2º) Crie outro script baseando em um seguro de vida com as seguintes regras:

Idade: **Grupo de Risco :**

18 a 24 - Baixo

25 a 40 - Médio

41 a70 - Alto

5.4 SWITCH

Observe que quando temos muitos “if’s encadeados” estamos criando uma estrutura que não é considerada uma boa prática de programação. Para resolver esse problema temos uma estrutura onde sua funcionalidade é semelhante ao ELSE IF. O comando SWITCH é uma estrutura que simula uma bateria de teste sobre uma variável. Frequentemente é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um desses valores.

```
switch(expressão) {
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    case "valor 1":
```

Sintaxe:

```
<script type="text/javascript">
    var numero = 2;
    switch(numero){
        case 1:
            alert("opção 1");
        case 2:
            alert("opção 2");
        case 3:
            alert("opção 3");
        case 4:
            alert("opção 4");
        case 5:
            alert("opção 5");
    }
</script>
```

Exemplo:

Resultado: opção 2: opção 3:opção 4:opção 5:

Nesse exemplo temos o número = 2, onde o switch compara com os case's o valor recebido, o bloco que é executado é do segundo case, porém, os demais também são executados para que tenhamos um resultado satisfatório temos que usar em cada case um comando chamado **break**. No qual tem a função de para o bloco de execução.

5.5 SWITCH com BREAK

Break é uma instrução (comando) passada quando queremos parar o fluxo da execução de um programa. Em JavaScript, ele tem a mesma função que é “abortar” o bloco de código correspondente.

Observe o mesmo exemplo com o uso de break:

Temos agora como resultado “opção 2:”. O comando break fez com que os demais case's abaixo do 'case 2' não sejam executados.

```
<script type="text/javascript">
  var numero = 2;
  switch(numero){
    case 1:
      alert("opção 1");
      break;
    case 2:
      alert("opção 2");
      break;
    case 3:
      alert("opção 3");
      break;
    case 4:
      alert("opção 4");
      break;
    case 5:
      alert("opção 5");
      break;
  }
</script>
```

Obs.: Além de números podemos também comparar outros tipos como string, pontos flutuantes e inteiros, veja um exemplo abaixo:

```
<script type="text/javascript">
  var numero = "opc 2";
  switch(numero){
    case "opc 1":
      alert("opção 1");
      break;
    case "opc 2":
      alert("opção 2");
      break;
  }
</script>
```

Mas o que acontece se não tivermos um valor que seja satisfatório aos casos existentes no switch? A resposta é bem simples, nenhum dos blocos seriam executados, porém temos um comando onde determinamos uma opção padrão caso nenhuma das outras venha ter resultado que satisfaça a expressão passada para o switch chamada default (padrão).

```
<script type="text/javascript">
  var numero = "opc 3";
  switch(numero){
    case "opc 1":
      alert("opção 1");
      break;
    case "opc 2":
      alert("opção 2");
      break;
    default:
      alert("opção inválida");
  }
</script>
```

Veja um exemplo:

Resultado: opção inválida

A instrução passada não condiz com nenhum dos casos existentes. Por esse motivo o bloco pertencente ao comando default será executado.

O comando default pode ser inserido em qualquer lugar dentro do switch, porém caso isso aconteça, o uso do comando break deve ser adicionado para evitar que os case's abaixo

A partir de agora trabalharemos as estruturas de repetição. Elas muito utilizadas nas linguagens de programação.

1º) Faça um script em JavaScript usando switch, onde receba uma variável e mostre as seguintes opções: 1 - módulo.
2 - somar.
3 - subtrair.
4 - multiplicar.

5.6 WHILE

O WHILE é uma estrutura de controle similar ao IF, onde possui uma condição para executar um bloco de comandos. A diferença primordial é que o WHILE estabelece um laço de repetição, ou seja, o bloco de comandos será executado repetitivamente enquanto a condição passada for verdadeira. Esse comando pode ser interpretado como “ENQUANTO (expressão) FAÇA { comandos...}”.

Sintaxe:

```
while
  (expressão) {
    comandos;
```

Quando estamos usando um laço de repetição, podemos determinar quantas vezes ele deve ou não se repetir. Para trabalharmos com essa contagem de quantas vezes o laço deve se repetir, usaremos incremento ou decrecimento de uma variável conforme vimos no capítulo de operadores em JavaScript.

Observe o exemplo abaixo:

```
<script type="text/javascript">
  var a = 1;
  while(a < 10){
    alert(a);
    a++; //incrementa a variável
  }
</script>
```

Sequência do resultado em caixa de diálogo: 1 2 3 4 5 6 7 8 9

Nesse exemplo criamos um laço de repetição que tem como condição $a < 10$, a cada laço é executado um incremento na variável a , fazendo com que o seu valor aumente até a condição não ser mais satisfatória.



Dicas: Tenha cuidado quando estiver trabalhando com loop's (laço de repetição), pois caso a expressão passada esteja errada, pode ocasionar em um loop infinito fazendo com que o bloco de código se repita infinitamente. Isso pode ocasionar um travamento do navegador ou até mesmo do próprio servidor WEB.

Vamos ver agora um exemplo em que o laço se repete de forma automática, onde quem determina o loop são propriedades do JavaScript e não um número determinado pelo programador.

A propriedade length é chamada logo após uma variável de texto e retorna a quantidade de caracteres incluindo também os espaços em branco. Ele poderia ser aplicado diretamente no alert, mas no exemplo, ele determina a quantidade de loop's.

```
<script type="text/javascript">
    var a = 1;
    var texto = "Projeto e-Jovem";
    //Length conta o tamanho da string
    while(a < texto.length){
        a++; //incrementa a variável
    }
    alert("O texto possui "+a+" caracteres");
</script>
```

Resultado: a frase possui 15 caracteres

Exercício rápido:

- 1º) Faça um script que conte de 1 até 100.
- 2º) Faça um script que imprima na tela números de 3 em 3 iniciando com 0 até 90, ex: 0,3,6,9...

5.7 DO...WHILE

O laço do...while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço do...while possui apenas uma sintaxe que é a seguinte:

```
- do {
    comando
    ;
    ...
    comando;
} while
```

Sintaxe:

```
<script type="text/javascript">
    var a = 1;
    do{
        alert(a);
        a++;
    }while(a<10);
</script>
```

Exemplo:



Dicas: Talvez na criação de alguma página ou sistema web, seja necessário executar um bloco de código existente em um laço de repetição pelo menos uma vez, nesse caso podemos usar o do...while.

Sequência do resultado em caixa de diálogo: 1 2 3 4 5 6 7 8 9

Exercício rápido:

- 1º) Faça um script que conte de -1 até -100 usando “do while”.

2º) Faça um script que imprima na tela somente números pares de 2 até 20 com do while.

5.8 FOR

Outra estrutura semelhante ao while é o for, onde tem a finalidade de estabelecer um laço de repetição em um contador. Sua estrutura é controlada por um bloco de três comandos que estabelecem uma contagem, ou seja, o bloco de comandos será executado determinado número de vezes.

Sintaxe:

```
for( inicialização; condição;
      incremento ){ comandos;
}
```

Parâmetros	Descrição
Inicialização	Parte do for que é executado somente uma vez, usado para inicializar uma variável.
Condição	Parte do for onde é declarada uma expressão booleana.
Incremento	Parte do for que é executado a cada interação do laço.

Lembrando que o loop do for é executado enquanto a condição retornar expressão booleana verdadeira.

Outro detalhe importante é que podemos executar o incremento a cada laço, onde possibilitamos adicionar uma variável ou mais. Mostraremos agora um exemplo fazendo um comparativo entre a estrutura de repetição do while e também a do for de forma prática.

Exemplo:

```
<script type="text/javascript">
// COM WHILE
  var a = 1;
  while(a<10){
    alert(a);
    a++;
  }
</script>
```

WHILE

```
<script type="text/javascript">
// COM FOR
  for(a=1;a<10;a++){
    alert(a);
  }
</script>
```

FOR

Ambos exemplos geram a mesma sequência em caixa de diálogo com o resultado: 1 2 3 4 5 6 7 8 9

O **for** não precisa ter necessariamente todas as expressões na sua estrutura, com isso podemos criar um

exemplo de **for** onde suas expressões são declaradas externamente.

```
<script type="text/javascript">
    a = 1;
    for(;a<10;){
        alert(a);
        a++;
    }
</script>
```

Observe nesse exemplo uma proximidade muito grande do comando while. Apesar de ser funcional, não é uma boa prática de programação utilizar desta forma.

A estrutura “**while**” e “**do while**” normalmente é usado quando o programador não sabe quantos loops o sistema irá realizar, já o “**for**” informamos explicitamente qual será o ponto de partida, o seu limite, e valor de incremento ou decremento.

Exercício rápido:

1º) Faça um script que receba duas variáveis “a” e “b”, logo após imprima os números de intervalos entre eles com o uso de “for”. ex: a=5 ,b = 11, imprime : 5,6,7,8,9,10,11.

5.9 FOREACH

O **foreach** é um laço de repetição para interação em array's ou matrizes, o qual estudaremos com mais detalhes no próximo capítulo. Trata-se de um **for** mais simplificado que compõe um vetor ou matriz em cada um de seus elementos por meio de sua cláusula **IN**.

```
vetor = new Array();
foreach(incide in vetor) {
    comandos;
}
```

Sintaxe:

```
<script type="text/javascript">
    var texto = new Array("HTML","CSS","javascript");

    for (var indice in texto) {
        alert(texto[indice]);
    }
</script>
```

Exemplo:

Resultado: HTML CSS JAVASRIPT

Veremos adiante que um array é uma variável composta por vários elementos. No caso do exemplo anterior, esses elementos são nomes de cursos. A finalidade do foreach é justamente a cada laço, pegar cada índice desses valores e atribuir a uma variável, até que tenha percorrido todo array e assim, finalizar o laço. Podemos saber em qual posição o elemento se encontra no array, para isso basta imprimir o índice.

Observe o exemplo:

```
<script type="text/javascript">
    var texto = new Array("HTML","CSS","javascript");

    for (var indice in texto) {
        alert(indice+" - "+texto[indice]);
    };
</script>
```

Resultado:

0-HTML

1-CSS

2-JAVASCRIPT

Nesse exemplo observamos que cada elemento do array possui um índice (chave), imprimindo na tela o número da posição e o valor guardado.

5.10 BREAK.

Outro comando importante é o break, usado para abortar (parar) qualquer execução de comandos como SWITCH, WHILE, FOR, FOREACH, ou qualquer outra estrutura de controle. Ao encontrar um break dentro de um desses laços, o interpretador JavaScript interrompe imediatamente a execução do laço, seguindo normalmente o fluxo do script.

Sintaxe:

```
while.....
for.....
    break <quantidades de
    níveis>;
```

Vamos ver um exemplo com o uso de **break** dentro de um laço de repetição (no caso o **for**), onde criamos

```
<script type="text/javascript">
    var a = 1;
    for ( ; ; ) { //Loop Infinito
        document.write("<p>" + a + "</p>");
        if (a === 10) {
            break; //Para o Laço
        };
        a++; // Incremento
    };
</script>
```

um laço infinito, porém colocamos um if com a condição de parar o laço através do **break**. Observe:

Podemos notar nesse exemplo a criação de um laço(loop) infinito, que ocorre quando tiramos a condição

do **for**, ou atribuímos “`for(; true ;)`”, porém a condição fica na responsabilidade do **if**, quando o valor de `a` é igual a 10, faz com que o **if** execute o **break**, fazendo com que o laço pare de funcionar.

5.11 CONTINUE

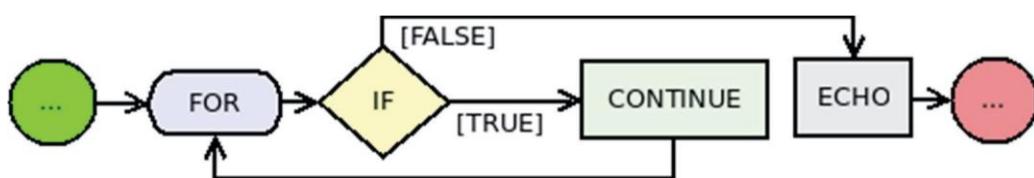
A instrução **continue**, quando executada em um bloco de comandos `for/while`, ignora as instruções restantes até o fechamento em “`}`”. Dessa forma, o programa segue para a próxima verificação da condição de entrada do laço de repetição, funciona de maneira semelhante ao `break`, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Veja um exemplo:

```
<script type="text/javascript">
  for ( var a = 0; a<20 ;a++) {
    if( a % 2){
      continue;
    }
    document.write(+a+" ");
  };
</script>
```

Resultado: 0,2,4,6,8,10,12,14,16,18

Podemos observar a seguinte lógica no exemplo acima:

Criamos um laço que tem 20 interações de repetição. Logo após temos um **if**, onde, quando o resto da divisão por 2 for igual a 0 (número par), o valor booleano será “`false`”. Quando não for igual a 0, significa que a variável `a` é um número ímpar (ex: `5%2 = 1`), então temos um valor booleano “`true`”. Isso significa que o **if** executa somente quando os números forem ímpares. Adicionamos um `continue`, que ao executar o **if**, faz com que volte novamente para o início do `for` impedindo de alcançar o `echo` em seguida. Com isso, em vez de mostrarmos os números ímpares, imprimimos somente os números pares incluindo o 0. Resumimos que o código só passa adiante quando o **if** não executa o `continue`. Fluxograma:



Exercícios Propostos

Qual a principal finalidade de uma estrutura de controle?

Qual a principal finalidade de uma estrutura de repetição?

Crie um código com uma condição ternária onde receba um valor booleano e de acordo com o valor passado na expressão, deve imprimir “sim” ou “não”.

Com o comando **IF** e **ELSE** crie um código que determine se uma expressão é verdadeira ou

falsa.

Qual a finalidade da estrutura de controle SWITCH e cite um exemplo onde comparamos uma opção com 4 casos diferente?

Crie um contador de 1 até 20 usando a estrutura de repetição WHILE.

Crie um contador de 1 até 100 usando DO WHILE. EP06.8: Crie um contador de 100 até 1 usando FOR.

Qual a finalidade de um FOREACH?

Crie um código onde podemos parar a execução de um laço infinito com o uso de BREAK.

Como podemos determinar o uso de CONTINUE e qual a sua aplicação prática em JavaScript.

Crie um código com as seguintes características:

Deverá receber um valor inicial e outro final (crie duas variáveis para esse fim).

Como o comando FOR crie um laço onde a contagem é determinada pelo valor inicial e final?

Dentro do for deverá conter um IF e ELSE responsável por comparar os valores passados a ele e imprimir os pares e ímpares. Exemplo:

```
IF(valor%2==0)
echo valor. "é um número par";
ELSE
echo valor. "é um número ímpar";
```

Exemplo prático: foi passado o número inicial 8 e o final 15, então o script JavaScript deverá imprimir o intervalo entre esse número ou seja 8,9,10,11,12,13,14,15, mostrando quais deles são pares e quais são ímpares.

6.0 Objetos em JavaScript

Quase “Tudo” em JavaScript é um objeto: uma String, um número, uma matriz, uma data, um botão

Em JavaScript, um objeto, são variáveis que guardam atributos e métodos. Atributos são características de um objeto.

Métodos são ações que um objeto pode realizar;

Vamos exemplificar com um objeto da vida real: Objeto Carro.



carro.

As propriedades do carro incluem a montadora, o modelo, o peso, a cor, etc...

Todos os carros têm essas propriedades, mas os valores dessas propriedades diferem de carro para carro.

Os métodos do carro são ligar(), acelerar(), freiar(), etc

Todos estes métodos pertencem ao objeto carro, mas eles são realizados em momentos diferentes.

6.1 Objeto String, Acessando atributos e métodos

Em JavaScript como já mencionado quase “tudo” é objeto. Quando declaramos uma variável dessa forma:

```
var txt = "Hello";
```

O que realmente estamos fazendo é criando um objeto String JavaScript. O objeto String tem vários métodos e alguns atributos já prontos para facilitar o seu trabalho quando for preciso manipular Strings, os quais são chamados built-in, ou seja, métodos que são nativos do JavaScript.

6.2 Acessando atributos

Para acessarmos um atributo ou métodos de um objeto usamos o operador ponto (.), logo após o nome do objeto.

Por exemplo, vamos acessar uma propriedade do Objeto String txt, criado anteriormente, chamado de **length**:

```
var txt = "Hello";
var tamanhoDaString = txt.length;
```

Se executarmos a instrução `document.write(tamanhoDaString)`; será mostrado o valor 5 na página.

6.3 Acessando Métodos

Para acessarmos os métodos de um objeto também utilizamos o operador ponto (.), logo após o nome do objeto, a diferença é que após o nome do método que será chamado digitamos parênteses (), nos quais podemos colocar ou não argumentos dentro.

Abaixo a sintaxe padrão para acesso de métodos de um objeto:

```
nomeObjeto.nomeMetodo(<argumento1,argumento2,argumentoN>)
```

Legenda:

- <argumento1,argumento2,argumentoN> < dependendo do método que você deseja chamar esse parâmetro pode existir ou não;
- nomeObjeto < nome do objeto JavaScript;
- nomeMetodo < nome do método que você deseja chamar/executar;

Neste exemplo vamos utilizar um dos métodos nativos do objeto String JavaScript, o método

toUpperCase().

```
<script type="text/javascript">
    var message = "Hello World!";
    var x = message.toUpperCase();
    document.write(x);
</script>
```

Este método é responsável por converter todos os caracteres de uma String para maiúsculo. A saída desse trecho de código JavaScript é essa:

HELLO WORLD!

Abaixo uma lista com todos os métodos e atributos que um objeto String JavaScript possui:

Atributos:

<i>length</i>	<i>prototype</i>	<i>constructor</i>
---------------	------------------	--------------------

Métodos:

<i>charAt()</i>	<i>charCodeAt()</i>	<i>concat()</i>	<i>fromCharCode()</i>
<i>indexOf()</i>	<i>lastIndexOf()</i>	<i>match()</i>	<i>replace()</i>
<i>search()</i>	<i>slice()</i>	<i>split()</i>	<i>substr()</i>
<i>substring()</i>	<i>toLowerCase()</i>	<i>toUpperCase()</i>	<i>valueOf()</i>

6.4 Objeto Number

JavaScript tem apenas um tipo de número, os quais podem ter ou não casas decimais.

JavaScript não é uma linguagem tipada, ao contrário de muitas outras linguagens de programação. O JavaScript não define diferentes tipos de números, como números int, short, long, float... Todos os números em JavaScript são armazenados como **float** de 64 bits(8 bytes).

JavaScript cria objetos do tipo **Number** quando uma variável é definido com um valor numérico, por exemplo var num = 255.336;. Raramente é necessário criar explicitamente objetos de **Number**.

Os números são convertidos em cadeias de caracteres em determinadas circunstâncias, por exemplo, quando um número é adicionado ou concatenado com uma String bem como utilizamos o método de **toString**.

O objeto do tipo **Number** tem suas próprias propriedades e métodos.

Atributos:

MAX_VALUE	MIN_VALUE	NEGATIVE_INFINITY	POSITIVE_INFINITY
NaN	<i>prototype</i>	<i>constructor</i>	

Métodos:

toExponential()	toFixed()	toPrecision()	toString()
valueOf()			

Atenção: consulte o nosso material de apoio para saber como utilizar cada método e atributo listado acima.

6.4.1 Atributos objeto Number - CONSTANTES

Objeto Number possui como atributos constantes: MAV_VALUE, MIN_VALUE, NEGATIVE_INFINITY, POSITIVE_INFINITY E NaN.

Para utilizar esses atributos, que são constantes, não precisamos criar ou instanciar um objeto do tipo Number como fizemos no Objeto String. Colocamos apenas a palavra-chave Number seguido de um ponto(.) e já conseguimos acessar os valores contidos nessa constante.

Veja um exemplo:

Number.MAX_VALUE	O maior número que pode ser representado em JavaScript. Retorna o valor: 1.7976931348623157e+308 .
Number.MIN_VALUE	o número mais próximo a zero que pode ser representado em JavaScript. Retorna o valor: 5.00E-324.

6.4.2 Representação de um número em Hexadecimal, Octal e notação científica

- **Hexadecimal:**

Podemos representar um número em hexadecimal em JavaScript colocando o valor: **0x** na frente do número a ser representado.

Exemplo:

```
var valor1 = 0x142;
```

- **Octal:**

Podemos também representar um número em Octal em JavaScript colocando o valor: **0** na frente do número a ser representado.

Exemplo:

```
var valor = 0377;
```

- **Notação científica:**

Para números muito grandes ou muito pequenos podemos representá-los usando a notação científica.

Veja abaixo como o JavaScript trabalha com esse tipo de notação:

```
var y = 123e5; // 12300000
var z = 123e-5 // 0.00123
```

6.5 Objeto Math

Em JavaScript, podemos fazer uso de um objeto próprio para cálculos matemáticos chamado **Math** que possui constantes, métodos para *calcular potências, raízes, arredondamentos, funções trigonométricas*, maneiras de encontrar o menor e o maior valor, além de um gerador de números randômicos. O objeto Math possui algumas constantes importantes para cálculos mais complexos, bem como métodos para executar operações matemáticas mais facilmente.

6.5.1 Constantes

O objeto Math possui 8 constantes que são:

E: constante do número de Euler. (2,718281828459045);

LN2: constante com o resultado do logaritmo natural na base 2. (0,6931471805599453); **LN10:** constante com o resultado do logaritmo natural na base 10. (2,302585092994046); **LOG2E:** constante com o resultado do logaritmo na base 2 do número de Euler.

(1,4426950408889634);

LOG10E: constante com o resultado do logaritmo na base 10 do número de Euler.

(0,4342944819032518);

PI: constante do pi (M). (3,141592653589793);

SQRT1_2: constante com o resultado da raíz quadrada de meio. ($\sqrt{1/2} = 0,7071067811865476$);

SQRT2: constante com o resultado da raíz quadrada de 2 ($\sqrt{2} = 1,4142135623730951$);

No caso, todas essas constantes são valores aproximados, levando-se em conta que são dízimas periódicas.

Abaixo está um exemplo de como obter o valor de todas as constantes.

```
<script type="text/javascript">
    document.write (Math.E +"  
");
    document.write (Math.LN2 +"  
");
    document.write (Math.LN10 +"  
");
    document.write (Math.LOG2E +"  
");
    document.write (Math.LOG10E +"  
");
    document.write (Math.PI +"  
");
    document.write (Math.SQRT1_2 +"  
");
    document.write (Math.SQRT2 +"  
");
</script>
```

6.5.2 Objeto Math e seus métodos

- **Raízes e Potências**

Podemos utilizar o objeto Math para obter raízes quadradas e potências. O método **sqrt()** extrai a raiz quadrada do número passado como argumento.

Exemplo:

```
<script type="text/javascript">
var1 = Math.sqrt(4);
document.write(var1);
</script>
```

Resultado: 2 (raiz quadrada de 4)

O método **pow()** retorna o valor da potência indicada em seus parâmetros, sendo o primeiro parâmetro o número base e o segundo o expoente.

```
<script type="text/javascript">
var1 = Math.pow(10,3); //Mesmo que  $10^3$ 
document.write(var1); //Resultado: 1000
</script>
```

Exemplo:

Vejamos o código e o resultado logo abaixo:

Código:

```
<script type="text/javascript">
var var1 = 4;
var var2 = 2;
document.write("A raiz quadrada de "+ var1+ " é " + Math.sqrt(var1)+"<br>");
document.write(var1+ " elevado a "+ var2 + " é " + Math.pow(var1,var2)+"<br>");
</script>
```

Saída:

```
A raiz quadrada de 4 é
24 elevados a 2 é 16
```

6.5.3 Arredondamentos

Quando tratamos com números que possuem a parte decimal extensa (como é o caso das constantes), podemos fazer uso de métodos para arredondar os números.

O método **round()** arredonda um número para o inteiro mais próximo, tanto para baixo quanto para cima. Por exemplo, o número 3.3 arredondado será 3, mas o número 3.8 arredondado será 4.

O método **floor()** arredonda um número para o inteiro mais baixo. Também considerado como piso. O método **ceil()** arredonda um número para o inteiro mais alto. Também considerado como teto.

O método **abs()** remove apenas a parte fracionada. Ou seja, retorna o valor absoluto.

Exemplo:

```
<script type="text/javascript">
var var1 = 4.5;
var var2 = -3.2;
document.write("O inteiro mais proximo de "+ var1+" é " + Math.round(var1)+"<br>");
document.write("O inteiro mais baixo(piso) de "+ var1+" é " + Math.floor(var1)+"<br>");
document.write("O inteiro mais alto de "+ var1+" é " + Math.ceil(var1)+"<br>");
document.write("O valor absoluto de "+ var2+" é " + Math.abs(var2)+"<br>");
</script>
```

6.5.4 Trigonometria

Usado para cálculos trigonométricos envolvendo principalmente ângulos. Com esses métodos fica fácil obter o resultado matemático dos ângulos sem a necessidade de vários cálculos ou tabelas prontas.

sin(): retorna o valor de seno; **cos()**: retorna o valor de cosseno; **tan()**: retorna o valor da tangente; **asin()**: retorna o valor do arco seno;

acos(): retorna o valor do arco cosseno;

atan(): retorna o valor do arco tangente;

Exemplo:

```
<script type="text/javascript">
var var1 = 90;
document.write("O seno de " + var1 + " é " + Math.sin(var1)+"<br>");
document.write("O cosseno " + var1 + " é " + Math.cos(var1)+"<br>");
document.write("A tangente " + var1 + " é " + Math.tan(var1)+"<br>");
document.write("O arco seno " + var1 + " é " + Math.asin(var1)+"<br>");
document.write("O arco cosseno " + var1 + " é " + Math.acos(var1)+"<br>");
document.write("O arco tangente " + var1 + " é " + Math.atan(var1)+"<br>");
</script>
```

6.5.5 Maior e Menor

Existem dois métodos do objeto Math que servem como comparativos. **min(valor1, valor2)**: retorna o menor valor entre os parâmetros passados. **max(valor1, valor2)**: retorna o maior valor entre os parâmetros passados.

Exemplo:

```
<script type="text/javascript">
var var1 = 7;
var var2 = 5;
document.write
    ("O maior valor entre "+var1+" e "+var2+" é " + Math.max(var1,var2)+"<br>");
document.write
    ("O menor valor entre "+var1+" e "+var2+" é " + Math.min(var1,var2)+"<br>");
</script>
```

6.5.6 Número Randômico

O objeto Math também possui um método para gerar automaticamente números randômicos.

O método **random()** retorna um número entre 0 e 1, ou seja, pode ser 0, 1, 0.5, 0.2, 0.8, 0.4567412, e assim por diante.

Se, por exemplo, quisermos fazer o limite entre 0 e 10, basta que multipliquemos por 10 o valor retornado por **random()**. Dessa forma conseguiremos um número entre randômico maior.

O problema de se usar isso é que os números retornados sempre serão muito fracionados, portanto, o ideal é utilizar junto uma das funções de arredondamento vistas nos tópicos anteriores.

Veja o exemplo abaixo:

```
<script type="text/javascript">
// Valor randomico
var valor1 = Math.random()*10;
var valor2 = Math.random()*10;
// Arredondando os valores
valor1 = Math.round(valor1);
valor2 = Math.round(valor2);
document.write
    ("Os numeros sorteados foram: "+valor1+" e "+valor2 );
</script>
```

Os números sorteados foram 4 e 2.

O que resulta em:

6.5.7 Objeto Boolean

Os objetos boolean servem para representar os valores booleanos (true/false). Foi acrescentado na versão 1.1 de JavaScript (com Netscape Navigator 3). Uma de suas possíveis características é a de conseguir valores booleanos a partir de dados de qualquer outro tipo.

Dependendo do que receba o construtor da classe Boolean o valor do objeto booleano que se cria será verdadeiro ou falso, da seguinte maneira:

- Inicia-se **false**: quando você não passa nenhum valor ao construtor, ou se passa uma cadeia vazia, o número 0 ou a palavra false sem aspas.
- Inicia-se **true**: quando recebe qualquer valor entre aspas ou qualquer número distinto de 0. Pode-se compreender o funcionamento deste objeto facilmente se examinarmos alguns exemplos.

```
<script type="text/javascript">
//Iniciando com FALSE (falso)
var b1 = new Boolean()
document.write(b1 + "<br>")

var b2 = new Boolean("")
document.write(b2 + "<br>")

var b3 = new Boolean(false)
document.write(b3 + "<br>")

var b4 = new Boolean(0)
document.write(b4 + "<br>")

//Iniciando com TRUE (verdadeiro)
var b5 = new Boolean("Brasil")
document.write(b5 + "<br>")

var b6 = new Boolean(3)
document.write(b6 + "<br>")
</script>
```

6.5.8 Objeto Date

O objeto Date é usado para trabalhar com datas e tempo. Para trabalhar com datas necessitamos instanciar um objeto da classe Date e com ele já podemos realizar as operações que necessitamos.

6.5.9 Criando um objeto Date

Um objeto da classe Date pode ser criado de duas maneiras distintas. Por um lado, podemos criar o objeto com o dia e hora atuais e por outro podemos criá-lo com um dia e hora distintos aos atuais. Isto depende

dos parâmetros que passemos ao construir os objetos.

Para criar um objeto com o dia e hora atuais, colocamos os parênteses vazios ao chamar ao construtor da classe Date. Veja exemplo:

```
<script type="text/javascript">
    minhaDataAtual = new Date();
    document.write(minhaDataAtual);
    //Resultado: Wed Dec 16 2015 18:50:57 GMT-0300 (Hora oficial do Brasil)
</script>
```

6.5.10 Definindo uma data específica

Claro que o "var data = new Date()" nem sempre é o que a gente quer. Muitas vezes, queremos que o objeto criado represente uma data específica, seja ela lida de um campo da tela, seja ela calculada de alguma forma.

Para criar um objeto data com um dia e hora diferentes dos atuais temos que indicar entre parênteses o momento para iniciar o objeto. Existem várias maneiras de expressar um dia e hora válida, por isso podemos construir uma data nos guiando por vários esquemas. Estes são dois deles, suficientes para criar todo tipo de datas e horas.

Código:

```
<script type="text/javascript">
    var ano = 2015;
    var mes = 12;
    var dia = 16;
    var hora = 18;
    var minutos = 50;
    var segundos = 30;
    minhaData1 = new Date(ano,mes,dia,hora,minutos,segundos);
    minhaData2 = new Date(ano,mes,dia);
    document.write("Definindo data 1: "+minhaData1.toString()+"<br>");
    document.writeln("Definindo data 2: "+minhaData2.toString());
</script>
```

Saída:

```
Definindo data 1: Sat Jan 16 2016 18:50:30 GMT-0300 (Hora oficial do Brasil)
Definindo data 2: Sat Jan 16 2016 00:00:00 GMT-0300 (Hora oficial do Brasil)
```

Os valores que devem ser passados para os construtores acima são sempre numéricos.

Um detalhe, o mês começa por 0(zero), ou seja, janeiro é o mês 0. Se não indicamos a hora, o objeto data

se cria com hora 00:00:00.

Assim é possível criar um objeto Date com o valor de qualquer data. Lembre-se de que o valor do mês parece sempre diminuído de 1, porque janeiro=0, fevereiro=1, março=2, ..., dezembro=11!! Até o argumento "dia" é obrigatório, os outros são opcionais.

6.5.11 Métodos do Objeto Date

O objeto Date não possui atributos/propriedades, por outro lado, possui muitos métodos. A seguir serão apresentados alguns dos seus principais métodos.

1. **getDate()**: devolve o dia do mês, um inteiro entre 1 e 31.
2. **getDay()**: devolve o dia da semana, inteiro entre 0 e 6 (0 para Domingo).
3. **getHours()**: retorna a hora, inteiro entre 0 e 23.
4. **getMinutes()**: devolve os minutos, inteiro entre 0 e 59.
5. **getSeconds()**: devolve os segundos, inteiro entre 0 e 59.
6. **getMonth()**: devolve o mês, um inteiro entre 0 e 11 (0 para Janeiro).
7. **getTime()**: devolve os segundos transcorridos entre o dia 1 de janeiro de 1970 e a data correspondente ao objeto ao que se passa a mensagem.
8. **getYear()**: retorna o ano, os últimos dois números do ano. Por exemplo, para o 2006 retorna 06. Este método está obsoleto em Netscape a partir da versão 1.3 de JavaScript e agora se utiliza **getFullYear()**.
9. **getFullYear()**: retorna o ano com todos os dígitos com datas posteriores desde 2000.
10. **setDate(d)**: atualiza o dia do mês.
11. **setHours(h)**: atualiza a hora.
12. **setMinutes(m)**: muda os minutos.
13. **setMonth(m)**: muda o mês (atenção ao mês que começa por 0).
14. **setSeconds(s)**: muda os segundos.
15. **setTime(t)**: atualiza a data completa. Recebe um número de segundos desde 1 de janeiro de 1970.
16. **setYear(y)**: muda o ano, recebe um número, ao que lhe soma 1900 antes de colocá-lo como ano data. Por exemplo, se receber 95 colocará o ano 1995. Este método está obsoleto a partir de JavaScript 1.3

em

Netscape. Agora se utiliza setFullYear(), indicando o ano com todos os dígitos.

17. setFullYear(): muda o ano da data ao número que recebe por parâmetro. O número se indica completo ex: 2005 ou 1995. Utilizar este método para estar certo de que tudo funciona para datas posteriores a 2000.
18. getimezoneOffset(): Devolva a diferença entre a hora local e a hora GMT (Greenwich, UK Mean Time) sob a forma de um inteiro representando o número de minutos (e não em horas).
19. toGMTString(): devolva o valor do atual em hora GMT (Greenwich Mean Time)

Exercícios Propostos:



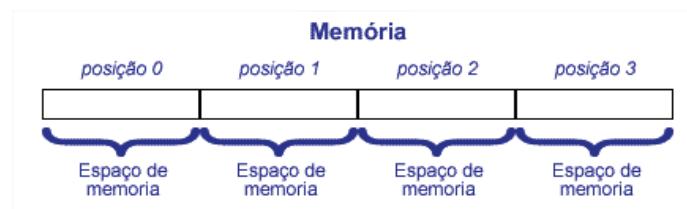
Leia a data de hoje e monte uma página com o calendário, destacando o dia de hoje (em vermelho, por exemplo).

Escreva um programa que receba uma data através de um campo de textos (prompt()) no formato dd/mm/aa. O programa deve reclamar (alert()) se o formato digitado for incorreto e dar uma nova chance ao usuário. Recebido o string, ele deve ser interpretado pelo programa que deverá imprimir na página quantos dias, meses e anos faltam para a data digitada.

7.0 Array

Nesse capítulo iremos abordar de forma clara as principais características de um array; Mostrar a sua criação e manipulações possíveis; Definir arrays multidimensionais ou matrizes; Determinar formas de interações e acessos.

Um array no JavaScript é atualmente um conjunto de valores ordenado por índices. Podemos relacionar cada valor com um índice/chave, para indicar em qual posição um determinado valor está armazenado dentro do array.



Ele é otimizado de várias maneiras, então podemos usá-lo como um array real, lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente muito mais. Além disso, JavaScript nos oferece uma gama enorme de funções para manipulá-los.

A explicação dessas estruturas está além do escopo dessa apostila, mas todo conteúdo aqui abordado traz uma boa base para quem estiver iniciando o conteúdo de array.

7.1 Criando um Array

Existe várias formas de construção de um array em JavaScript, veja:

- **Construção simples sem dimensionamento:** Quando criamos um array, mas não especificamos quantas posições ele irá possuir;
- **Construção simples com dimensionamento:** Quando criamos um array e especificamos quantas posições ele irá possuir. Estas posições são vazias.
- **Construção inserindo valores:** Quando criamos um array e especificamos exatamente quais valores ele irá possuir. Cada valor deve ser separado por vírgula e qualquer tipo de dado pode ser utilizado para popular nosso vetor.

Veja um exemplo de cada forma de criar um vetor/array:

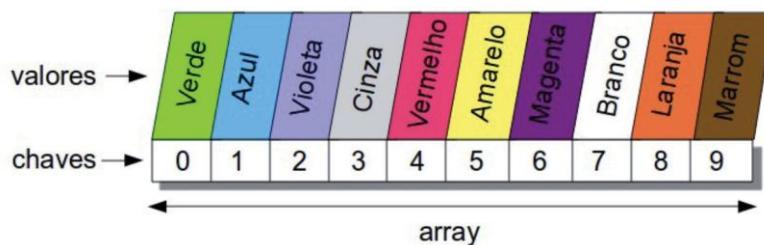
```
<script type="text/javascript">
    //Sem dimensionamento
    meuVetor1 = new Array();

    //Com dimensionamento
    meuVetor2 = new Array(4);

    //Com valores inseridos
    meuVetor3 = new Array("Raquel","Fabiana","Diana","Alana");
</script>
```

Detalhe importante é que todo array não associativo começa pela chave ou índice de número 0.

A figura abaixo representa um array que tem como valor representação de cores, e possui dez posições,



cada posição representa uma cor, seu índice (chave) vai de 0 até 9. Veja:

Em código temos:

```
<script type="text/javascript">
    var cores = new Array("verde","azul","violeta","cinza",
        "vermelho","amarelo","magenta","branco","laranja","marrom");
</script>
```

7.2 Arrays associativos.

Os arrays associativos associa-se um determinado valor ou nome a um dos valores do array.

O array associativo usa strings como índice, onde cada string pode representar uma chave.

Observe a sintaxe:

```
<script type="text/javascript">
    var vetor = {"chave1":"valor1","chave2":"valor2","chaveN":"valorN",}
</script>
```

Observe que quando usamos arrays associativos, a compreensão é mais fácil, dando mais legibilidade ao código. Porém, não é adequado utilizá-lo dentro de um loop tradicional(while,do..while,for), por isso temos um loop especial para percorrer arrays associativos chamado de for..in.

```
<script type="text/javascript">
//Forma 01
var vetor1 = {"nome": "Valdenia", "rua": "São João", "bairro": "Messejana"};
//Forma 02
var vetor2 = new Array[3];
vetor2["nome"] = "Valdenia";
vetor2["rua"] = "São João";
vetor2["bairro"] = "Messejana";
</script>
```

Veja um exemplo de array associativo escrito de duas formas diferentes:

Umas das vantagens do array associativo é quando fazemos o acesso ao array, onde temos de forma clara e compreensível o valor que aquela chave pode conter. Como por exemplo nome, onde só vai existir o nome de pessoas. Veja abaixo um exemplo de acesso aos valores armazenados em um array dessa natureza.

Exemplo:

```
<script type="text/javascript">
var vetor = new Array(3);
vetor['nome'] = "Valdenia";
vetor['rua'] = "São João";
vetor['bairro'] = "Messejana";

document.writeln(vetor['nome']);
document.writeln(vetor['rua']);
document.writeln(vetor['bairro']);
</script>
```

Dessa forma podemos acessar o array. Basta determinar o nome do array e qual a chave, onde cada chave tem um valor já determinado. Resultará em um erro o uso de uma chave errada.

7.3 Interações

Quando falamos de interações em um array estamos dizendo o mesmo que percorrer esse array usando mecanismos da própria linguagem. Como isso as interações podem ser feitas de várias formas, mas no JavaScript podem ser iterados pelo operador FOR(chave IN array) que já vimos anteriormente.

Exemplo:

```
<script type="text/javascript">
    var vetor = new Array(3);
    vetor['nome'] = "Valdenia";
    vetor['rua'] = "São João";
    vetor['bairro'] = "Messejana";

    for (chave in vetor){
        document.write(chave + " = " + vetor[chave] + "<br>");
    }
</script>
```

Resultado:

```
nome = Valdenia
rua = São João
bairro = Messejana
```

Esse tipo de interação é muito utilizado, principalmente quando temos arrays associativos.

7.4 Acessando um Array

Quando criamos um array temos que ter em mente que estamos criando uma variável que possui vários valores e que os mesmos podem ser acessados a qualquer momento. Cada valor está guardado em uma posição que pode ser acessada através de uma chave.

A sintaxe para acesso simplificado de um array é a seguinte:

```
nome_do_array[chave_de_acesso];
```

Temos que ter cuidado ao passar uma chave para o array, pois ela deve conter o mesmo nome de qualquer uma das chaves existentes no array. Caso a chave não exista, o valor não poderá ser resgatado. A sintaxe acima retorna um valor contido no array, por esse motivo temos que atribuir esse valor como mostra o exemplo abaixo:

```

<script type="text/javascript">
//Array com Valores
    var meu_array = new Array("nome","telefone","rua");
// Acessando uma posição
    var elemento_array = meu_array[1];

document.write(elemento_array);
</script>

```

Resultado: telefone.

7.5 Alterando um Array

Podemos alterar qualquer valor de um array. É muito semelhante ao acesso, onde, a diferença está na chamada do array. É nesse momento que atribuímos um novo valor.

nome_do_array[chave_de_acesso] = <novo_valor>;

Observe o exemplo abaixo:

```

251<script type="text/javascript">
252    //criando um array com valores
253    var meu_array = new Array("nome","telefone","rua","cidade");
254
255    //alterando o valor de uma posições
256    meu_array[1] = "sobrenome";
257
258    //imprime na tela.
259    for(chave in meu_array){
260        document.write(meu_array[chave]+<br>);
261    }
262</script>

```

Resultados:

```

nome
sobrenome
rua
cidade

```

Vimos no exemplo anterior o valor da posição 1 do array ('telefone') foi alterada para sobrenome. Vale ressaltar que esse array tem suas chaves definidas de forma automática. A primeira posição é 0, a segunda é 1, e assim sucessivamente. Veja mais um exemplo onde alteramos o valor, mas usando o operador “+=”:

```

268<script type="text/javascript">
269    //criando um array vazio
270    var produto = new Array();
271    //adicionando valores
272    produto["nome_produto"] = "Arroz";
273    produto["valor_produto"] = 1.35;
274
275    //alterando valor do pruduto
276    produto["valor_produto"] += .63;
277
278    //alterando nome do produto
279    produto["nome_produto"] = " Tio João ";
280
281    //imprime na tela.
282    for(chave in produto){
283        document.write(produto[chave]+"<br>");
284    }
285 </script>

```

Podemos observar que assim como as variáveis “comuns”, a forma de alterar o valor de um array é igual. A diferença está na chamada do elemento de um array, pois temos que passar a chave além do valor que queremos atribuir.

7.6 Arrays multidimensionais

Os arrays multidimensionais são estruturas de dados que armazenam os valores em mais de uma dimensão. Os arrays que vimos até agora armazenam valores em uma dimensão, por isso para acessar às posições utilizamos somente um índice ou chave. Os arrays de 2 dimensões salvam seus valores de alguma forma como em filas e colunas e por isso, necessitaremos de dois índices para acessar a cada uma de suas posições.

Em outras palavras, um array multidimensional é como um “contêiner” que guardará mais valores para cada posição, ou seja, como se os elementos do array fossem por sua vez outros arrays.

Outra ideia que temos é que matrizes são arrays nos quais algumas de suas posições podem conter outros arrays de forma recursiva. Um array multidimensionais pode ser criado pela função array():

Na figura abaixo temos a representação de um array com duas dimensões.

	0	1	2	3	4	Colunas
0	0.0	0.1	0.2	0.3	0.4	
1	1.0	1.1	1.2	1.3	1.4	
2	2.0	2.1	2.2	2.3	2.4	
3	3.0	3.1	3.2	3.3	3.4	
4	4.0	4.1	4.2	4.3	4.4	

Uma diferença importante de um array comum para um multidimensional é a quantidades de chaves (índices), onde cada um dos índices representa uma dimensão.

Código:

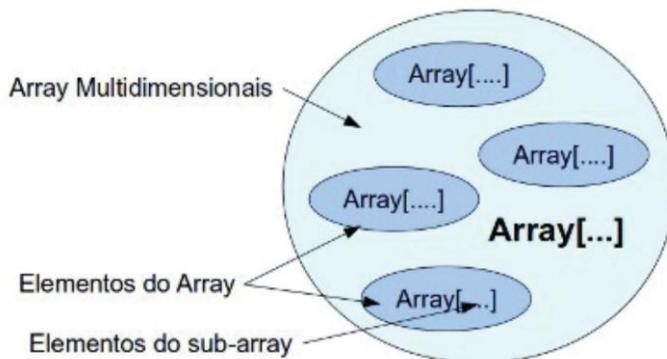
```
289<script type="text/javascript">
290     var matriz = new Array(
291         new Array("0.0" , "0.1" , "0.2" , "0.3" ) ,
292         new Array("1.0" , "1.1" , "1.2" , "1.3" ) ,
293         new Array("2.0" , "2.1" , "2.2" , "2.3" ) ,
294         new Array("3.0" , "3.1" , "3.2" , "3.3" ) ;
295 </script>
```

Outra forma de iniciar o array:

```

301<script type="text/javascript">
302  var matriz = new Array(4,4);
303  matriz[0] = new Array("0.0" , "0.1" , "0.2" , "0.3");
304  matriz[1] = new Array("1.0" , "1.1" , "1.2" , "1.3");
305  matriz[2] = new Array("2.0" , "2.1" , "2.2" , "2.3");
306  matriz[3] = new Array("3.0" , "3.1" , "3.2" , "3.3");
307 </script>
```

Observe que temos uma chave para representar a linha e outra para representar a coluna, assim, determinando uma matriz 4x4. Podemos ver também que inicializamos um array dentro do outro. Cada sub- array é uma linha, e cada elemento do array maior representa as colunas.



Para acessarmos o valor de um array multidimensional, basta colocar as duas ou mais chaves da posição que queremos acessar. É muito semelhante ao array de uma única dimensão.

Observe o acesso aos exemplos anteriores:

```
nme_do_array[chave_1][chave_2]...[chave_n];
```

Exemplo :

```

310<script type="text/javascript">
311  var matriz = new Array(4,4);
312  matriz[0] = new Array("0.0" , "0.1" , "0.2" , "0.3");
313  matriz[1] = new Array("1.0" , "1.1" , "1.2" , "1.3");
314  matriz[2] = new Array("2.0" , "2.1" , "2.2" , "2.3");
315  matriz[3] = new Array("3.0" , "3.1" , "3.2" , "3.3");
316
317  //acessando array linha 1 coluna 2
318  document.write(matriz[1][2]); //Resultado: 1.2
319 </script>
```

Dessa forma podemos acessar o elemento “1.2” que está guardado na posição linha 1 coluna 2, lembrando que o primeiro elemento de um array é 0. Abaixo, um exemplo que acessa todos os valores do array e imprime com quebra de linha:

Código:

```

324<script type="text/javascript">
325     var matriz = new Array(
326         new Array("0.0" , "0.1" , "0.2", "0.3"),
327         new Array("1.0" , "1.1" , "1.2" , "1.3"),
328         new Array("2.0" , "2.1" , "2.2" , "2.3"),
329         new Array("3.0" , "3.1" , "3.2" , "3.3") );
330     //percorrendo matriz
331     for(i=0 ; i<matriz.length;i++){
332         for(j=0 ; j<matriz[i].length;j++){
333             //retirando a ultima vírgula da linha na hora da impressão
334             if(j != matriz[i].length-1){
335                 document.write(matriz[i][j]+", ");
336             }
337             else{
338                 document.write(matriz[i][j]);
339             }
340         }
341         document.write("<br>");
342     }
343 </script>
```

Resultado:

0.0, 0.1, 0.2, 0.3
1.0, 1.1, 1.2, 1.3
2.0, 2.1, 2.2, 2.3
3.0, 3.1, 3.2, 3.3

Explicando o código:

Linha 325 à 329 – criamos um array de duas dimensões.

Linha 331 – temos um for que irá percorrer as linhas de nossa matriz. Esse for tem uma variável i iniciando com 0 e será incrementado de um em um enquanto i for menor que o número de linhas da matriz

Linha 332 – temos um for que irá percorrer as colunas de nossa matriz. Esse for tem uma variável j iniciando com 0 e será incrementado de um em um enquanto j for menor que o número de colunas da matriz

Linha 334 –temos um if que irá verificar se não estamos na última coluna da linha que está sendo impressa. Se não estiver na última coluna irá imprimir o valor da coluna concatenado com uma “,”.

Linha 337 – caso a condição da linha 334 não seja verdade, é porque iremos imprimir a última coluna da linha que está sendo impressa, então irá imprimir somente o valor da coluna, sem a vírgula.

Linha 341 – temos uma quebra de linha, que será executada todas as vezes que terminarmos de imprimir todos os valores de uma linha, ou seja quando o for mais interno for finalizado

7.6 Funções com Arrays

O objeto Array possui uma propriedade chamada length. Esta propriedade mostra quantos elementos o vetor possui.

Código:

```
<script type="text/javascript">
    vetor = new Array("João", "Roberto", "José");
    document.write(
        "<p>Este vetor possui "+vetor.length+" elementos. Que são: </p>");
    document.write("<ul>");
    for (indice = 0; indice < vetor.length; indice++) {
        document.write ("<li>" + vetor [indice] + "</li>")
    };
    document.write("</ul>");
</script>
```

Saída:

Este vetor possui 3 elementos. Que são: João

Roberto José

7.7 Junção e Concatenação de Arrays (Vetores)

Como vimos anteriormente, os vetores em JavaScript são constituídos pelo objeto Array.

Há momentos que queremos unir um ou mais vetores. JavaScript nos possibilita fazer isso resultando duas formas diferentes.

7.8 Método join()

O vetor construído com o objeto Array de JavaScript pode ter a junção de seus elementos. A junção consiste em criar uma string única usando separadores.

O método **join()** nos possibilita a junção de forma simples, precisa e correta em uma string.

Por exemplo, vamos imaginar que temos um vetor que possui os dados de uma data qualquer em seus índices e que queremos mostrar a saída dessa data inteira. Há duas formas que podemos fazer isso e são complicadas.

Uma delas é criar um loop entre os dois primeiros índices e depois mostrar apenas o último índice.

```

<script type="text/javascript">
    var data = new Array(3);
    data[0] = 27;
    data[1] = 3;
    data[2] = 2009;

    for (i = 0; i<data.length-1; i++) {
        document.write (data[i] + "/");
    }
    document.write (data[2]);
</script>

```

A outra forma, apesar de mais simples, ainda não é a mais indicada.

```

<script type="text/javascript">
    var data = new Array(3);
    data[0] = 27;
    data[1] = 3;
    data[2] = 2009;
    document.write (data.join("/"));
</script>

```

Vale lembrar que o método join não modifica o vetor original, mas é possível guardar seu resultado em uma variável. Ex.: dataCompleta = data.join("/") .

7.8.1 Método concat()

O método **concat()** consiste em unir um ou mais arrays (vetores). Este método usa como argumento um objeto do tipo array. Se desejarmos unir mais de um vetor, cada objeto array do argumento deve vir separado por vírgula (,).

Código:

```

<script type="text/javascript">
    var alunosAno1 = new Array ("André","Sansão","Raquel");
    var alunosAno2 = new Array ("Marcos","Valdenia","Fabiana");
    var alunosAno3 = new Array ("Thiago","Elinardy","Krystian");
    var alunosAno4 = new Array ("Fabricio","Patricia","Atila");
    var todosAlunos = alunosAno1.concat(alunosAno2,alunosAno3,alunosAno4);
    document.write ("Esta escola tem " + todosAlunos.length +
        " alunos no projeto e-Jovem.<br>" + "Eles se chamam:<br>" + todosAlunos);
</script>

```

Aqui, o resultado obtido na variável todosAlunos é o nome contido nas quatro variáveis alunosAno.

Saída:

Esta escola tem 12 alunos no projeto e-Jovem. Eles se chamam:

André,Sansão,Raquel,Marcos,Valdenia,Fabiana,Thiago,Elinardy,Krystian,Fabricio,Patricio,Atila

Assim como no método join(), concat() também não altera o conteúdo original dos vetores.

7.8.2 Método push()

Além da forma algorítmica de inserir um elemento no final de um array, o próprio objeto Array possui um método que faz isso por nós.

O método push() insere um ou mais elementos no final de um array. A sintaxe é: push (elemento1, elemento2, elemento3, ...)

Dessa forma, não precisamos nos preocupar com o tamanho e nem com o último índice do array.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Cristiane");
    document.write ("Este array possui "+vetor.length+" elementos.<br>");
    vetor.push ("Nogueira");
    document.write("Agora possui "+vetor.length+" elementos.<br>");
    vetor.push ("Wallison","Ledruwick","Crispiano","Mateus");
    document.write ("Este vetor possui "+vetor.length+
        " elementos.<br>"+Que são: "+vetor.join(", "));
</script>
```

Saída:

Este array possui 1 elementos.
Agora possui 2 elementos.
Este vetor possui 6 elementos.
Que são: Cristiane, Nogueira, Wallison, Ledruwick, Crispiano, Mateus

7.8.3 Método unshift()

Outra forma de inserir elementos é utilizar o método unshift(). O método unshift() insere um ou mais elementos no início de um array.

A sintaxe é: unshift (elemento1, elemento2, elemento3, ...)

Um detalhe importante é o fato de que este método possivelmente não funcione corretamente em versões

anteriores ao Internet Explorer 6.

Código:

```
<script type="text/javascript">
    vetor = new Array ("5","6","7","8","9","10","J","Q","K");
    document.write ("Este baralho tem apenas as cartas "+vetor.join(", ") +
    ". Vamos completá-lo com as cartas iniciais. <br>");
    vetor.unshift ("A","1","2","3","4");
    document.write("Agora temos o baralho completo: "+vetor.join(", "));
</script>
```

Saída:

Este baralho tem apenas as cartas 5, 6, 7, 8, 9, 10, J, Q, K. Vamos completá-lo com as cartas iniciais.
Agora temos o baralho completo: A, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

7.8.4 Método pop()

O método **pop()** remove um elemento do final de um array e retorna o conteúdo do índice removido.

Não precisamos nos preocupar com o tamanho e nem com o último índice do array, pois **pop()** fará todo o trabalho sozinho.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Marcelo","Patricia","Joelma","Diana");
    document.write ("Este array possui "+vetor.length+" elementos.<br>");
    document.write ("Vamos remover "+vetor.pop()+" da lista.<br>");
    document.write ("Agora, temos "+vetor.length+" elementos. <br>" +
    "Que são: "+vetor.join(", "));
</script>
```

Saída:

Este array possui 4 elementos.
Vamos remover Diana da lista.
Agora, temos 3 elementos.
Que são: Marcelo, Patricia, Joelma

7.8.5 Método shift()

Outra forma de remover elementos é utilizar o método **shift()**. O método **shift()**remove um elemento do início de um array e retorna o conteúdo do índice removido.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Marcelo","Patricia","Joelma","Diana");
    document.write ("Este array possui "+vetor.length+ " elementos.<br>");
    document.write ("Vamos remover "+vetor.shift()+" da lista.<br>");
    document.write ("Agora, temos "+vetor.length+ " elementos. <br>" +
        "Que são: "+vetor.join(", "));
</script>
```

Saída:

Este array possui 4 elementos.
 Vamos remover Marcelo da lista.
 Agora, temos 3 elementos.
 Que são: Patricia, Joelma, Diana

7.8.6 Método reverse()

O método `reverse` do objeto array serve para inverter a ordem dos elementos de um array. Dessa forma podemos usar apenas uma forma de organização e depois inverter o array para conseguir a ordem desejada.

Código:

```
<script type="text/javascript">
    var vetor = Array (1,2,3,4,5);
    document.write ("Array inicial: " + vetor);
    document.write ("<br> Array invertido: " + vetor.reverse());
</script>
```

Saída:

Array inicial: 1,2,3,4,5
 Array invertido: 5,4,3,2,1

7.8.6 Método sort()

O método `sort` é o que realmente faz a organização do array. Porém, `sort()` apenas organiza o array de forma alfabética, se quisermos organizar o array de forma numérica devemos criar uma função para isso.

Abaixo está um exemplo de uma organização simples feita em ordem alfabética.

Código:

```
<script type="text/javascript">
    vetor = new Array ("Elinny","André","Adriano","Elisabete");
    document.write (vetor.sort());
</script>
```

Veja que *vetor* é organizado corretamente de forma alfabética.

Saída:

Adriano,André,Elinny,Elisabete

Agora, podemos perceber que o resultado da organização de números não é dado de forma satisfatória.

Código:

```
<script type="text/javascript">
    vetor = new Array (3000,20,100,4);
    document.write (vetor.sort());
</script>
```

Saída:

100,20,3000,4

Exercícios Resolvido

- Quais serão as letras que serão geradas a partir desse código?

```
347<script type="text/javascript">
348    var valor = 2;
349    var indice = valor;
350    var letras = new Array('d','b','e','a','i','o','r');
351
352    document.write(letras[indice++]);
353    document.write(letras[indice]);
354    document.write(letras[indice-2]);
355    document.write(letras[0]);
356    document.write(letras[valor+1]);
357 </script>
```

Entendendo o algoritmo

- No escopo temos uma variável chamada “valor” = 2 e “índice” que recebe o conteúdo que está em valor que é 2 também.
- Temos também um array que irá receber um conjunto descrito, lembrando que o índice inicial de um vetor é zero até a sua dimensão menos um. Nesse caso temos 7 elementos, então os índices para o meu vetor são de zero até 6.

- Para cada impressão, temos uma letra seguido de uma concatenação com um espaço em branco, então podemos chegar a conclusão que irá ser impresso um conjunto de letras com e espaços.
- O primeiro “document.write” irá imprimir o valor de “índice” = 2 que é representado pela letra “e”, em seguida índice será incrementado passando a ser igual a 3.
- O segundo “document.write” irá imprimir o valor de índice = 3 que é representado pela letra “a”.
- O terceiro “document.write” irá imprimir o valor de (índice - 2) = (3-2) = 1 que é representado pela letra “b”
- O quarto “document.write” irá imprimir o índice zero do meu array representado pela letra “d”
- O quinto “document.write” irá imprimir o índice (valor +1) = (2 + 1) = 3 representando pela letra “a”
- Por fim podemos identificar a mensagem no JAVASCRIPT: e a b d a.

Exercícios Propostos:

O que é um array, e qual a sua principal finalidade? Declare um array chamado “nomes” com 8 posições, e grave nomes de pessoas que você conhece em cada uma delas. Após criar o array responda as questões 2, 4, 5, 10, 11:

Utilizado o array responda.

a) Qual nome é impresso no navegador se colocarmos o código:

```
document.write(nomes[3]);
```

b) Quais nomes aparecerá se adicionamos os seguintes códigos:

```
for(i= 6; i>1 ; i--){ document.write(nomes[i]); }
```

c) O que acontece se chamarmos uma posição que não existe, exemplo: nomes[15];

O que é um array associativo, de exemplos:

Usando o comando *for...in*, crie uma interação onde todos os nomes possa ser impresso na tela.

Utilizando o mesmo código, altere alguns nomes do array.

O que é um array multidimensional?

Crie um array “palavras” multidimensional 5x3 com os valores da tabela abaixo, e responda as questões 7,8,9:

“oi”	“tudo”	“estar”
“você”	“vai”	“?”
“com”	“dia”	“!”
“,”	“bem”	“sim”
“casa”	“hoje”	“em”

Crie um código JAVASCRIPT onde com os valores do array possa ser impresso na tela com a frase “oi, tudo bem com você?”.

Utilizando as posições da sequência [1][0],[1][1],[0][2],[4][2],[4][0],[4][1],[1][2] do array palavras, qual frase podemos formular? Utilize a função document.write() para mostrar na tela do navegador.

Construa um código JAVASCRIPT para mostra uma resposta para a pergunta da questão 7. (Use o comando document.write para imprimir na tela).

Utilizando a função sort(), imprima em ordem alfabética os nomes do array “nomes”.

Use o comando unshift() para adicionar mais dois nomes no array.

Crie um documento HTML com um código JavaScript que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais e em seguida imprima-os na tela. Em seguida, escolha duas posições aleatoriamente e troque os valores de uma posição pelo da outra. Repita essa operação 10 vezes. Ao final, imprima o array novamente.

Crie um documento HTML com um código JavaScript que armazene 10 números inteiros em um array. Preencha todas as posições com valores aleatórios e em seguida imprima-os. Ordene do menor valor para o maior e imprima novamente.

8.0 Manipulação de Funções

Quando queremos um código funcional para determinado fim, com por exemplo fazer um cálculo ou alguma interação dentro do JavaScript, usamos o que chamamos de função. As funções são um pedaço de código com o objetivo específico, encapsulado sob uma estrutura única que recebe um conjunto de parâmetros e retorna ou não um determinado dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes. É uma das estruturas mais básicas para prover reusabilidade ou reaproveitamento de código, deixando as funcionalidades mais legíveis.

8.1 Declarando uma Função.

Declaramos uma função, com o uso do operador `function` seguido do nome que devemos obrigatoriamente atribuir, sem espaços em branco e iniciando sempre com uma letra. Temos na mesma linha de código a declaração ou não dos argumentos pelo par de parênteses “()”. Caso exista mais de um parâmetro, usamos vírgula(,) para fazer as separações. Logo após encapsulamos o código pertencente a função por meio das chaves ({}). No final, temos o retorno com o uso da cláusula `return` para retornar o resultado da função que pode ser um tipo inteiro, array, string, ponto flutuante etc. A declaração de um retorno não é obrigatória. Observe a sintaxe:

```
function nome_da_função( argumento_1, argumento_2, argumento_n )
{
    comandos; return $valor;
}
```

Observe um exemplo onde criamos uma função para calcular o índice de massa corporal de uma pessoa (IMC), onde recebe como parâmetro dois argumentos. Um é a altura representada pela variável \$altura e o outro é o peso representada pela variável \$peso. Passamos como parâmetros para essa função o peso = 62 e a altura = 1.75. Observe:

```
<script type="text/javascript">
    //Declarando a função
    function calculo_IMC(peso, altura){
        return peso/(altura*altura);
    }
    //Chamando a função
    document.write(calculo_IMC(62,1.75));
</script>
```

Exemplo:

Resultado: 20.244897959183675

Nesse exemplo temos a declaração e logo após a chamada da função, onde é nesse momento que passamos os dois parâmetros na ordem que foi declarada na função. Lembrando que essa ordem é obrigatória. Observe mais um exemplo, onde a função declarada, porém não possui a cláusula **return**.

```
<script type="text/javascript">
    //Declarando a função
    function imprime(texto){
        document.write("<h1>" + texto + "</h1>");
    }
    //Chamando a função
    imprime("Testando a função.");
</script>
```

8.2 Escopo de Variáveis em Funções

Um conceito importante em programação são os tipos de declarações de variáveis, onde sua visibilidade vai depender de onde ela é declarada. O acesso a essas variáveis pode ser definido da seguinte forma:

Variáveis locais < São aquelas declaradas dentro de uma função e não tem visibilidade fora dela.

Variáveis Globais < São variáveis declaradas fora do escopo de uma função, porém tem visibilidade (pode ser acessada) ao contexto de uma função sem passá-la como parâmetro. Para isso declaramos a variável no escopo fora da função e fazemos a sua chamada dentro da função.

8.3 Valor de Retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum. Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou array's. As operações aritméticas podem ser feitas de forma direta no retorno. Observe um exemplo onde temos uma operação direta:

```
<script type="text/javascript">
    function somar(n1,n2){
        return n1+n2;
    }
    function texto () {
        return "O resultado é: ";
    }
    document.write(texto() + somar(115,23));
    //Resultado: 138
</script>
```

Também podemos determinar mais de um retorno desde que eles não sejam acessados ao mesmo tempo, observe o exemplo:

```
<script type="text/javascript">
    function teste(caso){
        switch(caso){
            case 1:
                return "opção 1";
            case 2:
                return "opção 2";
            case 3:
                return "opção 3";
            default:
                return null;
        }
    }
    alert(teste(2));
</script>
```

Esse código mostra de forma clara que não existe a possibilidade de retornarmos mais de um **return**, caso isso ocorresse, teríamos um erro, ou não funcionamento da função.

8.4 Recursão.

Função recursiva é uma definição usada tanto na programação quanto na matemática, onde, significa que uma função “faz a chamada” de si mesma na sua execução. Um exemplo é o cálculo do fatorial de um número. Observe:

```
<script type="text/javascript">
    function fatorial(numero){
        if (numero === 1) {
            return numero;
        }else{
            return numero * fatorial(numero-1);
        }
    }
    alert(fatorial(5)); //Resultado: 120
</script>
```

Fatorial de 5: $5! = 5 \cdot 4!$, $4! = 4 \cdot 3!$, $3! = 3 \cdot 2!$, $2! = 2 \cdot 1!$ ou $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.

Exercícios Propostos

Diga com suas palavras uma definição para função, e como podemos declará-la em PHP.

Qual a diferença de variáveis globais para variáveis locais e como podemos defini-las em PHP?

Quais as funções que podemos usar para criarmos uma função onde seus parâmetros são passados por argumentos variáveis?

O que é um valor de retorno e qual o comando usado quando queremos retornar algo dentro de uma função?

O que é recursão?

Crie uma função que determine se um numero é par ou ímpar. E faça uma chamada dessa função imprimindo o resultado.

Crie uma função que calcule a fatorial de um número.

Crie uma função para determina se um numero é primo ou não. Numero primo é aquele que possui dois divisores, 1 e ele mesmo. Crem um laço de repetição e use estrutura de controle.

9.0 Eventos JavaScript

Quando um usuário visita uma página web e interage com ela se produzem os eventos e com JavaScript podemos definir o que queremos que ocorra quando se produzam.

Com JavaScript podemos definir o que acontece quando se produz um evento, como por exemplo quando um usuário clica sobre um botão, edita um campo de texto ou abandona a página um evento é lançado e podemos definir que alguma função seja executada quando isso acontecer.

9.1 Relação de eventos

Abaixo uma lista de eventos suportados pelo JavaScript com suas respectivas descrições e a partir de que versão do JavaScript esse evento foi incorporado.

- **onabort**

Este evento se produz quando um usuário cancela o carregamento de uma imagem, seja porque para o carregamento da página ou porque realiza uma ação que a detém, como por exemplo, sair da página. JavaScript 1.1

- **onblur**

Desata-se um evento onblur quando um elemento perde o foco da aplicação. O foco da aplicação é o lugar onde está situado o cursor, por exemplo, pode estar situado sobre um campo de texto, uma página, um botão ou qualquer outro elemento. JavaScript 1.0

- **onchange**

Desata-se este evento quando muda o estado de um elemento de formulário, às vezes não se produz até que o usuário retire o foco da aplicação do elemento. JavaScript 1.0

- **onclick**

Produz-se quando se clica o botão do mouse sobre um elemento da página, geralmente um botão ou um link. JavaScript 1.0

- **ondragdrop**

Produz-se quando um usuário solta algo que havia arrastado sobre a página web. JavaScript 1.2

- **onerror**

Produz-se quando não se pode carregar um documento ou uma imagem e esta fica quebrada. JavaScript 1.1

- **onfocus**

O evento onfocus é o contrário de onblur. Produz-se quando um elemento da página ou a janela ganham o

foco da aplicação. JavaScript 1.0

- **onkeydown**

Este evento é produzido no instante que um usuário pressiona uma tecla, independentemente que a solte ou não. É produzido no momento do clique. JavaScript 1.2

- **onkeypress**

Ocorre um evento onkeypress quando o usuário deixa uma tecla clicada por um tempo determinado. Antes deste evento se produz um onkeydown no momento que se clica a tecla. JavaScript 1.2

- **onkeyup**

Produz-se quando o usuário deixa de apertar uma tecla. É produzido no momento que se libera a tecla. JavaScript 1.2

- **onload**

Este evento se desata quando a página, ou em JavaScript 1.1 as imagens, terminaram de se carregar. JavaScript 1.0

- **onmousedown**

Produz-se o evento onmousedown quando o usuário clica sobre um elemento da página. onmousedown se produz no momento de clicar o botão, soltando ou não. JavaScript 1.2

- **onmousemove**

Produz-se quando o mouse se move pela página. JavaScript 1.2

- **onmouseout**

Desata-se um evento onmouseout quando a seta do mouse sai da área ocupada por um elemento da página. JavaScript 1.1

- **onmouseover**

Este evento desata-se quando a seta do mouse entra na área ocupada por um elemento da página. JavaScript 1.0

- **onmouseup**

Este evento se produz no momento que o usuário solta o botão do mouse, que previamente havia clicado. JavaScript 1.2

- **onmove**

Evento que se executa quando se move a janela do navegador, ou um frame. JavaScript 1.2

- **onresize**

Evento que se produz quando se redimensiona a janela do navegador, ou o frame, no caso de que a página

os tenha. JavaScript 1.2

- **onreset**

Este evento está associado aos formulários e se desata no momento que um usuário clica no botão de reset de um formulário. JavaScript 1.1

- **onselect**

Executa-se quando um usuário realiza uma seleção de um elemento de um formulário. JavaScript 1.0

- **onsubmit**

Ocorre quando o visitante aperta sobre o botão de enviar o formulário. Executa-se antes do envio propriamente dito. JavaScript 1.0

- **onunload**

Ao abandonar uma página, seja porque se clique em um link que nos leva a outra página ou porque se fecha a janela do navegador, se executa o evento onunload.

Destes eventos iremos selecionar e estudar os principais nos próximos tópicos

9.2 Evento onclick

O evento onclick é lançado/disparado quando clicamos em um elemento da página, geralmente um botão ou um link. Esse evento é valido para os objetos ou componentes de página HTML: Button, Checkbox, Radio, Link, Reset e Submit.

Sintaxe

```
<elemento onclick="AlgumCodigoJavaScript">
```

Em HTML:

Exemplo: onclick

Quando o usuário clicar no botão “Clique aqui” o evento onclick será disparado e chamara automaticamente a função de nome “minhaFuncao” a qual irá imprimir a seguinte String em um caixa de diálogo alert: “O evento onclick foi disparado”.

```
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>
            Onclick
        </title>

        <script type="text/javascript">
            function minhaFuncao(numero){
                alert("Evento Onclick");
            }
        </script>
    </head>

    <body>
        <p>Clique no botão </p>
        <button onclick="minhaFuncao()">Clique Aqui</button>
    </body>
</html>
```

O mesmo exemplo poderia ser implementado dessa forma:

```
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>
            Onclick
        </title>
    </head>

    <body>
        <p>Clique no botão </p>
        <button onclick="alert('Evento Onclick')">Clique Aqui</button>
    </body>
</html>
```

9.3 Evento onsubmit

O evento onsubmit é suportado apenas por formulários e disparado quando o usuário clica no botão Enviar de seus formulários. É neste evento que você fará a validação do formulário, assim como enviará o usuário para uma página, informando do sucesso/falha do envio de dados, ou outra ação que você queira definir no momento.

Sintaxe em HTML:

```
<form onsubmit="AlgumCodigoJavaScript">
```

Exemplo: onsubmit

Para observarmos o evento onsubmit iremos fazer a validação de um formulário de login que irá conter os campos nome de usuário e senha que serão verificados quando o usuário clicar no botão logar. Essa ação do usuário irá gerar um evento onsubmit o qual irá chamar a função “validacao()”.

Vamos construir o nosso formulário de login com a nossa função validacao().

```
1  <!DOCTYPE html>
2  <html lang="pt">
3      <head>
4          <meta charset="utf-8">
5          <title>
6              Página Principal
7          </title>
8          <script type="text/javascript">
9              function validacao(){
10                  var user = "jonas";
11                  var password = "qwe123";
12                  if ((user == document.form1.usuario.value)
13                      &&(password==document.form1.senha.value)){
14                      window.alert("Seja bem-vindo "+user+"");
15                      return true;
16                  }else{
17                      window.alert("Usuário ou senha inválido");
18                      return false;
19                  }
20              }
21          </script>
22      </head>
23
24      <body>
25          <form action="pagina1.html" name="form1" method="post" onsubmit="return validacao();">
26              <center>
27                  <table>
28                      <tr>
29                          <td>Digite o nome de usuário</td>
30                          <td><input type="text" name="usuario"></td>
31                      </tr>
32                      <tr>
33                          <td>Digite a senha:</td>
34                          <td><input type="password" name="senha"></td>
35                      </tr>
36                  </table>
37                  <table>
38                      <tr>
39                          <td><input type="submit" value="Logar"></td>
40                          <td><input type="reset" value="Limpar"></td>
41                      </tr>
42                  </table>
43              </center>
44          </form>
45      </body>
46  </html>
```

Vamos analisar nossa função validacao():

```
<script type="text/javascript">
    function validacao(){
        var user = "jonas";
        var password = "qwe123";
        if ((user == document.form1.usuario.value)
            &&(password==document.form1.senha.value)){
            window.alert("Seja bem-vindo "+user+"");
            return true;
        }else{
            window.alert("Usuário ou senha inválido");
            return false;
        }
    }
</script>
```

Observe que nossa função retorna **true** se o usuário e senha digitados no componente de name “usuario” e de name “senha” do formulário de name “form1”, sejam for “jonas” e “qwe123” respectivamente, caso contrário ela retorna **false**.

Esse valor retornado pela função **validacao()** irá ser salvo no evento onsubmit que caso seja **false** irá impedir que o formulário seja direcionado para a página html de nome pagina1.html , caso contrário irá permitir o redirecionamento da página.

- **Atenção**

Perceba que adicionamos a palavra reservada **return** antes da chamada de nossa função, essa palavra foi utilizada com intuito de que o evento onsubmit espere e receba um retorno da função **validacao()**.

Evento onmouseover e onmouseout

O evento onmouseover ocorre quando o ponteiro do mouse se posiciona sobre um elemento da página HTML, já o evento onmouseout acontece o oposto, o evento é disparado quando retiramos o ponteiro do mouse de sobre um determinado elemento.

Sintaxe em HTML:

```
<elemento onmouseover="AlgumCodigoJavaScript">
<elemento onmouseout="AlgumCodigoJavaScript">
```

Exemplo:

```
<!DOCTYPE html>
<html>
<head>
<title>Usando eventos no Javascript</title>

<script type="text/javascript">
function mOver(obj) { obj.innerHTML="Obrigado" }
function mOut(obj) { obj.innerHTML="Passe o mouse" }
</script>
</head>
<body>
<div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-
color:gray; width:120px; height:20px; padding:40px;">Passe o mouse
</div>
</body>
</html>
```

9.4 Eventos: onfocus e onblur

O evento onfocus ocorre quando um elemento/objeto recebe o foco. Ex: quando você clica em um campo de texto para digitar algo, aquele elemento ganhou foco e nessa ação é gerado um evento do tipo onfocus, já o onblur é o oposto o evento é gerado quando o elemento perde o foco, ou seja você tem 2 campos de textos para digitar seus dados. O primeiro você deve digitar seu nome e no outro o seu email. Após digitar o seu nome você terá que digitar o email, certo? Se você tivesse colocado um evento onBlur no campo onde digitou o nome, ao clicar no campo de e-mail um evento do tipo onBlur será gerado, pois o campo nome acaba de perder o foco na aplicação.

```
<element onfocus="AlgumCodigoJavaScript">
<element onblur="AlgumCodigoJavaScript">
```

Sintaxe em HTML:

Exemplo:

Vamos implementar dois campos nome e e-mail, os quais inicialmente estarão preenchidos com a String “digite aqui” quando for dado foco a um dos campos o evento onfocus será gerado e o campo de texto que ganhou foco e que estava inicialmente com uma frase se encontrará vazio para ser digitado uma informação do usuário, caso o usuário não digite nenhuma informação, por esquecimento, e retire o foco do campo de texto. O evento onblur será gerado e uma frase em vermelho irá surgir dentro do campo que perdeu o foco “campo obrigatório”.

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>Usando eventos no Javascript</title>
6          <script type="text/javascript">
7              function alertaCampoVazio(campo){
8                  if (campo.value == '') {
9                      campo.value = 'campo obrigatório';
10                     campo.style.color = "red";
11                 } else{
12                     if (campo.value == 'digite aqui' || campo.value == 'campo obrigatório') {
13                         campo.value = '';
14                         campo.style.color = "black";
15                     };
16                 }
17             }
18         </script>
19     </head>
20     <body>
21         <label>Nome:</label>
22         <input value="digite aqui" type="text"
23             onblur="alertaCampoVazio(this);"
24             onfocus="alertaCampoVazio(this);">
25         <label>E-Mail:</label>
26         <input value="digite aqui" type="text"
27             onblur="alertaCampoVazio(this);"
28             onfocus="alertaCampoVazio(this);">
29     </body>
30 </html>
31

```

9.5 Evento: onchange

O evento onchange é disparado sempre que o texto de um elemento foi alterado. Nós podemos usá-lo para fazer uma verificação em um valor digitado pelo usuário e informá-lo ou instruí-lo a digitar novamente. Este evento está disponível para os campos texto, áreas de texto e listas de seleção.

Sintaxe em HTML:

```
<elemento onchange="AlgumCodigoJavaScript">
```

Exemplo:

Como demonstração do uso desse evento, vamos validar um campo de senha. Iremos criar um campo para o usuário digitar sua senha e outro para que ele repita a senha digitada. O evento onchange será gerado quando o usuário modificar o conteúdo da caixa senha e fazer com que a mesma perca o foco. Quando o evento for gerado será verificado se o tamanho da senha digitada é maior ou igual a 6, caso seja verdade a sentença “Senha válida” será exibida em uma caixa de diálogo alert, caso contrário será exibido a frase “Tamanho da senha inválido, digite uma senha de no mínimo 6 dígitos”, o campo será limpo e ganhará o foco da aplicação novamente.

Veja abaixo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando eventos no Javascript</title>
    <script type="text/javascript">
      function valida(campo){
        if (campo.value.length >=6) {
          alert("Senha válida");
        } else{
          alert("A senha deve ter no mínimo 6 dígitos");
          campo.value = '';
          campo.focus();
        }
      }
    </script>
  </head>
  <body>
    <label>Senha:</label>
    <input value="" type="password" onchange="valida(this);">
    <label>Repita a senha:</label>
    <input value="" type="password">
  </body>
</html>
```

9.6 Eventos: **onload** e **onunload**

O onload é um dos eventos mais importantes em JavaScript. Ele é disparado sempre que a página estiver carregada por completo. Todas as vezes que você escrever um script que só possa ser executado usando outros elementos da página, é sempre uma boa idéia usar este evento para dar início à execução do código. Por só ocorrer quando a página estiver carregada por completo, este evento evita que o seu código tente acessar algum elemento que não esteja baixado para a página ainda.

Já o evento onunload é disparado assim que usuário encerra a execução da página atual para exibir uma nova página. Ele pode ser usado para dar alguma informação, agradecer ao usuário a sua visita ou exibir uma nova página. Este evento também ocorre se você usar o botão Voltar do browser.

Sintaxe m HTML:

```
<body onload="AlgumCodigoJavaScript">
<body onunload="AlgumCodigoJavaScript">
```

Exemplo:

Vamos ver um exemplo da aplicação destes eventos. Queremos exibir uma mensagem de boas-vindas

quando o visitante acessar a nossa página e outra mensagem quando ele sair da nossa página.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando eventos no Javascript</title>
    <script type="text/javascript">
      function saudacaoChegada(){
        alert("Seja bem vindo ao nosso site!");
      }
      function saudacaoSaida(){
        alert("Volte sempre ao nosso site!");
      }
    </script>
  </head>
  <body onload="saudacaoChegada()" onunload="saudacaoSaida()">
    <h1>Teste do onload e onunload</h1>
    <p>
      "Tinha um monte de coisas que eu queria fazer...<br>
      Eu quero ser gerente, eu quero tbm ser astronauta...<br>
      Eu quero ter minha propria loja de bolos...<br>
      Eu quero ir numa loja de rosquinhas e dizer "eu quero todas!"...<br>
      Eu quero poder viver umas 5 vezes...<br>
      Entao eu renasceria em 5 cidades diferentes...<br>
      Me encheria de coisas diferentes e deliciosas 5 vezes cada...<br>
      E entao nessas 5 vezes... eu me apaixonaria pela mesma pessoa" <br>
      <i>Inoue Orihime - Bleach</i>
    </p>
  </body>
</html>
```

Atenção! O evento onunload não é suportado pelo navegador Google Chrome ou Opera, nem por algumas versões do Mozilla Firefox.

Exercícios Propostos

Criar um contador do número de dias transcorridos desde uma determinada data, conforme formulário mostrado abaixo:

Entre com a data no formato MM/DD/AAAA	
Data:	<input type="text"/>
Transcorridos	<input type="text"/> dias.
<input type="button" value="Limpar"/>	<input type="button" value="Calcular"/>

Crie um programa que apresente cinco retângulos de 100px por 200 px de cores vermelho, verde, azul amarelo e rosa. O sistema deverá então, randomicamente, o retângulo que contém um prêmio. Quando o usuário clicar em um dos retângulos, o sistema deverá então informar se aquele é o premiado.

Caso, contrário, deverá informar que o usuário errou, informando qual era o correto. (Dica: estude o método random() contido no Objeto Math)

Crie um botão em um ficheiro html, que ao ser clicado aparece um número aleatório entre 0 e 50 e 50

Crie um botão em um ficheiro html, que ao ser clicado aparece um número aleatório entre 0

Crie uma caixa de texto em html. Crie um botão em html que chama código em JavaScript que dependendo do valor introduzido mostra uma mensagem diferente;

- Entre 0 e 10, 10 excluído, mostra “Insuficiente”;
- Entre 10 e 14, 14 excluído, mostra “Bom”;
- Maior que 14, mostra “Muito Bom”

Crie uma caixa de texto e um botão em html; ao clicar no botão, chama uma função com um parâmetro que é o valor que está dentro da caixa de texto; A função mostra mensagens de acordo com o parâmetro da caixa de texto; por exemplo, no caso de ter introduzido três, a função seria chamado com o parâmetro 3, assim, as mensagens seriam “AIA1”, “AIA 2” e “AIA 3”

10.0 Document Object Model – DOM

Para ter um domínio completo de JavaScript, é preciso que você passe pelo estudo do DOM (Document Object Model), que é o modelo dos objetos que formam a hierarquia de toda a estrutura de uma página web. Essa estrutura começa a partir da janela do navegador e vai até o último elemento da página.

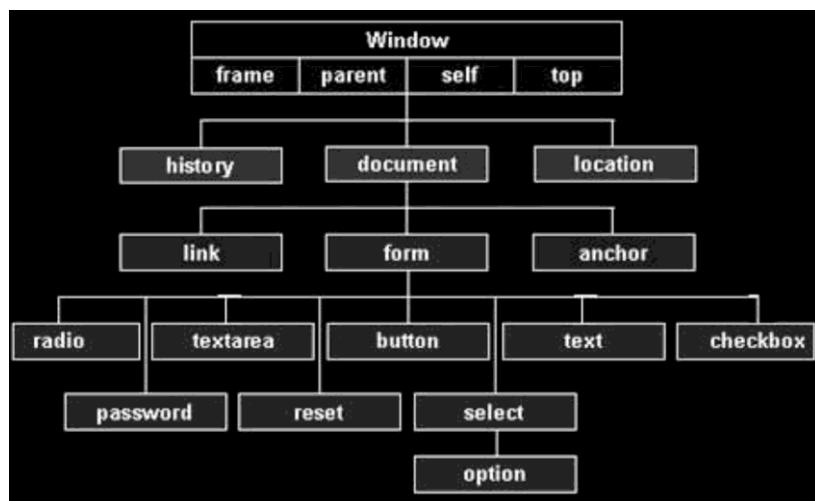
Para entendermos melhor o que é uma hierarquia de objetos, considere o seguinte exemplo.

Suponhamos que você esteja abrindo a porta de seu carro. No momento que abre a porta, se pudéssemos demonstrar em código o que você acaba de fazer, teríamos algo como o mostrado em seguida:

```
carro.porta.fechadura.aberta = true;
```

Observe que acessamos a propriedade aberta da porta, que retorna dois valores. Se estiver aberta, o valor será true e se estiver fechada, será false. Observe o ponto (.) que separa os elementos da hierarquia.

Em JavaScript as coisas não são diferentes. Observe a figura em seguida que mostra bem toda a hierarquia dos objetos em JavaScript:



Vamos agora escrever uma página HTML bem simples para que possamos ver o DOM em ação. Crie uma nova página HTML, digitando o código seguinte no seu editor favorito. Veja como a página ficará.

Lembre-se de criar as imagens que farão parte da página.

Demonstração do DOM em JavaScript



JavaScript

Para receber nossa newsletter digite o seu nome:


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Usando DOM</title>

  </head>
  <body align="center">
    <h3>Demonstração do DOM em JavaScript</h3>
    
    <a href="http://ead.projetojovem.seduc.ce.gov.br/ejovem/"></a>
    <p>Para receber nossa newsletter dífite o seu nome:</p>
    <form name="form1">
      <input type="text" name="text1">
      <input type="button" value="Enviar"><br><br>
    </form>
  </body>
</html>
```

E aqui está o código para a página:

Observe que a partir de agora, sempre que você criar um formulário, deverá dar nome a ele. Isso é fácil de entender. Como todos os elementos de uma página seguem o modelo de hierarquia, você deve seguir a para conseguir acessar as propriedades dos objetos. Vamos ver algum exemplo como isso é possível.

Depois de criada a página apresentada, salve-a em uma pasta. É essa página que usaremos para os nossos testes.

O primeiro script que vamos criar vai obter o título da página e exibi-lo em uma caixa de mensagem.

Abra a nossa página de testes e digite o código seguinte dentro das tags <head></head>.

```
<script type="text/javascript">
    window.alert(document.title);
</script>
```

Observe que tudo que fizemos aqui foi criar uma mensagem alert e definir que a mensagem a ser exibida será o título da página. Para acessar a propriedade title do objeto document, você só precisa usar a linha de código:

```
document.title;
```

Como este objeto está logo abaixo do objeto window na hierarquia, você pode perfeitamente acessar a propriedade acima, usando a linha de código seguinte:

```
window.document.title;
```

Por padrão, não é necessário usar o nome do objeto window sempre que for acessar o objeto document. Por esta razão nós o omitimos no script.

O nosso próximo script é um pouco mais avançado. Agora vamos criar uma função que altera o título da página em tempo de execução. Pense como uma visitante ficaria satisfeita se você exibisse o nome dele na barra de títulos do navegador.

Ainda com a nossa página de testes, digite o código abaixo na parte <body></body> da página para criarmos um botão. Para separar esse botão do outro conteúdo da página, digite algumas quebras de linhas
 para colocá-lo mais abaixo na página:

```
<form name="form1">
    <input type="text" name="text1">
    <input type="button" value="Enviar"><br><br>
    <input type="button" value="Alterar o Título da página" onclick="alterarTitulo()">
</form>
```

```

<script type="text/javascript">
    window.alert(document.title);
    function alterarTitulo () {
        window.document.title = "Seja bem vindo Samia";
    }
</script>

```

Agora, com o botão devidamente criado, digite o código seguinte na parte <head></head> da página:

Nossa página ficará dessa forma.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Usando DOM</title>
        <script type="text/javascript">
            window.alert(document.title);
            function alterarTitulo () {
                window.document.title = "Seja bem vindo Samia";
            }
        </script>
    </head>
    <body align="center">
        <h3>Demonstração do DOM em JavaScript</h3>
        
        <a href="http://ead.projetoejovem.seduc.ce.gov.br/ejovem/"></a>
        <p>Para receber nossa newsletter digite o seu nome:</p>
        <form name="form1">
            <input type="text" name="text1">
            <input type="button" value="Enviar"><br><br>
            <input type="button" value="Alterar o Título da página" onclick="alterarTitulo()">
        </form>
    </body>
</html>

```

Salve a página e execute.

Você vai perceber que, assim que clica no botão, ele chama a função alterarTitulo() e o título da página é alterado. Legal, não?

Agora, que tal colocar o nome do visitante na barra de títulos? Vamos ao nosso script:

O código seguinte pede que o visitante digite o seu nome e depois o exibe na barra de títulos juntamente com o título da página. Digite o código seguinte na parte <body></body> da página de teste:

```

<script type="text/javascript">
    var nome = window.prompt("Qual é o seu nome?", "Seu nome");
    var titulo = document.title + " - Visitante: " + nome;
    document.title = titulo;
</script>

```

Que tal usar isso em alguma página? Tenho certeza de que os seus visitantes ficarão deslumbrados.

Vamos entender um pouco como o código foi feito.

Primeiro nós criamos uma variável chamada nome e atribuímos a ela uma mensagem prompt, perguntando o nome do visitante. Criamos também uma variável chamada "titulo" que receberá o texto do título do documento mais o valor da variável "nome", que será informada pelo visitante. A partir daí é só alterar o título por meio da linha de código:

```
document.title = titulo;
```

Fácil, não?

Fico imaginando o que seria de nós, programadores em JavaScript, se não pudéssemos acessar os objetos por meio da hierarquia do DOM do JavaScript.

O script seguinte mostra como acessar uma caixa de texto e pegar o valor que foi digitado pelo usuário. Observe que sempre que quiser realizar tal tarefa, terá que dar nomes tanto ao formulário quanto aos seus elementos. Se você ainda não tem um conhecimento muito aprofundado de HTML, veja como dar nomes aos elementos:

```
<form name="meu_form">  
<input type="text" name="texto1">
```

Quando você dá nomes aos elementos, pode acessá-los e manipular suas propriedades do jeito que quiser. Você pode dar qualquer nome aos seus elementos. Apenas tente dar nomes sugestivos que façam com que você se lembre mais facilmente na hora de acessar cada um deles.

Voltando à nossa página de teste, você pode ver que temos um formulário chamado form1, uma caixa de texto chamada text1 e um botão. Observe que aqui não demos um nome ao botão, mas se quisermos acessar suas propriedades (faremos isso no próximo script), temos que dar um nome a ele.

O script que vamos escrever agora exibe uma mensagem assim que o usuário digita o nome na caixa de texto e clica no botão Enviar.

O texto a ser exibido usa o valor digitado na caixa de texto. Vamos ao código: Digite este código na parte <head></head> de sua página:

```
function enviar () {  
    var nome = document.form1.text1.value;  
    window.alert("Você digitou: " + nome);  
}
```

Agora, precisamos chamar esta função a partir do botão Enviar. Para tanto, vá até o botão e digite a seguinte linha de código:

```
onclick="enviar()"
```

O seu código completo para o botão deve ficar assim:

```
<input type="button" value="Alterar o Título da página" onclick="alterarTitulo()">
```

Salve a página, execute, digite o seu nome na caixa de texto e clique o botão Enviar. Se você realizou todos os procedimentos corretamente, verá uma caixa de mensagem exibindo o texto Você digitou "seu nome". Como nos exemplos anteriores, pratique bastante alterando valores e mixando estes scripts com as técnicas aprendidas anteriormente.

Exercícios Propostos

Crie uma função que seja capaz de receber a informação de 2 campos de texto e em seguida mostre no HTML os valores digitados.

No mesmo formulário melhore a função verificando se existe ou não algo digitado em um dos campos, se não tiver nada digitado, o html deverá mostrar uma mensagem dizendo campo vazio.

Crie um formulário para que irá receber 3 valores, converta-os para valores numéricos usando Objeto Number() e imprima a média dos valores digitados.

Crie um formulário com um campo de texto para uma nota e um combobox que terá as matérias: Física, matemática e português. Mostrei abaixo qual nota foi digitada e qual curso foi selecionado.

11.0 Introdução a jQuery

O FrameWork jQuery nada mais é que uma um conjunto de bibliotecas escritas em JavaScript, contendo funções que tratam de efeitos interativos em processos Web. O jQuery veio para facilitar os efeitos do JavaScript, realizando em poucas linhas uma vastas possibilidades de efeitos. Para ter um bom aprendizado, será importante conhecer bem o JavaScript, especialmente modelo de Objeto de Documento ou seja (**DOM**) e **Eventos**. Pois podemos ter interações com todos os elementos da estrutura básica de um site. Nesse curso iremos ver **alguns** comandos simples do jQuery, em seguida, nos focaremos na biblioteca **jQueryUI**, utilizado para abstrair a programação complexa de baixo nível em interações WEB/usuário aumentando a produtividade, funções e beleza do site.



Um exemplo de sua funcionalide:

Enquanto em Java Script escrevíamos: **document.getElementById("elemento")**

No jQuery só precisamos : **\$("#elemento")**

11.1 Instalação e configuração

Para realizar a instalação e configuração, primeiramente devemos acessar o site oficial da jQuery: "<http://code.jquery.com/>" E acessar o link de download. Você deverá copiar o conteúdo de texto e salvar no seu computador como "**jquery.js**". Como iremos trabalhar na versão: 2.1.4, o link direto que contém o a jquery é : "<http://code.jquery.com/jquery-2.1.4.min.js>". Assim que o arquivo tiver salvo, adicione-o dentro do seu projeto, de preferência em uma pasta para separá-la dos demais elementos do seu site .

Pronto, falta apenas importar a biblioteca para sua página Web, para isso, basta importar o jQuery na sua

```
<script type="text/javascript" src="js/jquery.js"></script>
```

página WEB digitando o seguinte comando na estrutura <head>:

Perceba que fizemos esse mesmo procedimento em capítulos anteriores para importação dos nossos projetos em JavaScript. A diferença que copiamos um projeto já feito pelo grupo da jQuery que importamos no nosso site. Podemos também criar o nosso projeto e importá-lo abaixo, unindo a nossa biblioteca com a do jQuery. Veja no exemplo abaixo;

```
<script type="text/javascript" src="js/anima.js"></script>
<script type="text/javascript" src="js/jquery.js"></script>
```

Obs.: Se por acaso precisarmos migrar um site para um outro projeto que tiver recursos do jQuery implementados, precisamos exportar a biblioteca junto ou linkar o arquivo externamente alterando o caminho src para um endereço da página web onde encontra-se a biblioteca jQuery, no nosso caso: “code.jquery.com/jquery-2.1.4.min.js”.

11.2 Primeiros Passos

Para inicializar as funções do JavaScript e jQuery precisamos primeiro criar um evento de inicialização do documento DOM. Inicialmente vamos gerar um evento do tipo “ready”, ou seja, quando o documento raiz da nossa página estiver pronto e iniciado veja a imagem abaixo.

```
<script type="text/javascript">
    $(document).ready(
        function () {
            alert("Função principal na inicialização da página");
        });
</script>
```

O termo “\$(document)” indica que estamos acessando o documento DOM como um todo. O termo “ready” indica que quando o DOM estiver pronto, a página irá inicializar um evento. O evento está indicado

por “function()”. Dentro dessa “função principal” podemos realizar nossas animações, criar outros eventos, etc.

```
<script>
$(document).ready(
    function(){
        $("a").click(
            function(){
                alert("o botão foi pressionado");
            }
        );
    });
</script>
```

No exemplo acima, geramos uma mensagem ao entrar na página, iremos agora gerar um, ao evento, ao clique de um botão obedecendo o mesmo padrão.

Usando o mesmo raciocínio, ativamos um evento de alerta, quando um botão com referência de link “a” for clicado. Assim podemos manipular qualquer elemento que esteja presente na nossa página WEB.

Estrutura

HTML para script acima:

```
<body>
    Clique no botão: <br />
        <a href=""> Primeiro Evento </a><br />
</body>
```

Exercício Rápido: Como seria ativar um botão ao clicar ou ao selecionar uma div?

Usando o método do exemplo acima, criamos um mesmo evento caso clicássemos em qualquer elemento de link “a”. Porém como poderíamos criar eventos isolados por referência de elemento? Para isso, podemos usar “#id” ou “.class” conteúdo esse visto no nosso curso de HTML/CSS. Iremos agora gerar um evento isoladamente para um botão, veja no exemplo abaixo.

```
<script>
$(document).ready(
    function(){
        $("#evento1").click(
            function(){
                alert("Evento 1 ativo");
            }
        );

        $("#evento2").click(
            function(){
                alert("Evento 2 ativo");
            }
        );
    });
</script>
```

Perceba que agora não iremos mais ativar um evento ao clicar em uma tag “a” qualquer, podemos direcionar um evento para um elemento específico do nosso HTML usando “#id”. Sabemos que um ID só poderá ser usado em um elemento HTML por página, então podemos usar esse parâmetro para gerar eventos em elementos individuais.

Estrutura HTML para script acima:

```
<body>
    Clique no botão: <br />
        <a id="evento1" href=""> Primeiro Evento </a><br />
        <a id="evento2" href=""> Segundo Evento </a>
    </body>
</html>
```

Podemos também criar funções separados do HTML, da mesma forma que fizermos no curso de JavaScript, faça um teste, adapte o código acima com funções em um aquivo chamado “anima.js” e chame-o dentro do seu código.

```

<script>
    $(document).ready(
        function(){
            $("#content a").click(
                function(){
                    alert("Evento acionado");
                });
        });
    </script>
</head>
<body>

    <div id="content">
        <a href=""> Pronto Para Evento JS</a>
    </div>
        <a href=""> Sem evento JS</a>
</body>

```

Se você quiser acessar um elemento que está dentro de um outro elemento identificado por algum “id”, “class” ou “tag”, podemos colocar o caminho completo do elemento que queremos acessar da seguinte forma:

Perceba que um dos links “<a>” está dentro da div “content”. Então geramos um evento para ser acionado apenas no link que está dentro dessa div adicionando o caminho :

`$("#content a").evento`

Veremos no próximo tópico outras formas de acesso a elementos HTML pelo jQuery:

Exercício Rápido: Como seria ativar esse mesmo evento levando em consideração que a camada “#content” estaria dentro de uma outra camada chamada “fundo”?

11.2.1 Métodos de acesso a Elementos HTML

Mostraremos algumas formas de selecionar um elemento dentro da sua estrutura HTML. Conhecer os exemplos abaixo é bastante importante, pois em nossos algoritmos estaremos sempre acessando e alterando valores das nossas páginas WEB. Estaremos usando alguns desses seletores no decorrer do nosso conteúdo. Seletores:

Seletor	Descrição
<code>\$(this)</code>	Seleciona o elemento referendado por ela mesma.

<code>\$('*')</code>	Seleciona todos os elementos do documento HTML
<code>\$(' div ')</code>	Seleciona todos os elementos <DIV> poderá ser usado para outros elementos. Ex: “img”, “a” , “table”.
<code>\$(' .curso ')</code>	Seleciona todos os elementos cuja a classe seja “curso”.
<code>\$(' #camada ')</code>	Seleciona um único elemento de id = “camada”.
<code>\$('#camada img')</code>	Seleciona todas as imagens que estiver dentro da camada (exemplo).

Pseudo-Seletores:

Seletor	Descrição	Exemplo
<code>:first</code>	Seleciona por um primeiro item	<code>\$(“ul li:first”)</code>
<code>:last</code>	Seleciona por um ultimo item	<code>\$(“ul li:last”)</code>
<code>:not</code>	Ignora um item selecionado	<code>\$(“ul li:not(:last)”)</code>
<code>:contains</code>	Seleciona um item que contenha texto indicado.	<code>\$(“p:contains(fab)”)</code>
<code>:empty</code>	Seleciona itens vazio	<code>\$(“p:empty”)</code>
<code>:visible</code>	Seleciona itens visiveis	<code>\$(“div:visible”)</code>

Pseudo-Seletores de formulários:

Seletor	Descrição	Exemplo
<code>:text</code>	Seleciona campos do tipo text	<code>\$(“:text”)</code>
<code>:hidden</code>	Seleciona campos invisíveis	<code>\$(“:hidden”)</code>
<code>:file</code>	Seleciona arquivos de um formulário	<code>\$(“:file”)</code>
<code>:button</code>	Seleciona botões do tipo “button” em um formulário	<code>\$(“:button”)</code>
<code>:checkbox</code>	Seleciona um checkbox de formulário	<code>\$(“:checkbox”)</code>
<code>:image</code>	Seleciona campo/Imagem de um formulário	<code>\$(“:image”)</code>
<code>:password</code>	Seleciona um campo de texto do tipo Senha de um formulário	<code>\$(“:password”)</code>
<code>:radio</code>	Seleciona uma caixa de texto do tipo Radio	<code>\$(“:radio”)</code>
<code>:submit</code>	Seleciona um botão do tipo submit de um formulário.	<code>\$(“:submit”)</code>

:reset	Seleciona um botão do tipo reset de um formulário.	<code>\$(":reset")</code>
---------------	--	---------------------------

Seletores de Formulários:

Seletor	Descrição
<code>\$('input [name = "bot"]')</code>	Seleciona todos os elementos de formulário com o nome “bot”.
<code>\$('input [name != "curso"]')</code>	Seleciona todos os elementos de formulário que contenham um valor diferente de “curso”.

12.0 Biblioteca jQuery

Neste capítulo estudaremos algumas funções da biblioteca jQuery, não estudaremos toda a biblioteca, pois são muitas funções, o objetivo do nosso curso é mostrar através de algumas funções o uso prático dessa ferramenta, se você pretende se especializar nesse framework indicamos esse link: “<http://api.jquery.com/>”. Este link contém uma documentação completa de todas as funções do jQuery com exemplos práticos do jQuery para as mais diferentes situações.

12.1 Funções de Estilo

Com a biblioteca jQuery, Podemos alterar dinamicamente os estilos de elementos HTML de forma simples, podendo adicionar ou alterar estilos através dos comandos “css()”, “add()” , “addClass()” , “removeClass()” e “toggleClass()”. Veremos em exemplos suas diferenças, como podemos utilizar esses recursos para cada caso utilizando também recursos aprendidos no módulo anterior de JavaScript.

12.2 Função .css()

Com esse comando podemos adicionar um estilo CSS mesmo que o elemento html não tenha estilo pronto para ela, vejamos como podemos aplicar isso.

```
$(“elemento”).css(“Atributo CSS”, “valor”)
```

```
<script>
    $(document).ready(
        function(){
            $("#bot").click(
                function(){
                    $("#texto").css("color", "red");
                });
        }
    );
</script>
</head>
<body>
    <p id="texto"> Texto a ser modificado uma cor </p>
    <button id="bot"> Aletar estilo </button>
</body>
```

No exemplo abaixo, temos um botão que aciona um evento responsável por criar um estilo ao texto Html veja:

Este código mostra como alterar a cor de um texto usando o jQuery. Criamos um evento de clique em um botão chamado “bot” que será responsável por alterar a cor do texto, perceba a cor do texto que será

alterada está em um parágrafo que tem como seu id = “texto”. Alteramos o conteúdo do texto para cor vermelha juntamente no na função .css(“color”, “red”); . Podemos alterar qualquer CSS de qualquer elemento usando essa função.

Podemos também guardar em uma variável JS, informações de algum atributo CSS já criado usando:

```
var corAtual = $("elemento").css("Atributo CSS");
```

Exercício Rápido: Como seria alterar a cor de plano de fundo de um campo de texto de formulário ao selecionar o clique?

12.3 Função .add()

Com este comando podemos adicionar estilo a vários elementos distintos com um mesmo comando, podemos também definir os estilos mais genéricos, e específicos entre os elementos envolvidos no relacionamento.

No exemplo abaixo, temos um exemplo de uma tabela editada com recursos de estilo “add()”. Iremos posteriormente analisar o código.

```

<script>
    $(document).ready(
        function(){
            $("th").css("color","navy")
                .add("#tabelaEdit td")
                .css("background-color", "#DDDDDD");
        });
    </script>
</head>
<body>
    <table id="tabelaEdit" border="1">
        <thead>
            <tr>
                <th>Nome</th>
                <th>e-mail</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Fabricio</td>
                <td>emailFabricio@gmail.com</td>
            </tr>
            <tr>
                <td>Rosal</td>
                <td>emailRosal@gmail.com</td>
            </tr>
        </tbody>
    </table>
</body>

```

Se olharmos o código JS de trás para frente, podemos perceber que incluímos em todas as nossas células “td” que estão dentro da tabela com id = “tabelaEdit” um plano de fundo de cor cinza. Continuando a leitura, as tags “th” recebem um estilo de cor de texto com nome “navy”, porém as tags “th” também irão “herdar” as características de estilos que foram atribuídas à tag “td”. Portanto o plano de fundo da tag “th” também será preenchido como cinza em seu plano de fundo.

Veja como ficou a nossa tabela.

Nome	e-mail
Fabricio	emailFabricio@gmail.com
Rosal	emailRosal@gmail.com

Exercício Rápido: Como seria aplicar esse mesmo conceito em uma lista HTML?

13.1.1. Função `.addClasse()` e `removeClass()`.

Estas funções são responsáveis por alterar um estilo estruturado de CSS de um determinado elemento html substituindo um estilo por outro. Veja sua Sintaxe:.

```
$(“elemento”).addClass(“.classNova”);
```

E para remover o estilo da classe adicionada temos:

```
$(“elemento”).removeClass(“.classNova”);
```

Iremos criar um algoritmo para alterar o estilo de uma DIV utilizando o evento hover, ou seja, quando o mouse estiver selecionado sobre a camada. Para isso iremos inicialmente criar 2 classes de estilo para uma

<pre><code>.estilo1{ width: 200px; height: 100px; background-color: #999900; margin: 10px; color: yellow; } .estilo2{ width: 200px; height: 100px; background-color: blue; margin: 10px; color: white; }</code></pre>	<pre><code><script> \$(document).ready(function(){ \$(".estilo1").hover(function(){ \$(this).addClass("estilo2"); },function(){ \$(this).removeClass("estilo2"); }); }); </script> </head> <body> <div class="estilo1"> Texto Teste. </div></code></pre>
--	--

camada e usar os comandos da seguinte forma:

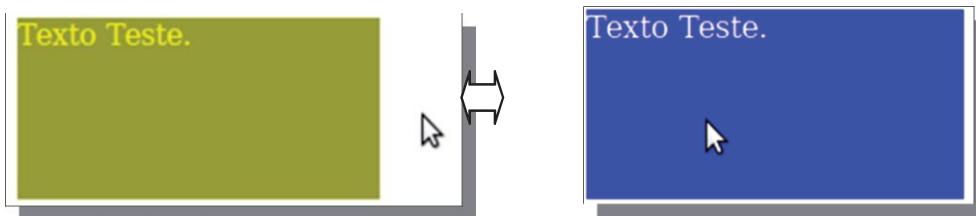
Vamos analisar cada parte do algorítimo:

`$(“.estilo1”).hover()`- Com esse comando, iremos chamar um evento ao selecionar o mouser por cima do elemento. No nosso caso na DIV que estiver com estilo “.estilo1” associado.

`$(this).addClass(“estilo2”)` - Com esse comando estamos querendo alterar a própria camada que foi selecionada, indicado pelo termo “`this`”, para o “`.estilo2`”.

`$(this).removeClass(“estilo2”)` - Quando o mouser for retirado da camada, o queremos retirar o estilo que foi incluído anteriormente na camada. Voltando ao seu estado original.

Veja como ficaria essa operação no browser:



Exercício Rápido: Crie 2 camadas, onde cada camada seja responsável por alterar temporariamente o estilo da outra camada.(camada 1 altera estilo da camada 2 e vice-versa)

Podemos alterar não só as Div, mas também qualquer elemento HTML

12.4 Função `toggleClass()`

Com essa função podemos montar de forma mais simples a animação do tópico passado. podemos modificar um estilo de um elemento e quando esse elemento for ativado novamente, a função irá voltar para o seu estado original. Veja sua Sintaxe::

```
$(“elemento”).toggleClass(“.classNova”);
```

Como será a modificação do nosso código.

```
<script>
    $(document).ready(
        function(){
            $(".estilo1").hover(
                function(){
                    $(this).toggleClass("estilo2");
                });
        });
</script>
```

12.5 Funções de visibilidade

Com a biblioteca jQuery, podemos alterar dinamicamente a visibilidade de elementos HTML de forma simples, podendo mostrar ou ocultar os elementos através dos comandos “show()”, “hide()” , “toggle()”.Estas funções da Biblioteca jQuery são referentes ao modo de visibilidade, sendo possível alterar a visão de qualquer elemento da nossa estrutura HTML dinamicamente. Iremos ver nos próximos tópicos como utilizá-los.

12.6 Funções **show()** e **hide()**

Podemos mostrar um determinado elemento HTML através do comando “show()” ou esconder-los usando o comando “hide()” de acordo com as nossas necessidades. Veremos a sintaxe:

Para ocultar um elemento dinamicamente por um evento:

```
$(“elemento”).hide()
```

Para mostrar um elemento dinamicamente por um evento:

```
$(“elemento”).show()
```

As 2 funções podem receber como parâmetro uma String com 2 tipos de valores.

Fast - Tornará oculto / visível um elemento com o efeito de transição de forma rápida. Slow - Tornará oculto / visível um Elemento com o efeito de transição de forma lenta. Sintaxe com Parâmetros:

```
$(“elemento”).show(“fast”) / $(“elemento”).show(“slow”)  
$(“elemento”).hide(“fast”) / $(“elemento”).hide(“slow”)
```

Se quisermos editar o tempo de duração do evento, podemos também atribuir um valor em milisegundos referente ao tempo da animação:

```
$(“elemento”).show(2000);
```

Para esse exemplo iremos mostrar uma camada em um tempo de 2000 milissegundos, ou seja, 2 Segundos.

Obs.: existe um estilo CSS chamado “display” Quando ele está atribuído inicialmente como “none”, queremos dizer que inicialmente o elemento que tiver esse atributo, inicialmente estará oculto, sendo assim podemos torná-lo visível através do show() e tornar oculto novamente através do hide();

Vamos analisar um exemplo prático na prima página, mostrando como mostrar ou ocultar uma pequena

```

<script>
    $(document).ready(
        function(){
            $("#bot").click(
                function(){
                    $("p").hide();
                    $("#divFormulario").show();
                });
            $("#voltar").click(
                function(){
                    $("p").show();
                    $("#divFormulario").hide();
                });
        });
    </script>
</head>
<body>
    <p>
        Deseja Responder nossa enquete?<br />
    </p>
    <button id="bot">Responder</button>

    <div id="divFormulario">
        Quantas horas de estudo você realiza por dia ?
        <form name="form1" action="#" method="GET">
            <input type="radio" name="escHoras" value="hora1" />
            Pelo menos 1 hora<br />
            <input type="radio" name="escHoras" value="hora2" />
            Pelo menos 2 horas<br />
            <input type="radio" name="escHoras" value="hora3" />
            Pelo menos 3 horas<br />
            <input type="radio" name="escHoras" value="horaMais" />
            Mais de 3 horas<br />
            <input type="submit" value="Enviar" name="botao"/>
            <button id="voltar"> Voltar </button>
        </form>
    </div>
</body>

```

enquete que poderá está dentro do seu site:

Exercício Rápido: Crie 2 formulários, onde o segundo só aparecerá quando o usuário terminar de preencher o primeiro formulário.

A camada “idFormulario” estará inicialmente com “display : none;” em seu CSS. Assim ela estará oculta incialmente. O Botão chamado “#bot” realizará um evento ao clique do botão que será responsável por esconder o paragrafo de texto e mostrar a camada “idFuncionário” que é justamente a camada que contém a enquete. Dentro da camada enquete podemos perceber que temos um botão chamado “voltar” ele fará o inverso, irá ocultar a enquete e tornará visível novamente o texto inicial que está dentro do parágrafo.

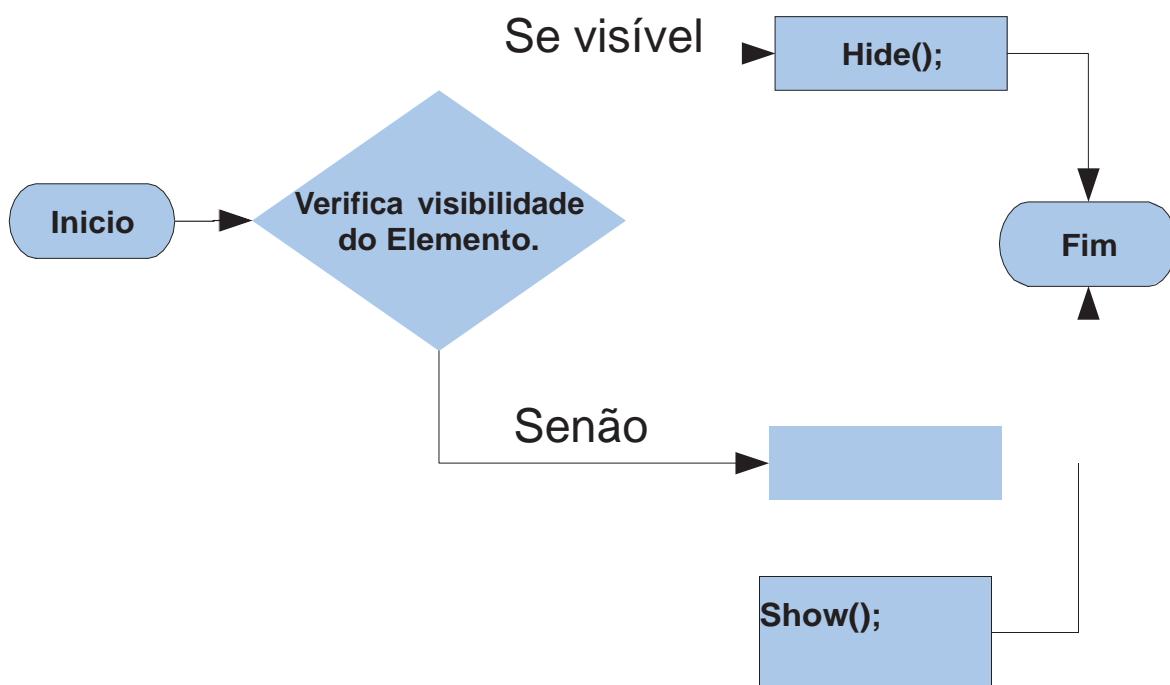
Perceba que não colocamos nenhum parâmetros na função: show() ou hide(). Veja o que acontecerá se colocarmos os parâmetros aprendidos nesse tópico.

12.6.1 Função Toggle()

Esta função, realiza o mesmo resultado das funções, show() e hide() estudados no tópico anterior.

Porém ele tem uma pequena implementação de algoritmo que permite analisar se o elemento está visível ou

não, alterando dinamicamente o seu estado de visibilidade. Veja no Fluxograma abaixo seu processo.



Veja sua Sintaxe:

`$(“elemento”).toggle();`

Obs.: Esta função permite os mesmos parâmetros aprendidos nas funções “show()” e “hide()” estudados do tópico anterior.

Vejamos um exemplo abaixo usando o “toggle();” levando em consideração a mesma estrutura HTML do tópico anterior.

```
<script>
    ocult = true;
    $(document).ready(
        function(){
            $("#bot").click(
                function(){
                    $("p").toggle("slow");
                    $("#divFormulario").toggle("slow");
                    if(ocult){
                        document.getElementById('bot').innerHTML = 'Ocultar';
                        ocult = false;
                    }else{
                        document.getElementById('bot').innerHTML = 'Responder';
                        ocult = true;
                    }
                });
        });
    </script>
```

Perceba que com apenas um botão podemos mostrar e ocultar vários conteúdos simultaneamente

12.6.2 Função delay()

Essa função poderá ser combinada com outras funções jQuery, ela irá programar cada evento com um tempo de espera estimado em milissegundos como parâmetro. Iremos mostrar a seguir a sua sintaxe e um exemplo de sua aplicação:

```
$(“elemento”).delay(tempo).outrasFunções();
```

Iremos mostrar 2 eventos que poderá ser disparado através do clique, um deles terá um elemento com delay entre o título e o conteúdo de texto, o outro não terá, vejamos a diferença:

```
<script>
    $(document).ready(
        function(){
            $("#b1").click(
                function(){
                    $('div').add('p').hide();
                    $('#conteudo1').show('slow');
                    $('#conteudo1 p').show('slow');
                }
            );
            $("#b2").click(
                function(){
                    $('div').add('p').hide();
                    $('#conteudo2').show('slow');
                    $('#conteudo2 p').delay(1500).show('slow');
                }
            );
        });
    </script>
</head>
<body>
    <button id="b1">Conteúdo 1</button>
    <button id="b2">Conteúdo 2</button>

    <div id="conteudo1" style="display: none">
        <h2>Título1</h2>
        <p>Texto para conteúdo 1</p>
    </div>

    <div id="conteudo2" style="display: none">
        <h2>Título2</h2>
        <p>Texto para conteúdo 2</p>
    </div>
</body>
```

Perceba que todos os eventos irão inicializar ao acionar o botão “b2”. Porém o conteúdo do parágrafo da camada div “conteudo2” irá esperar 1.5 segundos até começar seu efeito de “fadeIn()”;

12.7 Funções de Opacidade

Com a biblioteca jQuery, podemos controlar dinamicamente a opacidade de elementos HTML de forma simples, podendo tornar completamente opaco ou invisível, também podemos controlar o nível de opacidade de um elemento através dos comandos “fadeIn()”, “fadeOut()” , “fadeTo()” , e “fadeToggle()”. Veremos nos próximos tópicos como essas funções funcionam

12.7.1 Funções **FadeIn()** e **Fadeout()**

A função fadeIn() faz tornar visível um elemento HTML, enquanto fadeOut() torna invisível. Veja sua sintaxe abaixo:

Para tornar visível:

```
$(“elemento”).fadeIn();
```

Para tornar invisível;

```
$(“elemento”).fadeOut();
```

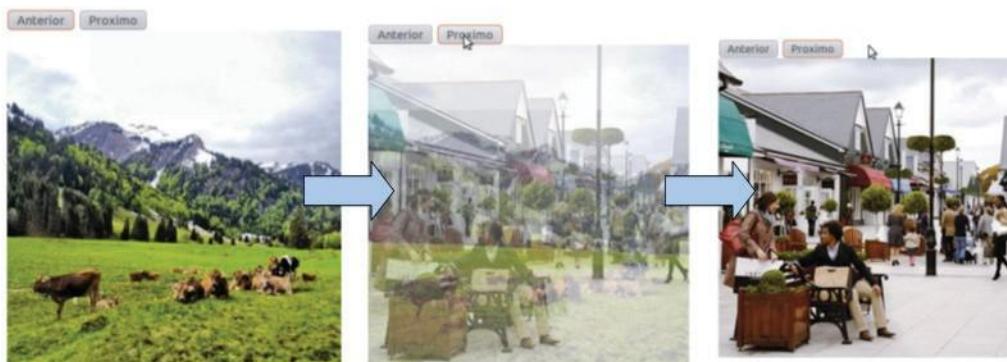
Podemos também controlar a velocidade que esses elementos poderão aparecer ou sumir desta forma:

Obs.: Podemos também controlar o tempo dos efeitos de opacidade colocando um parâmetro “slow” ou “fast” ou atribuir valores de parâmetro em milisegundos ao invés de um texto, da mesma forma que fazemos para os comandos **show()** e **hide()** estudado nos tópicos anteriores.

Veja um exemplo abaixo usando as funções aprendidas:

```
<script>
    $(document).ready(
        function(){
            $('input[name="bot1"]').click(
                function(){
                    $('img').fadeOut('slow');
                    $("#img1").fadeIn("slow");
                });
            $('input[name="bot2"]').click(
                function(){
                    $('img').fadeOut('slow');
                    $("#img2").fadeIn("slow");
                });
        });
    </script>
</head>
<body>
    <input type="button" name="bot1" value="Anterior" />
    <input type="button" name="bot2" value="Próximo" /><br />
    
    
</body>
```

Neste exemplo temos 2 imagens que foram colocadas no nosso site. Existem 2 botões que são responsáveis por fazer um efeito de transição entre as imagens inseridas, perceba que para cada botão,



enquanto uma imagem desaparece (fadeOut) uma outra vai aparecendo (fadeIn). Veja:

12.7.2 FadeTo();

Esta função é responsável por controlar o nível de opacidade de um elemento HTML, veja sua sintaxe.

```
$(“elemento”).fadeTo(velocidade, opacidade);
```

onde:

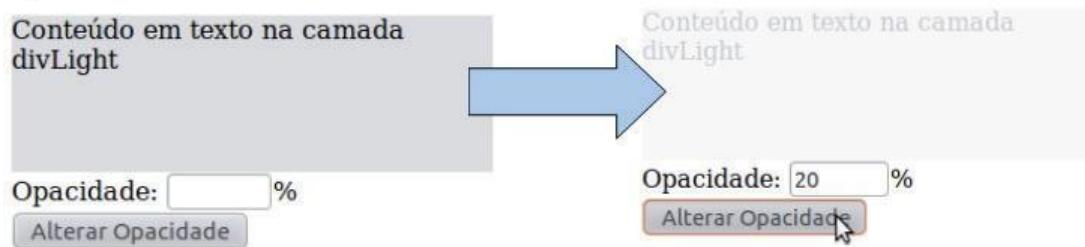
Velocidade = Velocidade do efeito de transição da opacidade antiga com a novo valor de opacidade poderá ser numérico em milissegundos ou uma String “slow” e “fast”.

Opacidade = nível de opacidade do elemento, poderá variar de 0(zero) até 1(100%) onde os valores intermediários seriam o nível da opacidade. Exemplo: 0.5 = (50% de opacidade).

```
<script>
    $(document).ready(
        function(){
            $('input[name="bot"]').click(
                function(){
                    var valor = $('input[name="opacidade"]').val();
                    valor = valor/100;
                    $(".divLight").fadeTo("slow",valor);
                });
        });
    </script>
</head>
<body>
    <div class="divLight">
        Conteúdo em texto na camada divLight
    </div>
    Opacidade:
    <input type="text" name="opacidade" size="5"/>%<br />
    <input type="button" name="bot" value="Alterar Opacidade" />
</body>
```

Abaixo temos um exemplo que muda a opacidade de uma camada ativado por um botão que verifica o valor de uma caixa de texto e realiza a animação de mudança de opacidade.

Usamos também um novo Método para receber valores em um formulário usando jQuery, através do comando: (“elemento input”).val(). Em JavaScript, usávamos outros métodos, não fará diferença para esse exemplo, nos 2 casos podemos receber valores de campos HTML. Temos a seguir o seguinte resultado:



12.7.3 FadeToggle()

Essa função é responsável por alterar o modo de visibilidade de um elemento HTML voltando ao seu estado inicial quando ativado novamente, ele segue o mesmo raciocínio das funções que contém “Toggle” nos tópicos anteriores, alterando apenas a maneira como é feita a animação. Veja sua sintaxe:

```
$(“elemento”).fadeToggle(velocidade);
```

Velocidade = Velocidade do efeito de transição da opacidade antiga com a novo valor de opacidade poderá ser numérico em milissegundos ou uma String “slow” e “fast”. Esse parâmetro é opcional. Veja um Exemplo:

```
<script>
    $(document).ready(
        function(){
            $("a").hover(
                function(){
                    $("#resposta").fadeToggle();
                });
        });
    </script>
</head>
<body>

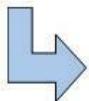
    <p> Qual a função básica do Jquery <a href="">Ver resposta</a> </p>
    <p id="resposta" style="display: none">
        jQuery foi feito para tornar mais simples a navegação<br />
        do documento HTML, a seleção de elementos DOM e <br />
        criar animações </p>
</body>
```

Resultado:

Qual a função básica do Jquery [Ver resposta](#) 

Qual a função básica do Jquery [Ver resposta](#) 

jQuery foi feito para tornar mais simples a navegação do documento HTML, a seleção de elementos DOM e criar animações



Qual a função básica do Jquery [Ver resposta](#) 

jQuery foi feito para tornar mais simples a navegação do documento HTML, a seleção de elementos DOM e criar animações

Exercício rápido: Como seria por uma legenda com uma camada semitransparente usando fade por cima das imagens em efeito?

13.0 Biblioteca jQuery UI

A jQuery UI proporciona abstrações de baixo nível de interação, animação e avançados efeitos especiais de alto nível, construída em cima do jQuery JavaScript Library, que pode ser utilizada para construir aplicações web altamente interativas.

13.1 Instalação e configuração

A instalação e configuração é feita de forma semelhante a que fizemos com jQuery, vamos para o site Oficial do projeto jQuery UI : <http://jqueryui.com/>. Onde iremos baixar um pacote para a instalação. Iremos trabalhar com a versão **jquery-ui-1.11.4.custom**.

O jQuery UI nos disponibiliza um pacote contendo um jQuery e jQuery UI, junto com uma pasta chamada CSS. Essa página também deverá ser importada no seu projeto, pois alguns recursos utilizando de estilos pré-definidos. Acesse o site da jQuery UI e você poderá receber outros pacotes com outros efeitos, por padrão usaremos o estilo “**“ui-lightness”** pacote esse que você escolhe no momento do download do pacote. Você poderá acessar o CSS modificando configurações existentes ou criando um próprio estilo para você interagindo com seu jQuery.

O link de importação deverá ser da seguinte forma:

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/jquery-ui.min.js"></script>
<script type="text/javascript" src="js/jquery-ui.js"></script>
<link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
```

Pronto, agora além de melhorarmos algumas funções existentes do pacote jQuery já estudados, estamos também implementando novas funções da biblioteca jQuery UI. Estudaremos nesse curso algumas funções dessa biblioteca, recomendamos que veja a documentação do site oficial da jQuery UI para estudos complementares.

13.2 Datepicker.

Essa função é responsável por mostrar um calendário para o usuário, as informações do dia selecionado pelo usuário, poderá diretamente ser alterado em uma caixa de texto, oferecendo estilo e poupança ao usuário de ter que digitar uma data manualmente.

Sua Sintaxe:

```
$(“elemento”).datepicker();
```

Elemento – Elemento HTML poderá ser responsável por representar o retorno da data escolhida. Veja um exemplo:

```
<html>
  <head>
    <title></title>
    <meta charset="UTF-8">
    <script type="text/javascript" src="js/jquery.js"></script>
    <script type="text/javascript" src="js/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui.js"></script>
    <link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
    <script type="text/javascript">
      $(document).ready(
        function () {
          $("#data").datepicker();
        });
    </script>
  </head>
  <body>
    Faça seu agendamento: <br>
    <input type="text" id="data" size="10">
    <button>Enviar</button>
  </body>
</html>
```



Temos então o resultado a seguir no navegador:

13.3 Accordion

Essa função é responsável por segmentar conteúdos de forma mais elegante, podemos mostrar um conteúdo selecionado ocultando os outros conteúdos que tiverem dentro dessa camada. Sua Sintaxe é descrita da seguinte forma:

```
$(“elemento”).accordion();
```

Onde esse elemento descrito na sintaxe poderá ser uma ID de uma camada. Para organizar o conteúdo interno, devemos estruturar da seguinte forma:

<H3>: Dentro da camada indicada com “**accordion()**” irá representar o título de cada camada de conteúdo.

<div>: as Camadas internas à camada indicada pelo “**accordion()**” irá representar o conteúdo interno que será visível ou não dependendo de sua seleção.

Todos os efeitos já serão gerados automaticamente se organizados os títulos e conteúdo dessa forma.

Vejamos um exemplo a seguir. Resultado no navegador:

Código:

```

<html>
  <head>
    <title></title>
    <meta charset="UTF-8">
    <script type="text/javascript" src="js/jquery.js"></script>
    <script type="text/javascript" src="js/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui.js"></script>
    <link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
    <script type="text/javascript">
      $(document).ready(
        function () {
          $("#conteudo").accordion();
        });
    </script>
  </head>
  <body>
    <div id="conteudo" style="width=500px">
      <h3> Artigo I</h3>
      <div>
        <p>
          Blockchain talvez seja o recurso mais interessante da criptomoeda, com sua capacidade de registrar transações com segurança e publicamente. Considerando que o mercado de ações é uma confusão de troca de negociações, usar o blockchain para essas transações pode ser uma ideia surpreendentemente boa.
        </p>
      </div>
      <h3> Artigo II</h3>
      <div>
        <p>
          O Overstock.com é uma loja online que abraçou o Bitcoin e o blockchain, e agora está levando o amor à criptografia a um novo nível: com aprovação da Comissão de Valores Mobiliários (SEC, na sigla em inglês) dos EUA, o Overstock planeja emitir títulos públicos. É um grande passo para o blockchain como tecnologia, considerando que até hoje se manteve limitado ao Bitcoin.
        </p>
      </div>
      <h3> Artigo III</h3>
      <div>
        <p>
          O blockchain é uma plataforma para tornar registros de transações publicamente disponíveis e que não pode ser modificado. Usuários podem escrever transações na cadeia (e elas são verificadas por uma vasta rede de computadores), mas não podem, posteriormente, eliminar essas entradas. Na ausência de um banco central, é isso o que mantém o Bitcoin funcionando.
        </p>
      </div>
    </div>
  </body>
</html>

```

Além de texto, podemos colocar outros elementos HTML, como Imagens, links, tabelas, listas etc.

Podemos usar esses recursos para uma área de notícias ou artigos por exemplo, pesquise em outros sites novos contextos onde podemos utilizar **accordion** para facilitar o acesso à conteúdos WEB.

13.4 Tabs

Essa função é uma outra forma elegante de organizar o conteúdo de uma página WEB através de abas por link para visualização de conteúdo de forma dinâmicas. Vejamos sua sintaxe e como aplica-las.

```
$(“elemento”).tabs();
```

Onde esse elemento descrito na sintaxe poderá ser uma ID de uma camada. Para organizar o conteúdo interno, devemos estruturar da seguinte forma:

<a>: Inicialmente criamos uma lista de links para representar as abas de conteúdo. Os índices deverão linkar um ID responsável por alterar o conteúdo.

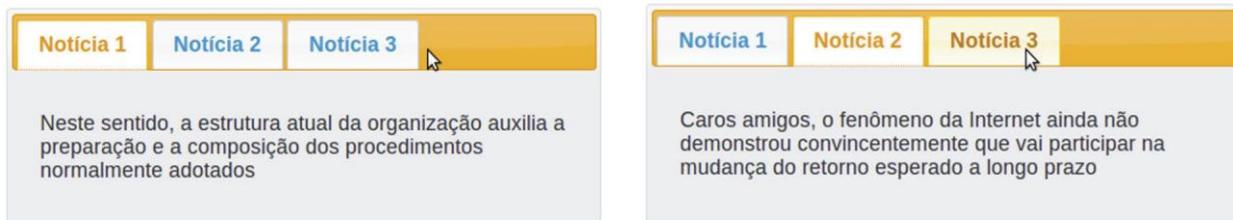
<div>: Cada camada deverá ter uma indicação “**ID**” que irá representar um conteúdo.

Todos os efeitos já serão gerados automaticamente se organizados os títulos e conteúdo dessa forma:
Vejamos um exemplo a seguir:

```
<script>
    $(document).ready(
        function(){
            $("#noticias").tabs();

        });
    </script>
</head>
<body> |
    <div id="noticias" style="width: 500px">
<ul>
    <li><a href="#not-1">Notícia 1</a></li>
    <li><a href="#not-2">Notícia 2</a></li>
    <li><a href="#not-3">Notícia 3</a></li>
</ul>
<div id="not-1">
    <p>
        Neste sentido, a estrutura atual da organização auxilia a
        preparação e a composição dos procedimentos normalmente adotados
    </p>
</div>
<div id="not-2">
    <p>Caros amigos, o fenômeno da Internet ainda não demonstrou
        convincentemente que vai participar na mudança do retorno
        esperado a longo prazo</p>
</div>
<div id="not-3">
    <p> Assim mesmo, a determinação clara de objetivos agrupa valor ao
        estabelecimento de todos os recursos funcionais envolvidos</p>
</div>
</div>
</body>
```

Temos então o resultado a seguir no navegador:



13.5 Menu

Esta função é responsável por montar um Menu DropDown de forma mais simples e elegante usando apenas listas e links como referência. Vejamos usa sintaxe:

```
$(“elemento”).menu();
```

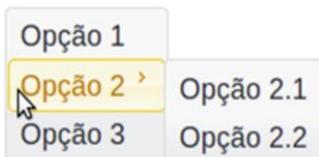
Onde esse elemento descrito na sintaxe poderá ser uma ID de uma camada. Para organizar o conteúdo interno, devemos estruturar da seguinte forma:

<a>: monta um novo botão para link no menu. Para criar um menu DropDown basta colocar uma lista dentro de outra lista, como fizemos no link “Opção 2” no link abaixo, veja o exemplo:

```

<script>
    $(document).ready(
        function(){
            $("#menu").menu();
        });
    </script>
</head>
<body>
    <ul id="menu" style="width: 100px">
        <li>
            <a href="">Opção 1</a>
        </li>
        <li>
            <a href="">Opção 2</a>
            <ul>
                <li>
                    <a href="">Opção 2.1</a>
                </li>
                <li>
                    <a href="">Opção 2.2</a>
                </li>
            </ul>
        </li>
        <li>
            <a href="">Opção 3</a>
        </li>
    </ul>
</body>

```



Temos então o resultado a seguir no navegador:

13.6 Tooltip

Essa função é responsável por indicar aos usuários alguma informação relevante para o preenchimento de algum dado do formulário. Esse método já existe usando apenas o HTML, porém o jQuery pode tornar esse recurso mais elegante. Vejamos sua sintaxe:

```
$(“elemento”).tooltip();
```

Você poderá utilizar “document” como parametro do “elemento”. Assim todos os campos de um formulário que tiver “title” será alterado com o novo designer baseado no tema escolhido do jQuery.

```
<script>
    $(document).ready(
        function(){
            $(document).tooltip();
        });
    </script>
</head>
<body>
    <form name="formulario" action="#" method="POST">
        <p>Nome:
            <input id="nome" size="20"/></p>
        <p>CPF :
            <input id="cpf" title="Digite seu CPF sem pontos e hifen"/></p>
            <input type="submit" value="enviar" name="enviar" />
            <input type="reset" value="limpar" name="limpar" />
    </form>
</body>
```

Temos então o resultado a seguir no navegador:

Nome:

CPF : 

Digite seu CPF sem pontos e hifen

1.0 PHP

Objetivos

Apresentar basicamente alguns tipos de dados a ser implementados e adotados; Mostrar sua importância; Diferenciar variável e constante; Apresentar comandos e processo de entrada.

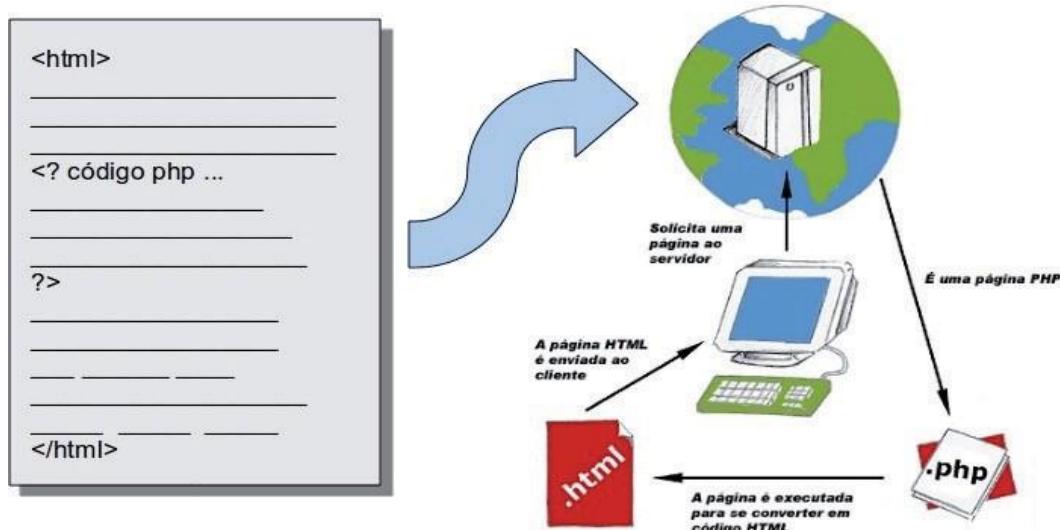
1.1 O que é PHP

PHP significa: Hypertext Preprocessor. O produto foi originalmente chamado de “Personal Home Page Tools”, mas como se expandiu em escopo um nome novo e mais apropriado foi escolhido por votação da comunidade. Você pode utilizar qualquer extensão que desejar para designar um arquivo PHP, mas os recomendados foram .php. O PHP está atualmente na versão 5.4.0, chamado de PHP5 ou, simplesmente de PHP, mas os seus desenvolvedores estão trabalhando para lançamento da versão 6, que causa uma preocupação para os programadores do mundo todo, uma vez que, algumas funcionalidades antigas deixam de funcionar quando passaram da versão 4 para a 5.



PHP é uma linguagem de criação de scripts embutida em HTML no servidor. Os produtos patenteados nesse nicho do mercado são as Active Server Pages(ASP) da Microsoft, o Coldfusion da Allaire e as Java Server Pages da antiga Sun que foi comprada pela Oracle. PHP é, às vezes, chamado de “o ASP de código-fonte aberto” porque sua funcionalidade é tão semelhante ao produto/conceito, ou o que quer que seja, da Microsoft.

Página Web (site)



Exploraremos a criação de script no servidor, mais profundamente, nos próximos capítulos, mas, no momento, você pode pensar no PHP como uma coleção de “supertags” de HTML que permitem adicionar funções do servidor às suas páginas da Web. Por exemplo, você pode

utilizar PHP para montar instantaneamente uma complexa página da Web ou desencadear um programa que automaticamente execute o débito no cartão de crédito quando um cliente realizar uma compra. Observe uma representação de como PHP e HTML se comportam:

O PHP tem pouca relação com layout, eventos ou qualquer coisa relacionada à aparência de uma página da Web. De fato, a maior parte do que o PHP realiza é invisível para o usuário final. Alguém visualizando uma página de PHP não será capaz de dizer que não foi escrita em HTML, porque o resultado final do PHP é HTML.

O PHP é um módulo oficial do servidor http Apache, o líder do mercado de servidores Web livres que constitui aproximadamente 55 por cento da World Wide Web. Isso significa que o mecanismo de script do PHP pode ser construído no próprio servidor Web, tornando a manipulação de dados mais rápida. Assim como o servidor Apache, o PHP é compatível com várias plataformas, o que significa que ele executa em seu formato original em várias versões do UNIX e do Windows. Todos os projetos da Apache Software Foundation – incluindo o PHP – são software de código-fonte aberto.

1.1.1 Um pouco da História do PHP.

Rasmus Lerdorf, engenheiro de software, membro da equipe Apache é o criador e a força motriz original por trás do PHP. A primeira parte do PHP foi desenvolvida para utilização pessoal no final de 1994. Tratava-se de um wrapper de PerlCGI que o auxiliava a monitorar as pessoas que acessavam o seu site pessoal. No ano seguinte, ele montou um pacote chamado de Personal Home Page Tools (também conhecido como PHP Construction Kit) em resposta à demanda de usuários que por acaso ou por relatos falados depararam-se com o seu trabalho. A versão 2 foi logo lançada sob o título de PHP/FI e incluía o Form Interpreter, uma ferramenta para analisar sintaticamente consultas de SQL.



Em meados de 1997, o PHP estava sendo utilizado mundialmente em aproximadamente 50.000 sites. Obviamente estava se tornando muito grande para uma única pessoa administrar, mesmo para alguém concentrado e cheio de energia como Rasmus. Agora uma pequena equipe central de desenvolvimento mantinha o projeto sobre o modelo de “junta benevolente” do código-fonte aberto, com contribuições de desenvolvedores e usuários em todo o mundo. Zeev Suraski e Andi Gutmans, dois programadores israelenses que desenvolveram os analisadores de sintaxe PHP3 e PHP4, também generalizaram e estenderam seus trabalhos sob a rubrica de Zend.com (Zeev, Andi, Zend).

O quarto trimestre de 1998 iniciou um período de crescimento explosivo para o PHP, quando todas as tecnologias de código-fonte aberto ganharam uma publicidade intensa. Em outubro de 1998, de acordo com a melhor suposição, mais de 100.000 domínios únicos utilizavam PHP de alguma maneira. Um ano depois, o PHP quebrou a marca de um milhão de domínios.

1.2 Instalação do Servidor PHP

As requisições são feita de forma cliente servidor, onde um determinado cliente faz uma requisição a um servidor, ele por sua vez recebe a requisição e faz todo o processo em diferentes camadas, retornando somente o que o cliente (browser) solicitou.

Existe uma aplicação chamada de LAMP (Linux, Apache, MYSQL, PHP) voltada para sistemas operacionais Linux, como também WAMP (Windows, Apache, MYSQL, PHP) voltada para sistemas operacionais Windows. Eles proporcionam uma simples configuração desses recursos e também muito indicado para ambiente de estudo PHP. Porém não iremos trabalhar com a instalação do LAMP. Pretendemos futuramente trabalhar com **frameWorks**, então o mais indicado será realizar a instalação e configuração manualmente. Iremos

ffLffl

ff

explicar um passo a passo de todo o processo de instalação e configuração desses recursos no tópico seguinte.

1.2.1 Instalação Apache.

O apache é um dos principais aplicativos para o funcionamento de programas web feito em PHP, ele é um dos responsáveis por compartilhar o nosso site dinâmico para outras outras máquinas, existe duas grandes versões, o apache 2.x e o Apache 1.3, que apesar de antigo ainda é muito utilizado em servidores. O Apache 2 trouxe muitas vantagens, sobretudo do ponto de vista do desempenho além de oferecer novos módulos e mais opções de segurança. Mas sua adoção foi retardada nos primeiros anos por um detalhe muito simples: o fato de ele ser incompatível com os módulos compilados para o Apache 1.3. Como os módulos são a alma do servidor web, muitos administradores ficavam amarrados ao Apache 1.3 devido à falta de disponibilidade de alguns módulos específicos para o Apache 2.

Iremos trabalhar com o Apache2 em sua versão para Linux, o procedimento de instalação é simples pois precisamos apenas de uma conexão com a internet e alguns comando, outra forma de instalação e baixando o mesmo no site: <http://httpd.apache.org/> e instalando de forma manual.

Antes de começar qualquer instalação, certifique-se que não existe nenhum pacote corrompido no sistema ou pacotes como LAMPP, XAMPP Instalados..

Para instalar o Apache precisamos abrir o terminal e atualizar o nosso repositório do Linux:

```
#apt-get update
```

Com os pacotes básicos atualizados podemos instalar o apache:

```
#apt-get install apache2
```

Espere até a conclusão de toda instalação. Por padrão o apache automaticamente inicializa, podendo ser utilizado após isso. Digite no <http://127.0.0.1/> ou <http://localhost/> no browser,

deverá aparecer a seguinte tela:



Isto mostra que o apache está funcionando corretamente. Trata-se de um texto dentro de um arquivo HTML que vem como padrão dentro da pasta “/var/www/” , essa é a pasta principal do apache, onde trabalharemos com os arquivos PHP.

filfl

to

O apache pode ser configurado de forma que podemos acessar mais de uma página com ip's ou portas diferentes, ou seja podemos ter www1,www2 ... em uma mesma máquina servidora fazendo acesso por endereços como <http://localhost:8060> ou <http://localhost:82>, entre outros. Mas trabalharemos apenas com a configuração padrão do apache, uma vez que existe muito conteúdo a respeito desse assunto na internet. Para obter mais informações consulte a documentação do site oficial do apache: <http://httpd.apache.org/docs/2.2/>

1.2.2 Instalação Php5.

Para o funcionamento dos programas em PHP precisamos de algumas dependências, onde funcionalidades, funções, classes e suporte a XML então embutidas. Por exemplo, se você quiser utilizar orientação a objeto ou simplesmente fazer uma conexão com o banco de dados, é necessário também a instalação do PHP5, onde algumas bibliotecas serão instaladas para dar suporte aos programas em PHP.

Após instalar o apache, faremos a instalação do PHP5 e algumas bibliotecas básicas com o seguinte comando:

```
#apt-get install php5 libapache2-mod-php5 php5-gd curl php5-curl php5-xmlrpc
php5-ldap php5-odbc
```

Atualizaremos o Ubuntu com alguns pacotes de serviço para servidor com o seguinte comando :

```
#apt-get install openssh-server unattended-upgrades
```

E por fim iremos reiniciar o serviço Apache:

```
#/etc/init.d/apache2 restart
```

Após a conclusão da instalação, podemos testar criando um arquivo dentro da pasta “**var/www**”.

Entre na pasta principal:

```
$ cd /var/www
```

Renomeie o arquivo “index.html” para outro nome:

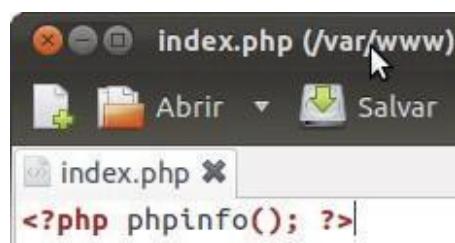
```
#mv  
index.html nome.html
```

Qualquer arquivo com o nome “index.html” dentro da pasta **www** será o primeiro arquivo a ser executado. Lembrando que pode existir somente um arquivo “index.html” na pasta. Após renomear

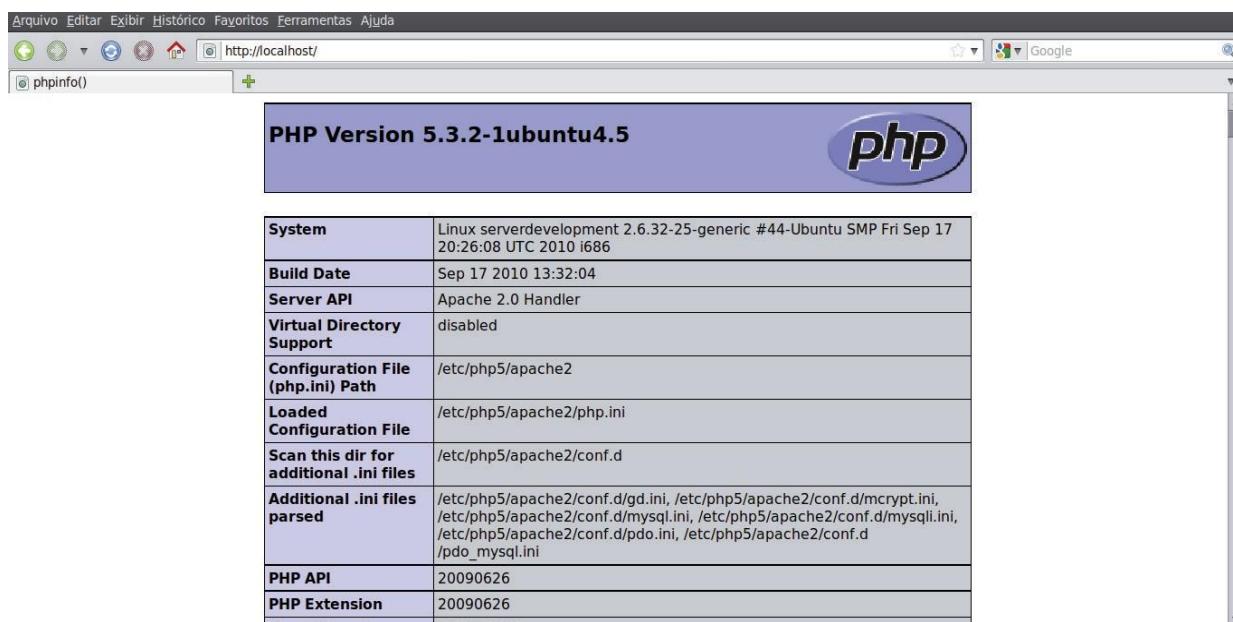
```
#gedit index.php
```

criaremos um novo arquivo.

Após executar esse comando, digite o seguinte código no editor de texto gedit:



Nesse momento estamos criando uma aplicação PHP que chama a função `phpinfo()` que tem como finalidade mostra informações sobre configuração do PHP. Lembre-se que toda operação em PHP estará dentro da Tag especial chamada: “`<?php ?>`”



Abra o browser e digite <http://localhost/>, se tudo estiver correto, observaremos seguinte tela:

1.3 Características de um programa PHP.

Assim como qualquer linguagem de programação, o PHP tem algumas características importantes, ao criamos um arquivo PHP podemos usar a seguinte extensão:

.php *

< Arquivo PHP contendo um programa.

.class.php <

adiante).

Arquivo PHP contendo uma classe(veremos o conceito de classe mais

.ini.php

< Arquivo PHP a ser incluído, pode incluir constantes ou configurações.

Outras extensões podem ser encontradas principalmente em programas antigos:

***.php3 <** Arquivo PHP contendo um programa PHP versão 3.

***.php4 <** Arquivo PHP contendo um programa PHP versão 4.

***.phtml <** Arquivo PHP contendo um programa PHP e HTML na mesma página.

1.3.1 Imprimindo algo no navegador WEB via PHP.

Usaremos comandos utilizados para gerar uma saída em tela (output). Se o programa PHP for executado via servidor de páginas web (Apache ou IIS), a saída será exibida na própria página HTML gerada no Browser (navegador), assim será mostrado de acordo com o conteúdo existente na saída, por exemplo, se tivermos o seguinte: "<h2> Hello Word!

<h2>", será mostrado no navegador apenas a mensagem Hello Word! em um tamanho maior, pois trata-se de um código HTML dentro de comandos PHP.

Podemos então usar os seguintes comandos para gerar comandos de saída: echo e print.

echo

É um comando que imprime uma ou mais variáveis ou textos, onde os mesmos são

colocados em aspas simples '' ou duplas “ ”. Sintaxe: echo 'a','b','c';

resultado:

abc

print

É uma função que imprime um *texto* na tela. Exemplo:

resultado:

apostila de PHP

print(' apostila de PHP ');

Observe um exemplo com o uso de **echo** e **print**:

Código:

```

1 <?php
2
3 echo '<h1>Bem Vindo!</h1>';
4 print('Site criado em PHP !');
5
6 ?>
7

```

Saída no Browser:



1.3.2 Comentários PHP

Usamos os comentários para nos ajudar a lembrar de partes do código ou para orientar outros programadores a algoritmo que está sendo escrito.

Para comentar uma única linha nos código PHP podemos utilizar tanto o separador “//” como também “#”, observe o exemplo abaixo:

```

1 | 1 <?php
2 | // echo "a";
3 | // imprime a;
4 | # echo b;
5 | # comentários!!
6 |
7 | ?>
8 |
9 |

```

Lembrando que os comentários são trechos de código que não são executados, onde eles servem somente para quem tem acesso aos códigos-fonte ou está ligado diretamente a programação desse código. Podemos também comentar muitas linhas, isso serve quando queremos que boa parte do código não execute ou simplesmente colocar um comentário mais extenso. Tudo que ficar entre: “/ ” e “ /” será um comentário, independente do número de linhas, observe o exemplo abaixo:

* * *

```

1 | 1 <?php
2 | /* parte menos importante do código!
3 | *
4 | * echo "<h1>Bem vindo ao Site</h1>"; ou
5 | * if($a==2){
6 | * echo "logout";
7 | * }
8 | */
9 |
10 |
11 | ?>
12 |

```

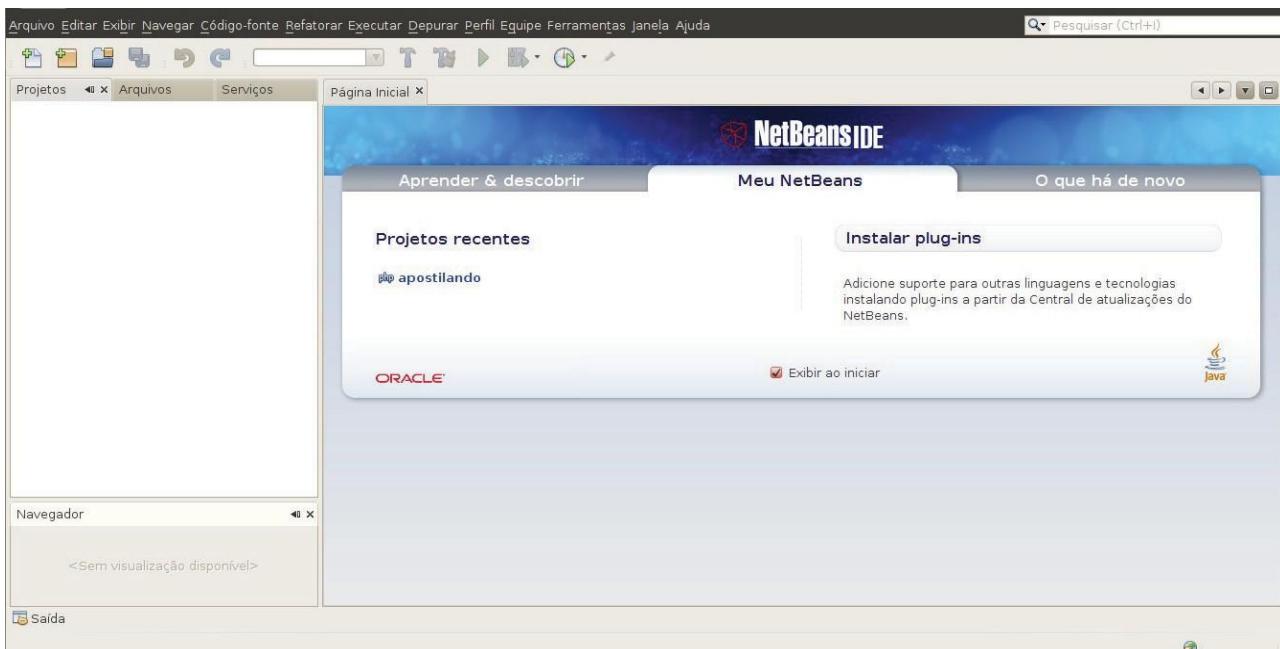
```

1 | 1 <?php
2 | /*
3 | 3 */ código principal!
4 | *
5 | * não inicializa as
6 | * variáveis
7 | */
8 |
9 | ?>

```

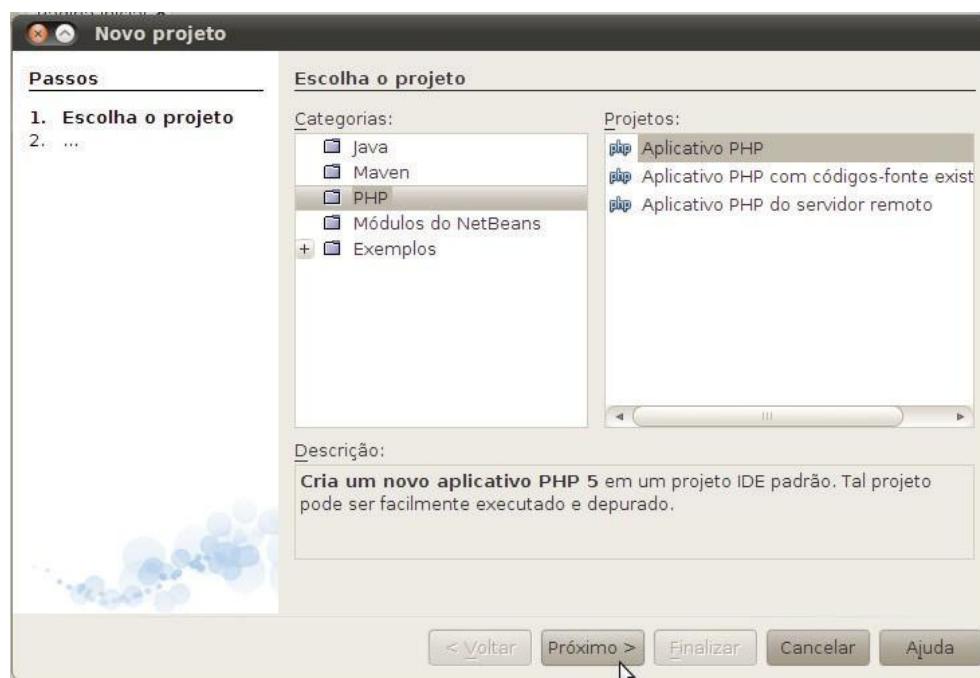
2.0 Iniciando um projeto com netbeans.

Para facilitar nosso trabalho, escolhemos usar a IDE Netbeans. Procure no site <http://netbeans.org/> Uma versão com o Netbeans instalada ou baixe plugin PHP em : Ferramentas > Plugins.



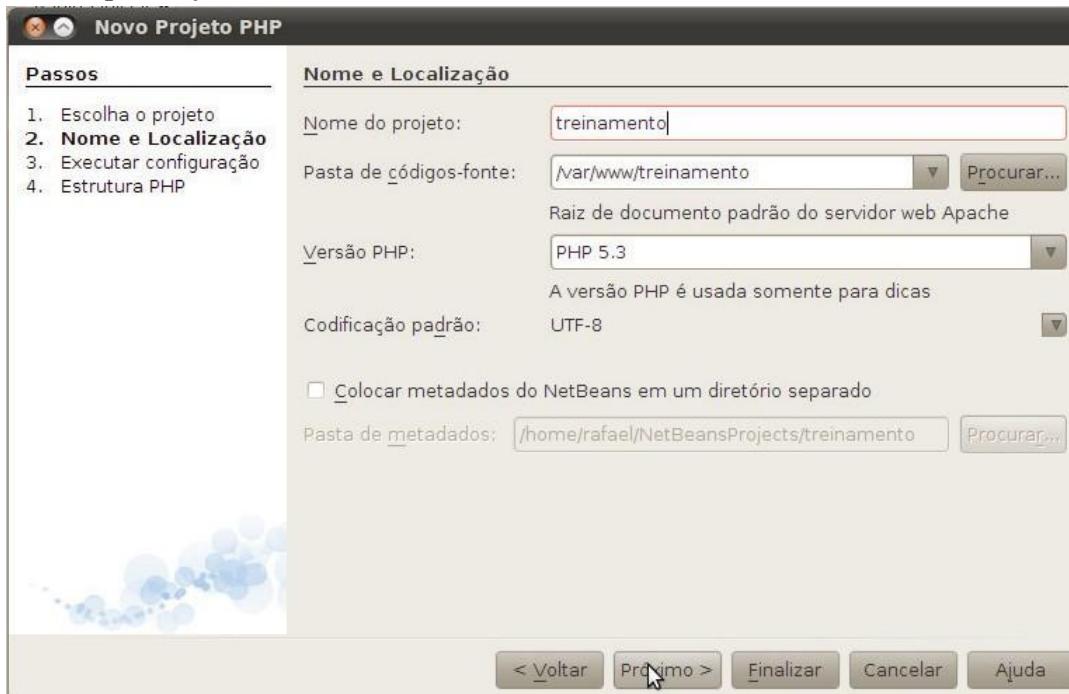
Com a IDE instalada devidamente, encontraremos o seguinte layout:

Para criar um novo projeto, vamos em: Arquivo > Novo Projeto .

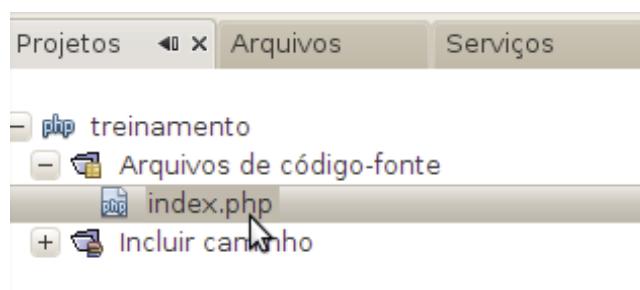


Escolhendo um projeto em PHP. Depois clicamos em Próximo:

Alteraremos o nome do Projeto para treinamento e certificar-se que ele estará dentro da pasta “/var/www/”. Clique ao final em Finalizar



Obs: Caso não tenha a pasta “/var/www/”, veja como está a instalação do servidor APACHE na sua máquina.



Na área de Projetos a esquerda, irá aparecer um novo projeto em PHP, através dele, estudaremos todo o conteúdo criando o novo projeto do site.

No palco Principal, o Netbeans já faz por nós toda a criação básica das estrutura HTML incluindo a TAG PHP na página. Para testar que a nossa aplicação, vamos dar um velho e conhecido : “Olá Mundo” em PHP. Para nos certificarmos que está tudo funcionando.

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5      <title>Página Inicial</title>
6    </head>
7    <body>
8      <?php
9        echo "Olá Mundo";
10     ?>
11   </body>
12 </html>
13

```

Veja o resultado no seu navegador: Acessando: "<http://localhost/treinamento/>".

Caso algo deu errado, veja novamente as configurações do APACHE ou PHP.

2.1 Tipos Primitivos

Todo o trabalho realizado por um computador é baseado na manipulação das informações contidas em sua memória. A grosso modo de expressar-se, estas informações podem ser classificadas em dois tipos:

As instruções, que comandam o funcionamento da máquina e determinam a maneira como devem ser tratados os dados. As instruções são específicas para cada modelo de computador, pois são funções do tipo particular de processador utilizado em sua implementação. Os dados propriamente ditos, que correspondem a porção das informações a serem processadas pelo computador.

2.1.1 Tipos de Primitivos de Dados

Quaisquer dados a ser tratado na construção de um algoritmo deve pertencer a algum tipo, que irá determinar o domínio de seu conteúdo. Os tipos mais comuns de dados são conhecidos como tipos primitivos de dados, são eles: inteiro, real, numérico ,caractere e lógico. A classificação que será apresentada não se aplica a nenhuma linguagem de programação específica, pelo contrário, ela sintetiza os padrões utilizados na maioria das linguagens.

- **Inteiro:** Todo e qualquer dado numérico que pertença ao conjunto de números inteiros relativos (negativo, nulo ou positivo). Exemplos: {...-4,-3,-2,-1,0,1,2,3,4,...}.
- **Real:** Todo e qualquer dado numérico que pertença ao conjunto de números reais(negativo, nulo ou positivo). Exemplos: { 15.34; 123.08 ; 0.005 -12.0 ; 510.20}.
- **Numérico:** Trata-se de todo e qualquer número que pertença ao conjunto dos inteiros ou reais, também abrange números binários, octal e hexadecimal.
- **Caracteres:** são letras isoladas. Exemplo : {'a', 'd', 'A', 'h'}.
- **Dados literais:** Conhecido também como **Conjunto de caracteres** ou String, é todo e qualquer dado composto por um conjunto de caracteres alfanuméricos (números, letras e caracteres especiais). Exemplos: {"Aluno Aprovado", "10% de multa", "Confirma a exclusão ??", "S", "99-3000-2", "email", "123nm", "fd54fd"}.
- **Lógico:** A existência deste tipo de dado é, de certo modo, um reflexo da maneira como

os computadores funcionam. Muitas vezes, estes tipos de dados são chamados de **booleanos**, devido à significativa contribuição de **Boole** a área da lógica matemática. A todo e qualquer dado que só pode assumir duas situações dados biestáveis, algo como por exemplo {0/ 1, verdadeiro/falso, sim/não, true/false }.

Veremos a seguir como esses valores serão armazenados e como o PHP irá interpretá-los.

EXERCÍCIOS PROPOSTOS

Como podemos definir PHP e qual a sua relação com HTML?

Descreva um exemplo estrutural relacionando tag's HTML e delimitações PHP.

Quais aplicativos básicos podemos instalar para criarmos um servidor de páginas em PHP?

Em relação ao apache, qual a sua principal finalidade?

Quais as principais extensões de arquivos PHP e diga a diferença entre elas?

Crie dois exemplos de código PHP usando seus delimitadores. **EP02.7:** Qual as finalidades de usar comentários dentro de um código-fonte? **EP02.8:** Cite os principais comandos de saída usados em PHP.

Qual a diferença entre **echo** e **print**?

Observe o seguinte código e diga qual o item correto:

```
<?php  
//página web.  
  
echo '<h1>Olá, essa é sua primeira página!<h1>';  
  
echo '<h2>Responda!</h2>' print 'O que é PHP?';  
print 'Qual sua finalidade?'; echo '<h4>Resposta: item a!</h4>';  
  
?>
```

I - Esse código pode ser interpretado pelo apache normalmente. II - As tag's HTML são interpretadas pelo servidor.

III - Esse código possui erros.

IV - As tag's são interpretada pelo navegador do cliente.

a) I, II, IV estão corretas.

b) somente a I está correta.

c) III, IV estão correta.

d) somente IV está correta.

e) I e IV estão corretas.

Prático:

Crie um arquivo PHP dentro da pasta www com o nome index.php, após isso pegue o código anterior e adicione a esse arquivo, defina quais textos são visualizadas em seu navegador. Caso exista erros, faça uma correção.

Crie dois arquivos diferentes, um com nome index.php, outro com o nome teste.php, após isso inclua o arquivo teste.php dentro do arquivo index.php, ambos arquivos deverá ter no seu código impresso mensagens diferentes utilize o comando print.

3.0 Atribuições em PHP

Objetivos

Mostrar a definição de variáveis e os seus tipos; Mostrar como pode-se atribuir um valor a ela, seu uso e aplicação em PHP; Mostrar a definição de constantes e sua importância.

3.1 Variáveis

Variáveis são identificadores criados para guardar valores por determinado tempo. Em PHP elas são declaradas e inicializadas, porém são armazenadas na memória RAM do servidor web. Esse é um dos motivos pelo qual os servidores precisam de grande quantidades de memória.

Imagine um servidor com mais de 20 mil acessos simultâneos ao mesmo tempo, onde cada usuário está acessando a mesma página feita em PHP. São criadas neste processo variáveis diferentes para cada usuário, logo, isso faz com que muitos processos sejam gerados e processados pelo servidor.

A tipagem em PHP é dinâmica, ou seja, as variáveis não precisam ser obrigatoriamente inicializadas após a declaração. Uma variável é inicializada no momento em que é feita a primeira atribuição. O tipo da variável será definido de acordo com o valor atribuído. Esse é um fator importante em PHP, pois uma mesma variável pode ser de um mesmo tipo ou não, e pode assumir no decorrer do código um ou mais valores de tipos diferentes.

Para criar uma variável em PHP, precisamos atribuir-lhe um nome de identificação, sempre procedido pelo caractere cifrão (\$). Observe um exemplo:

```

1 | <?php
2 |
3 | $nome = "João";
4 | $sobrenome = "da Silva";
5 | echo "$nome $sobrenome";
6 |
7 | ?>

```

Para imprimirmos as duas variáveis usamos aspas duplas no comando “echo”, no exemplo anterior temos a seguinte saída:



**Atenção!**

Obs.: Podem acontecer erros na exibição das mensagens por conta das codificações de acentuação. Caso isso aconteça, mude a codificação do seu navegador ou utilize as metas de codificação.

Algumas dicas Importantes:

- € Nomes de variáveis devem ser significativa e transmitir a ideia de seu conteúdo dentro do contexto no qual está inserido.
- € Utilize preferencialmente palavras em minúsculo (separadas pelo caracter “_”) ou somente as primeiras letras em maiúsculo quando da ocorrência de mais palavras.

Exemplo:

```
<?PHP  
    $codigo_cliente; //exemplo de variável  
    $codigoCliente; //exemplo de variável  
?>
```

- Nunca inicie a nomenclatura de variáveis com números. *Ex: \$1nota;*
- Nunca utilize espaço em branco no meio do identificado da variável. *Ex: \$nome um;*
- Nunca utilize caracteres especiais(! @ # ^ & */ | [] { }) na nomenclatura das variáveis.
- Evite criar variáveis com mais de 15 caracteres em virtude da clareza do código-fonte.

Com exceção de nomes de classes e funções, o PHP é **case sensitive**, ou seja, é sensível a letras maiúsculas e minúsculas. Tome cuidado ao declarar variáveis. Por exemplo a variável

\$codigo é tratada de forma totalmente diferente da variável *\$Codigo*.

Em alguns casos, precisamos ter em nosso código-fonte nomes de variáveis que podem mudar de acordo com determinada situação. Neste caso, não só o conteúdo da variável é mutável, mas também variante (variable variables). Sempre que utilizarmos dois sinais de cifrão (\$) precedendo o nome de uma variável, o PHP irá referenciar uma variável representada pelo conteúdo da primeira. Nesse exemplo, utilizamos esse recurso quando declaramos a variável \$nome (conteúdo de \$variável contendo 'maria').

```

1 |  <?php
2 | //define o nome da variável
3 | $variavel = 'nome';
4 | //cria variável identificada pelo conteúdo de $variavel
5 | $variavel = 'maria';
6 | // exibe variável $nome na tela
7 | echo "$nome";
8 | ?>
9 |

```

Resultado = maria.

Quando uma variável é atribuída a outra, sempre é criada uma nova área de armazenamento na memória. Veja neste exemplo que, apesar de \$b receber o mesmo conteúdo de \$a, após qualquer modificação em \$b, \$a continua com o mesmo valor,veja:

```

1 |  <?php
2 | $a = 5;
3 | $b = $a;
4 | $b = 10;
5 | echo $a; // resultado = 5
6 | echo $b; // resultado = 10
7 | ?>
8 |

```

Para criar referência entre variáveis, ou seja, duas variáveis apontando para a mesma região da memória, a atribuição deve ser precedida pelo operador &. Assim, qualquer alteração em qualquer uma das variáveis reflete na outra,veja:

```

1 |  <?php
2 | $a = 5;
3 | $b = &$a;
4 | $b = 10;
5 | echo $a; // resultado = 10
6 | echo $b; // resultado = 10
7 | ?>

```

No exemplo anterior percebemos que tanto \$a como \$b apontam para a mesma referência na memória, dessa forma se atribuirmos um novo valor em \$a ou em \$b, esse valor será gravado no mesmo endereço, fazendo com que, ambas variáveis possam resgatar o mesmo valor.

3.2 Tipos de Variáveis

Algumas linguagens de programação tem suas variáveis fortemente “tipadas”, diferentemente disso o PHP tem uma grande flexibilidade na hora de operar com variáveis. De fato, quando definimos uma variável dando-lhe um valor, o computador atribui-lhe um tipo. Isso permite que o programador não se preocupe muito na definição de tipos de variáveis, uma vez que isso é feita de forma automática. Porém deve ter cuidado com as atribuições de valores, evitando erros na hora de iniciar uma variável em PHP.

TIPO BOOLEANO.

Um booleano expressa um valor lógico que pode ser verdadeiro ou falso. Para especificar um valor booleano, utilize a palavra-chave TRUE para verdadeiro ou FALSE para falso. No exemplo a seguir, declaramos uma variável booleana \$exibir_nome, cujo conteúdo é TRUE para verdadeiro. Em seguida, testamos o conteúdo dessa variável para verificar se ela é realmente verdadeira imprimindo na tela caso seja. Usaremos a estrutura IF, uma estrutura de controle que veremos com mais detalhes no capítulo 4, para testar a variável. Observe:

```

1 | 1 <?php
2 | //Atribuição de valor.
3 | $exibir_nome = true;
4 | //Imprime na 1 na tela.
5 | print($exibir_nome);
6 | ?>
```

Resultado = 1 (esse valor representa verdadeiro ou true).

Também podemos atribuir outros valores booleanos para representação de valores falso em operações booleanas.

- Inteiro 0 ;
- Ponto flutuante 0.0 ;
- Uma String vazia “ ” ou “0” ;
- Um array vazio ;
- Um objeto sem elementos ;
- Tipo NULL .

TIPO INTEIRO

São os números que pertencem ao conjunto dos números inteiros, abrangendo valores negativos e positivos, trata-se de valores decimais.

```

1 | 1 <?php
2 | $a = 125;
3 | $b = -1456;
4 | $c = 7;
5 | ?>
```

TIPO PONTO FLUTUANTE:

Os números de ponto flutuante (floats e doubles) são números com casas decimais, onde a vírgula é substituída por um ponto. Exemplo:

```

1 | 1 <?php
2 | $a = 10.0;
3 | $a = 10.3e4;
4 | $a = 1.234;
5 | ?>
```

TIPO NUMÉRICO

Números podem ser especificados em notação decimal (base 10), hexadecimal (base 16) ou octal (base 8), opcionalmente precedido de sinal (- ou +), esse tipo abrange todos os valores abaixo:

```

1 1 <?php
2
3  $a = 1234; // número decimal
4  $b = -123; // um número negativo
5  $c = 0123; // numero octal (equivalente a 83 em decimal)
6  $d = 0x1a; // número hexadecimal (equivalente a 26 em decimal)
7  $e = 1.234; // ponto flutuante
8  $f = 4e23; // notação científica
9
10 ?>

```

Não entraremos em detalhes em relação a conversão desses valores, porém que fique claro a ideia de que uma variável numérica pode assumir diferentes tipos de valores.

TIPO STRING

Uma string é uma cadeia de caracteres alfanuméricos. Para declará-las podemos utilizar aspas

```

1 1 <?php
2  $a = 'Atribuição do tipo string'; //aspas simples
3  $a = "Atribuição do tipo string"; //aspas dupla
4  ?>

```

simples (' ') ou aspas duplas (""). Exemplo:

Observe na tabela abaixo o que podemos também inserir em uma String:

Sintaxe	Significado
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\"	A própria barra (\)
\\$	O símbolo \$
\'	Aspa simples
\\"	Aspa dupla

Observe o exemplo:

```

1 1 <?php
2  $a = '\\tipo de dado: \'inteiro\'';
3  echo $a;
4  ?>

```

Resultado: \tipo de dado: 'inteiro'

TIPO ARRAY

Array é uma lista de valores armazenados na memória, os quais podem ser de tipos diferentes (números, strings, objetos) e podem ser acessados a qualquer momento, pois cada valor é relacionado a uma chave. Um array também pode crescer dinamicamente com a adição de novos itens. Veja no capítulo 6 como manipular esse tipo de estrutura.

TIPO OBJETO

Um objeto é uma entidade com um determinado comportamento definido por seus métodos (ações) e propriedade (dados). Para criar um objeto deve-se utilizar o operador new. Para mais informações sobre orientação a objeto, consulte o site <http://php.net> e pesquise sobre object.

TIPO RECURSO

Recurso (resource) é uma variável especial que mantém uma referência de recursos externos. Recursos são criados e utilizado por funções especiais, como uma conexão ao banco de dados. Um exemplo é a função mysql_connect(), que ao conectar-se ao banco de dados, retorna um variável de referência do tipo recurso. Exemplo:

Resource mysql_connect(...) Outro exemplo: mysql_fetch_row(...)

TIPO NULL

Quando atribuímos um valor do tipo null (nulo) a uma variável estamos determinando que a mesma não possui valor, e que seu único valor é nulo. Exemplo:

```
$abc = null;
```

3.3 Constantes

3.3.1 Constantes pré-definidas

O PHP possui algumas constantes pré-definidas, indicando a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução e diversas outras informações. Para ter acesso a todas as constantes pré-definidas, pode-se utilizar a função phpinfo(), que exibe uma tabela contendo todas as constantes pré-definidas, assim como configurações da máquina, sistema operacional, servidor HTTP e versão do PHP instalada, como foi feito em exemplos anteriores.

3.3.2 Definindo constantes

Para definir constantes utiliza-se a função define. Uma vez definido, o valor de uma constante não poderá mais ser alterado. Uma constante só pode conter valores escalares, ou seja, não pode conter nem um array nem um objeto. A assinatura da função define é a seguinte:

```

1 <?php
2 define("NOME_DA_CONSTANTE", "valor inalterável");
3 echo NOME_DA_CONSTANTE;
4 ?>

```

define("NOME_DA_CONSTANTE", "valor inalterável"); Exemplo:

Resultado: valor inalterável

O nome de uma constante tem as mesmas regras de qualquer identificador no PHP. Um nome de constante válida começa com uma letra ou sublinhado, seguido por qualquer número de letras, números ou sublinhados. Você pode definir uma constante utilizando-se da função define(). Quando uma constante é definida, ela não pode ser mais modificada ou anulada.

Estas são as diferenças entre constantes e variáveis:

- Constantes podem ser definidas e acessadas de qualquer lugar sem que as regras de escopo de variáveis sejam aplicadas;
- Constantes só podem conter valores escalares;
- Constantes não podem ter um sinal de cífrão (\$) antes delas;
- Constantes só podem ser definidas utilizando a função define(), e não por simples assimilação;
- Constantes não podem ser redefinidas ou eliminadas depois que elas são criadas.

3.4 Conversão de variável

PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução.

Ainda assim, é permitido converter os valores de um tipo para outro desejado, utilizando o *typecasting* ou a função `settype` (ver adiante).

Assim podemos definir novos valores para terminadas variáveis:

typecasting	Descrição
(int),(integer)	<i>Converte em inteiro.</i>
(real),(float),(double)	<i>Converte em ponto flutuante.</i>
(string)	<i>Converte em string.</i>
(object)	<i>Converte em objeto.</i>

```

1 | 1 <?php
2 | 2 $a =(int)(48.56 + 145 + 15 - 0.21);
3 | 3 echo $a;
4 | 4 ?>
5 |

```

Convertendo de ponto flutuante para inteiro.

Resultado: 208

Convertendo de String para Object.

```

1 | 1 <?php
2 | 2 $a =(object) ("Bem vindo ao site!");
3 | 3 print_r($a);
4 | 4 ?>
5 |

```

Resultado:

stdClass Object ([scalar] =>Bem vindo ao site!)

Convertendo de inteiro para ponto flutuante.

```

1 | 1 <?php
2 | 2 $a = (double)(542);
3 | 3 print($a);
4 | 4 ?>
5 |

```

Resultado: 542

O resultado poderia ser 542.0, mas lembrando que o interpretador do PHP faz outra conversão ao notar que o numero 542.0 tem a mesma atribuição de 542. O resultado seria o mesmo se tentarmos atribuir \$a = 542.0.

EXERCÍCIOS PROPOSTOS

Qual a principal finalidade de uma variável?

O que significa tipagem automática.

Cite algumas dicas importantes na nomenclatura de variáveis:

Das variáveis abaixo, quais possuem nomenclaturas válidas.

\$a____b;	\$a_1_;	\$_início;
\$_@nome;	\$_val_!;	\$_--nome;
\$_a_ _;	\$_#valor;	\$_palavra;
\$_tele#;	\$_123;	\$_=_;
\$_VALOR_MAIOR;	\$______;	\$_all;

Resposta:

Crie dez variáveis atribuindo valores diversos, logo após use o comando **echo** pra imprimir na tela do browser, exemplo:

```
<?php  
$nome = “Maria Cavalcante”; echo $nome;  
....  
?>
```

Quais os tipos de variáveis que podemos citar em PHP.

Como podemos distinguir um tipo de variável de outro, uma vez que a tipagem é feita de forma automática em PHP.

Faça a ligação com os seguintes tipos:

- | | |
|------------------------|---------------------|
| 1 - \$var = -10; | ()ponto flutuante. |
| 2 - \$var = “palavra”; | ()tipo null. |
| 3 - \$var = 10.22; | ()tipo objeto. |
| 4 - \$var = true; | ()String. |
| 5 - \$var = null; | ()numérico. |
| 6 - \$var = new abc; | ()booleano. |

Qual a principal finalidade de um constante e como elas são definidas em PHP. **EP03.10:** Em que momentos precisamos converter uma variável de um tipo em outro. **EP03.11:** Quais os typecasting usados em PHP.

Crie uma constante com o comando define e imprima com o comando print();

Crie conversões e imprima na tela com o comando **print()** com as seguintes variável.

\$var1 = “paralelepípedo”, \$var2 = 15.20, \$var3 = 10.

- a) Converta a variável \$var1 em objeto.
- b) Converta a variável \$var3 em ponto flutuante.
- c) Converta a variável \$var2 em inteiro.

4.0 Operadores em PHP

Objetivos

Demonstrar os tipos e quais os operadores; Falar do conceito de atribuição e concatenação de strings;
Exemplificar os operadores, sua importância e funcionamento.

Os operadores tem seu papel importante dentro de qualquer linguagem de programação.

É através deles que podemos realizar diversos operações dentro de um programa, seja ela de atribuição, aritmética, relacional, lógico, dentre outros. Em PHP não é diferente, os operadores são utilizados constantemente, porém existem algumas regras que veremos mais adiante.

4.1 Operadores de strings

São operadores utilizados para unir o conteúdo de uma string a outra, com isso podemos dizer que há dois operadores de string. O primeiro é o operador de concatenação ('.') que já utilizamos em exemplos anteriores, ele retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação ('.='), que acrescenta o argumento do lado direito no argumento do lado esquerdo.

Observe o exemplo abaixo:

```
1 <?php
2
3 $a = " Bem vindo ";
4 $b = " ao site ";
5 $c = " de pesquisa ";
6 echo $a.$b.$c;    //imprime: Bem vindo ao site de pesquisa
7 $d = $a.$b.$c.=" tecnológica";
8 echo $d; //imprime: Bem vindo ao site de pesquisa tecnológica
9 $a .= " Aluno ";
10 echo $a; //imprime: Bem vindo Aluno
11 ?>
```

Nesse exemplo pode-se observar a declaração da variável **\$d**, logo após temos uma inicialização e atribuição de concatenação em uma mesma linha, isso é possível em PHP, deixando o código mais otimizado porém menos legível.

4.2 Operadores Matemáticos

4.2.1 Aritméticos

Chamamos de **operadores aritméticos** o conjunto de símbolos que representa as operações básicas da matemática.

Operações	Operadores	Exemplo	Resposta
Adição	+	3 + 5	8
subtração	-	20 - 5	15
Multiplicação		7 8	56
Divisão	/	15 / 3	5
Módulo (Resto da divisão)	%	10 % 3	1
Negação (Número Posto)	- (valor)	Se for 15 a atribuição	-15

Veja:

```

1 1 <?php
2 $a = 3;
3 $b = 4;
4 echo $a*$b+5-2*3; // 12+5-6 resultado: 11
5 echo $a*($b+5)-2*3; // 3*9-6 resultado: 21
6 ?>
7

```

Nesse exemplo fizemos algumas operações, porém ao utilizar parênteses, estamos determinando quem executa primeiro, no caso a soma de $\$b+5$.

4.2.2 Atribuição

O operador básico de atribuição é "=" (igual). Com ele podemos atribuir valores as variáveis como foi visto em exemplos anteriores. Isto quer dizer que o operando da esquerda recebe o valor da expressão da direita (ou seja, "é configurado para"). Mas podemos usar algumas técnicas, observe o exemplo abaixo:

```

1 1 <?php
2
3 $a = ($b=4)+5;
4 echo "a = $a,b = $b";
5
6 ?>

```

Resultado: $a = 9, b = 4$

Além do operador básico de atribuição, há "operadores combinados" usados para array e string, eles permitem pegar um valor de uma expressão e então usar seu próprio valor para o resultado daquela expressão.

Por exemplo:

```

1 |  <?php
2 |
3 |  $a = 3;
4 |  $a += 5; // $a recebe 8, então: $a = $a + 5;
5 |  $b = "Bom ";
6 |  $b .= "Dia!"; // $b recebe "Bom Dia!", então $b = $b . "Dia!";
7 |  echo "a = ".$a.",b = ".$b;
8 |
9 | ?>

```

Resultado: a = 8,b = Bom Dia!

Observe a expressão: \$a = 3 e logo após \$a+=5. Isto significa a mesma coisa de \$a = \$a + 5, ou, \$a = 3 +5. A ideia pode ser usada para string, como foi feito com a variável \$b, onde \$b = “Bom”, logo após usamos ponto(.) e igual(=) para concatenar os valores, ficando assim: \$b.=“Dia!”. Lembrando que isso significa a mesma coisa que \$b = \$b.“Dia”. Observe mais um exemplo:

```

1 |  <?php
2 |
3 |  $a = "Dia "; #inicia $a com Dia
4 |  $b = "Bom "; #inicia $b com Bom
5 |  $b .= $a.= "turma"; #concatena primeiro $a,logo depois $b
6 |  echo $b; #imprime na tela.
7 |
8 | ?>
9 |

```

Resultado: Bom Dia turma

Podemos definir uma sequência com duas concatenações, onde \$a = “Dia”.“turma” e

logo após temos \$b = “Bom”.“Dia turma”.

Os operadores de atribuição são usados para economizar linhas de código, deixando assim o código mais funcional e otimizado. A tabela abaixo mostra os principais operadores de atribuição:

Operadores	Descrição
=	Atribuição simples.
+=	Soma, depois atribui.
-=	Subtrai, depois atribui.
*=	Multiplica, depois atribui.
/=	Divide, depois atribui.
%=	Modulo(resto) da divisão, depois atribui.
.=	Concatena, depois atribui.

Exemplo:

```

1 | 1 <?php
2 |
3 | $a = 43; # inicia $a com 43
4 | $b = 62; # inicia $b com 62
5 |
6 | $a += 17; # atribui o valor anterior(43) mais 17 em $a
7 | echo $a; # imprime o valor(60) de $a na tela.
8 |
9 | $b -= 12; # atribui o valor anterior(62) menos 12 em $b
10 | echo $b; # imprime o valor(50) de $b na tela.
11 |
12 | ?>
13 |

```

Observe mais um exemplo aplicando os demais operadores.

```

1 | 1 <?php
2 | $a = 8; $b = 2; // inicializa duas variáveis
3 | # mutiplica o valor de $a(8) por 3 e atribui.
4 | echo ( $a *= 3 )."<br>" ;
5 | # divide o valor de $a(24) por 2 e atribui.
6 | echo ( $a /= 2 )."<br>" ;
7 | # resto da divisão de $a(12) por 5 e atribui.
8 | echo ( $a %= 5 )."<br>" ;
9 | // os pontos "." concatena com "<br>" .
10 | ?>
11 |

```

Resultado: 24

8
2

Vale ressaltar que a cada **echo**, o valor de **\$a** sofre modificações. Isso devido a atribuição feita após a operação. Usamos o operador ponto(.) para concatenar os valores obtidos com
 código usado em HTML para quebra de linha.

4.2.3 Operadores de decremento e incremento

São operadores usados para atribuir em 1 ou -1 a variável, isso pode ser feito antes ou depois da execução de determinada variável. A tabela abaixo mostra tais operadores:

Operadores	Descrição
++\$a	<i>Pré-incremento. Incrementa \$a em um e, então, retorna \$a.</i>
\$a++	<i>Pós-incremento. Retorna \$a, então, incrementa \$a em um.</i>
- -\$a	<i>Pré-decremento. Decrementa \$a em um e, então, retorna \$a.</i>
\$a- -	<i>Pós-decremento. Retorna \$a, então, decrementa \$a em um.</i>

Exemplo:

```

1 |  <?php
2 |  $a = 1;
3 |  print(++$a); // incrementa 1 em $a(1), depois imprime(2).
4 |  print($a++); // imprime $a(2) depois incrementa 1.
5 |  print($a);   // imprime $a(3)
6 |  print(--$a); // decrementa 1 em $a(3), depois imprime a$(2).
7 |  print($a--); // imprime $a(2), depois decremente 1.
8 |  print($a);   // imprime $a(1)
9 |
10| ?>

```

Nesse exemplo temos uma forma aplicada do uso de decremento e incremento, lembrando que a variável \$a pode ter qualquer nome. Também podemos fazer um comparativo com o Pré-incremento ou incremento-prefixado com operações que já conhecemos, observe:

Operador	Forma extensa.	Forma simplificada
++\$a	\$a = \$a + 1	\$a+=1
--\$a	\$a = \$a - 1	\$a -=1

4.3 Operadores relacionais

Os operadores relacionais ou conhecidos também como operadores de comparação, são utilizados para fazer determinadas comparações entre valores ou expressões, resultando sempre um valor booleano verdadeiro ou falso(TRUE ou FALSE). Para utilizarmos esses operadores usamos a seguinte sintaxe:

(*valor ou expressão*) + (*comparador*) + (*segundo valor ou expressão*)

Observe a tabela abaixo:

Comparadores	Descrição
==	<i>Igual. Resulta em TRUE se as expressões forem iguais.</i>
====	<i>Idêntico. Resulta em TRUE se as iguais e do mesmo tipo de dados.</i>
!= ou <>	<i>Diferente. Resulta verdadeiro se as variáveis foram diferentes.</i>
<	<i>Menor ou menor que. Resulta TRUE se a primeira expressão for menor.</i>
>	<i>Maior ou maior que. Resulta TRUE se a primeira expressão for maior.</i>
<=	<i>Menor ou igual. Resulta TRUE se a primeira expressão for menor ou igual.</i>
>=	<i>Maior ou igual. Resulta TRUE se a primeira expressão for maior ou igual.</i>

Veja um exemplo prático:

\$a <= \$b

Compara se \$a é menor ou igual a \$b, onde, retorna verdadeiro (TRUE), caso contrário retorna falso (FALSE).

Para testarmos essas comparações podemos utilizar o condicional “?:” (ou **ternário**), sua sintaxe é a seguinte:

(expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);

Agora podemos ver um exemplo envolvendo as sintaxes e empregabilidade dos comparadores:

```

1  <?php
2  $a = 15;
3  $b = 42;
4  $c = 42.0;                                #resposta:
5  echo $b == $c ? "verdadeiro" : "falso"; // verdadeiro
6  echo $b === $c ? "verdadeiro" : "falso"; // falso
7  echo $b != $c ? "verdadeiro" : "falso"; // falso
8  echo $a < $c ? "verdadeiro" : "falso"; // verdadeiro
9  echo $a < $c ? "verdadeiro" : "falso"; // verdadeiro
10 echo $a > $c ? "verdadeiro" : "falso"; // falso
11 echo $a <= $c ? "verdadeiro" : "falso"; // verdadeiro
12 echo $a >= $c ? "verdadeiro" : "falso"; // falso
13 ?>
```

Nesse exemplo declaramos e iniciamos três variáveis. Usamos então o comando **echo** para imprimir o resultado, onde o condicional “?:” foi utilizado. Iniciamos as comparações de \$a, \$b e \$c, caso a comparação individual retorne TRUE, imprime verdadeiro, caso retorne FALSE, imprime falso. Observe que o comparador “==” compara o valor e o tipo, retornando FALSE por \$b se tratar de um tipo inteiro, e \$c um tipo ponto flutuante, já o comparador “==” compara somente os valores onde 45 é igual a 45.0 retornando verdadeiro. Também podemos usar o operador “!==” onde tem a função semelhantemente ao operador “!=”, mas retorna TRUE se os tipos forem diferentes. Se a variável for do tipo booleano, podemos compará-los assim:

\$a == TRUE, \$a == FALSE

4.4 Operadores lógicos ou booleanos

São utilizados para avaliar expressões lógicas. Estes operadores servem para avaliar expressões que resultam em valores lógico sendo verdadeiro ou falso:

E	AND
OU	OR
Não	NOT

Esses operadores tem a finalidade de novas proposições lógicas composta a partir de outra proposições lógicas simples. Observe:

Operador	Função
<i>não</i>	negação
<i>e</i>	conjunção

ou	disjunção
-----------	-----------

Com isso podemos construir a Tabela verdade onde trabalhamos com todas as

possibilidades combinatória entre os valores de diversas variáveis envolvidas, as quais se encontram em apenas duas situações (V e F), e um conjunto de operadores lógicos. Veja o comportamento dessas variáveis:

Operação de Negação:

A	(not) não A
F	V
V	F

Trazendo para o nosso cotidiano:

Considerando que A = “Está chovendo”, sua negação seria : “Não está chovendo”. Considerando que A = “Não está chovendo” sua negação seria : “Está chovendo”

Operação de conjunção:

A	B	A e B
F	F	F
F	V	F
V	F	F
V	V	V

Trazendo para o nosso cotidiano:

Imagine que acontecerá um casamento:

Considerando que A = “Noivo presente” e B = “Noiva presente”.

Sabemos que um casamento só pode se realizar, se os 2 estejam presente.

Operação de disjunção não exclusiva:

A	B	A ou B
F	F	F
F	V	V
V	F	V
V	V	V

Trazendo para o nosso cotidiano:

Imagine que acontecerá uma prova e para realizá-la você precisará da sua Identidade ou título de eleitor no dia da prova.

Considerando que A = “Identidade” e B = “Título de eleitor”.

Sabemos que o candidato precisa de pelo menos 1 dos documentos para realizar a prova.

São chamados de operadores lógicos ou booleanos por se tratar de comparadores de duas ou mais expressões lógicas entre si, fazendo agrupamento de testes condicionais e tem como retorno um resultado booleano.

Na tabela abaixo temos os operadores e suas descrições:

Operador	Descrição
(\$a and \$b)	E : Verdadeiro se tanto \$a quanto \$b forem verdadeiros.
(\$a or \$b)	OU : Verdadeiro se \$a ou \$b forem verdadeiros.

(\$a xor \$b)	XOR : Verdadeiro se \$a ou \$b forem verdadeiro, de forma exclusiva.
(! \$a)	NOT : Verdadeiro se \$a for falso, usado para inverter o resultado da condição.
(\$a && \$b)	E : Verdadeiro se tanto \$a quando \$b forem verdadeiros.
(\$a \$b)	OU : Verdadeiro se \$a ou \$b forem verdadeiros.



Dicas: or e and tem procedência maior que && ou ||, ou seja, em uma comparação extensa, onde ambos estão aplicados. Eles tem prioridade de executar sua comparação primeiro.

No próximo exemplo usamos os operadores lógicos que tem procedência maior:

```

1  <?php
2   $a = 10 > 2;           // verdadeiro, pois 10 é maior que 2.
3   $b = 12 >= 12;         // verdadeiro, pois 12 é igual a 12.
4   $c = FALSE;            // inicia com FALSE(tipo booleano).
5   echo ($b and $a) ? "sim" : "não" ; //sim, pois $a e $b são verdadeiros.
6   echo ($b or $c) ? "sim" : "não" ; //sim, pois $b é verdadeiro.
7   echo ($b xor $a) ? "sim" : "não" ; //não, pois $b e $a são verdadeiros.
8
9

```

Em outro exemplo temos os operadores lógicos mais comuns:

```

1 | 1 <?php
2 | $a = 10 > 2;      // verdadeiro, pois 10 é maior que 2.
3 | $b = 12 >= 12;    // verdadeiro, pois 12 é igual a 12.
4 | $c = FALSE;       // inicia com FALSE(tipo booleano).
5 | echo (!$c)        ? "sim" : "não" ; //sim, pois $c é falso.
6 | echo ($a && $c)  ? "sim" : "não" ; //não, pois $c é falso.
7 | echo ($b || $c)   ? "sim" : "não" ; //sim, pois $b é verdadeiros.
8 |
9 | ?>

```

```

1 | 1 <?php
2 | $a = 0;
3 | $b = 0;
4 | ($a = 2 or $b = 3);
5 | echo $a.".". $b;
6 | ($a = 5 and $b = 3);
7 | echo $a.".". $b;
8 |
9 | ?>

```

Também podemos atribuir valores as variáveis usando os operadores lógicos:

O primeiro **echo** mostra 2 e 0, pois não atribui valor a
\$b uma vez que a primeira condição já é satisfatória.

O segundo **echo** mostra 5 e 3, pois tanto a primeira quanto a segunda precisam ser executadas.

4.5 Precedência de Operadores

Agora já conhecemos uma boa quantidade de operadores no PHP, falta agora conhecer a precedência de cada um deles, ou seja, quem é mais importante, qual operador é avaliado primeiro e qual é avaliado em seguida. Observe o seguinte exemplo:

```

1 | 1 <?php
2 |
3 | echo 5 + 2 * 6; // resultado: 17
4 |
5 | ?>

```

O resultado será **17**, pois o operador ***** tem maior precedência em relação ao operador **+**. Primeiro ocorre a multiplicação 2×6 , resultando em 12, em seguida a soma de $5 + 12$. Caso desejar realizar a operação com o operador **+** para só em seguida realizar a operação com o operador *****, temos que fazer conforme o exemplo abaixo:

```

1 | 1 <?php
2 |
3 | echo (5 + 2) * 6; // resultado: 42
4 |
5 | ?>

```

Observe que utilizamos os parênteses para determinarmos quem deve ser executado primeiro, assim alterando o resultado para **42**. Os parênteses determina qual bloco de código executa

primeiro, e também serve para isolar determinadas operações. Veja mais um exemplo onde as operações são feitas separadamente. Primeiro executa a soma, em seguida a subtração e só então é executado a multiplicação, imprimindo um resultado final **21**:

Exemplo:

```

1 <?php
2
3 echo (5 + 2) * (6 - 3); // resultado: 21
4
5 ?>
6

```

A tabela seguinte mostra a precedência dos operadores, da maior precedência no começo para os de menor precedência.

Operador	Descrição
- ! + +	--Negativo, negação, incremento e decremento
/ * %	Multiplicação, divisão e resto da divisão
+ - .	Adição, subtração e concatenação
> < >= <=	Maior que, menor que, maior ou igual e menor ou igual
== != <>	Igual e diferente
&&	E
	OU
= += -=	Operadores de atribuição
= /= %=	
AND	E com menor prioridade
XOR	Ou exclusivo
OR	Ou com menor prioridade

É importante lembrar que primeiro o PHP executará todas as operações que estiverem entre parênteses, se dentro dos parênteses houver diversas operações, a precedência dos operadores será utilizada para definir a ordem. Após resolver todas as operações dos parentes, o PHP volta a resolver o que está fora dos parênteses baseando-se na tabela de precedência de operadores. Havendo operadores de mesma prioridade o PHP resolverá a operação da esquerda para direita.

Também podemos trabalhar com precedência de parênteses, fazendo associações com um ou mais operadores, observe o seguinte exemplo:

```

1 <?php
2
3 echo (3 + 2) * (9 - 3) / (16 - ((5+2)*2));
4 // resultado: 15
5
6 ?>
7

```

Seguindo a ordem de procedência temos:

$$(5) \quad *(6) / (16 - ((7) *2)) >>> 5 *6 / (16 -(14)) >>> 5 *6 / 2 >>> 30 / 2$$

Resultado : 15

Observe que primeiro executa todos os parênteses, e só então temos as procedência das demais operações.

EXERCÍCIOS PROPOSTOS

Qual a finalidade dos operadores de strings?

Quais os operadores de decremento e incremento? Cite alguns exemplos:

Qual a finalidade do operador aritmético %(modulo)?

Cite os operadores relacionais, mostre alguns exemplos.

Quais operadores lógicos ou booleanos?

Quais os operadores de atribuição?

Qual a sintaxe do uso de ternário e cite um exemplo?

Quais os operadores utilizados e o resultado final do código abaixo:

```
<?php
$a =10;
$b = 12.5;
$c = $a+$b;
print($a>$b? "verdadeiro" : "falso"); print($c>=$b? "verdadeiro" : "falso");
?>
```

Observe o código abaixo e diga quais das operações são executadas primeiro, coloque a resposta em ordem decrescente.

```
$a = 8*5-3+4f2+19%5f2+1;
```

Faça testes com os operadores relacionais substituindo o operando > do código-fonte abaixo.

```
<?php
$var1 = 2.2564;
$var2 = 2.2635;
print($var1 > $var2 ? "sim" : "não");
?>
```

Usando o operador de String “.” para montar a seguinte frase abaixo:

```
<?php
$a = "de";
$b = "é um";
$c = "comunicação";
$c = "a";
$d = "internet";
$e = "meio"; print( ..... );
?>
```

Observe o código-fonte abaixo e diga qual o resultado booleano final. Justifique sua resposta.

```
<?
$a = 12.0 < 11.2;
$b = 10*2-3 > 19%3+10;
$c = 10;
print( ($a || $c = 10 && $b) ? "true" : "false");
?>
```

5.0 Interações PHP com HTML

Objetivos

Apresentar ao aluno como trabalhar com interações PHP dentro do código HTML; Mostrar exemplos de formulário, o uso de métodos POST e GET, cookies, listagem, seção e suas interações com o Browser.

Abordaremos neste capítulo algumas formas de interações utilizando a linguagem de programação PHP e a linguagem de marcação HTML. Além disso, mostraremos alguns componentes mais utilizados para a construção de um sistema ou uma página web, e de que forma o PHP pode receber, processar, e enviar essa informação.

Utilizamos linguagens para exibir os dados da aplicação, seja ela para simples conferência, em relatório ou ainda possibilitando a adição e exclusão de registros. Criaremos a princípio formulários e listagem.

5.1 Formulários

Podemos definir fomulário como um conjuntos de campos disponíveis ao usuário de forma agrupada para serem preenchidos com informações requisitada pela aplicação (sistemas web ou páginas). Um formulário é composto por vários componentes, além de possuir botões de ação, no qual define o programa que processará os dados.

Em uma aplicação determinamos então a entrada de dados (no caso os formulários), e a saída de dados, que é toda e qualquer informação apresentada ao usuário pelo browser, de forma que ambas tenham uma ligação lógica e possa retornar um resultado onde todos os componentes da aplicação trabalhem de forma coerente.

ELEMENTOS DE UM FORMULÁRIO.

Para criarmos um formulário, utilizamos a tag <form> e dentro dela podemos dispor diversos elementos, onde, cada um deles representa uma propriedade em particular. A seguir explicaremos os principais componentes de um formulário.

Criando um formulário:

Todo formulário deve conter no mínimo as seguintes características, observe:

```
1 |  <form name="" method="" action="">
2 |    ...
3 |  </form>
```

name < Nome atribuído ao formulário para sua identificação. **method** < Método POST ou GET como veremos mais adiante.

action < Caminho do arquivo que receberá os dados do formulário ao ser enviado.

Os elementos do formulário serão preenchidos com os componentes input onde a tag é <input> conforme o exemplo abaixo:

```

1 | <form name="" method="" action="">
2 |   ...
3 |   <input name="" value="" type="">
4 |   ...
5 | </form>

```

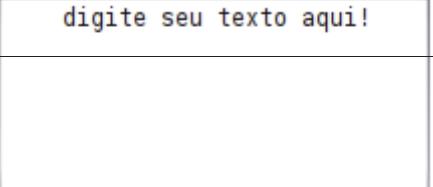
A tag input pode ser composta por vários elementos onde neles podemos definir o tipo, nome, e o valor padrão além de outras propriedades que não apresentaremos nessa apostila.

name < Nome atribuído ao componente para sua identificação. value < Valor padrão que pode ser atribuído no formulário.

type < Tipo de input, é nele onde definimos o tipo elemento o input vai representar.

Observe a tabela abaixo com a definição dos tipos que podemos atribuir ao elemento type do input.

type	Descrição	Exemplo:
texto	Elemento utilizado para entra de texto simples, é um dos mais utilizados.	<input type="text" value="texto"/>
password	Elemento utilizado para entrada de senhas, exibe “ ” no lugar dos caracteres inseridos pelo usuário.	<input type="password" value="●●●●"/>
checkbox	Utilizado para exibir caiar de verificação, muito utilizado para perguntas booleanas.	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2
radio	Exibe botões para seleção exclusiva, utilizado para seleção de apenas um item.	sex: M <input checked="" type="radio"/> F <input type="radio"/>
file	Utilizado para seleção de arquivos, esse tipo é muito comum quando queremos enviar algo pelo navegador(browser).	<input type="file"/> Enviar arquivo...
hidden	É utilizado para armazenar um campo escondido dentro do formulário.	Não é visível ao usuário.
button	Usado para exibir um botão na tela, porém sua ação é definida por outra linguagem como javascript.	<input type="button" value="enviar"/>
submit	Botão usado para submeter os dados do formulário no servidor, ele envia as informações de acordo com as informações preenchidas na tag <form>.	<input type="submit" value="enviar"/>
reset	Utilizado para limpar todos os campo do formulário, voltando ao valor inicial.	<input type="reset" value="resetar"/>

select	Tipo utilizado para exibir uma lista de valores contido na lista de seleção do usuário (cada valor é guardado em uma tag <option>, porém só pode ser selecionado uma opção.	
Tag: <textarea>	Área de texto disponibilizado ao usuário, possui múltiplas linhas (rows) e colunas (cols) que podem ser determinados dentro de sua tag. Não é um tipo de input, mas é uma tag HTML para formulário.	

```
<textarea name="area" rows = "5" cols="27">
digite seu texto aqui!
</textarea>
```

A tag <textarea> pode ser definida conforme o exemplo de código abaixo:

Observe que definimos uma caixa de texto com 5 linhas e 27 colunas, isso define o tamanho que essa caixa vai ter dentro do formulário.

5.2 Exemplo de formulário.

Veremos agora exemplos de alguns formulários comumente encontrados, usando os componentes apresentado anteriormente. Não entraremos em detalhes do HTML, sendo apresentado de maneira direta, pois está subentendido que o aluno já conheça HTML. Observe os exemplos abaixo:

Tela de login:

código-fonte:

```

1  <form name="cadastro" method="post" action="gravar.php">
2    <table border="1" width="400px" bgcolor="#f0f0f0"
3      style="border-collapse: collapse">
4        <tr>
5          <td>Login :</td>
6          <td><input type="text" value="Nome completo."></td>
7        </tr>
8        <tr>
9          <td>Senha :</td>
10         <td><input type="password" value="12345678"></td>
11       </tr>
12
13     </table>
14     <input type="submit" value="Entrar">
15   </form>

```

Resultado:

Login :	<input type="text" value="Nome completo."/>	
Senha :	<input type="password" value="●●●●●●●●"/>	
<input type="button" value="Entrar"/>		

Tela de Cadastro:

código-fonte:

```

1  <form name="cadastro" method="post" action="gravar.php">
2    <table border="1" width="400px" bgcolor="#f0f0f0"
3      style="border-collapse: collapse">
4        <tr>
5          <td>Nome : </td>
6          <td><input type="text" ></td>
7        </tr>
8        <tr>
9          <td>Endereço : </td>
10         <td><input type="text" ></td>
11       </tr>
12       <tr>
13         <td>Cidade : </td>
14         <td><input type="text" ></td>
15       </tr>
16       <tr>
17         <td>Estado : </td>
18         <td><input type="text" size="3" maxlength="2" ></td>
19       </tr>
20     </table>
21     <input type="submit" value="Cadastrar">
22   </form>
```

Resultado com dados preenchidos:

Nome :	<input type="text" value="Alex Sousa"/>
Endereço :	<input type="text" value="Rua 123"/>
Cidade :	<input type="text" value="Fortaleza"/>
Estado :	<input type="text" value="CE"/>
<input type="button" value="Cadastrar"/>	

Tela de envio de dados e arquivos:
código-fonte:

```

1  <form name="cadastro" method="post" action="gravar.php">
2      <table border="1" width="400px" bgcolor="#f0f0f0"
3          style="border-collapse: collapse">
4          <tr>
5              <td>Titulo:</td>
6              <td><input type="text" ></td>
7          </tr>
8          <tr>
9              <td>Nome do autor:</td>
10             <td><input type="text" ></td>
11         </tr>
12         <tr>
13             <td>Imagen:</td>
14             <td><input type="file" ></td>
15         </tr>
16         <tr>
17             <td>Descrição:</td>
18             <td><textarea rows="6" cols="35">
19                 Digite sua descrição aqui.
20             </textarea></td>
21         </tr>
22     </table>
23     <input type="submit" value="Enviar dados">
24 </form>
```

Resultado:

Titulo:	<input type="text"/>
Nome do autor:	<input type="text"/>
Imagen:	<input type="file"/> Enviar arquivo...
Descrição:	Digite sua descrição aqui.
Enviar dados	

Com esses exemplos já podemos trabalhar com as informações até agora mostrada. Para isso vamos conhecer os dois principais métodos de envio de dados de um formulário para um arquivo PHP.

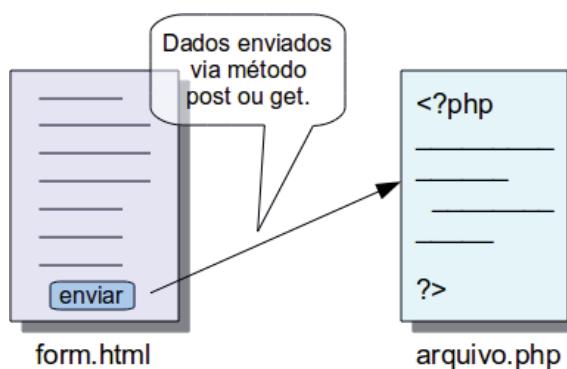
5.3 Métodos Post e Get

Quando falamos em como enviar dados para um formulário, deve vir em mente os métodos GET e POST, que são os métodos utilizados. Mas quando fazemos uma requisição HTTP, sempre utilizamos um desses métodos, normalmente o GET.

Se você digita um endereço na barra de endereço seu navegador e aperta a tecla enter (ou clica no botão ir), o navegador faz uma requisição HTTP para o servidor do endereço digitado e o método dessa requisição é o GET. Se você clica em um link em um site, o navegador também se encarrega de fazer um requisição HTTP com o método GET para buscar o conteúdo da página que você clicou.

Esse mecanismo funciona da seguinte forma. Primeiro temos em um formulário, um botão ou link. Quando clicamos em umas dessas propriedades, estamos enviando uma requisição. A forma de enviar pode ser definida pelo método get ou post e deve ser enviada para algum arquivo, que ao receber, processe a informação devolvendo resultado ou não.

Veja abaixo:



a ilustração

5.4 Método Get

O método GET utiliza a própria URI (normalmente chamada de URL) para enviar dados ao servidor. Quando enviamos um formulário pelo método GET, o navegador pega as informações do formulário e coloca junto com a URI de onde o formulário vai ser enviado e envia, separando o endereço da URI dos dados do formulário por um “?” (ponto de interrogação) e “&”.

Quando você busca algo no Google, ele faz uma requisição utilizando o método GET, você pode ver na barra de endereço do seu navegador que o endereço ficou com um ponto de interrogação no meio, e depois do ponto de interrogação você pode ler, dentre outros caracteres, o que você pesquisou no Google.

Abaixo temos um exemplo de uma URL do gmail da google. Observe:



Podemos notar a passagem de dois valores:

?hl = pt-br, logo após &shva=1, ou seja, temos então a criação da “variável” **hl** que recebe o valor pt-br, e também a “variável” **shva** que recebe como valor 1. Veja exemplos de códigos com links enviando valores por métodos GET.

```

1| 1 <a href="gravar.php?pagina=3&conteudo=4">Informação</a>
2| 2 <a href="gravar.php?pg=24&user=anonimo">Entrar</a>
3| 3 <form method="get" action="gravar.php">
4| 4     Nome:<input name="nome" type="text">
5| 5     <input type="submit" value="Enviar">
6| 6 </form>

```

Ao clicarmos em um desses links, podemos observar o seguinte comportamento na URL do navegador:

Link informação < Estamos enviando dois valores via método GET, um contendo 3, e o outro 4, para o arquivo gravar.php .

Link Entrar < Estamos enviando dois valores via método GET, um contendo 24, e o outro anonimo, para o arquivo gravar.php .

Formulário(<form>)

< Envias as informações preenchidas no **input nome** via GET para o arquivo gravar.php.

RECEBENDO DADOS VIA MÉTODO GET

Agora trabalharemos com o arquivo PHP, onde podemos resgatar os valores enviados pelo método **\$_GET**, sua sintaxe é a seguinte:

\$_GET['nome_da_campo'] < retorna o valor passado pelo campo.

\$_GET < retorna um **array** com todos os valore enviados e seus supostos índices.

Quando queremos um valor específico, colocamos o nome da “variável” da URL ou o nome atribuído na propriedade name do input do formulário.

Exemplo com links:

código – HTML:

```
<a href="gravar.php?pagina=3&conteudo=4">Informação</a>
```

código – PHP (nome do arquivo: gravar.php):

Resultado:

```

1| 1 <?php
2| 2
3| 3 // Imprime valores específico
4| 4 echo $_GET['pagina']."<br>";
5| 5 echo $_GET['conteudo']."<br>";
6| 6 // monta um array com todos valores enviados.
7| 7 $a = $_GET;
8| 8 foreach($a as $nome => $valor)
9| 9     echo $nome." = ".$valor."<br>";
10|
11| ?>

```

3
4
pagina = 3
conteudo = 4

Exemplo com Formulário:

código – HTML:

```
1 |  <form method="get" action="gravar.php">
2 |      Login: <input name="login" type="text">
3 |      Senha: <input name="senha" type="password">
4 |          <input type="submit" value="Logar">
5 |  </form>
```

Resultado no Browser:

Login: Alex	Senha: ●●●●●●●●	Logar
-------------	-----------------	-------

código – PHP (nome do arquivo: gravar.php):

```
1 |  <?php
2 |
3 | echo $_GET['login']."<br>";
4 | echo $_GET['senha']."<br>";
5 |
6 | ?>
```

Resultado:



Em casos que precisamos de mais segurança, onde o ocultamento de informação é necessário, o método GET não é uma boa opção. Observe no exemplo anterior que qualquer valor passado pela URL fica visível, e o usuário pode ver informações pessoais como um login e uma senha.



Dica: Em casos semelhante, utilize sempre o método POST.

5.5 Método Post

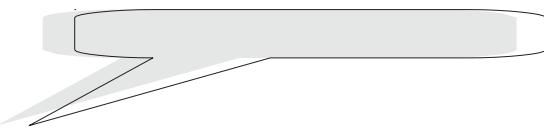
Muito semelhante ao método GET, porém a principal diferença está em enviar os dados encapsulado dentro do corpo da mensagem. Sua utilização é mais viável quando trabalhamos com informações segura ou que poder ser alteradas somente por eventos do Browser. Sintaxe:

`$_POST['nome_da_campo']` < retorna o valor passado pelo campo.
`$_POST` < retorna um `array` com todos os valore enviados e seus supostos índices.

Veja o mesmo exemplo anterior, porém com o uso de POST:

Exemplo com Formulário:

código – HTML:



```
1 | 1<form method="post" action="gravar.php">
2 | 2    Login: <input name="login" type="text">
3 | 3    Senha: <input name="senha" type="password">
4 | 4        <input type="submit" value="Logar">
5 | 5</form>
```

código – PHP (nome do arquivo: gravar.php):

```
1 | 1<?php
2 |
3 | 3echo $_POST['login']."<br>";
4 | 4echo $_POST['senha']."<br>";
5 |
6 | 6?>
```

Resultado após o envio dos dados preenchidos:

Alex
12345678

Mudança no método

Com o uso do método post, as informações ficam invisíveis ao usuário, isso evita que alguma causa mal-intencionada venha tornar os dados inconsistentes(dados sem fundamentos ou falsos).

6.0 Estruturas de controle e repetição

Objetivos

Mostra estruturas de controle e sua aplicação prática em PHP; Definir qual a principal finalidade dessas estruturas; Mostrar exemplos em sua sintaxe; Mostrar aplicação e uso de foreach.

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando descrever a sintaxe de cada uma delas resumindo o funcionamento. Independente do PHP, boa parte das outras linguagens de programação tem estruturas iguais, mudando apenas algumas sintaxes.

6.1 Blocos de controle

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em PHP são delimitados pelos caracteres “{” e “}”. A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador. Outro detalhe importante: usar as estruturas de controle sem blocos delimitadores faz com que somente o próximo comando venha ter ligação com a estrutura. Observe os exemplos:

```

1 |  <?php
2 |
3 |  $a = 0;
4 |  if($a>2)
5 |      echo "comando1";
6 |      echo "comando2";
7 |
8 |  ?>
9 |

```

Observe que temos um comando IF, onde é passado a ele uma expressão booleana que retorna verdadeiro ou falso.

O resultado da expressão é FALSE(falso), pois 0 não é maior que 2, fazendo com que o IF não execute o echo com “comando1”. Somente o segundo echo é executado, pois não pertence ao IF declarado.

Mas se quisermos que mais de um comando pertença a estrutura de controle, será usado blocos de comandos

({ comando; }), onde através deles podemos delimitar e organizar os códigos.

```

1 |  <?php
2 |
3 |  $a = 0;
4 |  if($a>2){ // início do bloco de código.
5 |      echo "comando1";
6 |      echo "comando2";
7 |  } // fim do bloco de código.
8 |
9 |

```

No código acima, temos um bloco onde inserimos dois comandos. Observe que eles não serão

executados, pois a expressão booleana passada para o IF é falsa.

6.2 If e else

Essa estrutura condicional está entre as mais usadas na programação. Sua finalidade é induzir um desvio condicional, ou seja, um desvio na execução natural do programa. Caso a condição dada pela expressão seja satisfeita, então serão executadas as instruções do bloco de comando. Caso a condição não seja satisfeita, o bloco de comando será simplesmente ignorado. Em lógica de programação é o que usamos como “SE (expressão) ENTÃO {comando:}”.

Sintaxe:

```
if (expressão)
    comando; if (expressão) {
        comando1;
        comando2; comando3;
    }
```

exemplo:

```
1 <?php
2
3 $idade = 16;
4 if($idade > 18 ){
5     $texto = "maior idade";
6     echo $texto;
7 }
8 ?>
9
```

Caso a condição não seja satisfatória (FALSE), podemos atribuir outro comando pertencente ao IF chamado ELSE, como se fosse a estrutura SENÃO em lógica de programação.

Sintaxe:

```
if (expressão)
    comando;
else
    comando;
if (expressão) {
    comando1;
    comando2;
    comando3;
} else {
    comando1;
    comando2;
    comando3;
}
```

Exemplo:

```
1 <?php
2
3 $idade = 16;
4 if($idade > 18 ){
5     $texto = "maior idade";
6 }
7 else {
8     $texto = "menor idade";
9     echo $texto;
10 }
11 ?>
12
```

Nesse exemplo temos uma expressão booleana onde retorna falso, com isso o IF não executa, passando a execução para o else, que por sua vez executa e atribui o valor “menor idade” a variável \$texto.

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Isso é o que podemos chamar de “If's encadeados”, onde usamos a estrutura **IFELSE**. Para facilitar o entendimento de uma estrutura do tipo:

Sintaxe:

```

if (expressao1)
    comando1;

    else
if (expressao2)

    comando2;
else
    if (expressao3)
        comando3;
    else

```

comando4;

Exercício rápido:

1º) Faça um script em PHP que possua 4 notas de um aluno (cada uma em uma variável). Depois calcule e imprima a média aritmética das notas e a mensagem de aprovado para média superior ou igual a 7.0 ou a mensagem de reprovado para média inferior a 7.0.

2º) Faça um script em PHP que receba a idade de um nadador (representada por uma variável chamada “\$idade”) e imprima a sua categoria seguindo as regras:

Categoria	Idade
Infantil A	5 - 7 anos
Infantil B	8 - 10 anos
Juvenil A	11- 13 anos
Juvenil B	14- 17 anos
Sênior	maiores de 18 anos

6.3 Atribuição condicional (ternário)

Como já vimos exemplos de atribuição condicionais (ternários), podemos defini-los usando a sintaxe:

(expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);

Isso se aplica quando queremos uma estrutura resumida, onde podemos ter um resultado mais direto, como por exemplo, atribuir um valor a uma variável dependendo de uma expressão. Observe o exemplo abaixo onde envolvemos uma variável do tipo string, porém o valor

```

1 | 1| <?php
2 |
3 | $idade =16;
4 | $texto = $idade > 18 ? "maior idade" : "menor idade";
5 | echo $texto;
6 |
7 | ?>

```

atribuído a essa variável deverá ser de acordo com o valor da idade:

É uma estrutura parecida com IF e ELSE, onde dependendo da expressão booleana podemos executar um bloco ou não.

Exercício rápido:

1º) Faça uma script em PHP que receba um número representado por uma variável. Verifique se este número é par ou ímpar e imprima a mensagem.

2º) Crie outro script baseando em um seguro de vida com as seguintes regras:

Idade: **Grupo de Risco :**

18 a 24 - Baixo

25 a 40 - Médio

41 a70 - Alto

6.4 Switch

Observe que quando temos muitos “if's encadeados” estamos criando uma estrutura que não é considerada uma boa prática de programação. Para resolver esse problema temos uma estrutura onde sua funcionalidade é semelhante ao IFELSE. O comando SWITCH é uma estrutura que simula uma bateria de teste sobre uma variável. Frequentemente é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um desses valores.

Sintaxe:

```
switch(expressão)
{
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    ...
}
```

Exemplo:

```
1 <?php
2 $numero = 2;
3 switch ($numero){
4     case 1:
5         echo "opção 1:";
6     case 2:
7         echo "opção 2:";
8     case 3:
9         echo "opção 3:";
10    case 4:
11        echo "opção 4:";
12    case 5:
13        echo "opção 5:";
14    }
15 ?>
16
17
```

Resultado: opção 2:opção 3:opção 4:opção 5:

Nesse exemplo temos o número = 2, onde o switch compara com os case's o valor recebido, o bloco que é executado é do segundo case, porém os demais também são executados para que tenhamos um resultado satisfatório temos que usar em cada case um comando chamado **break**. No qual tem a função de para o bloco de execução.

6.4.1 Switch com break

Break é uma instrução (comando) passada quando queremos parar o fluxo da execução de um programa. Em PHP, ele tem a mesma função que é “abortar” o bloco de código correspondente.

Observe o mesmo exemplo com o uso de break:

Obs.: Além de números podemos também comparar outros tipos como string, pontos flutuantes e inteiros, veja um exemplo abaixo:

```

1  <?php
2  $numero = 2;
3  switch ($numero){
4      case 1:
5          echo "opção 1:";
6          break;
7      case 2:
8          echo "opção 2:";
9          break;
10     case 3:
11         echo "opção 3:";
12         break;
13     case 4:
14         echo "opção 4:";
15         break;
16     case 5:
17         echo "opção 5:";
18         break;
19 }
20 ?>
```

```

1  <?php
2  $numero = "opc 2";
3  switch ($numero){
4      case "opc 1":
5          echo "opção 1:";
6          break;
7      case "opc 2":
8          echo "opção 2:";
9          break;
10 }
11 ?>
```

Temos agora como resultado “opção 2:”. O comando break fez com que os demais case's abaixo do 'case 2' não sejam executados.

Mas o que acontece se não tivermos um valor que seja satisfatório aos casos existentes no switch? A resposta é bem simples, nenhum dos blocos seria executados, porém temos um comando onde determinamos uma opção padrão caso nenhuma das outras venha ter resultado que satisfaça a expressão passada para o switch chamada default (padrão).

Veja um exemplo:

```

1 <?php
2 $numero = "opc 3";
3 switch ($numero){
4     case "opc 1":
5         echo "opção 1:";
6         break;
7     case "opc 2":
8         echo "opção 2:";
9         break;
10    default:
11        echo "opção inválida";
12    }
13 ?>
14

```

Resultado: opção inválida

A instrução passada não condiz com nenhum dos casos existentes. Por esse motivo o bloco

O comando default pode ser inserido em qualquer lugar dentro do switch, porém caso isso aconteça, o uso do comando break deve ser adicionado para evitar que os case's abaixo sejam executados.

pertencente ao comando default será executado.

A partir de agora trabalharemos as estruturas de repetição. Elas muito utilizadas nas linguagens de programação.

Exercício rápido:

- 1º) Faça um script em PHP usando switch, onde receba uma variável e mostre as seguintes opções: 1 - módulo.
2 - somar.
3 - subtrair.
4 - multiplicar.
5 - dividir.

6.5 While

O WHILE é uma estrutura de controle similar ao IF, onde possui uma condição para executar um bloco de comandos. A diferença primordial é que o WHILE estabelece um laço de repetição, ou seja, o bloco de comandos será executado repetitivamente enquanto a condição passada for verdadeira. Esse comando pode ser interpretado como “ENQUANTO (expressão) FAÇA { comandos...}”.

Sintaxe:

```

while (expressão) {
    comandos;
}

```

Quando estamos usando um laço de repetição, podemos determinar quantas vezes ele deve ou não se repetir. Isso pode ser feito de forma manual, onde o programador determina, ou automaticamente, onde quem vai determinar é fluxo de execução o código- fonte através de funções do PHP, funções estas já existentes. Por exemplo a função “sizeOf”.

Para trabalharmos com essa contagem de quantas vezes o laço deve se repetir, usaremos incremento ou decrecimento de uma variável conforme vimos no capítulo de operadores em PHP.

```

1 | <?php
2 | $a = 1; // iniciamos uma variável para contagem.
3 | while($a < 10){
4 |     print $a;
5 |     $a++; // incrementamos a variável.
6 |
7 | }
8 | ?>

```

Observe o exemplo abaixo:

Resultado: 123456789

Nesse exemplo criamos um laço de repetição que tem como condição `$a < 10`, a cada laço é executado um incremento na variável `$a`, fazendo com que o seu valor aumente até a condição não ser mais satisfatória.



Dicas: Tenha cuidado quando estiver trabalhando com loop's (laço de repetição), pois caso a expressão passada esteja errada, pode ocasionar em um loop infinito fazendo com que o bloco de código se repita infinitamente. Isso pode ocasionar um travamento do navegador ou até mesmo do próprio servidor WEB.

Vamos ver agora um exemplo em que o laço se repete de forma automática, onde quem determina o loop é uma função do PHP e não um numero determinado pelo programador.

A função `strlen()` recebe uma string e retorna a quantidade de caracteres incluindo também os espaços em branco. Ele poderia ser aplicado diretamente no `echo`, mas no exemplo, ele determina a quantidade de loop's.

```

1 | <?php
2 | $a = 1;
3 | $b = "Projeto ejovem";
4 | while($a < strlen($b)) // conta o tamanho da string.
5 |     $a++;
6 |
7 | echo "a frase possui ".$a." caracteres";
8 | ?>
9 |

```

Resultado: a frase possui 14 caracteres

Exercício rápido:

- 1º) Faça um script que conte de 1 até 100.
 2º) Faça um script que imprima na tela números de 3 em 3 iniciando com 0 até 90, ex: 0,3,6,9...

6.6 Do...while

O laço do...while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço do...while possui apenas uma sintaxe que é a seguinte:

Sintaxe:

```
do {
    comando;
    ...
    comando;
} while (expressão);
```

Exemplo:

```
1 <?php
2
3 $a = 1;
4 do{
5
6     echo $a;
7     $a++; // incrementa +1 a cada laço.
8
9 }while($a < 10)
10
11 ?>
12
```

Resultado: 123456789



Dicas: Talvez na criação de alguma página ou sistema web, seja necessário executar um bloco de código existente em um laço de repetição pelo menos uma vez, nesse caso podemos usar o do...while.

Exercício rápido:

- 1º) Faça um script que conte de -1 até -100 usando “do while”.
 2º) Faça um script que imprima na tela somente números pares de 2 até 20 com do while.

6.7 For

Outra estrutura semelhante ao while é o **for**, onde tem a finalidade de estabelecer um laço de repetição em um contador. Sua estrutura é controlada por um bloco de três comandos que estabelecem uma contagem, ou seja, o bloco de comandos será executado determinado número de vezes.

Sintaxe:

```
for( inicialização; condição; incremento ){
    comandos;
}
```

Parâmetros	Descrição
inicialização	Parte do for que é executado somente uma vez, usado para inicializar uma variável.
condição	Parte do for onde é declarada uma expressão booleana.
incremento	Parte do for que é executado a cada interação do laço.

Lembrando que o loop do **for** é executado enquanto a condição retornar expressão booleana verdadeira. Outro detalhe importante é que podemos executar o incremento a cada laço, onde possibilitamos adicionar uma variável ou mais. Mostraremos agora um exemplo fazendo um comparativo entre a estrutura de repetição do while e também a do **for** de forma prática.

Exemplo:

while

```
1 <?php
2
3     $a = 1;          //inicia a variável.
4     while($a < 10){ //condição.
5         echo $a;      //comando.
6         $a++;          //incremento.
7     }
8
9 ?>
```

for

```
1 <?php
2
3     //inicia a variável.
4     //condição
5     //incremento
6
7 for($a = 1 ; $a < 10 ; $a++){
8     echo $a;
9
10}
11
12 ?>
```

Ambos exemplo geram o mesmo resultado: 123456789

O **for** não precisa ter necessariamente todas as expressões na sua estrutura, com isso podemos

```
1 <?php
2
3     $a = 1;          //inicia a variável.
4     for( ; $a < 10 ; ){ //condição.
5         echo $a;      //comando.
6         $a++;          //incremento.
7     }
8
9 ?>
```

criar um exemplo de **for** onde suas expressões são declaradas externamente.

Observe nesse exemplo uma proximidade muito grande do comando while. Apesar de ser funcional, não é uma boa prática de programação utilizar desta forma.

Exercício rápido:

1º) Faça um script que receba duas variáveis \$a e \$b, logo após imprima os números de intervalos entre eles com o uso de “for”. ex: a=5 ,b = 11, imprime : 5,6,7,8,9,10,11.

6.8 Foreach

O **foreach** é um laço de repetição para interação em array's ou matrizes, o qual estudaremos com mais detalhes no próximo capítulo. Trata-se de um **for** mais simplificado que compõe um vetor ou matriz em cada um de seus elementos por meio de sua cláusula AS. **Exemplo:**

Sintaxe:

```
foreach( expressão_array as $valor) {
    comandos;
}
```

```
1 □ <?php
2
3 $nomes = array("ana","maria","joão","alex");
4 foreach($nomes as $nome){
5     echo $nome."<br>";
6 }
7
8 ?>
9
```

Resultado: ana maria João alex

Veremos adiante que um array é uma variável composta por vários elementos. No caso do exemplo anterior, esses elementos são nomes de pessoas. A finalidade do **foreach** é justamente a cada laço, pegar um desses valores e atribuir a uma variável \$nome até que tenha percorrido todo array e assim, finalizar o laço. Também podemos saber em qual posição o elemento se encontra no array, para isso basta adicionar uma nova variável logo após o AS seguido de =>.

Observe o exemplo:

```
1 □ <?php
2
3 $nomes = array("ana","maria","joão","alex");
4 foreach($nomes as $chave => $nome){
5     echo $chave." - ".$nome."<br>";
6 }
7
8 ?>
9
10
```

Resultado:

1. ana
2. maria 2-joão 3-alex

Nesse exemplo observamos que cada elemento do array possui um índice (chave), imprimindo na tela o numero da posição e o valor guardado.

6.9 Break

Outro comando importante é o break, usado para abortar (parar) qualquer execução de comandos como SWITCH, WHILE, FOR, FOREACH, ou qualquer outra estrutura de controle. Ao encontrar um break dentro de um desses laços, o interpretador PHP interrompe imediatamente a execução do laço, seguindo normalmente o fluxo do script.

Sintaxe:

```
while....
for....
break <quantidades de níveis>;
```

Vamos ver um exemplo com o uso de **break** dentro de um laço de repetição (no caso o **for**), onde criamos um laço infinito, porém colocamos um if com a condição de parar o laço através do **break**. Observe:

```
1 <?php
2
3 $a = 1;           // inicia a variável.
4 for( ; ; ){      // loop infinito.
5     echo "-".$a; // imprime na tela.
6     if($a == 10) // condição.
7         break;    // break, para o laço.
8     $a++;        // incremento
9 }
10
11 ?>
12
```

Podemos notar nesse exemplo a criação de um laço(loop) infinito, que ocorre quando tiramos a condição do **for**, ou atribuímos “**for(; true ;)**”, porém a condição fica na responsabilidade do if, quando o valor de \$a é igual a 10, faz com que o if execute o **break**,

fazendo com que o laço pare de funcionar.

Mas se tivéssemos mais de um laço, como poderíamos definir qual deles deixaria de funcionar? Para responder essa pergunta usamos a quantidades de níveis que pode existir em um break, observe o exemplo abaixo:

<pre>1 <?php 2 3 for(\$a = 1;\$a < 5;\$a++){ // primeiro laço 4 echo \$a." ° ----- laço
"; 5 6 for(\$b = 1;\$b < 5;\$b++){ // segundo laço 7 echo \$b." - segundo for
"; 8 if(\$a == 2) // condição 9 break 2; // break 10 11 } 12 13 ?></pre>	<i>Resultado:</i> 1 ° ---- laço 1 - segundo for 2 - segundo for 3 - segundo for 4 - segundo for 2 ° ---- laço 1 - segundo for
--	--

Observe que para definir qual nível podemos parar utilizamos o break, ou seja, o primeiro nível é onde o break está localizado, no exemplo citado temos dois níveis, e determinamos pelo “break 2;” que o segundo for(que é o de fora!) deixaria de funcionar.

6.10 Continue

A instrução **continue**, quando executada em um bloco de comandos for/while, ignora as instruções restantes até o fechamento em “}”. Dessa forma, o programa segue para a próxima verificação da condição de entrada do laço de repetição, funciona de maneira semelhante ao break, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Veja um exemplo:

```

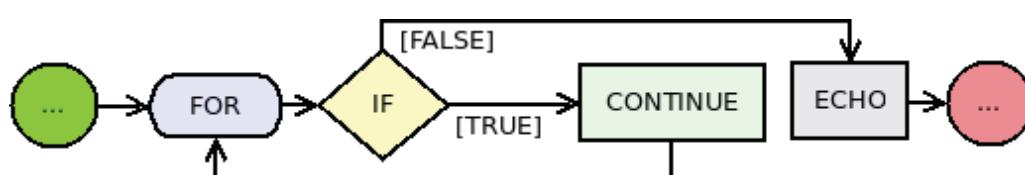
1  <?php
2
3  for ($i = 0; $i < 20; $i++) { //laço de repetição.
4      if ($i % 2) continue; //continue.
5      echo $i.",";
6  }
7
8  ?>
9

```

Resultado: 0,2,4,6,8,10,12,14,16,18,

Podemos observar a seguinte lógica no exemplo acima:

Criamos um laço que tem 20 interações de repetição. Logo após temos um if, onde, quando o resto da divisão por 2 for igual a 0 (numero par), o valor booleano será “false”. Quando não for igual a 0, significa que a variável \$i é um numero ímpar(ex: 5%2 = 1), então temos um valor booleano “true”. Isso significa que o if executa somente quando os números forem ímpares. Adicionamos um **continue**, que ao executar o if, faz com que volte novamente para o início do for, impedindo de alcançar o **echo** em seguida. Com isso, em vez de mostrarmos os números ímpares, imprimimos somente os números pares incluindo o 0. Resumimos que o código só passa adiante quando o if não executa o continue. Fluxograma:



Assim como o break, também podemos definir em qual nível queremos que a execução continue. Veja o exemplo abaixo:

```

1 <?php
2
3 $i = 0;
4 while ($i++ < 5) {           //laço de 5 interações, 3º nível.
5     echo "---primeiro<br>";
6     while (1) {               //laço infinito, 2º nível.
7         echo "-segundo<br>";
8         while (1) {           //laço infinito, 1º nível.
9             echo "-terceiro<br>";
10            continue 3;       //continua do 3º nível.
11        }
12        echo "Esta saída não é Alcançada.<br>";
13    }
14    echo "Esta saída não é Alcançada.<br>";
15 }
16
17 ?>
18

```

```

--primeiro
-segundo
-terceiro

```

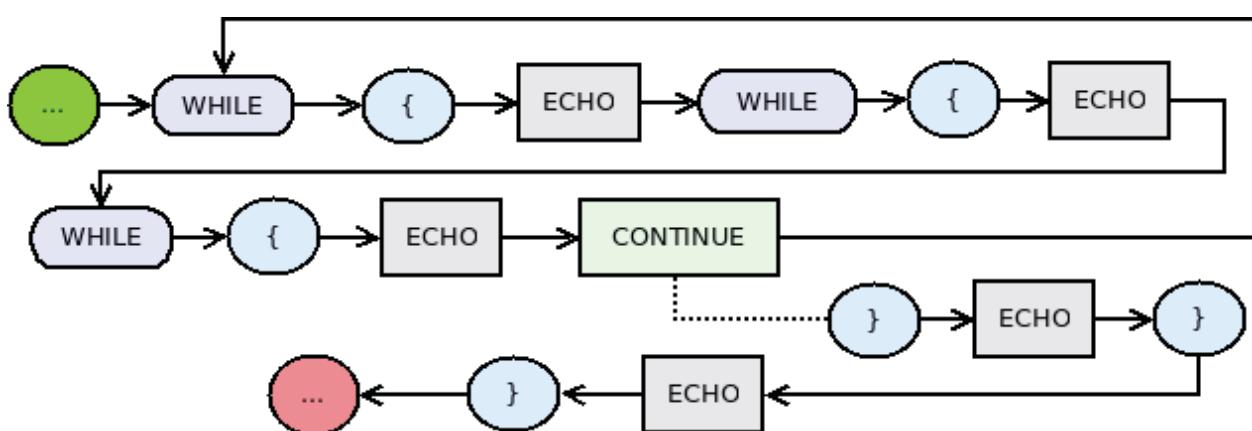
Resultado:

Podemos observar então o uso de **continue** dentro de um laço infinito. Ele faz com que o laço de nível 3 seja executado toda vez que a execução passe pela linha 10 do código, logo, impede que o programa fique sempre executando dentro do while de nível 1. Com isso, o while da linha 4 do código tem um ciclo de vida de 5 laços.

Observe também que os dois últimos echo's nunca serão alcançados, pois o comando continue impede que o fluxo do código passe adiante, fazendo voltar ao nível determinado.

Resumindo: O **continue** é usado dentro de estruturas de loops para saltar o resto da execução do loop atual e continuar a execução na avaliação do estado, em seguida, o início da próxima execução.

Fluxograma:



Exercício Resolvido com switch

Qual a mensagem que será gerada para o usuário no final desse código?

```
<?php

$texto = "Olá";
$op = 2;

switch($op){
    case 1:
        $texto .= ", crianças";
        break;
    case 2:
        $texto .= ", senhores";
    case 3:
        $texto .= ", senhoras";
        break;
    default:
        $texto .= ", idosos";
        break;
}

$texto .= " e Jovens de todo o Ceará";

echo $texto;

?>
```

Entendendo o algoritmo:

- No escopo temos uma variável com conteúdo de \$texto = “Olá”
- No escopo temos uma variável \$op = 2.
- O valor da variável \$op será avaliado no switch.
- O switch redireciona para o caso 2, pois o valor da variável \$op = 2.
- Uma operação de concatenação é realizada, a variável \$texto agora apresenta : “Olá, senhores”.
- Abaixo da concatenação não existe o comando break, podendo prosseguir com o caso 3 logo abaixo.
- Uma nova operação de concatenação é feita no caso 3 com a variável \$texto, tornando-se : “Olá, senores, senhoras”.
- Ao encontrar o break, a estrutura de seleção multipla será finalizada. realizando abaixo da estrutura uma ultima operação de concatenação : “Olá, senores, senhoras e Jovens de todo o Ceará”
- Por fim, temos a variável \$texto sento impressa com : “**Olá, senhores, senhoras e**

Jovens de todo o Ceará”

Exercício Resolvido com “do while”.

Existe algum erro nesse código? Se existe, onde estaria o erro, caso contrário qual mensagem

```
<?php

$vidas = 0;
$pontos = 0;

do{
    if($pontos == 0 ){
        $pontos++;
    }
}while($vidas>0);

echo "Você acumulou $pontos Pontos";
```

?>

será gerada para o usuário?

Entendendo o algoritmo:

- No escopo temos uma variável \$vidas = 0;
- No escopo temos uma variável \$pontos = 0;
- A estrutura do{ }while(\$vidas>0) realiza pelo menos 1 vez o bloco de código independente da condição.
- Na estrutura de condição if(\$pontos==0) podemos identificar um condição “true” pois o valor da variável pontos é igual a 0. Assim pontos será incrementado em 1.
- A estrutura de repetição “do while” é verificada, como o valor da variável \$vidas é igual a 1, essa estrutura irá retornar falsa encerrando a estrutura de repetição.
- Por fim, a impressão de um texto que será de :

Você acumulou 1 Pontos.

Obs.: Perceba que nesse algoritmo não diferencia a palavra “PONTOS” e “PONTO” de acordo com o número de vidas..

Pergunta: Como melhora riamos esse código para caso eu tenha apenas 1 vida, eu possa imprimir:

Você acumulou 1 Ponto.

EXERCÍCIOS PROPOSTOS

Qual a principal finalidade de uma estrutura de controle?

Qual a principal finalidade de uma estrutura de repetição?

Crie um código com a um condição ternária onde receba um valor booleano e de acordo com o valor passado na expressão, deve imprimir “sim” ou “não”.

Com o comando IF e ELSE crie um código que determine se uma expressão é verdadeira ou falsa.

Qual a finalidade da estrutura de controle SWITCH e cite um exemplo onde comparamos uma opção com 4 casos diferente?

Crie um contador de 1 até 20 usando a estrutura de repetição WHILE.

Crie um contador de 1 até 100 usando DO WHILE. **EP06.8:** Crie um contador de 100 até 1 usando FOR. **EP06.9:** Qual a finalidade de um FOREACH?

Crie um código onde podemos para a execução de um laço infinito com o uso de BREAK.

Como podemos determinar o uso de CONTINUE e qual a sua aplicação prática em PHP.

Crie um código com as seguintes características:

- a) Deverá receber um valor inicial e outro final (crie duas variáveis para esse fim).
- b) Como o comando FOR crie um laço onde a contagem é determinada pelo valor inicial e final?
- c) Dentro do for deverá conter um IF e ELSE responsável por comparar os valores passados a ele e imprimir os pares e ímpares. Exemplo:

IF(\$valor%2==0)

echo \$valor. “é um numero par”; ELSE

echo \$valor. “é um numero ímpar”;

- d) Exemplo prático: foi passado o numero inicial 8 e o final 15, então o script PHP deverá imprimir o intervalo entre esse numero ou seja 8,9,10,11,12,13,14,15, mostrando quais deles são pares e quais são ímpares.

7.0 Manipulação de arrays

Objetivos

Abordar de forma clara as principais estruturas de um array; Mostrar a sua criação e manipulações possíveis; Definir arrays multidimensionais ou matrizes; Determinar formas de interações e acessos.

Um array no PHP é atualmente um conjunto de valores ordenado. Podemos relacionar cada valor com uma chave, para indicar qual posição o valor está armazenado dentro do *array*. Ele é otimizado de várias maneiras, então podemos usá-lo como um array real, lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente muito mais. Além disso, o php nos oferece uma gama enorme de funções para manipulá-los.

A explicação dessas estruturas estão além do escopo dessa apostila, mas todo conteúdo aqui abordado trás uma boa base para quem estar iniciando o conteúdo de array.

7.1 Criando um Array

Arrays são acessados mediante uma posição, como um índice numérico. Para criar um array pode-se utilizar a função `array([chave =>] valor, ...)`. Exemplo:

Sintaxe:

```
$nomes = array('Maria', 'João', 'Alice', 'Alex');
```

```
$nomes = array(0=>'Maria', 1=>'João', 2=>'Alice', 3=>'Alex');
```

ou

Nessa sintaxe temos duas formas de declarar uma variável do tipo array. Onde a chave o índice podem ser de forma automática como no primeiro exemplo, ou manual como no segundo. Outro detalhe importante é que: todo array começa pela chave ou índice de numero 0, quando o mesmo não é declarado.

Também temos outras formas de criar um array, onde simplesmente podemos adicionar valores conforme a sintaxe abaixo:

```
$nome[] = 'Maria';
$nome[] = 'João';
$nome[] = 'Carlos';
$nome[] = 'José';
```

A figura abaixo representa um array que tem como valor representação de cores, e possui dez posições,

Cada posição representa uma cor, seu índice (chave) vai de 0 até 9. Veja:



Em código temos:

```
$cores = array('Verde','Azul','Violeta','Cinza','Vermelho','Amarelo',
               'Margenta','Branco','Laranja','Marrom');
```

7.2 Arrays Associativos.

Aos arrays associativos associa-se um determinado valor ou nome a um dos valores do array.

O array associativo usa strings como índice, onde cada string pode representar uma chave.

Observe a sintaxe:

```
$var = array('texto1'=>'valor1', 'texto2'=>'valor2', ..., 'textoN'=>'valorN');
```

Observe que quando usamos arrays associativos, a compreensão é mais fácil, dando mais legibilidade ao código. Porém não é utilizado quando usamos um array dentro de um laço

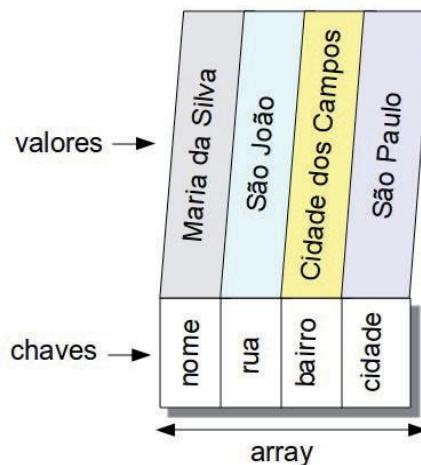
```
1 <?php
2 $dados = array('nome'=>'Maria Cristina', 'rua'=>'São João',
3                 'bairro'=>'Cidade dos Campos',
4                 'cidade'=>'São Paulo');
5 ?>
```

(loop), mas em outros casos sempre é bom utilizar arrays associativos. Veja um exemplo:

Outra forma de iniciarmos o mesmo array é adicionar valores conforme abaixo:

```
1 <?php
2 $dados['nome']='Maria Cristina';
3 $dados['rua']='São João';
4 $dados['bairro']='Cidade dos Campos';
5 $dados['cidade']='São Paulo';
6 ?>
```

A imagem abaixo representa os exemplos anteriores:



Umas das vantagens do array associativo é quando fazemos o acesso ao array, onde temos de forma clara e compreensível o valor que aquela chave pode conter. Como por exemplo nome, onde só vai existir o nome de pessoas. Veja abaixo um exemplo de acesso ao valores armazenados em um array dessa natureza.

Exemplo:

```

1 | 1<?php
2 | $dados['nome']='Maria Cristina';
3 | $dados['rua']='São João';
4 | $dados['bairro']='Cidade dos Campos';
5 | $dados['cidade']='São Paulo';
6 | echo $dados['nome']; # Resultado = Maria Cristina
7 | echo $dados['rua']; # Resultado = São João
8 | echo $dados['bairro']; # Resultado = Cidade dos Campos
9 | echo $dados['cidade']; # Resultado = São Paulo
10| ?>

```

Dessa forma podemos acessar o array. Basta determinar o nome do array e qual a chave, onde cada chave tem um valor já determinado. Resultará em um erro o uso de uma chave errada.

7.3 Interações

Quando falamos de interações em um array estamos dizendo o mesmo que percorrer esse array usando mecanismos da própria linguagem. Como isso as interações podem ser feitas de várias formas, mas no PHP podem ser iterados pelo operador FOREACH que já vimos anteriormente.

Exemplo:

```

1 <?php
2 $dados['nome']='Maria Cristina';
3 $dados['rua']='São João';
4 $dados['bairro']='Cidade dos Campos';
5 $dados['cidade']='São Paulo';
6 //loop que percorre todo array.
7 foreach($dados as $chave => $dados){
8     echo $chave." = ".$dados."<br>";
9 }
10 ?>

```

resultado:

```

nome = Maria Cristina
rua = São João
bairro = Cidade dos Campos
cidade = São Paulo

```

Esse tipo de interação é muito utilizado, principalmente quando temos arrays associativos.



Dicas: Sempre que se deparam com arrays, onde haja a necessidade de percorrer suas informações independentemente da chave, procure sempre utilizar mecanismos de programação mais simplificados como FOREACH.

7.4 Acessando um Array

Quando criamos um array temos que ter em mente que estamos criando uma variável que possui vários valores e que os mesmos podem ser acessados a qualquer momento. Cada valor está guardado em uma posição que pode ser acessada através de uma chave.

nome_do_array[chave_de_acesso];

A sintaxe para acesso simplificado de um array é a seguinte:

Temos que ter cuidado ao passar uma chave para o array, pois ela deve conter o mesmo nome de qualquer uma das chaves existentes no array. Caso a chave não exista, o valor não poderá ser resgatado. A sintaxe acima retorna um valor contido no array, por esse motivo temos que

```

1 <?php
2 //criamos um array com valores.
3 $meu_array = array('nome','telefone','rua','cidade');
4
5 //vamos acessar uma das posições.
6 $r = $meu_array[1]; //atribuindo valor resgatado a $r.
7 //imprimi na tela.
8 echo $r;
9
10 ?>

```

atribuir esse valor como mostra o exemplo abaixo:

Resultado: telefone.

7.5 Alterando um Array

Podemos alterar qualquer valor de um array. É muito semelhante ao acesso, onde, a diferença está na chamada do array. É nesse momento que atribuímos um novo valor.

Veja a sintaxe:

```
nome_do_array[ chave_de_acesso ] = <novo_valor>;
```

Observe o exemplo abaixo:

```
1 | 1<?php
2 | //criamos um array com valores.
3 | $meu_array = array('nome','telefone','rua','cidade');
4 | //alterando o valor de uma das posições.
5 | $meu_array[1] = 'sobrenome';
6 | //imprimi na tela.
7 | foreach($meu_array as $valores)
8 | echo $valores."<br>";
9 |
10| ?>
```

Resultados:

```
nome
sobrenome
rua
cidade
```

```
1 | 1<?php
2 | //criamos um array com valores.
3 | $produto['produto'] = 'Arroz';
4 | $produto['valor'] = 1.35;
5 | // alterando o valor do produto.
6 | $produto['valor'] += .63;
7 | // alterando o nome do produto.
8 | $produto['produto'] .= ' Tio João ';
9 | //imprimi valores na tela.
10| foreach($produto as $valores)
11| echo $valores."<br>";
12|
13| ?>
```

Vimos no exemplo anterior o valor da posição 1 do array ('telefone') foi alterada para sobrenome. Vale ressaltar que esse array tem suas chaves definidas de forma automática. A primeira posição é 0, a segunda é 1, e assim sucessivamente. Veja mais um exemplo onde

alteramos o valor, mas usando o operador de atribuição “`+=`” e concatenação “`.=`”:

Resultados: Arroz Tio João 1.98

Podemos observar que assim como as variáveis “comuns”, a forma de alterar o valor de um array é igual. A diferença está na chamada do array, pois temos que passar a chave além do valor que queremos atribuir. Outro detalhe importante é o tipo de valor, onde supostamente devemos atribuir os tipos compatíveis. Ou seja, se o valor atribuído a chave produto for do tipo string, não podemos usar os operadores de atribuição para atribuir um outro tipo, porém podemos mudar o tipo do valor pelo operador de atribuição simples (`=`).

Exemplo:

```
$var[2] += 1.90;           ff o tipo é um ponto flutuante antes e depois.
$var[2] = 'bom dia' ;      ff agora temos a mudança de tipo ponto
                           flutuante para string.
```

7.6 Arrays multidimensionais

Os arrays multidimensionais são estruturas de dados que armazenam os valores em mais de uma dimensão. Os arrays que vimos até agora armazenam valores em uma dimensão, por isso para acessar às posições utilizamos somente um índice ou chave. Os arrays de 2 dimensões salvam seus valores de alguma forma como em filas e colunas e por isso, necessitaremos de dois índices para acessar a cada uma de suas posições.

Em outras palavras, um array multidimensional é como um “contêiner” que guardará mais valores para cada posição, ou seja, como se os elementos do array fossem por sua vez outros arrays.

Outra ideia que temos é que matrizes são arrays nos quais algumas de suas posições podem conter outros arrays de forma recursiva. Um array multidimensional pode ser criado pela função `array()`:

Na figura abaixo temos a representação de um array com duas dimensões.

	0	1	2	3	4	Colunas
0	0.0	0.1	0.2	0.3	0.4	
1	1.0	1.1	1.2	1.3	1.4	
2	2.0	2.1	2.2	2.3	2.4	
3	3.0	3.1	3.2	3.3	3.4	
4	4.0	4.1	4.2	4.3	4.4	

Uma diferença importante de um array comum para um multidimensional é a quantidades de chaves (índices), onde cada um dos índices representa uma dimensão. Observe o código da representação ao lado.

Código:

```

1 | 1 <?php
2 |
3 | $num = array(array(0.0 , 0.1 , 0.2 , 0.3),
4 |               array(1.0 , 1.1 , 1.2 , 1.3),
5 |               array(2.0 , 2.1 , 2.2 , 2.3),
6 |               array(3.0 , 3.1 , 3.2 , 3.3));
7 |
8 | ?>

```

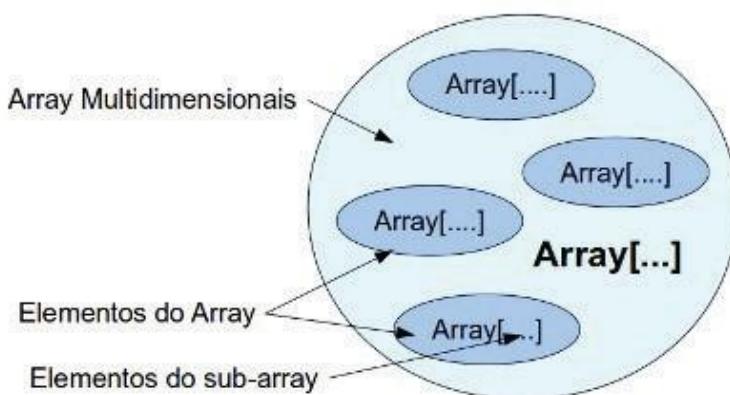
Outra forma de iniciar o array:

```

1 | 1 <?php
2 |
3 | $num[0][0] = 0.0; $num[0][1] = 0.1; $num[0][2] = 0.2; $num[0][3] = 0.3;
4 | $num[1][0] = 1.0; $num[1][1] = 1.1; $num[1][2] = 1.2; $num[1][3] = 1.3;
5 | $num[2][0] = 2.0; $num[2][1] = 2.1; $num[2][2] = 2.2; $num[2][3] = 2.3;
6 | $num[3][0] = 3.0; $num[3][1] = 3.1; $num[3][2] = 3.2; $num[3][3] = 3.3;
7 |
8 | ?>

```

Observe que temos uma chave para representar a linha e outra para representar a coluna, assim, determinando uma matriz 4x4. Podemos ver também que inicializamos um array dentro do outro. Cada sub-array é uma linha, e cada elemento do array maior representa as colunas.



Para acessarmos o valor de um array multidimensional, basta colocar as duas ou mais chaves da posição que queremos acessar. É muito semelhante ao array de uma única dimensão.

Observe o acesso aos exemplos anteriores: Sintaxe:

```
nome_do_array[ chave_1 ][ chave_2 ]... [chave_n] ;
```

Exemplo:

```

1 |  <?php
2 |
3 |  $num = array(array(0.0 , 0.1 , 0.2 , 0.3),
4 |                  array(1.0 , 1.1 , 1.2 , 1.3),
5 |                  array(2.0 , 2.1 , 2.2 , 2.3),
6 |                  array(3.0 , 3.1 , 3.2 , 3.3));
7 |  // acessando o array linha 1 coluna 2.
8 |  echo $num[1][2];
9 |  //resultado 1.2;
10|
11| ?>
```

Dessa forma podemos acessar o elemento numérico 1.2 que está guardado na posição linha 1 coluna 2, lembrando que o primeiro elemento de um array é 0. Abaixo, um exemplo que acessa todos os valores do array e imprime com quebra de linha:

```

1 |  <?php
2 |  $num = array(array(0.0 , 0.1 , 0.2 , 0.3),
3 |                  array(1.0 , 1.1 , 1.2 , 1.3),
4 |                  array(2.0 , 2.1 , 2.2 , 2.3),
5 |                  array(3.0 , 3.1 , 3.2 , 3.3));
6 |
7 |  foreach($num as $linha){
8 |      foreach($linha as $valores){
9 |          $i++;
10|         echo $valores." , ";
11|         if($i == 4){
12|             echo "<br>";
13|             $i = 0;
14|         }
15|     }
16| }
17| ?>
```

Resultado:

```

0 , 0.1 , 0.2 , 0.3 ,
1 , 1.1 , 1.2 , 1.3 ,
2 , 2.1 , 2.2 , 2.3 ,
3 , 3.1 , 3.2 , 3.3 ,
```

0.0 é igual a 0 , 1.0 é igual a 1 e assim sucessivamente.

Explicando o código:

Linha 2 – criamos um array de duas dimensões.

Linha 7 – temos um foreach para percorrer o primeiro array, ele retorna na variável \$linha os sub-arrays contido no array maior.

Linha 8 – agora temos outro foreach que vai percorrer os valores dos arrays passando pra

variável \$linha.

Linha 9 – criamos uma variável \$i para contar os elementos. Linha 10 – imprime os valores.

Linha 11 – temos um IF, quando \$i for igual a 4, significa que podemos executar o código pertencente ao If, determinando que chegou ao quarto elemento do array.

Linha 12 – quebra de linha com o
.

Linha 13 – zera a variável \$i para começar a contagem de novo.

7.7 Funções com Arrays

Em PHP temos um conjuntos de funcionalidades e que já vem prontas para serem utilizadas. Trata-se de funções que já estão pré-definidas, você pode encontrá-las facilmente no site php.net.

Abordaremos agora funções utilizadas exclusivamente para manipulação de array, funções de acesso, ordenação, dentre outras. Obviamente que não falaremos de todas, pois existem muitas funções, mas mostraremos as mais utilizadas, e outras que são definidas como principais.

var_dump

Essa função é muito usada por programadores que pretendem realizar debug (análise mais detalhado para encontrar supostos erros). Observe um exemplo prático:

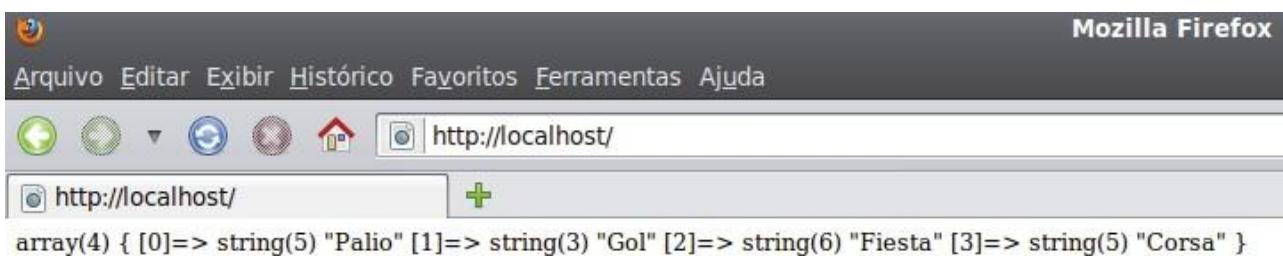
```

1 | <?php
2 |
3 | // declaração de um array
4 | $vetor = array('Palio','Gol','Fiesta','Corsa');
5 | //chamada da função var_dump passando $vetor
6 | var_dump($vetor);
7 |
8 | ?>
9 |

```

Saída no

Browser:



print_r

Imprime o conteúdo de uma variável assim como var_dump(), porém com um formato menos detalhado e mais legível ao programador. Exemplo:

código:

```

1  <?php
2
3  // declaração de um array
4  $vetor = array('Palio','Gol','Fiesta','Corsa');
5  //chamada da função print_r passando $vetor
6  print_r($vetor);
7
8  ?>
9

```

Saída no Browser:

Dicas: Ao olhar o código fonte da página aberta, nota-se o código bem organizado, porém os comentários não podem ser vistos. Procure olhar o código-fonte sempre que tiver dúvida de que código HTML o PHP está gerando. Clique com botão direito do mouse, procure código fonte, e observe o que é gerado!

[array_push](#)

Adiciona elementos ao final de um array. Tem o mesmo efeito de utilizar a sintaxe:

```
var_dump( nome_do_array , novo_valor)
```

Exemplo:

```

1  <?php
2  $var = array("valor-1","valor-2","valor-3");
3  array_push($var,"novo_valor");
4  var_dump($var);
5  ?>

```

Resultado:

```

array(4) {
  [0]=>
  string(7) "valor-1"
  [1]=>
  string(7) "valor-2"
  [2]=>
  string(7) "valor-3"
  [3]=>
  string(10) "novo_valor"
}

```

array_pop

Remove um valor no final de um array. Sintaxe:

array_pop (nome_do_array) ;

Exemplo:

```

1 <?php
2 $var = array("valor-1","valor-2","valor-3");
3 array_pop($var);
4 var_dump($var);
5 ?>

```

Resultado:

```

array(2) {
    [0]=>
    string(7) "valor-1"
    [1]=>
    string(7) "valor-2"
}

```

array_shift

Remove um elemento do início de um array, sintaxe:

array_shift(nome_do_array)

Exemplo:

```

1 <?php
2 $var = array("valor-1","valor-2","valor-3");
3 array_shift($var);
4 var_dump($var);
5 ?>

```

Resultado:

```

array(2) {
    [0]=>
    string(7) "valor-2"
    [1]=>
    string(7) "valor-3"
}

```

Como já sabemos como lidar com essas funções, observemos que basta conhecer a sintaxe para montarmos um exemplo. Apresentaremos agora de forma resumida as demais funções.

**Atenção!**

Todas as funções aqui apresentadas são para mostrar ao aluno as formas de trabalharmos com determinadas funções. Fica a critério do aluno se aprofundar ou não nesse conhecimento, uma vez que exista inúmeras funções.

Observe a tabela abaixo com outras funções:

Funções	Definição	Sintaxe:
array_unshift	Adicionar um elemento no inicio de um array.	array_unshift(nome_array , novo_valor)
array_pad	Preenche um array com valores, determina a quantidade de posições.	array_pad(nome_array, tamanho ,valor)
array_reverse	Recebe um array e retorna-o na ordem inversa.	array_reverse(nome_array, valor_booleano)
array_merge	Uni dois arrays criando um novo array.	\$novo_array = array_merge(array_1,array_2)
array_keys	Retorna somente as chaves do array.	array_keys(nome_array)
array_values	Cria um novo array com os valores de outro.	\$novo_array = array_values(outro_array)
array_slice	Extrai posições de um array.	\$var = array_slice(nome_array, inicio, tamanho)
count	Conta quantos elementos tem um array	\$var = count(nome_array)

Funções	Definição	Sintaxe:
in_array	Verifica se um array possui determinado valor.	in_array(valor_buscado, nome_array)
sort	Ordena um array pelos valores.	sort(nome_array)
rsort	Ordena um array pelos valores de ondem reversa.	rsort(nome_array)
explode	Converte uma string em um array.	explode(separador, nome_string)
implode	Converte um array em uma string	implode(separador, nome_array)

Essa tabela mostra as funções mais comuns. De repente você pode se deparar com algumas delas em códigos já feito, ou baixados da internet. Exemplo disso são as ferramentas CMS, como joomla ou wordpress, que são feitas em PH. Existem muitas combinações dessas funções e outras a mais em seu código-fonte. Veja alguns exemplos utilizando algumas das funções anteriores.

```

1  <?php
2   //Criamos dois arrays.
3   $array_1 = array("i","r","c","o","u","g","t","k","y","q","b","n");
4   $array_2 = array("f","m","l","v","j","x","z","a","p","h","d","s","e");
5   //Adicionar o elemento w no inicio do $array_1.
6   array_unshift($array_1,"w");
7   //Unimos dois arrays criando um novo array $alfabeto.
8   $alfabeto = array_merge($array_1,$array_2 );
9   sort($alfabeto); // ordena em ordem alfabética de a-z.
10  foreach($alfabeto as $letra){echo "-".$letra;} //imprime na tela
11  rsort($alfabeto); // ordena em ordem alfabética de z-a.
12  echo "<br>"; // quebra de linha.
13  foreach($alfabeto as $letra){echo "-".$letra;} //imprime na tela
14 ?>

```

Resultado: -a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-q-r-s-t-u-v-w-x-y-z
-z-y-x-w-v-u-t-s-r-q-p-o-n-m-l-k-j-i-h-g-f-e-d-c-b-a

Exercício resolvido com array.

Quais serão as letras que serão geradas a partir desse código?

```
<?php
```

```
$valor = 2;
$indice = $valor;

$letras = array('d','b','e','a','i','o','r');

echo $letras[$indice++]." ";
echo $letras[$indice]." ";
echo $letras[$indice-2]." ";
echo $letras[0]." ";
echo $letras[$valor+1]." ";
```

```
?>
```

Entendendo o algoritmo:

- No escopo temos uma variável chamada “valor” = 2 e “indice” que recebe o conteúdo que está em valor que é 2 também.
- Temos também um array que irá receber um conjunto descrito, lembrando que o indice inicial de um vetor é zero até a sua dimensão menos um. Nesse caso temos 7 elementos, então os indices para o meu vetor são de zero até 6.
- Para cada impressão, temos uma letra seguido de uma concatenação com um espaço em branco, então podemos chegar a conclusão que irá ser impresso um conjunto de letras com espaços.
- O primeiro “echo” irá imprimir o valor de “indice” = 2 que é representado pela letra “e”, em seguida indice será incrementado passando a ser igual a 3.
- O segundo “echo” irá imprimir o valor de indice = 3 que é representado pela letra “a”.
- O terceiro “echo” irá imprimir o valor de (indice - 2) = (3-2) = 1 que é representado pela letra “b”
- O quarto “echo” irá imprimir o indice zero do meu array representado pela letra “d”
- O quinto “echo” irá imprimir o indice (valor +1)= (2 + 1) = 3 representando pela letra “a”
- Por fim podemos identificar a mensagem no PHP : **e a b d a.**

EXERCÍCIOS PROPOSTOS

O que é um array, e qual a sua principal finalidade?

Declare um array chamado “nomes” com 8 posições, e grave nomes de pessoas que você conhece em cada uma delas. Após criar o array responda as questões 2, 4, 5, 10, 11:

Utilizado o array responda.

- Qual nome é impresso no navegador se colocarmos o código: echo nomes[3];
- Quais nomes aparecerá se adicionamos os seguintes códigos: for(\$i= 6; \$i>1 ; i--) echo nomes[\$i];
- O que acontece se chamarmos uma posição que não existe no array, exemplo: nomes[15];

O que é um array associativo, de exemplos:

Usando o comando *foreach*, crie uma interação onde todos os nomes possa ser impresso na tela.

Utilizando o mesmo código, altere alguns nomes do array.

O que é um array multidimensional?

Crie um array “palavras” multidimensional 5x3 com os valores da tabela abaixo, e responda as questões 7,8,9:

“oi”	“tudo”	“estar ”
“você ”	“vai”	“?”
“com”	“dia”	“!”
“,”	“bem”	“sim”
“casa”	“hoje”	“em”

Crie um código PHP onde com os valores do array possa ser impresso na tela com a frase “oi, tudo bem com você?”.

Utilizando as posições da sequencia [1][0],[1][1],[0][2],[4][2],[4][0],[4][1], [1][2] do array palavras, qual frase podemos formular?utilize a função print() para mostrar na tela do navegador.

Construa um código PHP para mostra uma resposta para a pergunta da questão 7. (use o comando echo para imprimir na tela).

Utilizando a função sort, imprima em ordem alfabética os nomes do array “nomes”.

Use o comando array_unshift() para adicionar mais dois nomes no array.

8.0 Manipulação de funções

Objetivos

Apresentar as estrutura de funções em PHP; Mostrar qual a definição entre variável global e estática e sua relação com o uso de funções; Mostrar também o uso de passagem de parâmetros, recursão e qual a sua principal finalidade.

Quando queremos um código funcional para determinado fim, com por exemplo fazer um cálculo ou alguma interação dentro do PHP, usamos o que chamamos de função. As funções são um pedaço de código com o objetivo específico, encapsulado sob um estrutura única que recebe um conjunto de parâmetros e retorna ou não um determinado dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes. É uma das estruturas mais básicas para prover reusabilidade ou aproveitamento de código, deixando as funcionalidades mais legíveis.

8.1 Declarando uma Função.

Declaramos uma função, com o uso do operador `function` seguido do nome que devemos obrigatoriamente atribuir, sem espaços em branco e iniciando sempre com uma letra. Temos na mesma linha de código a declaração ou não dos argumentos pelo par de parênteses “()”. Caso exista mais de um parâmetro, usamos vírgula(,) para fazer as separações. Logo após encapsulamos o código pertencente a função por meio das chaves ({}). No final, temos o retorno com o uso da cláusula `return` para retornar o resultado da função que pode ser um tipo inteiro, array, string, ponto flutuante etc. A declaração de um retorno não é obrigatório. Observe a sintaxe:

```
function nome_da_função( $argumento_1, $argumento_2, $argumento_n )
{
    comandos; return $valor;
}
```

Observe um exemplo onde criamos uma função para calcular o índice de massa corporal de uma pessoa (IMC), onde recebe como parâmetro dois argumentos. Um é a altura representada pela variável `$altura` e o outro é o peso representada pela variável `$peso`. Passamos como parâmetros para essa função o peso = 62 e a altura = 1.75. Observe:

Exemplo:

```
1  <?php
2  // declaração da função:
3  function calculo_IMC($peso, $altura){
4
5      return $peso/($altura*$altura);
6
7  }
8  // fazendo a chamada da função:
9  echo calculo_IMC( 62 , 1.75 );
10
11 ?>
```

Resultado: 20.244897959184

Nesse exemplo temos a declaração e logo após a chamada da função, onde é nesse momento que passamos os dois parâmetros na ordem que foi declarada na função.

Lembrando que essa ordem é obrigatória.

Observe mais um exemplo, onde a função declarada porém não possui a cláusula **return**.

```

1 | 1 <?php
2 | // declaração da função:
3 | function imprime($texto){
4 |
5 |     echo "<h1>$texto</h1>";
6 |
7 | }
8 | // fazendo a chamada da função:
9 | imprime("Testando a função!!");
10 |
11 | ?>

```

Resultado:

Testando a função

8.2 Escopo de Variáveis em Funções

Um conceito importante em programação são os tipos de declarações de variáveis, onde sua visibilidade vai depender de onde ela é declarada. O acesso a essas variáveis podem ser definidas da seguinte forma:

Variáveis locais < São aquelas declaradas dentro de uma função e não tem visibilidade fora dela. Veja um exemplo:

O valor da variável \$a não é impresso na tela, pois ela só existe dentro da função, qualquer outra variável declarada com o mesmo nome fora da função é uma nova variável.

```

1 | 1 <?php
2 | function Teste() {
3 |     $a = 25; // variável local
4 | }
5 | Teste(); /* ativa a função */
6 | echo $a; /* não imprime */
7 |
8 | ?>

```

Variáveis Globais < São variáveis declaradas fora do escopo de uma função, porém tem visibilidade (pode ser acessada) ao contexto de uma função sem passá-la como parâmetro. Para isso declaramos a variável e fazemos a sua chamada logo após com o uso do termo **global**.

Exemplo:

```

1 | <?php
2 | $a = 0;          /* escopo global */
3 | function Teste() {
4 |     global $a;   /* variável global */
5 |     $a = 25;      }
6 | Teste();         /* ativa a função */
7 | echo $a;        /* imprime 25 na tela*/
8 | ?>
9 |

```

Resultado: 25

Variáveis estáticas < Podemos armazenar variáveis de forma estática dentro de uma função. Significa que ao fazermos isso, temos o valor preservado independente da ultima execução. Usamos o operador static para declaramos a variável.

Exemplo:

```

1 | <?php
2 | function Teste() {
3 |     static $a; // variável statica
4 |     $a += 10; // atribuição +10
5 |     echo $a.",";
6 | }
7 | Teste(); // 1º chamada da função
8 | Teste(); // 2º chamada da função
9 | Teste(); // 3º chamada da função
10| ?>
11|

```

Resultado: 10,20,30,

Observe que o valor é mantido e a cada chamada é acrescentado +10, caso não exista o static o resultado seria: 10,10,10, .

8.3 Passagem de Parâmetro.

Como vimos anteriormente, podemos passar ou não parâmetros em uma função, porém existem dois tipos de passagem de parâmetros: Por valor (by value) e por referência (by reference).

Por Valor < Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original. Exemplo:

```

1 | <?php
2 | function valores($variavel , $valor) {
3 |     $variavel += $valor;
4 | }
5 | $a = 23;
6 | valores($a,26);
7 | print $a; #Resultado: 23
8 | ?>
9 |

```

O exemplo acima mostra que passamos um valor de \$a para a função, porém o valor temos a garantia que o valor continua íntegro, ou seja, não foi modificado ao longo do código.

Por Parâmetro < Para passarmos um valor por parâmetro, simplesmente colocamos o operador “&” na frente do parâmetro que queremos que o valor seja alterado, observe o

```

1 |<?php
2 |function valores( &$variavel , $valor) {
3 |    $variavel += $valor;
4 |
5 |    $a = 23;
6 |    valores($a,26);
7 |    print $a; #Resultado: 49
8 |
9 |?>

```

exemplo abaixo:

Observe agora nesse último exemplo que apenas acrescentamos o operador “&” no parâmetro que queríamos que alterasse a variável passada como parâmetro, fazendo com que o resultado fosse a soma de $23 + 26 = 49$.

Por argumentos variáveis < O PHP permite outras formas avançadas de passagem de parâmetros, onde o valor e a quantidade são definidas de forma automáticas por meio das funções **func_get_args()** e **func_num_args()**.

func_get_args() < diz os valores(argumentos) passado para a função.

func_num_args() < diz a quantidade de valores passados para a função.

Observe um exemplo mais complexo abaixo:

```

1 |<?php
2 |function Nomes() {
3 |    $argumentos = func_get_args();
4 |    $quantidade = func_num_args();
5 |
6 |    for($i=0;$i<$quantidade;$i++){
7 |        echo "nome = ".$argumentos[$i]." ,";
8 |    }
9 |
10}
11 Nomes("Alex","Sara","Maria","Bruna");
12 ?>
13

```

Resultado: nome = Alex , nome = Sara , nome = Maria , nome = Bruna ,

8.4 Valor de Retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum. Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou array's. As operações aritméticas podem ser feita de forma direta no retorno. Observe um exemplo onde

temos uma operação direta:

```

1 | 1 <?php
2 | 2
3 | 3 function Soma($n1, $n2) {
4 | 4     return $n1+$n2;
5 | 5 }
6 | 6 function Texto(){
7 | 7     return "O resultado é: ";
8 | 8 }
9 |
10| 10 echo Texto().Soma(115,24.5);
11|
12| 12 ?>
```

Resultado:

O resultado é: 139.5

Também podemos determinar mais de um retorno desde que eles não sejam acessados ao mesmo tempo, observe o exemplo abaixo:

```

1 | 1 <?php
2 | 2
3 | 3 function Teste($caso) {
4 | 4     switch($caso){
5 | 5         case 1:
6 | 6             return "opção 1"; break;
7 | 7         case 2:
8 | 8             return "opção 2"; break;
9 | 9         case 3:
10| 10            return "opção 3"; break;
11| 11        default:
12| 12            return null;
13| 13    }
14| 14 echo Teste(2);
15| 15 ?>
```

Esse código mostra de forma clara que não existe a possibilidade de retornarmos mais de um **return**, caso isso ocorresse, teríamos um erro, ou não funcionamento da função.

8.5 Recursão.

Função recursiva é uma definição usada tanto na programação quanto na matemática, onde, significa que uma função “faz a chamada” de si mesma na sua execução. Um exemplo é o cálculo do fatorial de um número. Observe:

Fatorial de 5: $5! = 5*4!$, $4! = 4*3!$, $3! = 3*2!$, $2! = 2*1!$ ou $5*4*3*2*1 = 120$.

Exemplo:

```

1  <?php
2  function Fatorial($numero) {
3      if($numero == 1) // Fatorial encerra quando o numero é igual a 1.
4          return $numero;
5      else
6          // Nesse retorno fazemos a chamada da função, passando o numero - 1.
7          return $numero * Fatorial($numero-1);
8      }
9      echo Fatorial(5);
10     ?>
```

Resultado: 120

EXERCÍCIO RESOLVIDO COM FUNÇÕES.

Realize o passo a passo de todas as operações matemáticas realizadas nas chamadas da função “calculo” junto com o resultado final.

```

<?php

function calculo($n1 , $n2 , $n3){
    return $n1 + $n2 *$n3;
}

echo calculo(5, calculo(1, 2, 3), 3);
```

Entendendo o algoritmo:

- Uma função chamada “calculo” foi definida com 3 parâmetros, “\$n1”, “\$n2”, “\$n3”
- A função “calculo” é responsável por retornar a multiplicação de “\$n2” e “\$n3” e com o resultado, realizar a soma com “\$n1” de acordo com a precedência matemática.
- Na sua chamada, temos 2 funções, a linguagem PHP começa realizando os cálculos das funções mais internas, sendo assim a primeira função que iremos analisar.
- A função calculo interna recebe 3 parâmetros.
 1. Numeral 1
 2. Numeral 2
 3. Numeral 3
- baseado na operação matemática temos : $1 + 2 \cdot 3 = 7$.
- Agora podemos resolver a função calculo mais externa, podemos encontrar mais 3 parâmetros
 1. Numeral 5
 2. Função que retorna um Numeral 7
 3. Numeral 3.

Baseado nos cálculos temos: $5 + 7 \cdot 3 = 26$.

O resultado que será impresso é : **26** baseado na lógica construída.

EXERCÍCIOS PROPOSTOS

Diga com suas palavras uma definição para função, e como podemos declará-la em PHP.

Qual a diferença de variáveis globais para variáveis locais e como podemos defini-las em PHP?

O que é um parâmetro, e quais os tipos de parâmetros em PHP?

Quais as funções que podemos usar para criarmos uma função onde seus parâmetros são passados pro argumentos variáveis?

O que é um valor de retorno e qual o comando usado quando queremos retornar algo dentro de uma função?

O que é recursão?

7º) Crie uma função que determine se um numero é par ou ímpar. E faça uma chamada dessa função imprimindo o resultado.

8º) Crie uma função que calcule a fatorial de um número.

9º) Crie uma função para determina se um numero é primo ou não. Número primo é aquele que possui dois divisores, 1 e ele mesmo. Criem um laço de repetição e use estrutura de controle.

9.0 Manipulação de arquivos e diretórios

Objetivos

Mostrar formas de manipulação de arquivos; Usar os principais comandos para trabalharmos com arquivo e diretórios; Aprender a trabalhar com leitura e escrita de arquivos, listagem, e criação de variáveis buffer de arquivos.

Assim como outras linguagens de programação, é muito importante trabalharmos com manipulações de arquivos e diretórios em PHP, onde temos a possibilidade de manipular um arquivo ou diretório dentro do servidor web, podendo criar arquivos responsáveis por guardar informações referentes aquele sistema ou página. Essas informações podem ser resgatadas futuramente, ou simplesmente são informações que ao invés de serem gravadas no bando de dados, foram gravadas em um arquivo ou log (arquivos que grava informações sobre o sistema, erros etc.).

Ao trabalhar com arquivos, no mínimo duas operações devem ser realizadas: abrir e fechar o arquivo.

9.1 Criando e Abrindo um Arquivo.

O comando utilizado para criar um arquivo é o mesmo que usamos para abri-lo, porém no Linux temos que dar permissões a pasta no qual o arquivo vai ser guardado.

Abra o console ou terminal do seu sistema(Linux). Digite:

```
chmod 777 fvarfwww
```

O comando chmod 777 dar todas as permissões possíveis na pasta www onde trabalharmos na criação de nossos arquivos.

Para abrir ou criar um arquivo utilizaremos o seguinte comando abaixo:

fopen

Com esse comando podemos abrir um arquivo e retornar um identificador. Sua sintaxe é a seguinte:

```
$identificador = fopen("string_do_arquivo","modo_do_arquivo");
```

string_do_arquivo

< é definido como o nome do arquivo mais a sua extensão, isso incluindo o caminho onde esse arquivo é localizado ou não, por exemplo:

“/home/aluno/meu_arquivo.txt”

Podemos observar um arquivo criado dentro da pasta alunos com o nome meu_arquivo.txt.

modo_do_arquivo

< **nesse parâmetro podemos determinar a forma que o arquivo vai ser aberto com os seguintes valores:**

“r”

< read, este modo abre o arquivo somente para leitura.

“w” < write, abre o arquivo somente para escrita, caso o arquivo não exista, tenta criá-lo. “a+” < append, abre o arquivo para leitura e escrita, caso o arquivo não exista, tenta * criá-lo.

Existem outros modos, mas trabalharemos somente com estes.

Dica!



Para trabalharmos com arquivos é sempre importante sabermos se a pasta ou o arquivo tem permissões dentro do Linux, caso isso não aconteça, o arquivo não será criado, lido ou até mesmo gravado.

Veja um exemplo do uso do comando fopen:

```
1  <?php
2
3  $a = fopen("meu_arquivo.txt", "w");
4
5  ?>
```

meu_arquivo.txt.

Caso o arquivo não exista, ele é criando dentro da pasta onde o arquivo .php foi criado, ou seja, no nosso exemplo o arquivo se chama index.php e estar dentro da pasta www, após executarmos esse comando teremos um novo arquivo com o nome

9.2 Gravando em um arquivo.

Após o uso do comando fopen, temos um identificador apontando para o arquivo, e com ele que podemos fazer alterações ou manipulações. Podemos gravar dados dentro do arquivo com o uso do seguinte comando:

fwrite

sintaxe:

```
fwrite("identificador", "conteúdo");
```

identificador < é o parâmetro retornado pelo comando fopen.

conteúdo < é o conteúdo a ser gravado no arquivo.

Vale ressaltar que para podermos gravar no arquivo ele deve ter permissão dentro do Linux e além disso ter como parâmetro “w” ou “a+” passado para o comando fopen.

Observe um exemplo onde escrevemos(gravamos) duas linhas dentro de um arquivo de texto criado com os comando visto até agora:

Exemplo:

```

1  <?php
2
3  $a = fopen("meu_arquivo.txt", "w");
4  $texto = "cidade: Fortaleza.\nrua: Pedro Pereira.";
5  fwrite($a, $texto);
6
7  ?>

```

Resultado:

O uso de “\n” antes da palavra rua faz com que ocorra uma quebra de linha escrevendo o resto do conteúdo na linha abaixo. Após a execução do script (colocando <http://localhost> no navegador e o nome do script criado), abrimos o arquivo de texto(meu_arquivo.txt) com um editor e percebemos o resultado final.

Observe mais um *exemplo*:

```

1  <?php
2
3  $a = fopen("meu_arquivo.txt", "w");
4  $textol = "Apostila de PHP.";
5  $texto2 = "Apostila de HTML.";
6  $texto3 = "Apostila de JAVA.";
7  fwrite($a, $textol."\n");
8  fwrite($a, $texto2."\n");
9  fwrite($a, $texto3."\n");
10
11  ?>

```

Resultado:

No exemplo, fizemos a chamada do comando **fwrite** três vezes e escrevemos a cada chamada um valor diferente concatenando com “\n”.

9.3 Fechando um arquivo.

Até agora trabalhamos com o comando fopen e não fechamos o arquivo, simplesmente abrimos e executamos os demais comandos. Isso faz com que, caso tenhamos de usar o mesmo arquivo em outra parte do código, ele não poderá ser utilizado, pois para isso é preciso fechá-lo para ele poder ser aberto novamente em outra parte do código. Para isso usamos o seguinte comando:

fclose

fclose("identificador");

sintaxe:

exemplo:

```

1  <?php
2  $a = fopen("meu_arquivo.txt","w");
3  fwrite($a,"escreva seu conteúdo aqui!!");
4  // fecha o arquivo
5  fclose($a);
6  ?>

```

Toda vez que abrimos um arquivo com fopen, devemos fechá-lo com o comando fclose conforme o exemplo ao lado.

9.4 Lendo um arquivo.

Após abrirmos um arquivo, outra operação que podemos efetuar é a leitura do conteúdo existente no arquivo. Essa operação é feita linha por linha, onde podemos resgatar valores existentes de acordo com a chamada do comando fread ou o índice do array criado pelo comando file.

file

Lê um arquivo e retorna um array com todo seu conteúdo, de modo que a cada posição do array representa uma linha do arquivo começando pelo índice 0.

```
$array = file("string_do_arquivo");
```

sintaxe:

string_do_arquivo < da mesma forma que é definida no comando fopen, usa-se o caminho com o nome do arquivo ou simplesmente o nome do arquivo caso ele exista na mesma pasta onde o arquivo PHP que contém o comando foi criado.

Exemplo:

```

1  <?php
2
3  $conteudo = file("meu_arquivo.txt");
4  echo $conteudo[0]."<br>";
5  echo $conteudo[1]."<br>";
6
7  ?>

```

Resultado:

Apostila de PHP
Apostila de HTML

Nesse exemplo utilizamos o arquivo anterior onde foi escrito três linhas, porém efetuamos a leitura somente da linha 1 (índice 0) e linha 2 (índice 1). Outra forma é percorrer o array usando um foreach(), dessa forma podemos ler todas as linhas existentes no arquivo, veja:

Exemplo:

```

1 | <?php
2 |
3 | $conteudo = file("meu_arquivo.txt");
4 | foreach($conteudo as $valor)
5 | echo $valor."<br>";
6 |
7 | ?>

```

Resultado:

Apostila de PHP
Apostila de HTML
Apostila de JAVA

9.5 Copiando, Renomeando e Apagando um Arquivo

Em PHP também é possível copiarmos um arquivo de uma origem para um determinado destino, como também apagar esse arquivo. Para isso usamos os seguintes comando:

copy

Cria um arquivo para outro local/nome. retornando um valor booleano verdadeiro(true) caso a copia tenha ocorrido sem erros ou falhas, caso contrário retorna falso(false). Sintaxe:

```
copy("string_origem","string_destino");
```

exemplo:

```

1 | <?php
2 | $origem = "meu_arquivo.txt";
3 | $destino = "meu_novo_arquivo.txt";
4 | $a = copy($origem,$destino); // copia o arquivo
5 | if($a)
6 |     echo "Cópia efetuada";
7 | else
8 |     echo "Erro ao copiar";
9 | ?>

```

Caso tudo ocorra corretamente, o resultado apresentado no navegador é “Cópia efetuada”, e será criado uma cópia dentro da pasta com o nome “meu_novo_arquivo.txt”. Vale lembrar que podemos também passar o caminho completo para onde deve ser copiado, como por exemplo:

fhomedafalunofmeu_novo_arquivo.txt

Para renomearmos um arquivo usamos:

rename Sintaxe:

```
rename("nome_do_arquivo", "novo_nome");
```

Para apagarmos um arquivo usamos:

unlink Sintaxe:

```
unlink("nome_do_arquivo");
```

Observe um exemplo, onde renomeamos o arquivo “meu_novo_arquivo.txt” para “arquivo_texto.txt” e apagamos o arquivo “meu_arquivo.txt”:

```

1  <?php
2      //renomeia o arquivo.
3      rename("meu_novo_arquivo.txt", "arquivo_texto.txt");
4      //apaga arquivo.
5      $a = unlink("meu_arquivo.txt");
6      if($a)
7          echo "arquivo apagado.";
8      else
9          echo "Erro ao apagar.";
10     ?>

```

Após executarmos isso no navegador, percebemos as mudanças ocorridas dentro do diretório.

9.6 Manipulando Diretório.

Alguns comandos básicos são necessários para manipulação de diretórios, mostraremos apenas como obter o diretório atual, como criar e apagar um diretório, para isso usamos os seguintes comandos:

mkdir

Cria um diretório de acordo com a localização e o modo. Sintaxe:

```
mkdir("string_localização", "int_modo");
```

string_localização<
nome.

é definido como o caminho com o nome do diretório, ou somente o **int_modo** < é onde definimos as permissões de acesso (como se fosse o chmod do Linux). Dessa forma podemos criar um diretório e já atribuirmos as permissões a ele.

getcwd

Retorna o diretório corrente, este comando é usado caso precise obter o diretório onde o arquivo PHP que possui este comando está guardado.

sintaxe:

```
getcwd();
```

rmdir

Apaga um diretório. Sintaxe:

```
1 | rmkdir("nome_diretório");
```

Observe o exemplo envolvendo os três comandos abaixo:

Exemplo:

```
1 | <?php
2 | // cria um diretório.
3 | $a = mkdir("minha_pasta",0777);
4 | if($a)
5 |     echo "pasta criada.";
6 | else
7 |     echo "erro!";
8 | //retorna o diretório onde a pasta foi criada.
9 | echo "<br>em ".getcwd()."<br>";
10 | //apaga o diretório.
11 | $a = rmdir("minha_pasta");
12 | if($a)
13 |     echo "pasta apagada.";
14 | else
15 |     echo "erro!";
16 |
17 |
18 | ?>
```

Resultado:

pasta criada.
em /var/www
pasta apagada.

Observe que o comando getcwd obtém o caminho completo de onde o arquivo PHP que contém o código-fonte estar guardado.

9.7 Interações com o Browser

PHP também permite interagir com informações do browser automaticamente. Isso pode ser muito útil quando queremos coletar informações sobre o cliente, como por exemplo, o tipo de browser (navegador), ou qual o sistema operacional, dentre outras informações.

O código a seguir mostra informações sobre o browser do usuário:

```
1 | <html>
2 |   <head>
3 |     <title>Apostila de PHP</title>
4 |   </head>
5 |   <body>
6 |     <?php echo $_SERVER['HTTP_USER_AGENT'] ?>
7 |   </body>
8 |
9 | </html>
```

Comando `$_SERVER[$HTTP_USER_AGENT]` tem como finalidade retornar informações do cliente que está acessando o arquivo PHP pelo browser, abaixo um exemplo desse retorno:

Mozilla/5.0 (X11; U; Linux i686; pt-BR; rv:1.9.2.10) Gecko/20100915 Ubuntu/10.04 (lucid) Firefox/3.6.10

Já vimos que o comando “HTTP_USER_AGENT” retorna uma string com várias informações, com isso podemos utilizar a função strpos(), que tem como finalidade procurar valores de uma string menor dentro de uma string maior. Podemos desta forma otimizar o nosso código. Sintaxe:

```
strpos( "string_maior" , "string_menor" );
```

Observe outro código com o uso da função strpos() :

```

2 |<html>
3 |<head>
4 |<title>Apostila de PHP</title>
5 |</head>
6 |<body>
7 |<?php
8 |$info = $_SERVER['HTTP_USER_AGENT'];
9 |if(strpos($info, "Firefox")!=0){
10 |    echo "Browser = Firefox";
11 |}else if(strpos($info, "Chrome")!=0){
12 |    echo "Browser = Chrome";
13 |}else{
14 |    echo "Desconhecido";
15 |}
16 |?>
17 |</body>
18 |</html>
```

Nesse exemplo procuramos as palavras “Firefox” e “Chrome” dentro do valor retornado pelo comando `$_SERVER["HTTP_USER_AGENT"]`. Dessa forma, podemos tratar o resultado comparando-o com “0”, ou seja, se a palavra existir, significa que seu valor é diferente de “0”. O resultado impresso na tela é de acordo com o navegador do cliente.

Exemplo de requisição HTTP_USER_AGENT:



9.10 Cookies

Cookies são mecanismos para armazenar e consultar informações. Eles são armazenados na máquina do cliente que acessa ao servidor php, e possui várias atribuições que são definidas pelo programador, por exemplo: imagine uma loja virtual, onde o cliente colocou em seu carrinho de compras vários produtos, mas por algum motivo ele não concluiu a compra, tendo que desligar a máquina que foi utilizada para fazer o acesso. No dia seguinte o cliente entra no mesmo site e percebe que todos os itens ainda estão no carrinho de compra do jeito que ele

deixou, esperando a conclusão da compra. Nesse exemplo, podemos perceber que as informações foram gravadas na máquina do cliente através dos cookies, que são simplesmente arquivos gerados pela página acessada dentro de alguma pasta do navegador que existe exclusivamente para esses arquivos.

O PHP atribui cookies utilizando a função **setcookie** que deve ser utilizada antes da tag <html> numa página. Além disso o uso de cookies não é recomendado quando se trata de informações sigilosas. Os dados dos cookies são armazenados no diretório de arquivos temporários do visitante, sendo facilmente visualizado por pessoas mal intencionadas.

Além da opção “aceitar cookies” que pode ser desativada a qualquer momento pelo visitante. Mas em cada navegador essa opção pode mudar de nome. Observe o comando abaixo:

setcookie

Sua sintaxe possui muitos parâmetros, abaixo está representada todos os valores que podem ser atribuído ao setcookie, mas vale ressaltar que não utilizaremos todos eles, somente os principais, veja sua sintaxe.

```
Setcookie("nome_do_cookie","seu_valor","tempo_de_vida","path",
"domínio","conexão_segura");
```

Onde na tabela abaixo temos a descrição de cada atributo:

Atributo	Descrição
nome_do_cookie	É o nome que, posteriormente, se tornará a variável e o que o servirá de referência para indicar o cookie.
seu_valor	É o valor que a variável possuirá. Esse valor pode ser de todos os tipos.
tempo_de_vida	É o tempo, em segundos, que o cookie existirá no computador do visitante. Uma vez excedido esse prazo o cookie se apaga de modo irrecuperável. Se esse argumento ficar vazio, o cookie se apagará quando o visitante fechar o browser.
path	Endereço da página que gerou o cookie – automático
domínio	Domínio ao qual pertence o cookie – automático
conexão_segura	Indica se o cookie deverá ser transmitido somente em uma conexão segura HTTPS.

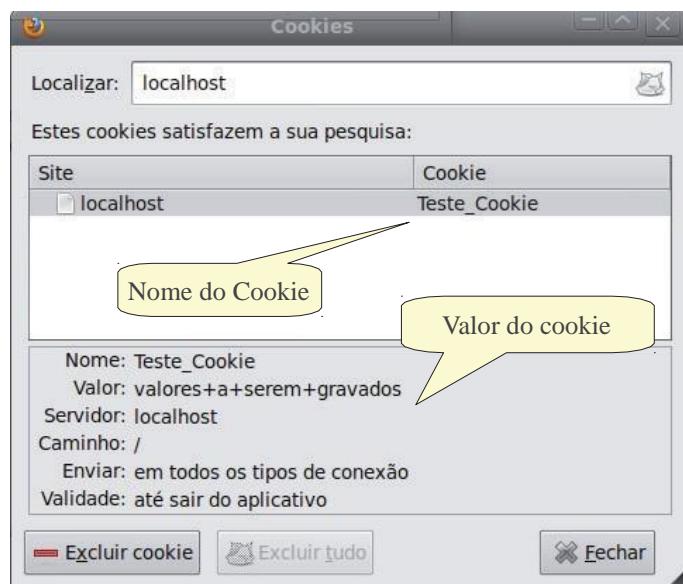
Observe um código onde criamos um cookie:

```
1 <?php
2
3 $valor = "valores a serem gravados";
4 setcookie ( "Teste_Cookie" , $valor );
5
6 ?>
```

Criamos então uma string, logo após a função setcookie recebendo como parâmetro somente o

seu nome e o valor a ser gravado.

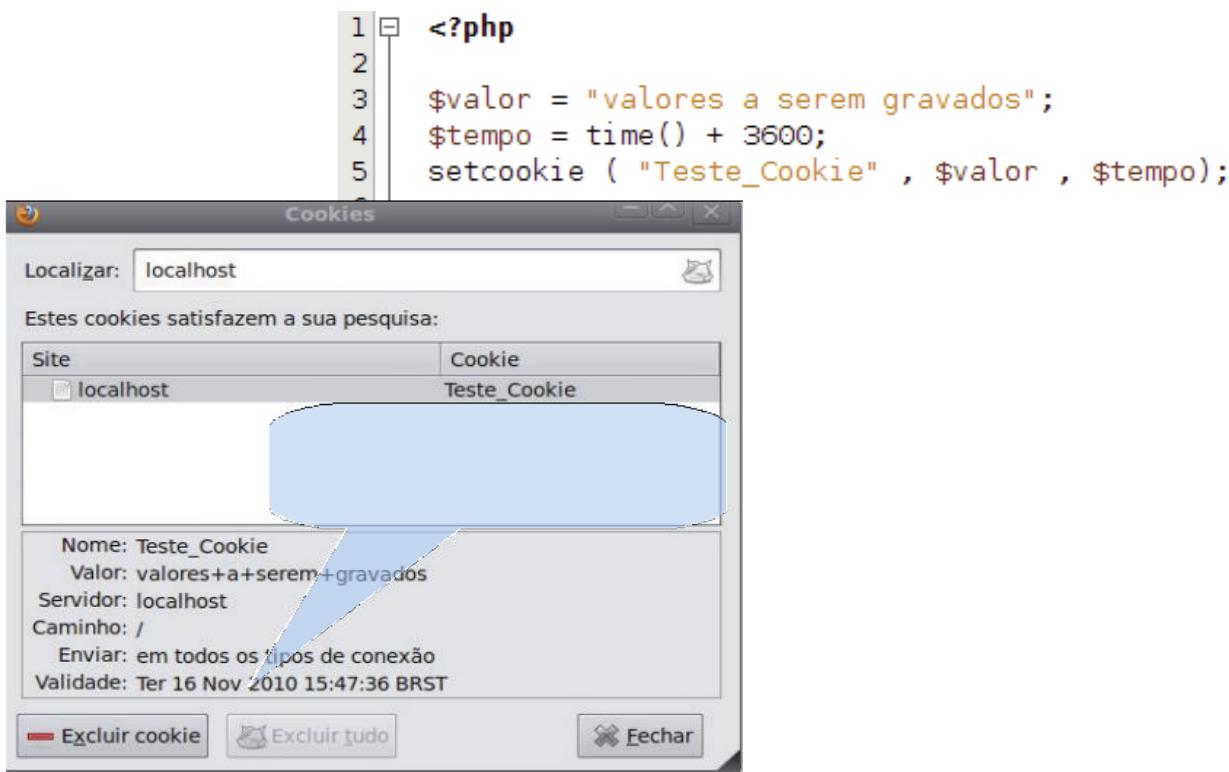
Usaremos o navegador Mozilla Firefox para visualizarmos o cookie criado, para isso basta digitar o endereço <http://localhost> na url, e logo após ir na opção: **Ferramentas < Propriedades da página < Segurança < Exibir cookie**. Lembre-se de criar o código acima primeiro e depois fazer a chamada pelo navegador de sua maquina. Se tudo ocorrer corretamente deverá aparecer a seguinte tela:



Veja que outras informações como caminho, enviar, e validade não foram especificados, porém podemos determiná-los na hora da criação do cookie dentro do código php.

Mostraremos agora um código onde atribuímos o tempo de vida do cookie, para isso devemos capturar o tempo com a função **time()** atual e somá-lo ao tempo que queremos em segundos, isso faz com que o cookie exista na máquina do cliente de acordo com a quantidade de tempo determinado pelo programador, observe um exemplo onde atribuirmos mais esse parâmetro o função **setcookie**:

Exemplo:



O novo resultado é o seguinte:

Data de vencimento do cookie, após ela ele é deletado automaticamente

Esse cookie tem a validade de 3600 segundos, ou seja 1 hora, com isso concluímos que o navegador fez seu acesso as 14:47:36. Isso é muito importante para a programação dos cookies. Se quisermos que ele exista por um determinado tempo, temos que calcular tudo em segundos da seguinte forma:

`$tempo = time()+(3600*24*7);`

Esse cookie tem seu tempo de vida de 7 dias, pois 3600 segundos = 1 hora, 24 horas = 1 dia e 7 horas_de_um_dia resulta em 7 dias.

Exemplo:

```

1 <?php
2     $valor = "valores a serem gravados";
3     $tempo = time() +(3600*24*7);
4     setcookie ( "Teste_Cookie" , $valor , $tempo);
5     ?>

```

Validade:

Validade: Sex 19 Nov 2010 18:55:29 BRST Validade de 7 dias a partir do dia 12.

ACESSANDO UM COOKIE:

Para acessarmos o valor gravado em um cookie é bem simples, basta utilizar o comando `$_COOKIE['coloque_aqui_o_nome_do_cookie']`, exemplo:

```

1  <?php
2  $valor = "valores a serem gravados";
3  $tempo = time()+(3600*24*7); // tempo de vida
4  setcookie ("Teste_Cookie" , $valor, $tempo); // grava o valor
5  echo $_COOKIE['Teste_Cookie']; //imprime
6  ?>

```

Resultado: *valores a serem gravados*

Observe agora um exemplo de um código utilizado para contar as visitas de um site usando cookie:

```

1  <?php
2  $valor=1;
3  $tempo = time()+(3600*24*7); // tempo de vida
4  $valor = $_COOKIE['contador']+1; // inicia a contagem
5  setcookie ("contador" , $valor, $tempo); // grava o valor
6  echo $_COOKIE['contador']." vistas no site!"; //imprime
7  ?>

```

O resultado é de acordo com a quantidade de vezes que o cliente entrou no site ou atualizou o mesmo.

9.11 Sessão

Sessões são mecanismos muito parecidos com os tradicionais cookies. Suas diferenças são que sessões são armazenadas no próprio servidor e não expiram a menos que o programador queira apagar a sessão.

As sessões são métodos de manter (ou preservar) determinados dados a mantê-los ativos enquanto o navegador do cliente (o internauta) estiver aberto, ou enquanto a sessão não expirar (por inatividade, ou porque em algum ponto você mandou que ela expirasse).

Para criarmos uma sessão utilizaremos a função abaixo:

```

1  <?php
2
3  session_start();
4
5  ?>

```

Dessa forma estamos iniciando um conjunto de regras. essa função deve sempre estar no início do código-fonte, com exceção de algumas regras.

Agora trabalharemos com essa sessão, primeiro podemos determinar o tempo de vida da sessão com o seguinte comando:

```

1 |  <?
2 |  session_cache_expire(5);
3 |  session_start();
4 | ?>

```

Neste caso, session_cache_expire vem antes de session start. Porque primeiro ele avisa que a sessão, quando iniciada, deve expirar em 5 minutos, e depois a inicia.

```

1 |  <?
2 |  session_start();
3 |  $var = "Sessão Criada!";
4 |  $_SESSION['minha_sessao'] = $var;
5 | ?>

```

Com o comando \$_SESSION podemos gravar valores na sessão, veja um exemplo:

Criamos uma sessão com o nome minha_sessao (não é uma boa prática de programação usar acentos em nomes de variáveis ou qualquer outra nomeação) e atribuímos a ela o valor gravado na variável string \$var. Essas informações ficam gravadas no servidor, logo após podemos resgatar o valor da seguinte forma:

```

1 |  <?php
2 |  session_start();
3 |  echo $_SESSION['minha_sessao'];
4 | ?>

```

Observe que o comando \$_SESSION tem seu tratamento igual a uma variável do tipo Array. Para resgatar o valor da sessão, basta fazer a chamada do comando passando o nome da sessão, no caso “minha_sessao”. O exemplo anterior foi adicionado em um outro arquivo, por esse motivo temos que chamar novamente o comando session_start(), para trazermos ao arquivo todas as regras usadas em sessão no PHP.

Abaixo temos um exemplo com o uso da função **isset()**, que verifica se uma variável existe ou não, retornando um valor booleano(true ou false):

```

session_start();
if(isset($_SESSION['minha_sessao'])){
    echo "Sessão Ativa";
} else{
    echo "Sessão não existe!";
}

```

O resultado é de acordo com a existência ou não da sessão.

Para desativarmos uma sessão podemos utilizar dois tipos de mecanismos: um deles é o uso da função **session_destroy()** que tem como finalidade destruir todas as sessões criadas pelo usuário, a outra forma é desalocarmos a sessão criada com o uso da função **unset()**.

Uso de **session_destroy()**:

```

1 |  <?php
2 |  session_start();
3 |  session_destroy();
4 | ?>
5 |

```

Uso de **unset()**:

Usamos unset() quando queremos desalocar uma determinada sessão, imaginamos que o usuário ao acessar uma determinada página, tenha criado várias sessões com nomes diferentes. Os nomes das sessões são determinados pelo programador, porém ao clicar em um link, o mesmo tem que destruir a seção escolhida. O exemplo abaixo destrói a sessão especificada:

```

1 | 1<?php
2 | session_start();
3 | // Destrói a sessão especificada.
4 | unset($_SESSION['minha_sessao']);
5 | ?>

```

Dessa forma desalocamos (destruirmos) a sessão “minha_sessao”, porém se existirem outras, elas ainda continuarão ativas.

9.12 Requisição de Arquivos

Assim como em muitas outras linguagens de programação também é possível incluir dentro de um script PHP outros arquivos contendo outras definições, constantes, configurações, ou até mesmo carregar um arquivo contendo a definição de uma classe. Para isso podemos usar os seguintes comandos:

include<arquivo>:

A instrução include() inclui e avalia o arquivo informado. O código existente no arquivo entram no escopo do programa que foi inserido, tornando-se disponível a partir da linha em que a inclusão ocorre. Se o arquivo não existir, produzirá uma mensagem de advertência (warning).

Exemplo onde temos dois arquivos:

código do, arquivo teste.php

```

1 | 1<?php
2 |
3 | echo "<h1>Bem vindo ao Site</h1>";
4 |
5 | ?>
6 |

```

código do arquivo index.php

```

1 | 1<?php
2 |
3 | include 'arquivo_teste.php';
4 | echo "Bom dia";
5 |
6 | ?>

```

Resultado:

Bem vindo ao Site

Bom dia

Nesse exemplo podemos notar que o código existente no “arquivo_teste.php” foi inserido dentro do arquivo index.php, tendo como resultado a execução dos dois códigos como se fossem apenas um, esse recurso é muito utilizado, pois podemos incluir até mesmo códigos de páginas inteiras em um arquivo.

require<arquivo>:

Comando muito parecido ao **include**. Difere somente na manipulação de erros. Enquanto o **include** produz uma warning, o **require** uma mensagem de Fatal Error caso o arquivo não exista.

Sintaxe: **require 'nome_do_arquivo.php';**

include once<arquivo>:

Tem funcionalidade semelhante ao **include**, a diferença é que caso o arquivo informado já esteja incluído, esse comando não refaz a operação, ou seja, o arquivo é incluído apenas uma vez. Este comando é útil para garantir que o arquivo foi carregado apenas uma vez. Caso o programa passe mais de uma vez pela mesma instrução, evitará sobreposição de arquivo.

Sintaxe: **include_once 'nome_do_arquivo.php'; require once<arquivo>;**

Tem funcionalidade parecida com o comando **require**. A diferença é justamente caso o arquivo já tenha sido incluído no programa, pois ele não carrega novamente o código. É

muito semelhante ao **include_once**, evitando redeclarações ou sobreposições, porém a mensagem exibida caso o arquivo não exista é de Fatal Error.

Sintaxe: **require_once 'nome_do_arquivo.php';**

9.13 Tratamentos de erro

São muito comuns erros na programação PHP, que podem partir do programador como pessoa física, do servidor, ou outros fatores envolvidos que juntos venham ocasionar em um erro. Existem quatro tipos de erros no PHP para indicar a gravidade do erro encontrado ou ocorrido. Eles são:

1. Erros de funções (function errors).
2. Avisos (warnings).
3. Erros de processamento (parser error).
4. Observações (notice).

Os programadores devem prestar muita atenção nas mensagens de erro, afinal nenhum programador quer por no ar um sistema que quando o primeiro visitante entra apareça uma mensagem de erro. Para evitar essas inconveniências use sempre um “@” antes de cada chamada as funções. Se a opção **track_errors** no arquivo **php.ini** estiver habilitada, a mensagem de erro poderá ser encontrada na variável global **\$php_errormsg**. Para exibir a mensagem direta no navegador procure ativar a opção **display_errors**, as funções serão ativadas para **On**(ligada) ou **Off** (desligada).

O arquivo **php.ini** no linux geralmente fica na pasta: “**/etc/php5/apache2/php.ini**”. Veja o

Exemplo:

```
1 |  <?
2 |  strtolower();
3 |  ?>
```

Imprimindo a mensagem de erro:

```

1 |  <?
2 |  strtolower();
3 |  echo "<h1>".$php_errormsg."</h1>";
4 | ?>

```

Resultado:

strtolower() expects exactly 1 parameter, 0 given

Mensagem de erro com a opção **display_errors = On** do php.ini ativada:

Warning: strtolower() expects exactly 1 parameter, 0 given in /var/www/index.php on line 2.

O erro mostra que devemos passar um parâmetro para a função strolower() na linha 2, mas podemos ignorar o erro usando o “@”.

```

1 |  <?
2 |  @strtolower();
3 | ?>

```

Essa função deixaria todos os caracteres em minúsculo, mas seu erro foi ignorado, não exibindo nenhuma mensagem de erro.

Lembrando: podemos utilizar tag's curtas no PHP “<? ?>” em vez de tags normais “<?php ?>”, mas isso vai depender da configuração do PHP (php.ini), em alguns casos não funciona.

Um vendedor de cestas básicas precisa de um sistema ao qual ele possa digitar quantas cestas básicas ele vendeu e qual o salário final que ele irá ter:

Um vendedor tem um salário fixo de R\$ 500,00 mais um bônus de 10% do valor de cada cesta básica vendida . Sabe-se também que cada cesta básica custa R\$30,00.

Crie código PHP que irá calcular e mostrar na página qual será o salário do mês.

OBS: Imagine que você já tem um formulário que têm um campo chamado : **qtdCestasV** que irá enviar essa informação via **POST** para uma “salarioVendedor.php”.

Crie um código PHP para solucionar o problema do vendedor.

```

<?php
    // variável que irá receber a informação do usuário
    $qtdCestasV = $_POST['qtdCestasV'];
    //Variáveis que tratam de valores da cesta e comissão
    $valorCesta = 30;
    $comissao = 0.1;

    //Calculo matemático para obter salário
    $salario = 500 + ($qtdCestasV*$valorCesta)*$comissao;

    //Imprimindo salário.
    echo "Salário Final = R$ ".$salario;
?>

```

EXERCÍCIOS PROPOSTOS

O que é manipulação de arquivos? **EP09.1:** Observe o código-fonte abaixo:

```

1      <?php
2          $arquivo = fopen("all.txt","w");
3          fwrite($arquivo, "oi tudo bem!");
4          fclose($arquivo); 5 ?>

```

- Que tipo de arquivo é gerado na linha 2 e aonde o mesmo é criado?
- Que o parâmetro “w” da linha 2 do código representa?
- Qual a finalidade do comando *fwrite* da linha 3 do código?
- Qual o principal motivo de fecharmos o arquivo com o comando *fclose* da linha 4 do código? *Crie um arquivo de texto chamado “frases.txt” usando o comando fopen, e responda as questões 3,4,5,6,7*

Grave uma mensagem dentro do arquivo criado.

Com base no arquivo criado, utilize o comando *fwrite* para ler o mesmo imprimindo na tela do navegador o conteúdo do arquivo.

Abra o arquivo “frases.txt” com um editor de texto, adicione cinco palavras, cada uma em uma linha diferente, após isso utilize o comando *file*, para efetuar a leitura do arquivo, e imprima na tela a primeira e ultima palavras com o comando *echo*.

Crie uma cópia do arquivo renomeando o novo arquivo para “palavras.txt”.

Crie um diretório com o comando *mkdir* e copie o arquivo “palavras.txt” para a pasta criada e apague o anterior, tudo com comandos PHP.

Crie um código que imprima na tela todo o caminho de pastas onde se localiza o arquivo “palavras.txt”.

Crie um formulário HTML com os seguintes campos: “nome”, “endereço”, “e-mail”, “senha”, e o botão “enviar”.

Utilize o método Get para visualizar os dados do array na URL do navegador ao clicar no botão enviar.

Qual a diferença do método POST e GET?

Como podemos receber dados via método GET? Exemplifique.

Como podemos receber dados via método POST? Exemplifique.

Crie um arquivo chamado dados.php, nesse arquivo deverá conter os seguintes requisitos:

-Os dados do formulário da questão 1 deverá ser enviado para esse arquivo via metodo POST.

-O arquivo dados.php deverá conter cinco variáveis, cada uma para determinado campo, exemplo:

```
$nome = $_POST['nome'];
```

-Os valores deverá ser impresso na tela.

Qual a finalidade do comando \$_SERVER ["HTTP_USER_AGENT"]?

Crie um cookie gravando nele o seu nome, logo após abra o Firefox e Exiba o valor gravado.

Crie um arquivo chamado “criar_sessao.php”, utilize comando PHP para cria uma sessão com a durabilidade de 3 minutos e adicione o valor “sessão ativa”.

Crie um novo arquivo chamado “ler_sessao.php” para verificar se a sessão criada na questão 9 existe ou não.

Utilize o comando \$_SESSION para ler o valor nela contido e imprima na tela.

Quais os possíveis erros existentes em PHP e qual a definição de cada um deles?

1.0 Introdução a Banco de Dados



Qualquer empresa necessita armazenar os dados relacionados ao seu negócio.

Por exemplo, uma livraria deve manter as informações dos livros que são comercializados por ela.

Um banco precisa registrar os dados dos seus clientes. Uma escola deve guardar as informações dos seus alunos.

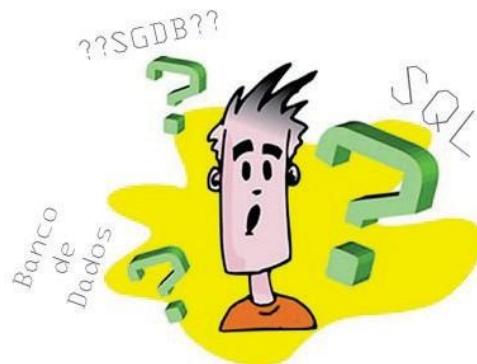
Atualmente, utilizar papel para registrar os dados de uma empresa não é uma boa alternativa. O espaço físico necessário gera custos altos para empresa. Em geral, a consulta das informações registradas é demorada. O risco de um acidente destruir os dados armazenados em papel é alto. Em vários aspectos, utilizar computadores para o armazenamento de dados é uma abordagem melhor do que usar papel.

Desta forma surgiu uma nova área ou vertente na Tecnologia da Informação chamada Banco de Dados.

1.1 Banco de dados? ou SGBD/SGDB? Ou SQL?

É comum estudantes iniciantes de banco de dados e até desenvolvedores não saberem diferenciar os conceitos de **Banco de dados** e **SGBD** ou até mesmo o que é **SGBD** e o que é **SQL**.

Tantos termos novos. Você deve estar confuso.

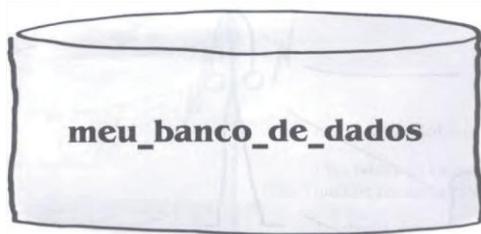


Não se preocupe vamos lhe apresentar e explicar as definições de cada termo (Banco de dados, SGBD e SQL) e suas diferenças.

1.2 Entendendo o que é Banco de Dados ou Base de Dados.

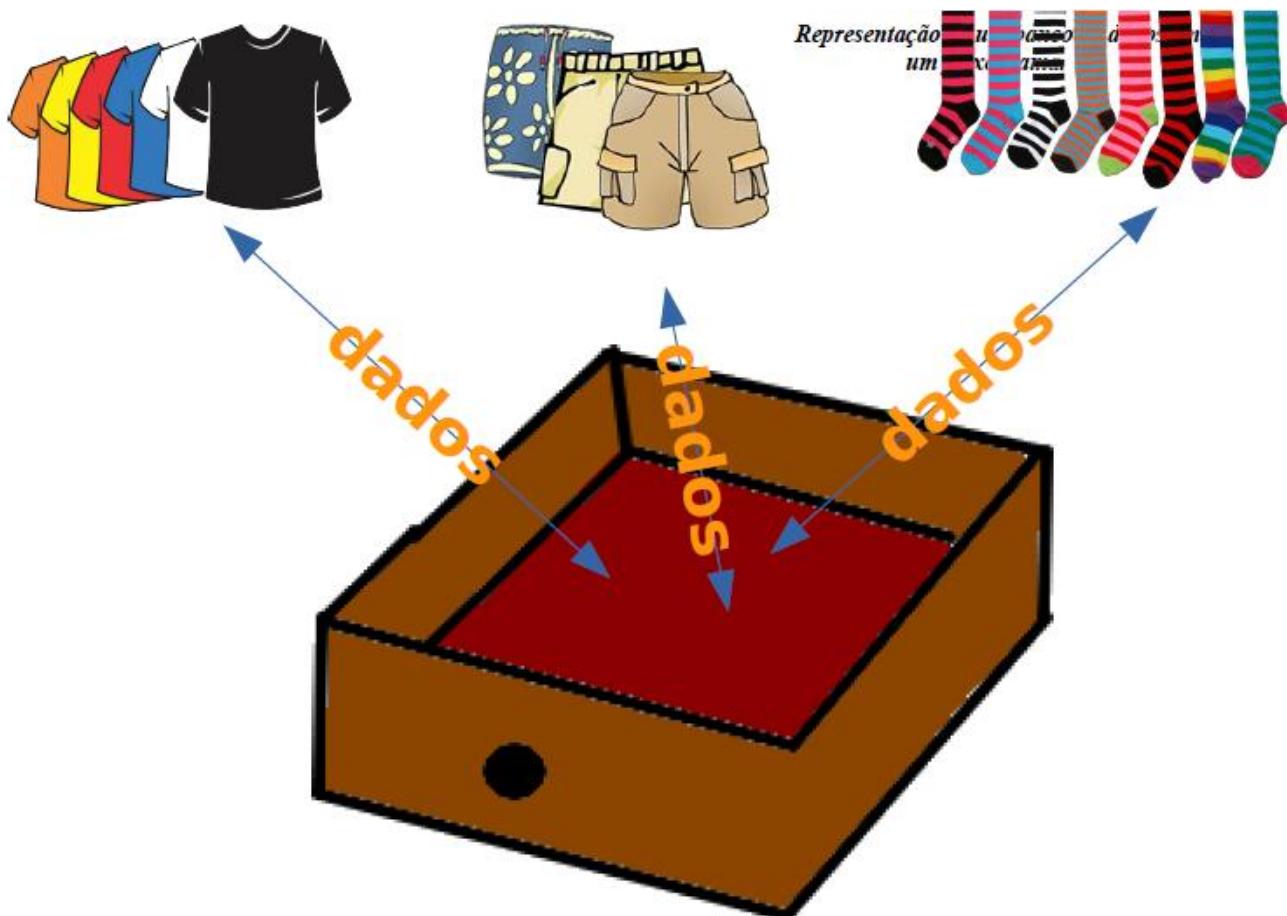
Um banco de dados ou base de dados (sua abreviatura é BD, em inglês DB, database) é uma **entidade** na qual é possível armazenar dados de maneira **estruturada** e com a menor redundância possível e esses dados estão inter-relacionados. Estes dados devem poder ser utilizadas por programas e por usuários.

Portanto, banco de dados é um repositório/contêiner onde os dados são armazenados para serem usados por algum software para um fim específico.



Ainda não entendeu? Deixa eu te explicar melhor.

Na sua casa você deve ter um guarda roupa e nesse uma gaveta para roupas. Vamos supor que nessa gaveta você guarda suas camisetas, bermudas e meias.



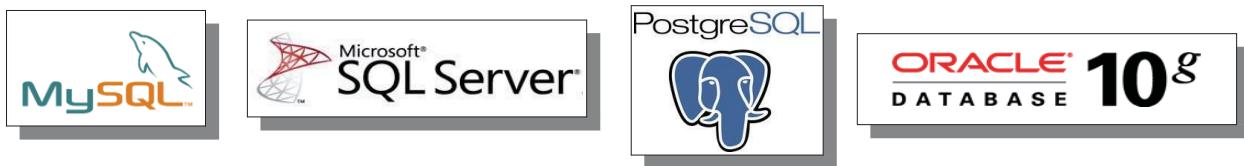
Fazendo uma analogia com a definição de banco de dados vista acima, os dados são suas roupas e a gaveta é o **repositório** ou **contêiner** que guarda os dados, ou seja, a gaveta seria o seu: **BANCO DE DADOS!**

Agora você deve ter entendido. De uma maneira bem resumida o Banco de dados seria o local onde são depositados os dados.

1.3 Sistema Gerenciador de Banco de Dados (SGBD)

Sistemas Gerenciadores de Banco de Dados(SGBD) é o conjunto de programas de computador (softwares) responsáveis pelo gerenciamento de uma base de dados, ou seja são softwares usados para administrar os dados armazenados.

Eis uma lista dos SGBDs mais utilizados nas empresas:



1.4 Banco de dados X SGBD

No dia a dia os termos se misturam: Banco de Dados e SGBD, mas são coisas diferentes.

Banco de dados é aquilo que os softwares ou gerenciadores de banco de dados produzem, ou seja, são os dados em si organizados agora em um ou mais arquivos que podem ser lidos e manipulados pelos SGBD.

Como definimos acima, o SGBD é um conjunto de software que usamos para manipular(inserir, consultar, alterar e deletar) dados de nossa base de dados.

Uma analogia ao Calc: O Calc é um software do grupo de planilhas eletrônicas, cujo produto ou resultado é um arquivo .ODS. Sendo assim, o .ODS seria a base de dados e o CALC o SGBD.

Lembre-se que muitos usam o termo **Banco de Dados** se referindo ao **SGBD**.

Por exemplo, muitos anúncios de vagas de estágios e empregos na área de BD colocam como requisito para a vaga que o candidato possua conhecimento do Banco de Dados Oracle, Banco de Dados Mysql, Banco de Dados PostgreSql.

Sendo que o mais apropriado seria que o candidato possua conhecimento do SGBD Oracle, SGBD Mysql, SGBD PostgreSql.

1.5 Linguagem SQL - Structure Query Language

Você já aprendeu o que é um banco de dados(BD), que é o contêiner onde são guardado as informações, também aprendeu que existe o SGBD que é um pacote de programas que manipulam os dados que estão dentro do BD, além disso

Você deve estar se perguntando:

- onde entra o SQL nessa história?!



Aprendeu que existem vários tipos de SGBD desenvolvidos por empresas diferentes como por exemplo o SGBD Oracle, SGBD Mysql, SGBD SqlServer.

A linguagem **SQL** – Structure Query Language(*Linguagem de Consulta Estruturada*) é a linguagem que usamos para controlar ou interagir com o SGBD, que por sua vez acessa e manipulam os dados que estão no Banco de Dados.

A linguagem SQL é extremamente estratégica e pode ser usada com algumas adaptações em qualquer SGBD.

Desenvolvida nos anos 70 nos laboratórios da IBM em San Jose, dentro do projeto System R, que tinha por objetivo demonstrar a viabilidade da implementação do modelo relacional proposto por E.

F. Codd. O nome original da linguagem era SEQUEL, acrônimo para "Structured English Query Language" (Linguagem de Consulta Estruturada, em Inglês), vindo daí o facto de, até hoje, a sigla, em inglês, ser comumente pronunciada "síquel" ao invés de "és-kiú-él", letra a letra. No entanto, em português, a pronúncia mais corrente é a letra a letra: "ésse-quê-éle".

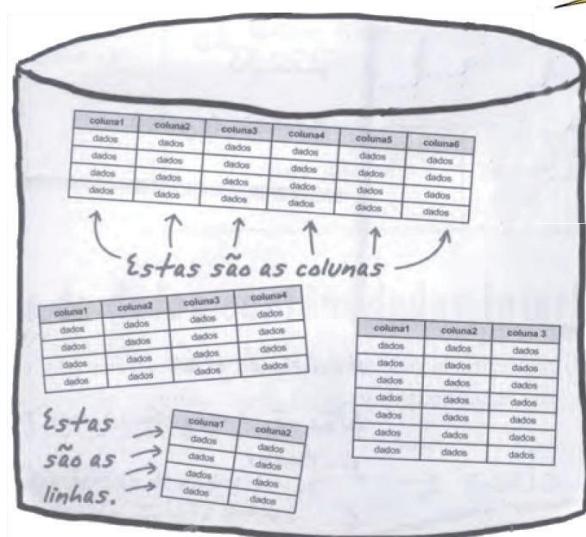
SQL tornou-se a linguagem padrão no mercado de banco de dados. Conhecê-la a fundo é um importante diferencial para aqueles que desejam uma carreira de sucesso nas áreas de programação e administração de bancos de dados.

Embora o SQL tenha se tornado um padrão para os bancos de dados, ele sofreu alterações específicas e dependendo do SGBD poderá ter recursos específicos.

1.6 Anatomia de um Banco de Dados

Como já definimos, o banco de dados é uma espécie de contêiner que guarda as informações

As informações dentro de um banco de dados estão organizados em **tabelas**.

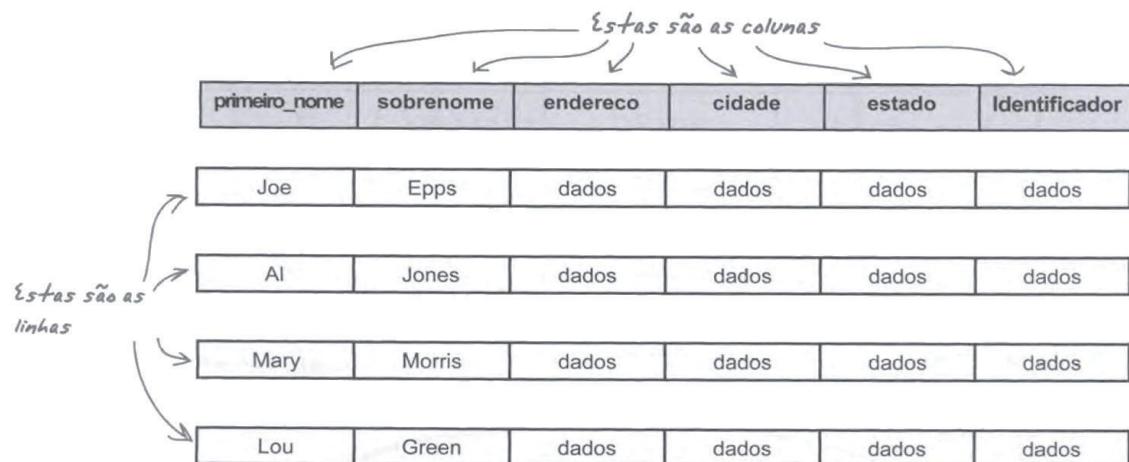


Tomando a analogia feita no tópico 1.2 em que a gaveta simboliza o banco de dados, e as camisas, bermudas e meias são os dados. Se não existir uma estrutura que organize as roupas dentro da gaveta, você terá dificuldade de achar uma peça de roupa(dado) dentro da gaveta(banco).

Dessa forma a solução definida para organizar os dados dentro de um BD foi as tabelas.

O banco de dados contém tabelas. Uma tabela é a estrutura interna de um banco de dados que contém dados em **linhas e colunas**. Uma linha da tabela contém todas as informações sobre um objeto na tabela.

No estudo de BD as tabelas são chamadas de *entidades*, as colunas de *campos* e as linhas de *registros*.



Quando juntamos as colunas (**campos**) com as linhas (**registros**) formamos um tabela (**entidade**) onde irão ser organizado os dados.

Coloque as colunas e as linhas juntas e você terá formado uma tabela!

primeiro_nome	sobrenome	endereço	cidade	estado	identificador
Joe	Epps	dados	dados	dados	dados
Al	Jones	dados	dados	dados	dados
Mary	Morris	dados	dados	dados	dados
Lou	Green	dados	dados	dados	dados

Exercícios Propostos

Qual a diferença entre banco de dados e SGBD?

O que é SQL e qual sua funcionalidade?

Complete as lacunas de forma que a sentença esteja correta:

No estudo de banco de dados, as colunas de uma tabela também são chamadas de _____, as linhas de _____ e a tabela de _____.

A sigla SGBD significa _____, e _____, _____, _____, _____ são exemplos de SGBD.

Utilizando o Writer ou Calc ou Inkscape crie uma entidade que represente:

- aluno

- escola
- usuario
- funcionario

Quais as vantagens da utilização de um SGBD em relação aos sistemas tradicionais de gerenciamento de arquivos?

O que é um DBA?

Dê um exemplo de aplicação de banco de dados.

O que é DML como são classificadas e qual a diferença entre as classificações?

2.0 Introdução ao MySQL



O MySQL é um sistema gerenciador de banco de dados (SGBD) de um banco de dados relacional, de licença dupla (sendo uma delas de software livre), projetado inicialmente para trabalhar com aplicações de pequeno e médio e portes, mas hoje atendendo a aplicações de grande porte e com mais vantagens do que seus concorrentes. Possui todas as características que um banco de dados de grande porte precisa, sendo reconhecido por algumas entidades como o banco de dados open source com maior capacidade para concorrer com programas similares de código fechado, tais como SQL Server(da Microsoft) e Oracle Database.

2.1 Um pouco da história do MySQL

O MySQL foi criado na Suécia por dois suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius, que têm trabalhado juntos desde a década de 1980. Hoje seu desenvolvimento e manutenção empregam aproximadamente 400 profissionais no mundo inteiro, e mais de mil contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito dele.

No dia 16 de Janeiro de 2008, a MySQL AB, desenvolvedora do MySQL, foi adquirida pela Sun Microsystems, por US\$ 1 bilhão, um preço jamais visto no setor de licenças livres. No dia 20 de Abril de 2009, foi anunciado que a Oracle compraria a Sun Microsystems e todos os seus produtos, incluindo o MySQL. Após investigações da Comissão Europeia sobre a aquisição para evitar formação de monopólios no mercado a compra foi autorizada e hoje a Sun faz parte da Oracle.

O sucesso do MySQL deve-se em grande medida à fácil integração com o PHP incluído, quase que obrigatoriamente, nos pacotes de hospedagem de sites da Internet oferecidos atualmente. Empresas como Yahoo! Finance, MP3.com, Motorola, NASA, Silicon Graphics e Texas Instruments usam o MySQL em aplicações de missão crítica. A Wikipédia é um exemplo de utilização do MySQL em sites de grande audiência.

O MySQL hoje suporta Unicode, Full Text Indexes, replicação, Hot Backup, GIS, OLAP e muitos outros

recursos de banco de dados.

2.2 Características

A seguir, algumas das principais características existentes no MySQL.

2.2.1 Portabilidade

Desenvolvido utilizando as linguagens de programação C e C++, unido com o uso de GNU Automake, Autoconf e Libtool, torna o MySQL uma aplicação altamente portável entre diferentes sistemas, plataformas (Linux, Unix, FreeBSD, Mac OS X Server, Windows) e compiladores. Além disso, fornece sua API para várias outras linguagens, como Java, Python, PHP, Perl, C, C++, entre outras.

2.2.2 Capacidades

O MySQL tem um alto poder de execução e de armazenamento. Dependendo da plataforma onde a ferramenta será utilizada, suas tabelas poderão armazenar espaços extraordinários, ficando limitadas somente ao tamanho máximo de arquivos com que a plataforma em questão pode manipular. No caso de tabelas do tipo InnoDB, cujo armazenamento pode ser realizado por um ou mais arquivos separados, é possível armazenar até 65.536 TB (terabytes).

No caso de expressões SQL, o SGBD suporta execuções de scripts SQL com até 61 uniões de tabelas (joins), e em se tratando de velocidade de execução, o MySQL pode ser enquadrado entre os mais velozes, se não o mais veloz, justamente por este ter sido um dos motivos que levou seus programadores a desenvolvê-lo, baseado em tecnologias que permitiram tal fato.

O MySQL é um banco de dados extremamente poderoso, pronto para executar mais de um bilhão de consultas por dia de um site, ou até mesmo processar milhares de transações por minuto.

2.2.3 Multithreads

Usa programação de threads utilizando-as diretamente no kernel da plataforma. Além de aumentar significativamente a velocidade de processamento, ainda facilita a integração da ferramenta em hardware com mais de uma CPU.

2.2.4 Licença de uso

O MySQL é desenvolvido e distribuído por meio de duas licenças que irão depender do tipo de uso da ferramenta. Na maioria dos casos, seu uso é livre. A primeira licença é Software Livre com base na GNU-

GPL (entretanto, se o programa que acessar o MySQL não for GPL, uma licença comercial deverá ser adquirida). A segunda licença é comercial. Essa licença lhe oferece suporte diferenciado, pacotes com mais ferramentas.

2.2.5 Formas de armazenamento

O MySQL disponibiliza vários tipos de tabelas para armazenamento de dados, tendo cada tipo suas próprias características. A vantagem dessa variedade de tabelas é a possibilidade de escolher o tipo em cada situação diferente. Enquanto um tipo prioriza velocidade, outro tipo prioriza volume de dados, entre outras características.

2.2.6 Suporta triggers

Os triggers(também conhecido como gatilhos) são blocos de código SQL armazenados no servidor, mas não para serem invocados pela aplicação integrada ao banco de dados, e sim ser iniciados a partir de algum evento pré-cadastrado que ocorre no sistema(determinado horário do dia no caso dos event schedulers, antes de uma inserção ou alteração entre outras possibilidades). Mostraremos como usaremos.

2.2.7 Suporta cursors (Non-Scrollable e Non-Updatable);

A partir da versão 5 do MySQL também é possível a utilização de cursores para navegação em conjuntos de resultados. De forma simples, é possível navegar pelos registros de uma tabela e partir de laços de repetição, permitindo realizar operações necessárias e transações à parte para cada linha da tabela.

2.2.8 Suporta stored procedures e functions;

Stored Procedures e Functions são blocos de código SQL armazenados no servidor, os quais são chamados(ou se preferir,invocados) a partir das aplicações integradas ao banco de dados.

2.2.9 Suporte a replicação

Visando aumentar ainda mais a disponibilidade do servidor, tornou-se possível a partir da versão 5 do MySQL configurar servidores réplicas(clones), unidireccionais e bidireccionais, ou seja, réplicas são outros servidores que estão com suas informações sincronizadas em um servidor principal, geralmente visando a aumentar o poder de processamento e disponibilidade na parte de hardware.

2.2.10 Suporte a clusterização

Cluster é um sistema que comprehende dois ou mais computadores ou sistemas (denominados nodos) na qual trabalham em conjunto para executar aplicações ou realizar outras tarefas. O MySQL 5 traz suporte a clusterização, ou seja, dois ou mais servidores trabalham juntos para responder solicitações feitas por suas aplicações.

2.2.11 Visões

Visões são consultas pré-programadas ao banco de dados que permitem unir duas ou mais tabelas e retorna uma única tabela como resultado, quando invocadas. Além disso, podem ser utilizados para filtrar informações, exibindo somente dados específicos de uma determinada categoria de uma ou mais colunas da tabela.

Exercícios Propostos

O que é o MySQL?

Pesquise sobre a licença usada pelo SGBD MySQL Server e descreva qual o nome da licença, quando é preciso comprar uma licença, quanto custa e como adquirir.

Quais as vantagens do MySQL diante dos SGBD listados no capítulo 01?

3.0 Instalando o Mysql Server

3.1 Instalação no Linux

Vamos aprender a instalação do MySQL Server no Sistema Operacional Linux, distribuição Ubuntu.

Passo 1: Acessando em modo root

O primeiro passo é acessar o terminal (*Aplicação < Acessórios < Terminal*) de comando com os direitos de administrador do sistema(**atelho: CTRL+ALT+T**).

Se seu usuário tem permissões administrativas pule para o Passo 2, todavia se seu usuário não tenha permissões de Administrador digite no terminal o comando abaixo e tecle Enter:

```
su -
```

Instalação-3.0

Após digitar o comando Instalação-3.0 o terminal irá pedir a senha do usuário root, digite-a e prontinho, você terá todos as permissões necessárias para a instalação do Mysql-Server.

Passo 2: Verificando a versão disponível para instalação

Antes de darmos início ao processo de instalação, é importante checar qual a versão do software que será instalada. Dependendo da versão do Linux Ubuntu, a versão 5 do MySQL não estará disponível como versão padrão de instalação. Portanto, é necessário executar um processo para verificar qual versão será instalada.

Digite o comando a seguir e tecle e Enter para verificar qual versão do MySQL o Ubuntu irá instalar em seu computador:

```
apt-cache policy mysql-server
```

Instalação-3.1

Um dos parâmetros de retorno do comando Instalação-3.1 é o valor de “**Candidate ou Candidato**” que informa qual pacote é candidato à instalação. Se a versão informada neste campo for **acima da versão 5.0**, avance para o **Passo 4**. Caso a versão deste campo seja inferior a 5.0, siga os passos descritos no **Passo 3** para ajustar o Ubuntu para utilizar a versão mais recente do MySQL.

Passo 3: Configurando o APT-GET

Somente leia este tópico se estiver tendo problemas para encontrar a versão 5.0 ou superior do MySQL na utilização do comando apt-get. Caso contrário, avance para o próximo passo.

Existe um modo para instalar a versão mais recente do MySQL caso o apt-get não esteja trazendo tal versão. Este processo é conhecido como Backport, cujo site oficial está descrito em <http://www.backports.org>.

Para obter informações detalhadas a respeito deste procedimento, acesse o site citado anteriormente. A seguir, serão encontrados apenas os passos necessários para a configuração do apt-get para o caso específico do MySQL. Esses passos farão com que o apt-get busque os pacotes do MySQL no repositório do Backports.org, em vez do depositório-padrão.

Adicione a linha a seguir em seu arquivo **/etc/apt/source.list**:

```
deb http://www.backports.org/debian/ sarge-backports main
```

Instalação-3.2

Essa linha incluirá o endereço citado anteriormente como uma base de requisição de pacotes para Ubuntu.

Na sequência, adicione as linhas a seguir no arquivo **/etc/apt/preferences**:

```
Package: *
Pin: release a=sarge-backports
Pin-Priority: 200
Package: mysql-server
Pin: release a=sarge-backports
Pin-Priority: 999
```

Instalação-3.3

As três primeiras linhas farão com que todos os pacotes solicitados pelo apt-get sejam requeridos da fonte de dados-padrão do Ubuntu. Já as três últimas linhas formam um filtro, informando ao Ubuntu que se o pacote solicitado for o MySQL Server, deverão ser utilizadas as bases de pacotes do Backports.org.

Para finalizar atualize a lista de pacotes do apt-get, por meio do comando:

```
apt-get update
```

Instalação-3.4

Passo 4 : Instalando o MySQL Client

Para realizar os downloads dos pacotes necessários e instalar o MySQL Client e MySQL Server, execute

```
apt-get install mysql-server
```

o comando a seguir:

Instalação-3.5

A partir deste ponto, o comando apt-get buscará informações sobre os pacotes a realizar o download, instalar e configurar. Após obter essas informações, que serão mostradas em sua tela, será solicitada uma confirmação para prosseguir.

Digite Y seguido de Enter

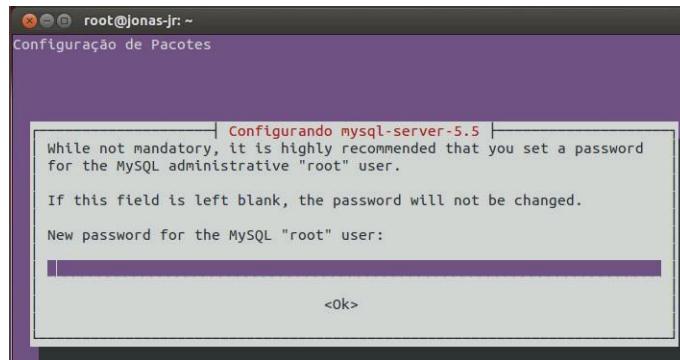
Instalação-3.6

Durante esse processo, outras solicitações poderão ser exibidas. Além disso, caso o seu Ubuntu não esteja com as informações de domínio e e-mail corretamente configurados, estas poderão ser solicitadas para correção.

Um das solicitações que podem surgir durante o processo de instalação é a tela para definição da senha do usuário root do MySQL Server, usuário esse que possui todos os privilégios de manipulação de todas as funções do SGBD MySQL Server.

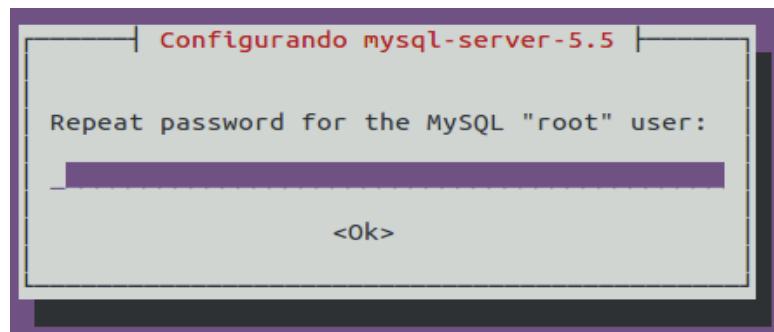
Atenção, esse usuário e essa senha que você está definindo não é o mesmo usuário root do sistema operacional Linux Ubuntu, é o usuário e senha necessário para que você consiga utilizar todas as ferramentas do SGBD.

Abaixo estão as telas de definição de senha do usuário root MySQL Server. Na tela *Instalação-3.7* você irá digitar com atenção a senha do usuário MySQL root e teclar enter, anote a senha para que você não esqueça, pois você irá precisar futuramente.



Instalação-3.7

Repete senha digitada na tela **Instalação-3.7**:



Instalação-3.8

Passo 5: Comandos de Inicialização e Interrupção

A seguir, os comandos de inicialização e interrupção do MySQL Server, bem como o de inicialização do MySQL Client para uso do Ubuntu. Lembre-se de que pode ser necessário estar com os privilégios de administrador (root) para iniciar ou interromper o MySQL Server (**Instalação-3.0**).

Digite no terminal o comando abaixo e tecle Enter para:

iniciar o serviço MySQL Server no Ubuntu:

```
/etc/init.d/mysql start
```

Instalação-3.9

interromper o serviço MySQL Server no Ubuntu:

```
/etc/init.d/mysql stop
```

Instalação-3.10

iniciar a ferramenta/software MySQL Client no Ubuntu:

```
mysql
```

Instalação-3.11

iniciar a ferramenta/software MySQL Client no Ubuntu com usuário root:

```
mysql -u root -p senhadoroot
```

Instalação-3.12

No comando **Instalação-3.12** no lugar de *senhadoroot* digite a senha que você cadastrou quando estava instalando o MySQL Server.

Vale a pena lembrar que o comando **apt-get**, dependendo das configurações do seu Ubuntu pode configurar para que o MySQL seja inicializado automaticamente ao iniciar o sistema e interrompê-lo da

maneira correta ao desligar o sistema.

Passo 6: Localização do arquivo de configuração

O arquivo **my.cnf**, responsável pela configuração do servidor, pode ser localizado no diretório /etc/mysql.

Passo 7: Criando a senha do root para o MySQL

Caso na instalação não tenha surgido a tela **Instalação-3.7** o usuário root do MySQL Server ficou sem senha. Então vamos aprender defini-la digitando o comando abaixo no terminal ubuntu:

```
mysqladmin -u root password suasenha
```

Instalação-3.13

No comando **Instalação-3.13** no lugar de **suasenha** digite a senha que você desejar, não esqueça essa senha, pois iremos precisar dela. Caso a senha contenha caracteres especiais digite a nova senha entre aspas dupla.

Passo 8: Testando a aplicação

Para verificar se o MySQL Server e o MySQL Client estão instalados corretamente em seu computador, inicialize o MySQL Client(**Instalação-3.11/Instalação-3.12**).

Em seguida digite o comando a seguir:

```
STATUS;
```

Instalação-3.14

Passo 9: Adicionando pacote extra ao PHP 5

Vamos instalar o pacote `php5-mysql`, pois esse nos fornece módulos para conexões ao banco de dados MySQL diretamente a partir de scripts PHP. Inclui o módulo genérico "mysql" que pode ser usado para conectar em todas as versões do MySQL, um módulo melhorado "mysqli" para versões MySQL 4.1 ou mais recentes e o módulo `pdo_mysql` para ser usado com a extensão PHP Data Object.

Para fazer a instalação do módulo você precisa de permissões de Administrador(Instalação 3.0). No terminal do Ubuntu digite:

```
apt-get install php5-mysql
```

Instalação-3.15

3.2 Instalação no Windows

Instalando e configurando a nova versão do MySQL

Primeiramente vamos acesso o site oficial do MySQL, através do seguinte link: <http://www.mysql.com/>



Figura 1: Site oficial do MySQL

Após o acesso a página, clique na aba DOWNLOADS(GA), para ter acesso a uma outra página, como mostra a figura 2:



Figura 2: Site oficial do MySQL mostrando a opção de download

Clique no botão DOWNLOAD, como está sendo mostrado na figura 2, em seguida aguarde ser redirecionado para outro passo, como mostra a figura 3:

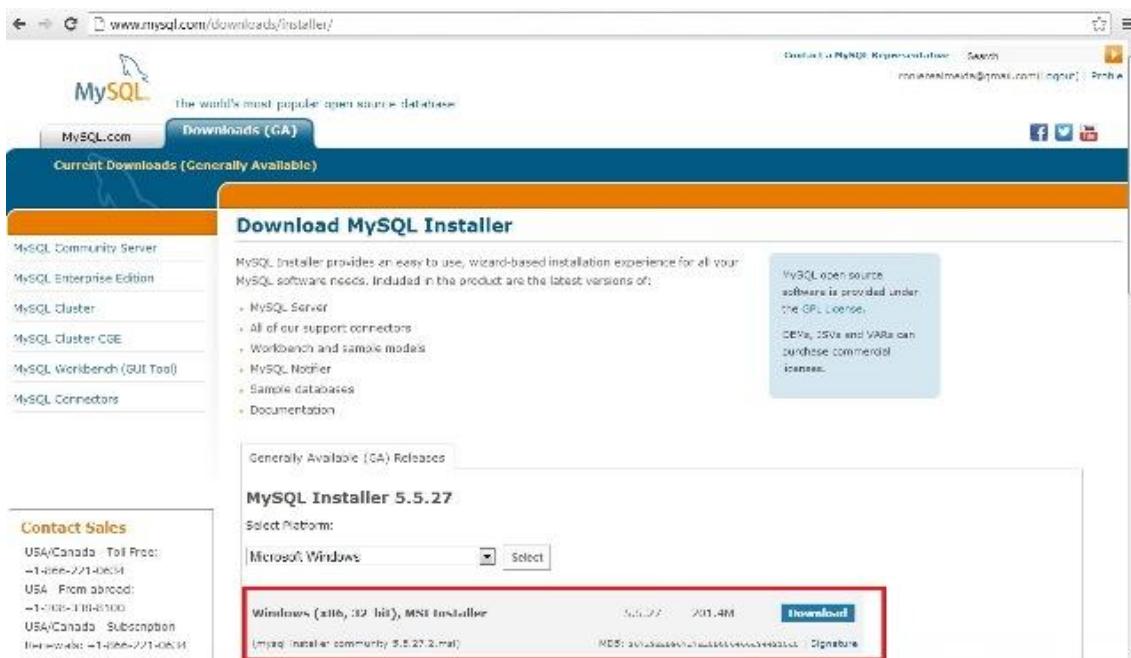


Figura 3: Site oficial do MySQL mostrando o botão de download

Após o clique no botão DOWNLOAD mostrado na figura 3, aparecerá uma nova janela, caso não tenha se cadastrado no site do MySQL não será possível fazer o download do mesmo, veja na figura 4 a janela que aparecerá em seguida:

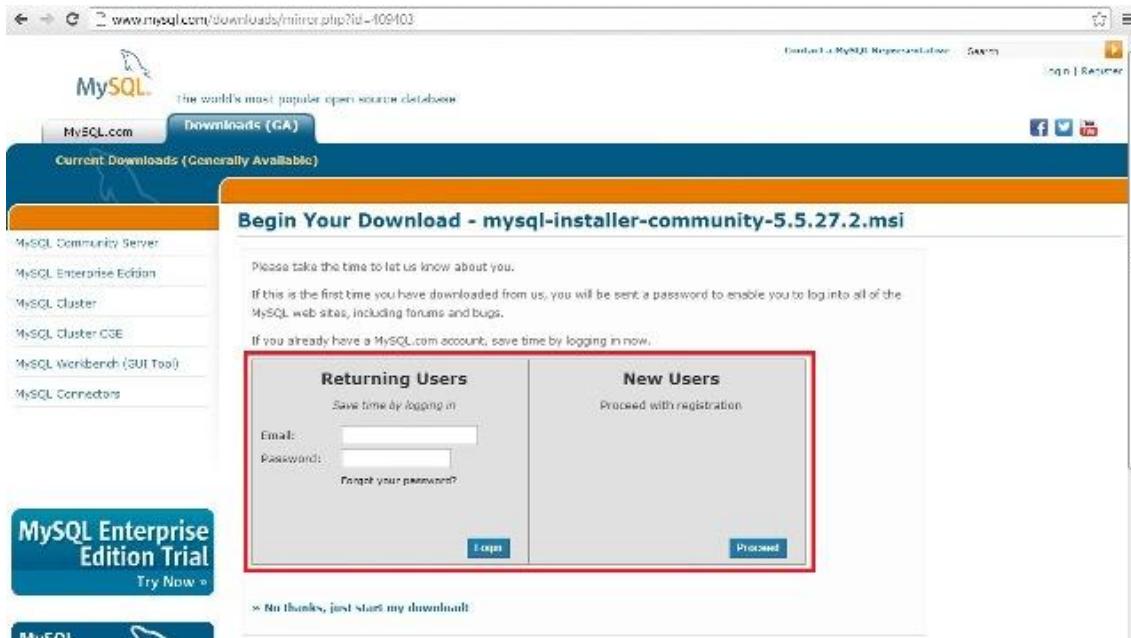


Figura 4: Site oficial do MySQL, fazer login ou cadastro

Como mostrado na figura 4, a primeira opção (esquerda) é para quem já é cadastrado no site, a segunda opção (direita) é para fazer um novo cadastro. Não se preocupem o processo é rápido e fácil, basta clicar em PROCEED e uma nova janela surgirá, no caso apresentado na figura 5:

Figura 5: Site oficial do MySQL, cadastro de novos usuários

Depois deste processo faça o download, geralmente, dependendo da sua conexão com a internet pode demorar alguns minutos.

Depois de baixar o arquivo de instalação do MySQL, vamos iniciar o processo de instalação. Dê um duplo clique no arquivo, e aparecerá uma nova janela, como mostra a figura 6:



Figura 6: Windows Configurando o MySQL Installer

O processo acima dependendo da configuração do seu computador pode demorar alguns minutos, e aparecerá sempre uma janela muito comum no Windows Seven e Windows Vista, sempre permita a execução dos arquivos na nossa instalação.

Finalmente chegamos a tela de boas vindas da instalação do MySQL, apresentada na figura 7:

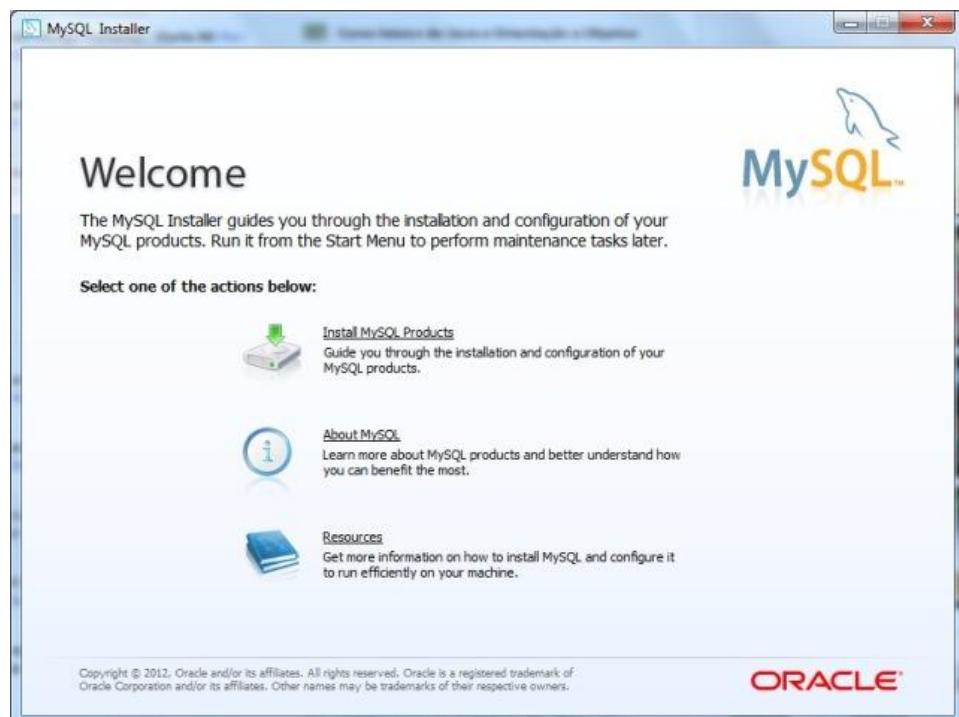


Figura 7: Janela de boas vindas da instalação do MySQL

Sobre a figura 7, temos três opções, sendo que uma delas é para instalação, vamos seguir adiante clicando em INSTALL MYSQL PRODUCTS, aparecerá a tela de acordo de licença do produto, apresentada na figura 8:

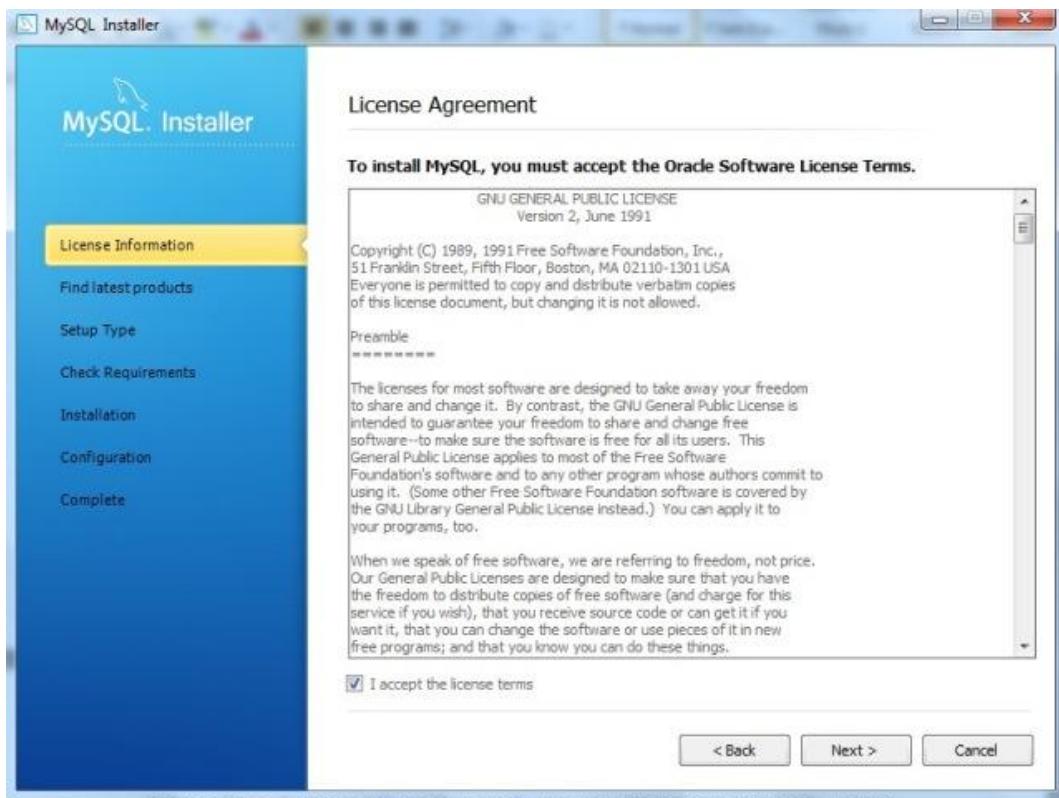


Figura 8: Janela de acordo de licença do produto

Sobre a janela acima, clique em I ACCEPT THE LICENSE TERMS e clique em NEXT, e a janela para atualizar ou encontrar produtos recentes aparecerá, como mostra a figura 9:

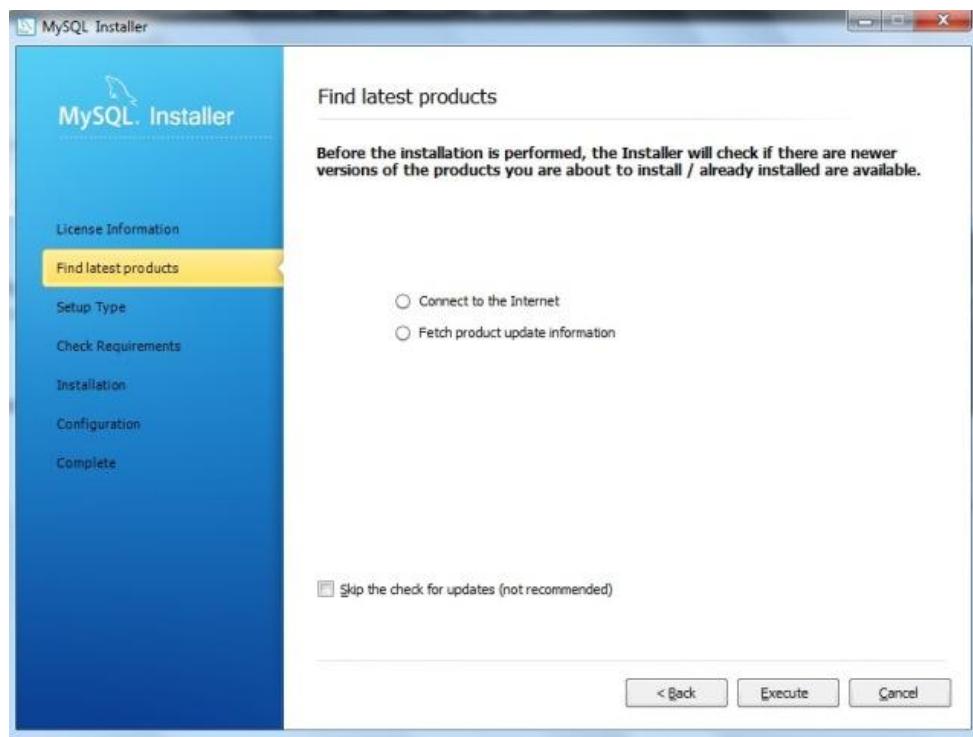


Figura 9: Janela para atualizar encontrar os produtos mais recentes

Na figura 9, temos as seguintes opções:

- Conectar a internet;
- Buscar informações de atualização de produto.

Estas duas opções servem para o momento da instalação, o próprio instalador do MySQL verifica se há versões mais recentes do produto, caso não ache necessário esta opção, simplesmente marque SKIP THE CHECK FOR UPDATES, ou se preferir, clique no botão EXECUTE.

Após clicar no botão EXECUTE aparecerá uma nova janela, com essa devemos ter um certo cuidado, veja na figura 10:

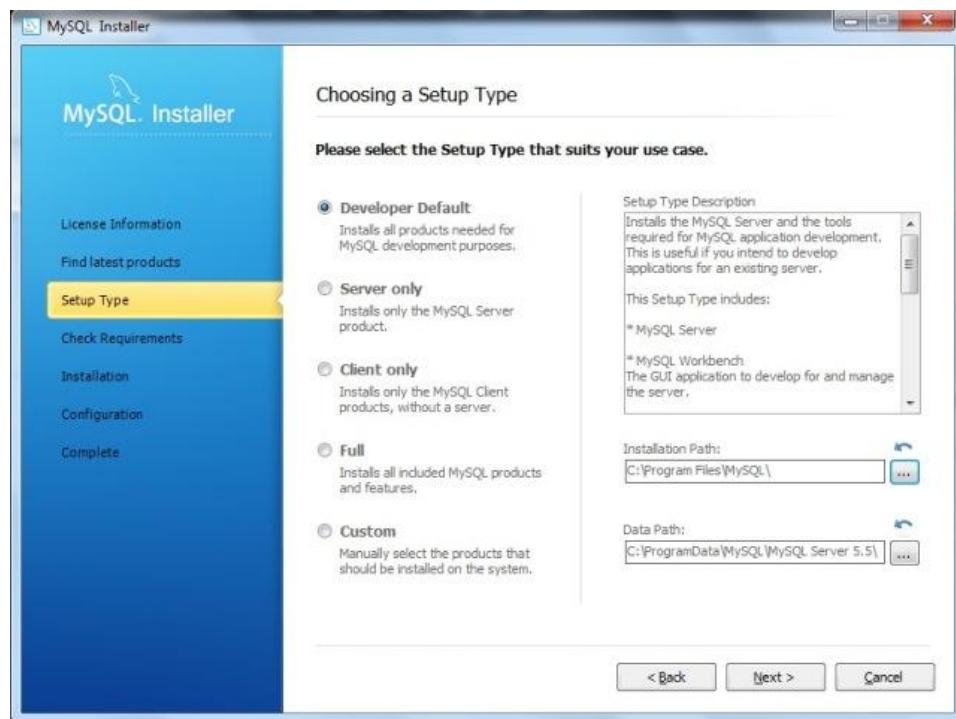
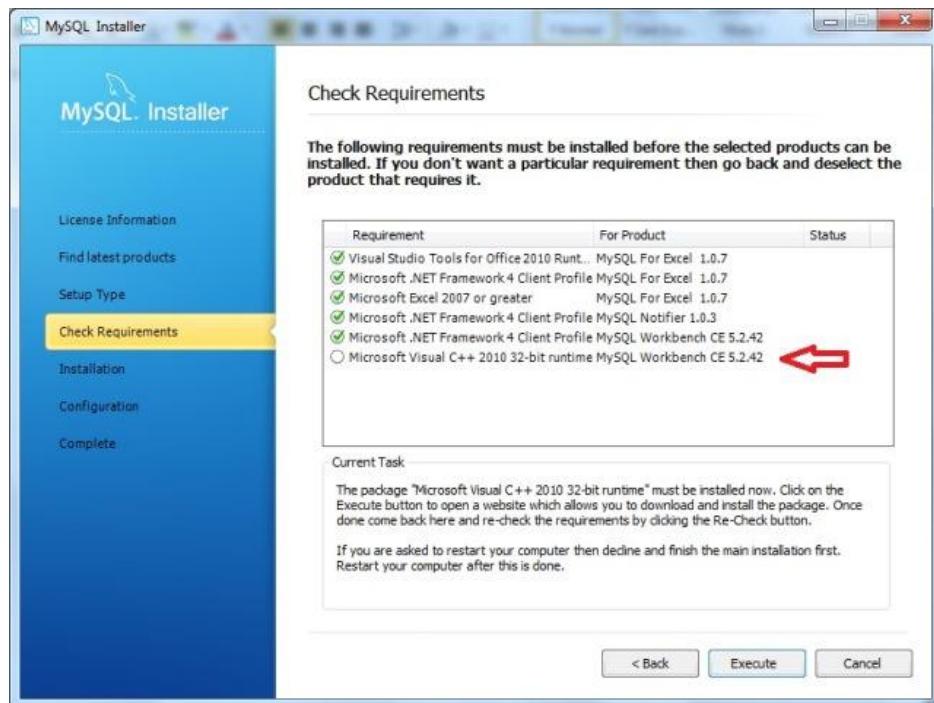


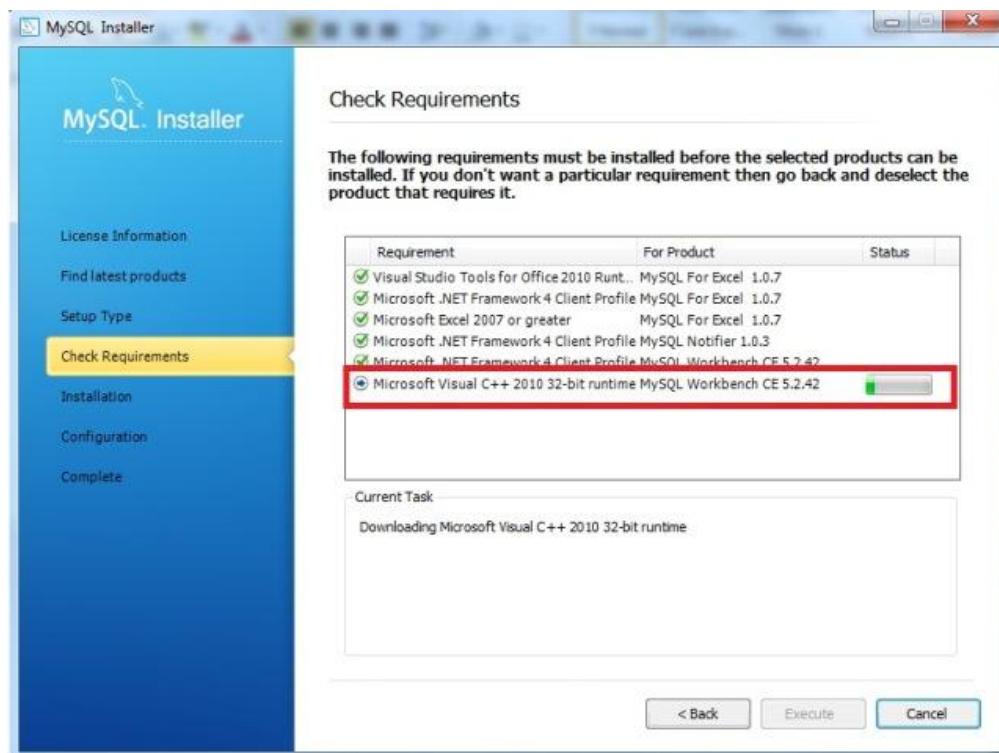
Figura 10: Janela de escolha do tipo de instalação

Na janela acima temos várias opções, neste caso, isso vai depender de cada profissional e para que ele vai usar o MySQL. Note que no lado esquerdo o aplicativo nos dá informações bem detalhadas do que será instalado. Após a sua escolha, marque a opção de sua escolha e clique em NEXT.

Depois deste processo, uma nova janela surgirá, veja na figura 11:

**Figura 11:** Janela para verificar os requisitos

Note que na figura 11, existe um componente detectado que não está instalado no seu computador, mas não há problema, clique em EXECUTE que o próprio assistente vai baixá-lo para você, como mostra a figura 12:

**Figura 12:** Janela para verificar os requisitos, baixando um aplicativo

Em seguida, após baixar o aplicativo que seria necessário para dar continuidade ao processo, veja mesma a janela como deverá ficar, na figura 13:

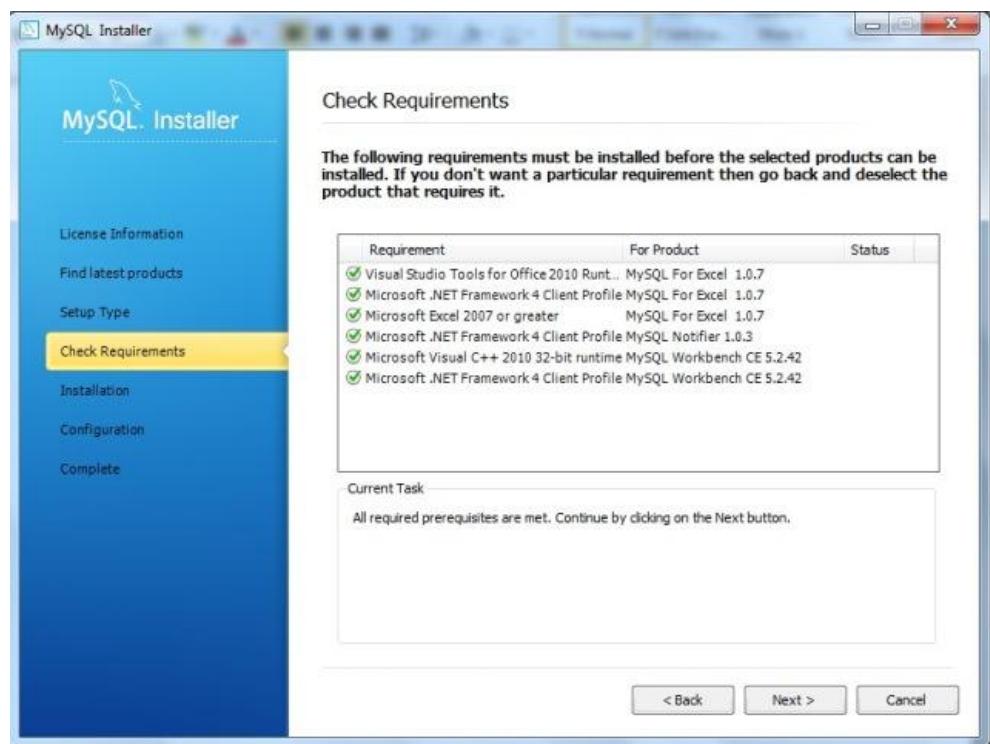


Figura 13: Janela para verificar os requisitos, sem pendencias

Após este processo, clique em NEXT para dar continuidade, a próxima janela mostrará tudo que será instalado, isso vai depender de quais opções você escolheu no processo mostrado na figura 10. Veja o resultado na figura 14:

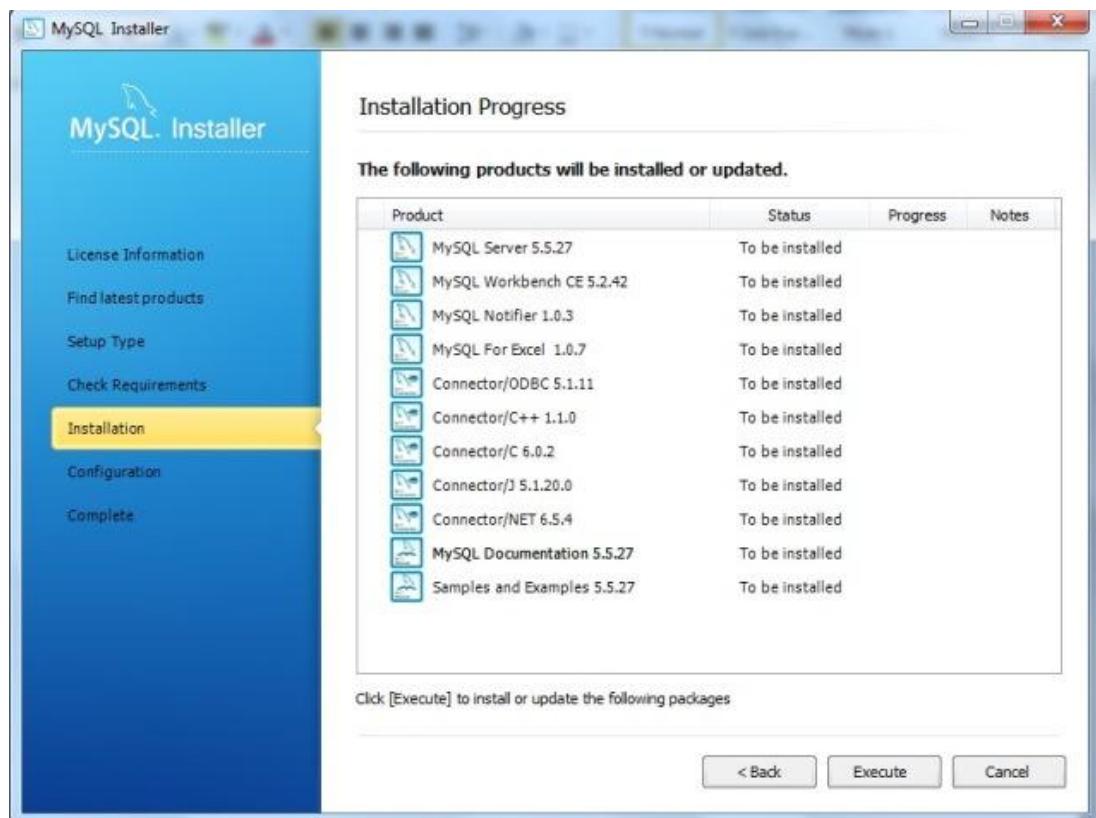


Figura 14: Janela do assistente mostrando tudo que será instalado

Após este passo clique em EXECUTE, e veja na figura 15 o processo que irá ocorrer com cada uma das opções a serem instaladas.

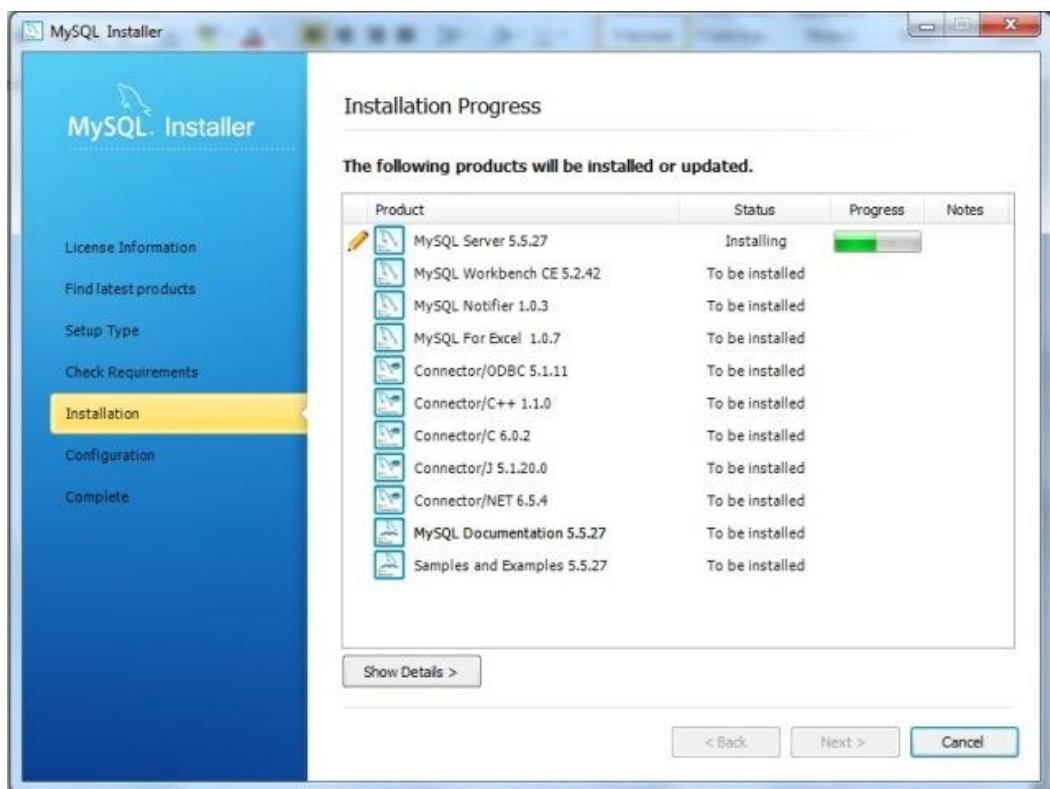


Figura 15: Processo de instalação

Após o término deste processo, o botão NEXT ficará habilitado, clique nele e continue até a próxima janela CONFIGURATION OVERVIEW (visão geral da configuração), apresentada na figura 16:

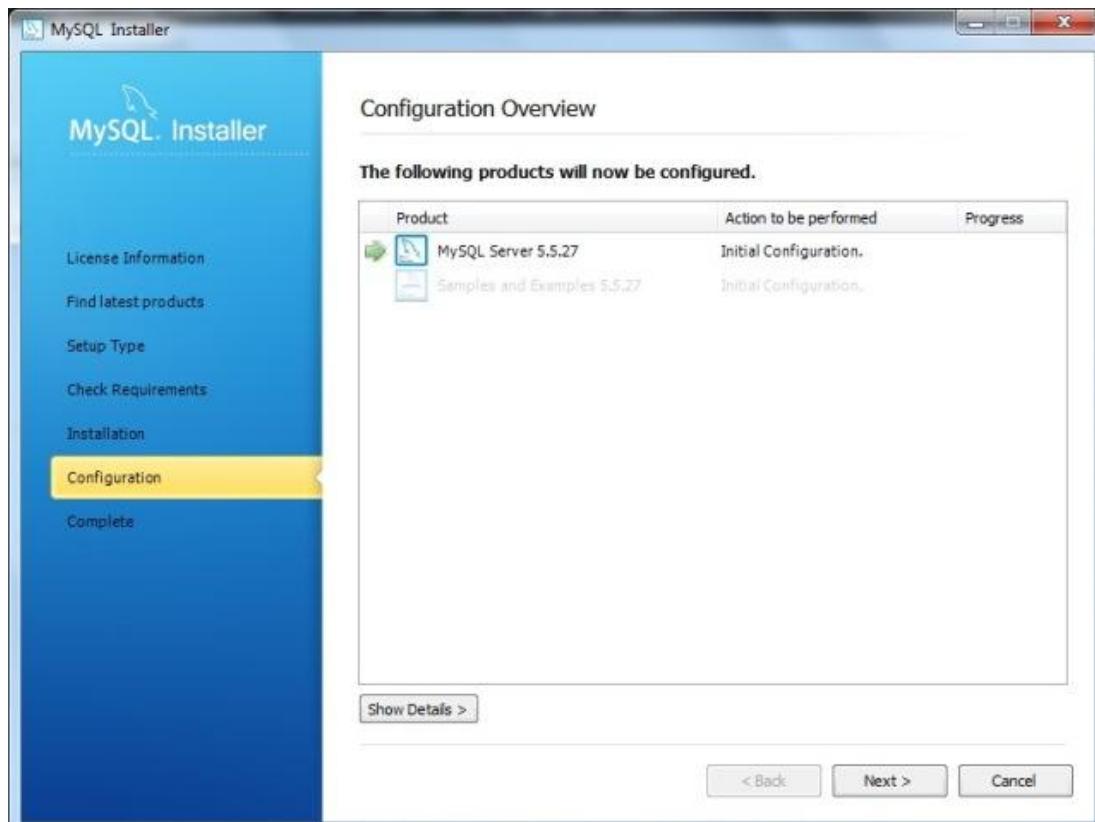


Figura 16: Janela CONFIGURATION OVERVIEW

Clique em NEXT para dar continuidade ao processo, em seguida aparecerá mais uma janela, como mostra a figura 17:

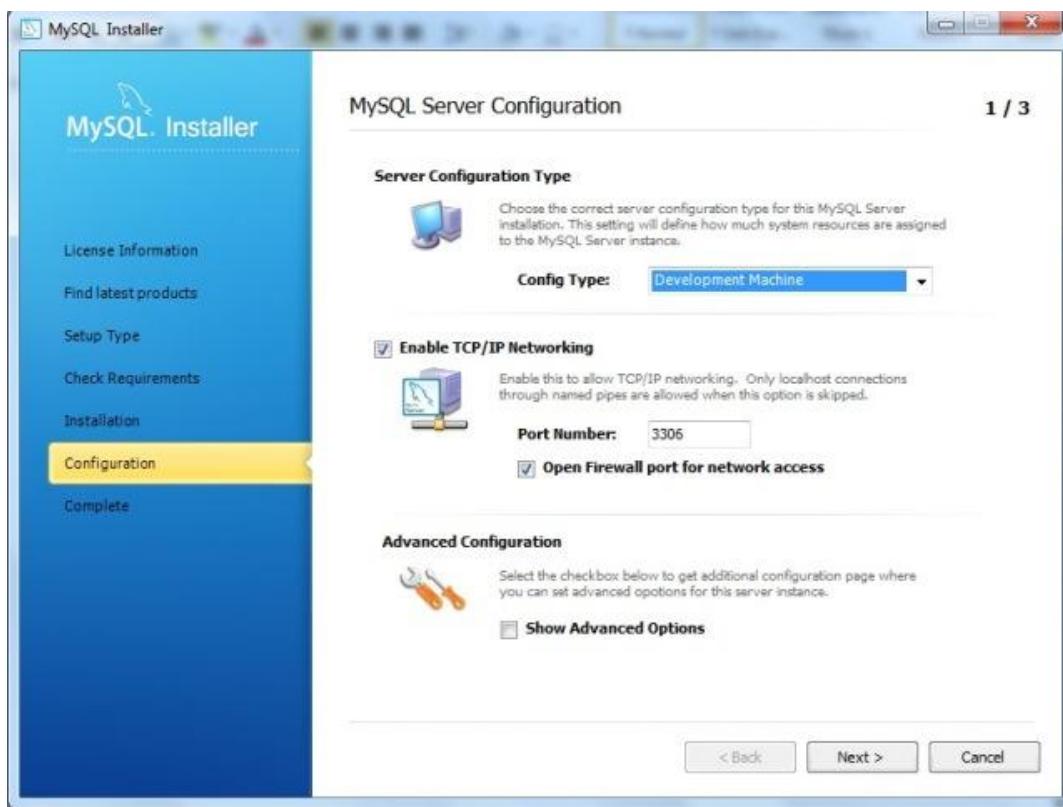


Figura 17.Configuração do MySQL.

Na configuração do MySQL, podemos mudar a porta utilizada e o tipo de configuração do servidor: Developer Machine, Server Machine e Dedicated Machine, para o nosso caso, vamos deixar do jeito que está, clique em NEXT para irmos ao próximo passo. Após clicar em NEXT, aparecerá uma janela para inserir uma senha para o administrador, como demonstrado na**Figura 18**:

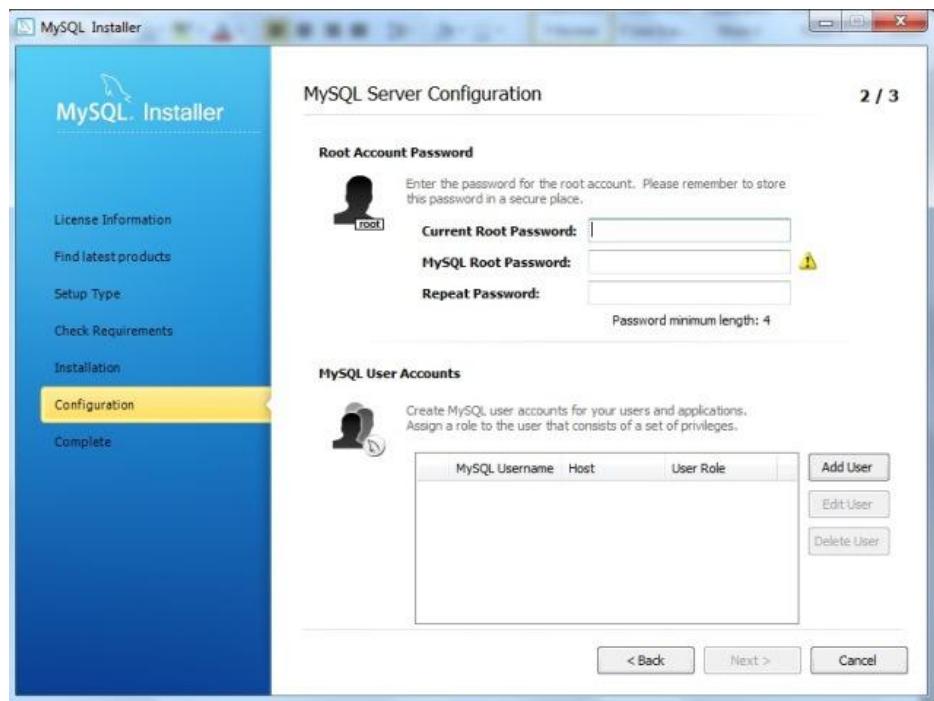


Figura 18: Definindo a senha do administrador

Após definir a senha do administrador clique em NEXT, novamente aparecerá uma nova janela, veja na figura 19:



Figura 19: Configurando o servidor

Na figura acima tem o WINDOWS SERVICE NAME, a opção de inicializar o MySQL com o sistema operacional e por último executar como conta do sistema padrão ou usuário personalizado. Caso deseje, desmarque a opção de inicializar o MySQL, já as outras configurações, deixe como está, clique em NEXT para o próximo passo, como mostra a figura 20:

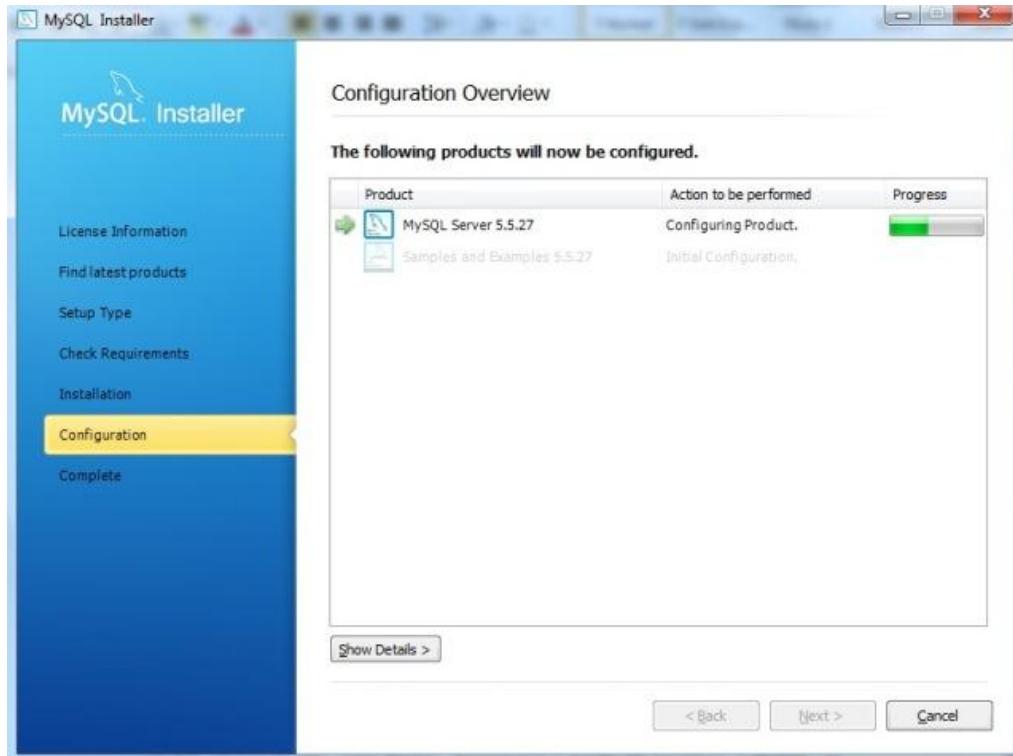


Figura 20: Configurando o servidor

Apos o término, clique em NEXT para seguir para o próximo passo, continuando a configuração do servidor, como apresentada na figura 21:

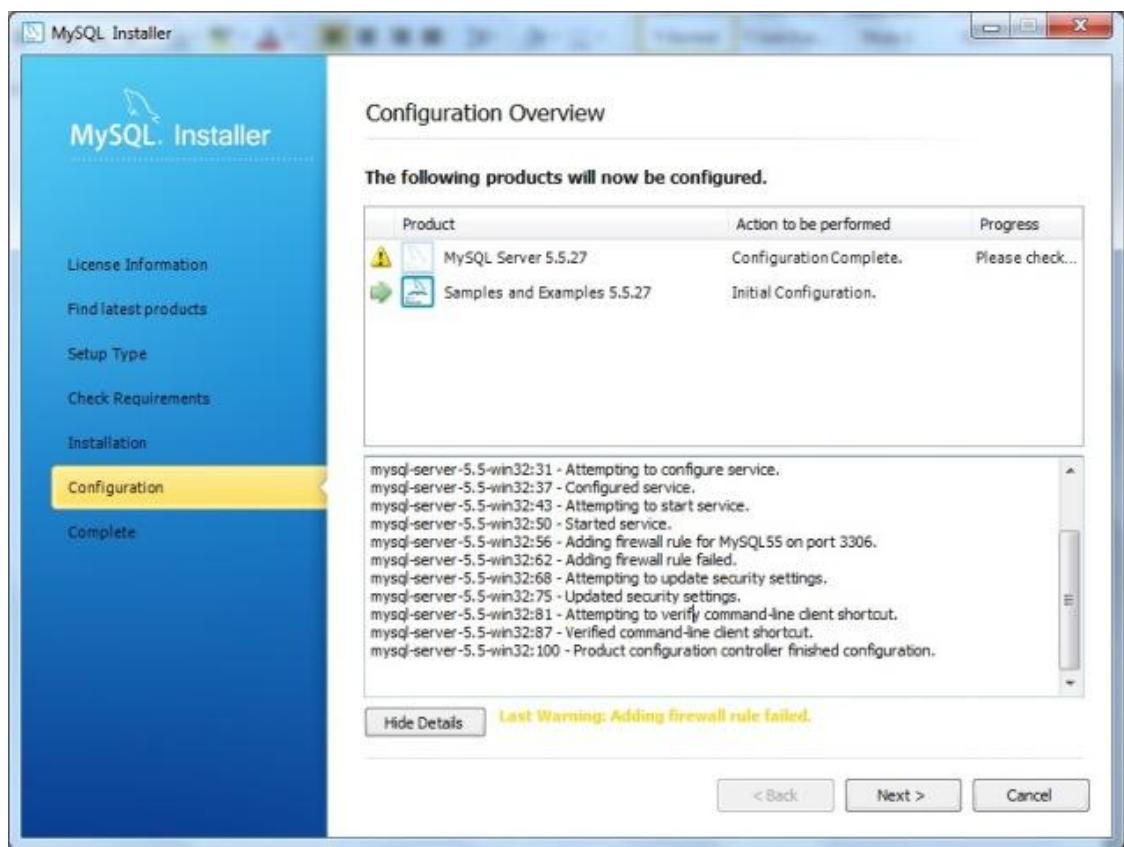


Figura 21: Configurando o servidor

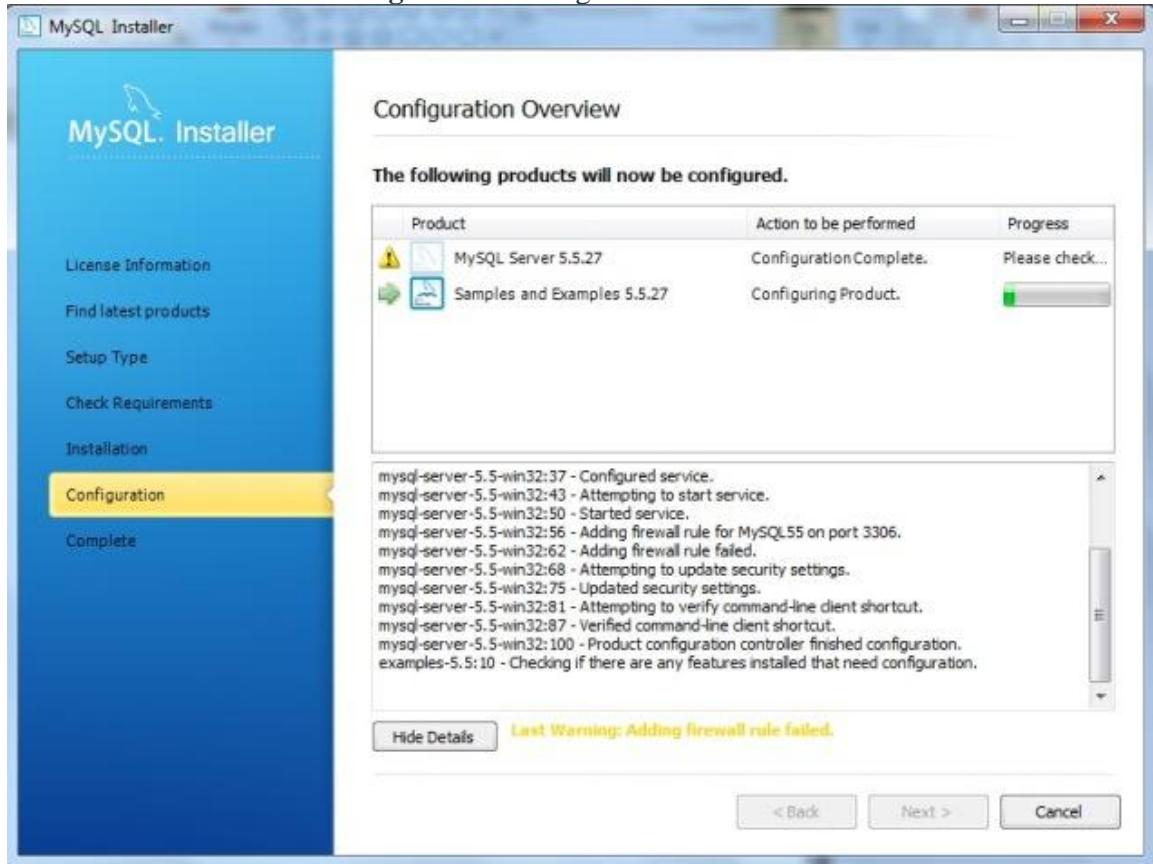


Figura 22: Continuação da configuração do servidor

Após este processo aparecerá uma janela informando a finalização do processo de instalação do MySQL, clique em finish e pronto.

Agora vamos visualizar o novo MySQL. Clique no botão INICIAR, TODOS OS PROGRAMAS, MYSQL, MYSQL WORKBENCH 5.2.CE, aparecerão as seguintes janelas em sequência, como mostram as figuras 23 e 24:



Figura 23: Janela de boas vindas do MySQL



Figura 24: Ferramenta do MySQL em execução

Para entrarmos na ferramenta, clique duas vezes em LOCALINSTANCE MYSQL55, em seguida aparecerá uma pequena tela com login e senha, vista na figura 25:



Figura 25: Janela de login para acesso ao MySQL

Entre com a senha informada na instalação e clique em OK, pronto, finalmente veremos um novo visual do MySQL, como mostra a figura 26:

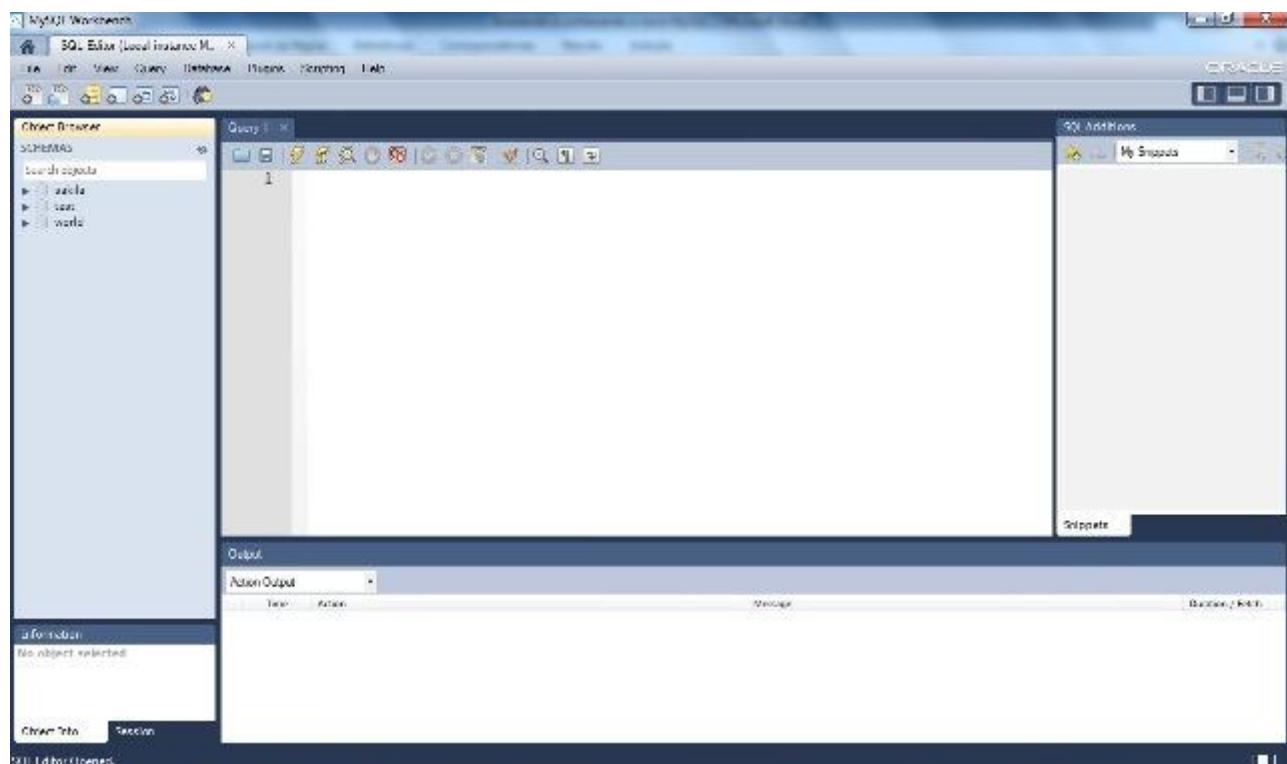


Figura 26: Janela de desenvolvimento do MySQL

Com isso finalizamos a instalação desse SGBD que agora está pronto para uso. Espero que as informações aqui apresentadas possam ser úteis para quem usa ou pretende usar essa nova versão do MySQL.

4.0 Bases de dados MySQL

Um sistema gerenciador de banco de dados é capaz de gerenciar informações de diversos sistemas ao mesmo tempo. Por exemplo, as informações dos clientes de um banco, além dos produtos de uma loja virtual ou dos livros de uma livraria.

Suponha que os dados fossem mantidos sem nenhuma separação lógica. Implementar regras de segurança específicas seria extremamente complexo. Tais regras criam restrições quanto ao conteúdo que pode ser acessado por cada usuário. Por exemplo, determinado usuário poderia ter permissão de acesso aos dados dos clientes do banco, mas não às informações dos produtos da loja virtual, ou dos livros da livraria.

Para obter uma organização melhor, os dados são armazenados separadamente em um SGBD. Daí surge o conceito de base de dados (database).

Uma base de dados é um agrupamento lógico das informações de um determinado domínio. Utilizaremos o programa que instalamos juntamente com o MySQL Server no capítulo anterior, o MySQL Command Line Client (também conhecido como MySQL Client), para controlarmos o SGBD MySQL Server usando a linguagem SQL, no decorrer do curso iremos conhecer outras formas de fazer essa manipulação.

Todas as instruções a seguir serão digitadas no Mysql Client(Instalação 3.11/Instalação 3.12)

4.1 Criando uma base de dados

Para criar uma base de dados no MySQL Server, utilizamos o comando **CREATE DATABASE...**

Legenda:

CREATE DATABASE nome_do_banco;

nome_do_banco < nome do banco a ser criado;

```
mysql> CREATE DATABASE livraria;
Query OK, 1 row affected (0.02 sec)
```

Exemplo:

Terminal 4.0: Criando uma base de dados.

4.2 Listando base de dados

Podemos utilizar o comando SHOW DATABASES para listar as bases de dados existentes.

```
SHOW DATABASES;
```

Exemplo:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| livraria |
| mysql |
| test |
+-----+
4 rows in set (0.03 sec)
```

Terminal 4.1: Listando as bases de dados existentes.

Repare que, além da base de dados **livraria**, há outras três bases. Essas bases foram criadas automaticamente pelo próprio MySQL Server para teste ou para armazenar configurações.

4.3 Ativando/Selecionando uma base de dados

Para a maioria das ações realizadas em um banco de dados no MySQL é necessário ativá-lo ou selecioná-lo para uso. Este procedimento é feito a partir do comando apresentado a seguir:

```
USE nome_do_banco;
```

Legenda:

nome_do_banco < nome do banco a ser selecionado;

Exemplo:

```
mysql> USE livraria;
Database changed
```

Terminal 4.2: Selecionando uma base de dados.

Nesse comando escrito no MySQL Client, selecionamos ou ativamos o banco de dados **livraria** criado no tópico 4.1. Com ele selecionado ou ativo podemos criar, editar, excluir e fazer consultas em tabelas

4.4 Deletando uma base de dado

Quando uma base de dados não é mais necessária, ela pode ser removida através do comando

DROP DATABASE...

```
DROP DATABASE nome_do_banco;
```

Legenda:

nome_do_banco < nome do banco a ser deletado;

Exemplo:

```
mysql> DROP DATABASE livraria;
Query OK, 0 rows affected (0.08 sec)
```

Terminal 4.3: Destruindo uma base de dados.

4.5 Alterando uma base de dados

Também podemos alterar as características de uma base de dados usando o comando **ALTER DATABASE...**

Veremos no tópico 4.6 como utilizar o **ALTER DATABASE...** e uma forma mais detalhada de criar uma base de dados, diferente do que já vimos no tópico 4.1.

Observação: o comando **ALTER DATABASE...** altera algumas propriedades da base de dados, mas não consegue renomeá-la. Na versão 5.1.7 do MySQL foi adicionado um comando próprio para fazer isso chamado **RENAME DATABASE...** mas, o mesmo foi retirado na versão 5.1.23 pois poderia resultar em perda de dados.

4.6 Conjunto de caracteres

4.6.1 Conjuntos de caracteres e collations em Geral

Um conjunto de caracteres é um conjunto de símbolos e códigos. Uma collation é um conjunto de regras para comparação de caracteres em um conjunto de caracteres. Vamos deixar a distinção clara com um exemplo de um conjunto de caracteres imaginário.

Suponha que temos um alfabeto com quatro letras: ‘A’, ‘B’, ‘a’, ‘b’. Damos um número a cada letra: ‘A’ = 0, ‘B’ = 1, ‘a’ = 2, ‘c’ = 3. A letra ‘A’ é o símbolo, o número 0 é o código para ‘A’, e a combinação de todas as quatro letra e seus códigos é um conjunto de caracteres.

Agora suponha que desejamos comparar duas strings, ‘A’ e ‘B’. O modo mais simples de se fazer isto é olhar o código — 0 para ‘A’ e 1 para ‘B’ — e como 0 é menor que 1, dizemos que ‘A’ é menor que ‘B’. Agora, o que fizemos foi apenas aplicar uma collation a nosso conjunto de caracteres.

A collation é um conjunto de regras (apenas um regra neste caso): “compara os códigos”.

Chamamos isto a mais simples de todas as collations possíveis como um collation binaria.

Mas e se você dissesse que letras minúsculas e maiúsculas são equivalentes? Então haveriam pelo menos duas regras: (1) tratar as letras minúsculas ‘a’ e ‘b’ como equivalentes a ‘A’ e ‘B’; (2) e então comparar os códigos. Chamamos isto de collation caso insensitivo. É um pouco mais complexo do que collation binária.

Na vida real, a maioria dos conjuntos de caracteres possuem muitos caracteres: não apenas ‘A’ e ‘B’ mas todo o alfabeto, algumas vezes alfabetos múltiplos ou sistemas de descritas ocidentais com milhares de caracteres, junto com muitos símbolos especiais e sinais de pontuação. Em geral as collations também possuem diversas regras: não apenas caso insensitivo mas acentos insensitivos e mapeamento de múltiplos caracteres (como a regra de que ‘ÄO’ = ‘OE’ em uma das duas collations alemãs)

4.6.2 Conjunto de caracteres e collations no MySQL

Um conjunto de carácter sempre tem pelo menos uma collation. Ele pode ter diversas collations. Por exemplo, conjunto de caracteres latin1 (“ISO-8859-1 West European”) tem os seguintes collations:

Collation	Significado
latin1_bin	Binário de acordo com a codificação latin1
latin1_danish_ci	Dinamarquês/Norueguês
latin1_german1_ci	latin1_german1_ci Alemão DIN-1
latin1_german2_ci	Alemão DIN-2
latin1_swedish_ci	latin1_swedish_ci Sueco/Finnish
latin1_general_ci	latin1_general_ci Multilíngua

Notas:

Dois conjuntos de caracteres diferentes não podem ter a mesma collation.

Cada conjunto de caracteres tem uma collation que é a collation padrão. Por exemplo, o collation padrão para latin1 é **latin1_swedish_ci**.

Perceba que existe uma convenção para nomes de collations. Elas iniciam com o nome do conjunto de

caracteres com o qual elas são associadas, eles normalmente incluem um nome de linguagem e finalizam com _ci (caso insensitivo), _cs (caso sensitivo), ou _bin (binário).

4.6.3 Conjunto de caracteres e collation de banco de dados

Há configurações padrões para conjuntos de caracteres e collations em quatro níveis: servidor, banco de dados, tabelas e colunas. Vamos ver como definimos e como funciona o conjunto de caracteres e o collation em um banco de dados no MySQL.

Todo banco de dados tem um conjunto de caracteres de banco de dados e uma collation de banco de dados, que não podem ser nulos. Os comandos CREATE DATABASE e ALTER DATABASE agora possuem cláusulas opcionais para especificarem o collation e conjunto de caracteres de banco de dados.

```
CREATE DATABASE nome_do_bd CHARACTER SET nome_do_conjunto_caracter  
COLLATE nome_do_collate;
```

Ou

```
ALTER DATABASE nome_do_bd CHARACTER SET nome_do_conjunto_caracter  
COLLATE nome_do_collate;
```

Legenda:

nome_do_bd < nome do banco;

nome_do_conjunto_caracter < nome do conjunto de caracteres que será utilizado no banco;
nome_do_collate < nome do collate que será utilizado no banco;

```
CREATEDATABASE livraria CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

Exemplo:

O MySQL escolhe o conjunto de caracteres e collations do banco de dados desta forma:

- Se CHARACTER SET X e COLLATE Y foram especificados, então o conjunto de caracteres é X e a é collation Y.
- Se CHARACTER SET X foi especificado sem COLLATE, então o conjunto de caracteres é X e a collation é o padrão.

Senão, utilize o conjunto de caracteres e a collation do servidor.

- A sintaxe CREATE DATABASE... DEFAULT CHARACTER SET... do MySQL é análoga

a sintaxe CREATE SCHEMA... CHARACTER SET... do padrão SQL. Por isto, é possível criar bancos de dados com conjunto de caracteres e collations diferentes, no mesmo servidor MySQL.

O conjunto de caracteres e collations do banco de dados são usados como valores padrões se o conjunto de caracteres e a collation de tabela não forem especificados nas instruções CREATE TABLE.

CHARSET pode ser usado como um sinônimo para **CHARACTER SET**.

Fica a dica: para a língua portuguesa, uma das opções de collations são **latin1_bin** e **latin1_swedish_ci** pertencente ao conjunto de caracteres **latin1** e para a língua inglesa temos a opção de collation **utf8_bin** encontrado no conjunto de caracteres **utf8**.

Exercícios Propostos

Crie um banco com as seguintes características:

- **NOME DO BANCO:** Agenda
- **COLLATE:** latin1
- **CHARACTER SET:** latin1_general_ci

Altere o collate da base de dados criado acima para **latin1_swedish_ci**.

Delete a base de dados usada no exercício 1 e 2.

Para que serve os comandos USE.. e SHOW DATABASES?

Qual o CHARSET e COLLATE que dar suporte ao conjunto de caracteres usados na língua portuguesa?

5.0 Manipulando tabelas

Um servidor de banco de dados é dividido em bases de dados com o intuito de separar as informações de domínios diferentes. Nessa mesma linha de raciocínio, podemos dividir os dados de uma base a fim de agrupá-los segundo as suas correlações. Essa separação é feita através de tabelas. Por exemplo, no sistema de um banco, é interessante separar o saldo e o limite de uma conta, do nome e CPF de um cliente. Então, poderíamos criar uma tabela para os dados relacionados às contas e outra para os dados relacionados aos clientes.

Cliente			Conta		
nome	idade	cpf	numero	saldo	limite
José	27	31875638735	1	1000	500
Maria	32	30045667856	2	2000	700

Tabela 5.0: Tabelas para armazenar os dados relacionados aos clientes e às contas

Uma tabela é formada por **registros (linhas)** e os registros são formados por **campos (colunas)**. Por exemplo, considere uma tabela para armazenar as informações dos clientes de um banco. Cada registro dessa tabela armazena em seus campos os dados de um determinado cliente.

Observação-5.0: Antes de utilizar os comandos abaixo, devemos selecionar uma base de dados em que você deseja trabalhar com a manipulação das tabelas. Para fazer essa seleção usamos o comando **USE**, estudado no capítulo 4 - tópico 4.3 . Todos as instruções abaixo foram escritas no MySQL Client.

5.1 Criando tabelas no MySQL Server

As tabelas no MySQL Server são criadas através do comando **CREATE TABLE**. Na criação de uma tabela, é necessário definir quais são os nomes e os tipos das colunas. Abaixo temos o comando básico para a criação de uma tabela:

```
CREATE TABLE
nome_da_tabela (nome_campo1 tipo_campo1, nome_campo2 tipo_campo2);
```

Legenda:

- **nome_da_tabela** < nome da tabela que você pretende criar;
- **nome_campo1** < nome do primeiro campo/coluna da tabela;
- **tipo_campo1** < tipo de dado que o campo/coluna da tabela irá armazenar;

Vamos criar uma tabela de nome **Livro** e com os campos de nome **nome_cod_livro** do tipo

```
mysql> CREATE TABLE Livro( cod_livro INTEGER, nome_livro VARCHAR(100) );
Query OK, 0 rows affected (0.09 sec)
```

INTEGER e **nome_livro** do tipo **VARCHAR**.

Terminal 5.0: Criando uma tabela.

Ainda, é possível criar uma tabela sem ativar/selecionar um banco de dados (ver **Observação-5.0 no início do capítulo**), contudo, é necessário informar no comando **CREATE TABLE** o nome do banco de dados e o nome da tabela, ficando o comando **CREATE TABLE** dessa forma:

```
CREATE TABLE nome_do_banco.nome_da_tabela
(nome_campo1 tipo_campo1, nome_campo2 tipo_campo2);
```

Legenda:

nome_do_banco < nome do banco onde será criado a tabela;

- < separa o nome do banco de dados e o nome da tabela que será criada;

Vamos criar a mesma tabela **Livro** criada no exemplo anterior, mas agora sem selecionar a base de dados

```
mysql> CREATE TABLE livraria.Livro( cod Integer, nome_livro varchar(100) );
Query OK, 0 rows affected (0.08 sec)
```

(ver **Observação-5.0 no início do capítulo**).

Terminal 5.1: Cria na base de dados livraria a tabela Livro.

5.2 Listando tabelas no MySQL Server

As tabelas de uma base de dados podem ser listadas através do comando:

```
SHOW TABLES;
```

Antes de utilizar esse comando (ver **Observação-5.0 no início do capítulo**).

Terminal 5.2: Listando as tabelas de uma base de dados.

```
mysql> USE livraria;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_livraria |
+-----+
| Livro |
+-----+
1 row in set (0.00 sec)
```

5.3 Alterando uma tabela no MySQL Server

Podemos alterar a estrutura de uma tabela e suas propriedades com o comando **ALTER TABLE**.

```
ALTER TABLE nome_da_tabela RENAME novo_nome_tabela;
```

5.3.1 Renomeando uma tabela com o comando ALTER TABLE.

Legenda:

nome_da_tabela < nome da tabela que você deseja alterar;

novo_nome_tabela < novo nome da tabela;

```
mysql> ALTER TABLE Livro RENAME livros;
Query OK, 0 rows affected (0.00 sec)
```

Exemplo:

Terminal 5.3: Alterando o nome da tabela.

5.3.2 Adicionando uma nova coluna com o comando ALTER TABLE.

```
ALTER TABLE nome_da_tabela ADD nome_do_novo_campo tipo_do_novo_campo;
```

Legenda:

nome_da_tabela < nome da tabela que você deseja adicionar o novo campo;

nome_do_novo_campo < nome do campo a ser adicionado a tabela;

tipo_do_novo_campo < tipo do campo que sera adicionado a tabela;

Exemplo:

```
mysql> ALTER TABLE Livro ADD paginas INTEGER;
Query OK, 0 rows affected (0.00 sec)
```

Terminal 5.4: Adicionando uma coluna.

5.3.3 Removendo uma coluna com o comando ALTER TABLE.

```
ALTER TABLE nome_da_tabela DROP COLUMN nome_da_coluna_deletada;
```

Legenda:

nome_da_tabela < nome da tabela que você deseja adicionar o novo campo;

nome_da_coluna_deletada < nome da coluna a ser deletada na tabela;

Exemplo:

```
mysql> ALTER TABLE Livro DROP COLUMN paginas;
Query OK, 0 rows affected (0.00 sec)
```

Terminal 5.5: Removendo uma coluna.

5.3.4 Renomeando um campo com o comando ALTER TABLE.

```
ALTER TABLE nome_da_tabela CHANGE nome_campo novo_nome_campo tipo;
```

Legenda:

nome_da_tabela < nome da tabela que você deseja alterar;

nome_campo < nome do campo que será alterado;

novo_nome_campo < novo nome do campo;

tipo < tipo do campo;

```
mysql> ALTER TABLE Livro CHANGE cod id_livro INTEGER;
```

Exemplo:

Terminal 5.6: Renomeando campo cod da tabela Livro para id_livro

5.4 Deletando uma tabela no MySQL Server

```
DROP TABLE nome_da_tabela;
```

Se uma tabela não for mais desejada, ela pode ser removida através do comando **DROP TABLE**.

Legenda:

nome_da_tabela < nome da tabela que você deseja deletar;

```
mysql> DROP TABLE Livro;
Query OK, 0 rows affected (0.00 sec)
```

Terminal 5.7: Eliminando tabela**5.5 Saiba mais...****5.5.1 Tipos de dados do MySQL**

Ao criar uma tabela você deverá especificar o tipo de dados a ser armazenado nela. O MySQL possui três tipos de dados básicos: **numéricos, data/hora e string**.

Tipos de dados numéricos

TIPO	INTERVALO	bytes	DESCRIÇÃO
TINYINT[(M)]	-127 a 128; ou 0 a 255	1	inteiros muitos pequenos
BIT			o mesmo que TINYINT
BOOL			o mesmo que TINYINT
SMALLINT[(M)]	-32768 a 32767	2	inteiros pequenos
MEDIUMINT[(M)]	-8388608 a 8388607; ou 0 a 16777215	3	inteiros de tamanho médio
INT[(M)]	-213 a 231-1; ou 0 a 232-1	4	inteiros regulares
INTEGER[(M)]			o mesmo que INT
BIGINT[(M)]	-263 a 263-1; ou 0 a 264-1	8	inteiros grandes
FLOAT(precisão)	depende da precisão	variável	números de ponto flutuante de precisão simples ou dupla
FLOAT[(M,D)]	1.175494351E-38 a ±3.402823466E+38	4	números de ponto flutuante de precisão simples. O mesmo que FLOAT(4)
DOUBLE[(M,D)]	±1.7976931348623157 E +308 a ±2.2250738585072014	8	números de ponto flutuante de precisão dupla. O mesmo que FLOAT(8)

	E -308		
DOUBLE			O mesmo que DOUBLE[(M,D)]
PRECISION[(M,D)]			O mesmo que DOUBLE[(M,D)]

REAL[(M,D)]			O mesmo que DOUBLE[(M,D)]
DECIMAL[(M,D)]	variável	M+2	número de ponto flutuante armazenado como char
NUMERIC[(M,D)]			O mesmo que DECIMAL
DEC[(M,D)]			O mesmo que DECIMAL

OBSERVAÇÕES:

- As opções entre colchetes ([e]) são opcionais;
- Dentre os tipos que se ajustam aos dados a serem inseridos, escolha sempre o de menor tamanho;
- Para dados do tipo inteiro você pode usar a opção **UNSIGNED** para especificar inteiros positivos ou zero;
- **M** especifica o tamanho máximo de exibição;
- **D** especifica o número de casas decimais. O valor máximo de D é 30 ou M-2;
- Tanto para números inteiros como para números de ponto flutuante você pode especificar a opção **ZEROFILL** que preenche os números com zeros iniciais. Colunas especificadas com ZEROFILL são automaticamente configuradas como UNSIGNED;

Tipos de dados data/hora

TIPO	INTERVALO	DESCRIÇÃO
DATE	1000-01-01 a 9999-12-31	data. Exibido como YYYY-MM-DD
TIME	-838:59:59 a 838:59:59	hora. Exibido como HH:MM:SS
DATETIME	1000-01-01 00:00:00 a 9999-12-31 23:59:59	data e hora. Exibido como YYYY-MM-DD HH:MM:SS
TIMESTAMP[(M)]	1970-01-01 00:00:00 a algum momento em 2037. Dependendo do limite do sistema operacional	<p>registro de data e hora útil para transações. Os formatos de exibição podem ser:</p> <p>TIMESTAMP YYYYMMDDHHMMSS TIMESTAMP(14) YYYYMMDDHHMMSS TIMESTAMP(12) YYMMDDHHMMSS TIMESTAMP(10) YYMMDDHHMM TIMESTAMP(8) YYYYMMDD TIMESTAMP(6) YYMMDD TIMESTAMP(4) YYMM TIMESTAMP(2) YY</p>
YEAR[(2)]	70 a 69 (1970 a 2069)	ano
YEAR[(4)]	1901 a 2155	ano

Tipos de dados string

TIPO	INTERVALO	DESCRIÇÃO
[NATIONAL] CHAR(M) [BINARY]	0 a 255 caracteres	string de comprimento fixo M. NATIONAL especifica que o conjunto de caracteres padrão (ANSI SQL) será utilizado. BINARY especifica que os dados devem ser tratados de modo a não haver distinção entre maiúsculas e minúsculas (o padrão é distinguir).
CHAR [NATIONAL]	1 1 a 255	o mesmo que CHAR(1) string de comprimento variável
VARCHAR(M) [BINARY]	variável	string de tamanho variável. O mesmo que [BINARY].
TINYBLOB	0 a 28 - 1 (255)	BLOB pequeno
TINYTEXT	0 a 28 - 1 (255)	TEXT pequeno
BLOB	0 a 216 - 1 (65535)	BLOB normal
TEXT	0 a 216 - 1 (65535)	TEXT normal
MEDIUMBLOB	0 a 224 - 1 (16777215)	BLOB médio
MEDIUMTEXT	0 a 224 - 1 (16777215)	TEXT médio
LONGBLOB	0 a 232 - 1 (4294967295)	BLOB longo
LONGTEXT	0 a 232 - 1 (4294967295)	TEXT longo
ENUM('valor1','valor2',...)	0 a 65535	armazenam um dos valores listados ou NULL
SET('valor1','valor2',...)	0 a 64	armazenam um ou mais dos valores listados ou NULL

OBSERVAÇÕES:

- CHAR e VARCHAR armazenam strings de comprimento fixo e variável respectivamente. VARCHAR trabalha mais lento.
- TEXT e BLOB armazenam textos grandes ou objetos binários (figuras, som, etc.). TEXT diferencia maiúsculas de minúsculas.

5.5.2 Tipo de tabelas/ Storage Engine/ Motor de Armazenamento do MySQL

O MySQL possui uma característica um pouco diferente dos outros sistemas gerenciadores de banco de

dados, uma vez que no MySQL é possível escolher o tipo da tabela no momento da criação da mesma.

O formato de armazenamento dos dados, bem como alguns recursos do banco de dados são dependentes do tipo de tabela escolhido. Para sabermos quais tipos de tabelas o MySQL instalado na sua máquina nos permite criar usamos o comando:

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	YES	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL
ARCHIVE	YES	Archive storage engine	NO	NO	NO
MyISAM	DEFAULT	Default engine as of MySQL 3.23 with great performance	NO	NO	NO

8 rows in set (0.00 sec)

Terminal 5.7: comando SHOW ENGINES;

Perceba pela figura Terminal 5.7 que o MySQL nos fornece flexibilidade na forma de armazenamento dos nossos dados, isso é uma característica forte do MySQL. Desses Storage Engines suportados por meu MySQL, vamos explanar somente dois deles, no geral os mais usados, são eles o MyISAM e o INNODB.

- **MyISAM**

O tipo de tabela ou Storage Engine MyISAM é o mecanismo instalado e configurado como padrão pelo MySQL até a versão inferior a 5.5, justamente por apresentar um dos melhores resultados em todos os aspectos possíveis.

É um método de armazenamento muito rápido, de bom armazenamento em disco, sem restrições de uso de tipos de dados, que permite o uso de todos os recursos do MySQL, com exceção do suporte a transações

A menos que a necessidade seja o uso de transações, este é o método indicado para a maioria dos casos, desde pequenos até grandes bancos de dados. É o único tipo de tabela do MySQL que suporta buscas do tipo Fulltext Searches.

Outra característica é que o seu nível de bloqueio(LOCK / UNLOCK) é de tabela, não tão prático quanto os níveis mais aprofundados. Contudo, como não utiliza o suporte a transações, o número geral de solicitações de bloqueio é menor, mantendo a boa disponibilidade deste tipo de Storage Engine.

Recomendado para sistemas que priorizam velocidade na pesquisa de dados.

Nome de uso: MyISAM **Suporte a índices:** SIM **Suporte a transações:** Não

Tipos de dados não suportados: Nenhum

Nível de bloqueio: Tabela

- **InnoDB**

Já o tipo de tabela InnoDB passou a ser a padrão a partir da versão 5.5 do MySQL, substituindo a Storage Engine MyISAM que até então era a padrão. O InnoDB é o mais recomendado para grandes e complexos bancos de dados, pois além de oferecer todos os recursos disponibilizados pelo método MyISAM, ainda permite o uso de transações com as propriedades ACID (Atomicidades, Consistência, Isolamento e Durabilidade).

Desenvolvido pela Inobase Oy, além do armazenamento de dados e índices em disco, algumas dessas informações são armazenadas também na memória enquanto o servidor está ativo, fazendo com que o seu processamento seja ainda mais veloz.

Outra característica que o diferencia do método MyISAM é que seu nível de bloqueio

(LOCKS) é de linha, sendo mais eficiente e aumentando sua disponibilidade, pois apenas os registros comprometidos em uma transação são bloqueados, e não a tabela toda.

Recomendado para todos os tamanhos de banco de dados, especialmente os de grande porte, além de ser o ideal para sistemas em que os dados estão em constante mudanças.

Nome de uso: InnoDB

Suporte a índices: Sim

Suporte a transações: Sim (ACID)

Tipos de dados não suportados: Nenhum

Nível de bloqueio: Linha

Exercícios Propostos

Crie uma base de dados com as seguintes características:

NOME DO BANCO: Vendas

CHARSET: latin1

COLLATE: latin1_general_ci

Selecione essa base de dados e crie uma tabelas com as seguintes características:

NOME DA TABELA: Vendedor

ENGINE: InnoDB

Campos:

cod – Integer – NOT NULL – AUTO_INCREMENT

nome – Varchar(50) – NOT NULL telefone – Varchar(15) – NOTNULL dataNascimento – DATE

Adicione um novo campo horaInicioTrabalho que será do tipo TIME a tabela criada na questão anterior.

Altere o campo **nome** para **nomeVendedor - varchar(60)** da tabela criada na questão 02;

6.0 CRUD

6.1 CRUD (Create, Read, Update e Delete)

As operações básicas para manipular os dados persistidos são: inserir, ler, alterar e remover.

Essas operações são realizadas através de uma linguagem de consulta denominada SQL (Structured Query Language). Essa linguagem oferece quatro comandos básicos: INSERT, SELECT, UPDATE e DELETE. Esses comandos são utilizados para inserir, ler, alterar e remover registros, respectivamente.

Os tópicos a seguir irão usar o banco de dados chamado **Livraria** e a tabela **Livro** que tem a seguinte estrutura:

titulo	preco
...	...
...	...
...	...

Tabela-6.0: Tabela Livro composta pelos campos título e preço

6.2 INSERT

A instrução **INSERT** é usada para inserir dados em uma tabela. Sintaxe básica do comando:

```
INSERT INTO nome_da_tabela ( nome_campo1, nome_campo2, ... nome_campoN )
VALUES
    ( valor_inserir_campo1, valor_inserir_campo2, ... valor_inserir_campoN );
```

Legendas:

nome_tabela < nome da tabela onde será inserido os dados;

nome_campo1 < nome do campo onde será inserido os dados;

valor_inserir_campo1 < valor que será inserido no campo;

Vamos para um exemplo. O comando abaixo insere os dados **Java**, **98.75** respectivamente nos campos **título**, **preço** da tabela **Livro**.

```
mysql> INSERT INTO Livro (titulo, preco) VALUES ('Java', 98.75);
Query OK, 1 row affected (0.00 sec)
```

6.3 SELECT

A instrução **SELECT** é usada para selecionar dados em uma tabela, iremos estudar a fundo esse comando nos próximos capítulos.

```
SELECT nome_campo1, nome_campo2, ... nome_campoN  
      FROM nome_tabela  
      [WHERE condicao];
```

Terminal - 6.0: Inserindo um registro.

Sintaxe básica do comando:

Legendas:

[...] < o comando que está entre colchetes é opcional;

condição < somente registros que obedecerem a condição serão selecionados, iremos estudar algumas condições lógicas mais a frente;

... nome_campo1, nome_campo2, ... nome_campoN... < nome dos campos que serão selecionados.

Caso seja digitado o caracter * no lugar de *... nome_campo1, nome_campo2, ... nome_campoN...* , todos os campos da respectiva tabela serão selecionados;

nome_tabela < nome da tabela onde se encontra os dados a serem buscados;

Abaixo um exemplo de como usar o comando SELECT para selecionar todos os campos da tabela Livro e todos os seus registros.

```
mysql> SELECT * FROM Livro;  
+-----+-----+  
| titulo | preco |  
+-----+-----+  
| Java   | 98.75 |  
+-----+-----+  
1 row in set (0.00 sec)
```

Terminal - 6.1: Selecionando todos os registros da tabela SELECT.

6.4 UPDATE

A instrução **UPDATE** é usada para atualizar dados em uma tabela. Para atualizar um registro em sua tabela usamos o comando abaixo:

```
UPDATE nome_da_tabela SET nome_do_campo = valor WHERE condicao;
```

Ou

```
UPDATE nome_da_tabela  

SET nome_do_campo1 = valor1, nome_do_campo2 = valor2, nome_do_campo3 = valor3  

WHERE condicao;
```

Legendas:

nome_da_tabela < nome da tabela onde o dado será atualizado;

nome_do_campo < nome dos campo que será alterado;

condição < somente os registros ou linhas que obedecem a(s) condição(ões) serão atualizados;

```
mysql> UPDATE Livro SET preco = 115.9 WHERE titulo = 'Java';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1    Changed: 1    Warnings: 0
```

Terminal-6.2: Alterando registro da tabela Livro.

```
mysql> SELECT * FROM Livro;
+-----+-----+
| titulo | preco |
+-----+-----+
| Java   | 115.9 |
+-----+-----+
1 row in set (0.00 sec)
```

6.5 DELETE

A instrução **DELETE** é usada para excluir dados em uma tabela. Para excluir um registro em sua tabela usamos o comando abaixo:

```
DELETE FROM nome_da_tabela WHERE condicao;
```

Terminal-6.3: Selecionando registros.

Legendas:

nome_da_tabela < nome da tabela onde será pesquisado o registro a ser excluído;

condição < somente os registros ou linhas que obedecerem a(s) condição(ões) serão deletados, caso não exista condição todos os registros da tabela serão deletados.

Exemplo:

```
mysql> DELETE FROM Livro WHERE titulo = 'Java';
Query OK, 1 row affected (0.00 sec)
```

Terminal 6.4: Removendo registros.

```
mysql> SELECT * FROM Livro;
Empty set (0.00 sec)
```

Terminal-6.5: Selecionando registros.

Crie uma tabela chamada categoria com os campos id do tipo Integer não nulo e auto_increment e outro campo do tipo varchar (50), não nulo que se chamará nomeCategoria.

Usando o comando Insert insira os dados abaixo na tabela categoria criada na questão 1:

id	nomeCategoria
1	Visitante
2	Registrado
3	Administrator
4	I am best

Altere o valor que tem no campo nomeCategoria do registro que tem id=3 para admin e altere o registro que tem id=4 para root.

Exclua o registro que tem o id=1;

Referências

<http://php.net/docs.php/>

<http://www.php.com.br/>

<http://www.planetphp.net/>

Use a Cabeça Php & Mysql , Beighley, Lynn; Morrison, Michael - Editora: Alta Books

•**Apostila HTML e CSS – Projeto e-Jovem**

•<http://jcmaxwell1.vi labol.uol.com.br/Html/texto.html>

• <http://www.itexto.net/devkico/?p=671>

Hino Nacional

Ouviram do Ipiranga as margens plácidas
De um povo heróico o brado retumbante,
E o sol da liberdade, em raios fúlgidos,
Brilhou no céu da pátria nesse instante.

Se o penhor dessa igualdade
Conseguimos conquistar com braço forte,
Em teu seio, ó liberdade,
Desafia o nosso peito a própria morte!

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, um sonho intenso, um raio vívido
De amor e de esperança à terra desce,
Se em teu formoso céu, risonho e límpido,
A imagem do Cruzeiro resplandece.

Gigante pela própria natureza,
És belo, és forte, impávido colosso,
E o teu futuro espelha essa grandeza.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Deitado eternamente em berço esplêndido,
Ao som do mar e à luz do céu profundo,
Fulguras, ó Brasil, florão da América,
Iluminado ao sol do Novo Mundo!

Do que a terra, mais garrida,
Teus risonhos, lindos campos têm mais flores;
"Nossos bosques têm mais vida",
"Nossa vida" no teu seio "mais amores."

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, de amor eterno seja símbolo
O lábaro que ostentas estrelado,
E diga o verde-louro dessa flâmula
- "Paz no futuro e glória no passado."

Mas, se ergues da justiça a clava forte,
Verás que um filho teu não foge à luta,
Nem teme, quem te adora, a própria morte.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Hino do Estado do Ceará

Poesia de Thomaz Lopes
Música de Alberto Nepomuceno
Terra do sol, do amor, terra da luz!
Soa o clarim que tua glória conta!
Terra, o teu nome a fama aos céus remonta
Em clarão que seduz!
Nome que brilha esplêndido luzeiro!
Nos fulvos braços de ouro do cruzeiro!

Mudem-se em flor as pedras dos caminhos!
Chuvas de prata rolem das estrelas...
E despertando, deslumbrada, ao vê-las
Ressoa a voz dos ninhos...
Há de florar nas rosas e nos cravos
Rubros o sangue ardente dos escravos.
Seja teu verbo a voz do coração,
Verbo de paz e amor do Sul ao Norte!
Ruja teu peito em luta contra a morte,
Acordando a amplidão.
Peito que deu alívio a quem sofria
E foi o sol iluminando o dia!

Tua jangada afoita enfune o pano!
Vento feliz conduza a vela ousada!
Que importa que no seu barco seja um nada
Na vastidão do oceano,
Se à proa vão heróis e marinheiros
E vão no peito corações guerreiros?

Se, nós te amamos, em aventuras e mágoas!
Porque esse chão que embebe a água dos rios
Há de florar em meses, nos estios
E bosques, pelas águas!
Selvas e rios, serras e florestas
Brotam no solo em rumorosas festas!
Abra-se ao vento o teu pendão natal
Sobre as revoltas águas dos teus mares!
E desfraldado diga aos céus e aos mares
A vitória imortal!
Que foi de sangue, em guerras leais e francas,
E foi na paz da cor das hóstias brancas!



GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação