

Sumário

Prefácio.....	7
Capítulo 1 - Lógica de Programação.....	8
1.1 - Noções de Lógica.....	8
1.2 - O que é Lógica.....	8
1.3 - A real lógica do dia a dia.....	9
1.4 - Lógica de Programação.....	10
1.5 - O que é um algoritmo.....	10
1.6 - Porque é importante construir um algoritmo.....	11
1.7 - Maneiras de Representar um algoritmo.....	12
1.7.1 - Descrição Narrativa.....	12
1.7.2 - Fluxograma Convencional.....	13
1.7.3 - Pseudocódigo.....	14
1.8 - Exercícios.....	15
Capítulo 2 - Introdução ao PHP.....	17
2.1 - O que é PHP e um pouco da história.....	17
2.2 - O que precisa pra desenvolver em PHP – Instalação.....	19
2.2.1 – Instalação Apache.....	19
2.2.2 – Instalação PHP.....	21
2.3 - Características do PHP.....	22
2.4 - Saída de Dados.....	22
2.4.1 - echo	23
2.4.2 - print.....	23
2.5- Comentários.....	23
2.6 – Estrutura de Dados.....	24
2.6.1 – Tipos primitivos de dados.....	24
2.6.3 - Exercícios.....	25
2.6.3.1 - Práticos.....	26
2.7 - Variáveis.....	26
2.7.1 – Algumas regras sobre variáveis.....	27
2.7.2 – Tipos de Variáveis.....	29
2.7.2.1 - Boolean(lógico).....	29
2.7.2.2 – Int (inteiro).....	30
2.7.2.3 – Float(Real).....	30
2.7.2.4 – String(cadeia de caracteres).....	30
2.7.2.5 – Array(Matrizes).....	31
2.7.2.6 – Object(Objeto).....	31
2.7.2.7 – Resource(Recurso).....	31
2.7.2.8 - Null(Nulo).....	31
2.7.3 - Conversão de Variáveis.....	31
2.7.4 - Exercícios.....	33
2.8 - Constantes.....	34
2.8.1 – Constantes Pré-definidas.....	34
2.8.2 – Definindo Constantes.....	34
2.8.3 - Exercícios.....	35
2.9 - Arrays.....	35

2.9.1 – Criando array Unidimensional.....	35
2.9.2 - Criando Array Associativo.....	37
2.9.3 - Arrays Multidimensionais.....	38
2.9.4 - Exercícios.....	39
2.10 - Operadores.....	40
2.10.1 – Operadores de String	40
2.10.2 - Operadores Matemáticos.....	41
2.10.2.1 – Operadores Aritméticos.....	41
2.10.2.2 - Atribuição.....	42
2.10.2.3 - Operadores de Incremento e Decremento.....	44
2.10.3 – Operadores Relacionais.....	44
2.10.4 – Operadores Lógicos ou Booleanos.....	46
2.10.5 – Precedência de Operadores.....	48
2.10.6 – Exercícios.....	50
2.11 - Estruturas de Controle.....	51
2.11.1 – Estrutura Sequencial.....	51
2.11.2 – Estruturas de Seleção/Decisão.....	51
2.11.2.1 - Simples(IF).....	52
2.11.2.2 - Composta(IF-ELSE).....	52
2.11.2.3 - Encadeada(IF-ELSE-IF-IF).....	53
2.11.2.4 - Múltipla Escolha(SWITCH-CASE).....	53
2.11.2.5 - Exercícios.....	55
2.11.3 - Estruturas de Repetição.....	55
2.11.3.1 - Condição no Início(WHILE).....	56
2.11.3.2 - Condição no Final(DO-WHILE).....	56
2.11.3.3 - Variável de Controle(FOR).....	57
2.11.3.4 – Exercícios.....	58
2.11.3.5 - Percorrer Arrays(Foreach).....	58
2.11.4 - Break.....	59
2.11.5 - Continue.....	60
2.11.6 - Exercícios.....	62
Capítulo 3 – Programando em PHP.....	63
3.1 - Interação PHP – HTML.....	63
3.1.1 - Revisão HTML - Formulários.....	63
3.1.1.1 - Elementos de um formulário.....	63
3.1.2 - Recebendo dados de Formulários(\$_GET - \$_POST).....	65
3.1.2.1 – Método GET.....	66
3.1.2.2 – Método POST.....	67
3.1.3 - Exercícios.....	67
3.2 - Especialidades do PHP	68
3.2.1 - Functions.....	68
3.2.1.1 – Declarando uma função.....	68
3.2.1.2 - Escopo de Variáveis em Funções.....	69
3.2.1.3 - Passagem de Parâmetro.....	70
3.2.1.4 - Valor de Retorno.....	71
3.2.1.5 - Recursão.....	71
3.2.1.6 - Funções nativas.....	71
3.2.1.7 – Exercícios.....	72

3.2.2 - Arrays Super Globais.....	72
3.2.2.1 - \$_COOKIES.....	72
3.2.2.2 - \$_SERVER.....	75
3.2.2.3 - \$_SESSION.....	75
3.2.2.4 - \$GLOBALS.....	76
3.2.2.5 – Outros Super Globais.....	77
3.2.2.6 - Exercícios.....	77
3.2.3 - Manipulação de Arquivos e Diretórios.....	77
3.2.3.1 - Criando e Abrindo um Arquivo.....	77
3.2.3.2 - Gravando em um arquivo.....	78
3.2.3.3 - Fechando um arquivo.....	79
3.2.3.4 - Lendo um arquivo.....	80
3.2.3.5 - Renomeando e Apagando um Arquivo.....	80
3.2.3.6 - Manipulando Diretório.....	82
3.2.3.7 – Exercícios.....	83
3.2.4 - Requisição de Arquivos.....	84
3.2.4.1 - INCLUDE.....	84
3.2.4.2 - REQUIRE.....	84
3.2.4.3 - INCLUDE_ONCE.....	85
3.2.4.4 - REQUIRE_ONCE.....	85
Capítulo 4 – PHP Orientado à Objetos.....	86
4.1 - Introdução a OO.....	86
4.1.1 - Como funciona? Pra que serve? Em que melhora?.....	87
4.1.2 - Classes.....	87
4.1.3 – Construindo uma classe em PHP.....	88
4.1.3.1 - Atributos.....	89
4.1.3.2 - Métodos.....	89
4.1.4 – Métodos GET e SET.....	89
4.1.4.1 - Método de Armazenamento de Atributos(SET)	90
4.1.4.2 - Método de Retorno de Atributos(GET).....	90
4.1.4.3 – Incluindo os Métodos GET e SET.....	91
4.1.5 - Instanciando a classe criada	91
4.2 - 4 pilares da OO.....	93
4.2.1 - Herança.....	93
4.2.1.1 - Extends.....	94
4.2.2 - Encapsulamento.....	96
4.2.3 - Polimorfismo.....	96
4.2.3.1 - Sobrescrita.....	96
4.2.3.2 – Usando o Polimorfismo.....	97
4.2.4 - Abstração.....	98
4.2.4.1 – Classe Abstrata.....	98
4.3 – Incrementando a OO.....	99
4.3.1 – Variáveis Estáticas.....	99
4.3.2 - Métodos Estáticos.....	99
4.3.3 - Exercícios	100
4.3.4 – Métodos Mágicos.....	101
4.3.4.1 - __construct.....	101
4.3.4.2 - __destruct.....	101

4.3.4.3 - <code>_toString</code>	102
4.3.4.4 - <code>_invoke</code>	102
4.4.5 - Interface.....	103
Capítulo 5 – Banco de Dados(MySQL).....	104
5.1 - O que é o MYSQL?.....	104
5.2 - Trabalhando com MYSQL.....	104
5.3 - Estruturas de Dados.....	105
5.4 - Criando Banco e Tabelas.....	106
5.4.1 - CREATE DATABASE.....	106
5.4.2 - CREATE TABLE.....	107
5.4.3 - DROP.....	107
5.5 - Manipulando dados das tabelas.....	108
5.5.1 - INSERT.....	108
5.5.2 - SELECT.....	108
5.5.3 - UPDATE.....	108
5.5.4 - DELETE.....	109
5.5.5 - COMANDOS ADICIONAIS.....	109
5.6 - Trabalhando com PhpMyAdmin.....	110
5.7 - Manipulando banco de dados no PhpMyadmin.....	111
5.8 – Exercícios.....	114
5.9 – Aprofundando no MySQL.....	115
5.9.1 – Usando SELECT em duas ou mais tabelas.....	115
5.9.2 – Usando o poder do WHERE.....	117
5.9.2.1 - Comparação de igualdade “=”	117
5.9.2.2 – Comparação de Diferença “!=” ou “<>”	117
5.9.2.3 – Comparação Menor ou igual “<=”	117
5.9.2.4 – Comparação Menor “<”	117
5.9.2.5 – Comparação Maior ou igual “>=”	118
5.9.2.6 – Comparação Maior “>”	118
5.9.2.7 – Comparação booleana “IS”	118
5.9.2.8 – Comparação booleana “IS NOT”	118
5.9.2.9 – Comparação booleana “IS NULL”	118
5.9.2.9 – Comparação booleana “IS NOT NULL”	118
5.9.2.10 – Comparação estão entre “BETWEEN...AND”	118
5.9.2.11 – Comparação não estão entre “NOT BETWEEN... AND”.....	118
5.9.2.12 – Comparação string inicial “LIKE”	119
5.9.2.13 – Comparação não é string inicial “NOT LIKE”	119
5.9.2.14 – Comparação “IN()”	119
5.9.2.15 – Comparação “NOT IN()”	119
5.9.2.16 – Operador “NOT” ou “!”	119
5.9.2.17 – Operador “AND” ou “&&”	119
5.9.2.18 – Operador “OU” ou “ ”	120
5.9.2.19 – Operador “XOR”	120
5.9.3 - ALIAS.....	120
5.9.4 – Exercícios.....	121
5.9.5 – Funções de Agrupamento.....	122
5.9.6 – GROUP BY.....	123
5.9.7 – DISTINCT.....	125

5.9.8 – LIMIT.....	125
5.9.9 – Índices, Otimizando Consultas.....	125
5.9.9.1 - Criando um índice	127
5.9.9.2 - Alterando um índice	128
5.9.9.3 - Removendo um índice	128
5.10 - Relacionamentos.....	128
5.10.1 - Unique.....	128
5.10.2 – Chaves Primárias.....	129
5.10.3 – Chaves Compostas.....	130
5.10.4 – Chaves Estrangeiras.....	131
5.10.5 – Relacionamento “One to One”	131
5.10.6 – Relacionamento “One to Many” ou “Many to One”	132
5.10.7 – Relacionamento “Many to Many”	133
Capítulo 6 – UML.....	134
6.1 – Pra que serve a UML?.....	134
6.1.1 – Visualização.....	134
6.1.2 - Especificação.....	134
6.1.3 - Construção.....	134
6.1.4 - Documentação.....	134
6.1.5 – Razões para Modelar.....	134
6.2 – Tipos de Diagrama.....	135
6.2.1 – Diagrama de Atividade.....	135
6.2.2 – Diagrama de Sequência.....	135
6.2.3 – Diagrama de Caso de Uso.....	135
6.2.4 – Diagrama de Classe.....	135
6.2.5 – Diagrama de Estado.....	135
6.3 - Documentação do Sistema.....	136
Capítulo 7 – MVC.....	137
7.1 – O que é? Pra que serve?	137
7.2 - Organização.....	137
7.2.1 - View.....	138
7.2.2 - Control.....	138
7.2.3 - Model.....	138
7.3 – Usando o MVC na prática.....	138
Capítulo 8 – Interação PHP & MySQL.....	140
8.1 – Conexão com MySQL.....	140
8.1.1 – mysql_connect ou mysql_pconnect.....	140
8.1.2 - mysql_select_db.....	141
8.1.3 – Fechando conexão - mysql_close.....	141
8.2 – Enviando querys - mysql_query.....	141
8.2.1 - Consultas.....	141
8.2.2 – Demais querys.....	142
8.3 – Manipulando os Resultados.....	142
8.3.1 – mysql_result.....	142
8.3.2 - mysql_num_rows.....	142
8.3.3 – mysql_fetch_array.....	143
8.4 – Gerar Erros – mysql_error.....	143
8.5 – Outras Funções do MySQL.....	143
Posfácio	144

Prefácio

Olá Jovem, seja bem vindo(a)!

Esta apostila foi criada para ser utilizada na aprendizagem em lógica voltada para programação WEB usando a linguagem PHP Orientado a Objetos e Banco de Dados Relacional, salientando que a mesma foi adaptada e finalizada baseando-se nas apostilas do **Projeto e-Jovem**.

A mesma foi elaborada com o intuito de enriquecer cada vez mais aqueles que buscam o sucesso profissional ou até mesmo pessoal, apresentamos um material dividido em 8 capítulos, desde lógica de programação, princípios básicos até os conceitos mais avançados dentro de um contexto lógico como Orientação a Objetos e Banco de Dados Relacional, levando ao futuro JEDI, uma base média para programação WEB ou em outras linguagens.

Esperamos que esse material seja um contribuinte para todos aqueles que procuram a força e o poder supremo da Programação, e que possam aprender de forma dedicada, e que principalmente tenham um significado importante para a criação e desenvolvimento de seres com o poder sem limites.

Desejamos a todos bons estudos, guardem o que acharem bom e descartem o que acharem sem importância .

E lembrem-se de 3 coisas básicas:

- Com grandes poderes vem grandes responsabilidades (e vitórias)
- Com Programação tudo é possível basta ter as palavras certas
- E que a Força está sempre com vocês!!!

De um Simples Programador – Alessandro Feitoza

Capítulo 1 - Lógica de Programação

Neste Capítulo você entenderá como funciona a interação Homem x Máquina.
E também as formas como essa interação pode ocorrer.



1.1 - Noções de Lógica

Lógica é algo muito comum na nossa vida, a prova disso é o que fazemos durante o dia, pois mesmo sem perceber usamos a lógica a todo momento, por exemplo:

Ao acordamos:

- Tomamos um banho.
- Escovamos os dentes.
- Tomamos o café da manhã.

Para irmos ao trabalho:

- Colocamos uma roupa apropriada.
- Pegamos um ônibus.
- Chegamos no trabalho.



Perceba que podemos detalhar mais coisas a respeito do exemplo anterior, nessa sequência foi definida alguns fatos comuns.

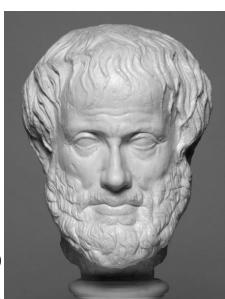
Quando queremos chegar a um objetivo final, temos que traçar uma lógica coerente que tenha sentido, onde em uma sequência lógica temos definido todos os passos que devemos seguir até o final.

1.2 - O que é Lógica

Lógica vem do grego clássico logos ($\lambda\sigma\gamma\kappa\nu$), que significa palavra, pensamento, ideia, argumento, relato, razão lógica ou princípio lógico. É uma parte da filosofia que estuda o fundamento, a estrutura e as expressões humanas do conhecimento. A lógica foi criada por Aristóteles no século IV a.C. para estudar o pensamento humano e distinguir interferências e argumentos certos e errados.

Já que o pensamento é a manifestação do conhecimento, e que o conhecimento busca a verdade, é preciso estabelecer algumas regras para que essa meta possa ser atingida.

Para que tenhamos um bom raciocínio lógico, precisamos praticar diariamente,



mudar a maneira de pensar e tomar decisões. Vamos praticar sua lógica, resolva essas questões em conjunto com a turma, procurem solucionar o problema desses casos:

DESAFIO ACEITO?



Quantos Animais de cada espécie Moisés colocou na arca?

A mãe de Maria tem 5 filhas: Lalá, Lelé, Lili, Loló, e ?

Você entra em um quarto escuro(e sem iluminação) com uma caixa de fosfóro, uma vela, e uma lamparina. Que objeto você acende primeiro?

1.3 - A real lógica do dia a dia

Sempre que estamos pensando em algo, a lógica nos acompanha. Usamos a lógica quando falamos ou quando escrevemos para por em ordem o nosso pensamento que está sendo expressado. A automação é o processo em que uma tarefa deixa de ser desempenhada pelo homem e passa a ser realizada por máquinas, sejam estes dispositivos mecânicos, eletrônicos (como os computadores) ou de natureza mista. Para que a automação de uma tarefa seja bem sucedida é necessário que a máquina que irá realizá-la seja capaz de desempenhar cada uma das etapas constituintes do processo a ser automatizado com eficiência, de modo a garantir a repetição do mesmo.

Assim, é necessário que seja especificado com clareza e exatidão o que deve ser realizado em cada uma das fases do processo a ser automatizado, bem como a sequência em que estas fases devem ser realizadas.

A essa especificação da sequência ordenada de passos que deve ser seguida para a realização de uma tarefa, garantindo a sua repetibilidade, dá-se o nome de algoritmo. Ao contrário do que se pode pensar, o conceito de algoritmo não foi criado para satisfazer as necessidades da computação. Pelo contrário, a programação de computadores é apenas um dos campos de aplicação dos algoritmos. Na verdade, há inúmeros casos que podem exemplificar o uso (involuntário ou não) de algoritmos para a padronização do exercício de tarefas rotineiras. No entanto, daqui por diante a nossa atenção se volta para automação de tarefas em programação.

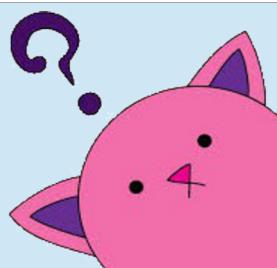


1.4 - Lógica de Programação

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, assim, ela permite definir a sequência lógica para a criação de aplicativos. Significa o uso correto das leis do pensamento, da “ordem da razão”, do processo de raciocínio e simbolização formais na programação de computadores. Com isso criam-se técnicas que cooperam para a produção de soluções logicamente válidas e coerentes, que resolvam com qualidade os problemas que se deseja programar.

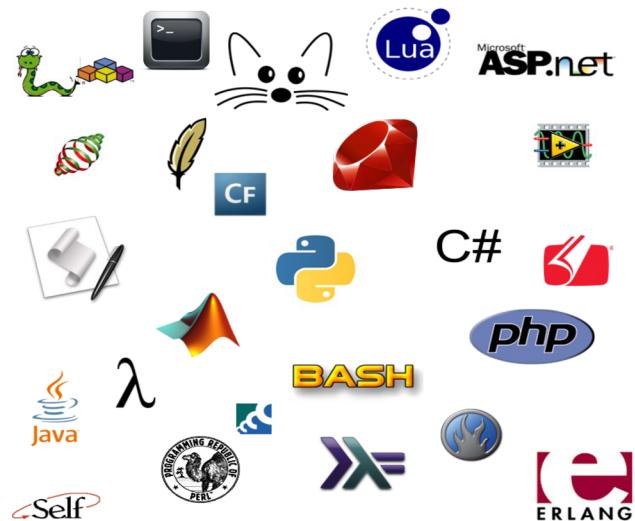
E então o que é Lógica de Programação?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.



Usar o raciocínio é algo abstrato, intangível. Os seres humanos têm a capacidade de expressá-lo através de palavras ou escrita, que por sua vez se baseia em um determinado idioma, que segue uma série de padrões(gramática). Um mesmo raciocínio pode ser expresso em qualquer um dos inúmeros, mas continuará representando o mesmo raciocínio, usando apenas outra convenção.

Em programação temos várias linguagens como, java, C, PHP, Python, ShellScript, Lua dentre muitas outras que existem, porém cada uma delas tem um padrão próprio. Essas por sua vez, são muito atreladas a uma grande diversidade de detalhes computacionais, mas com o mesmo raciocínio, seguindo as regras da linguagem, pode ser obtido o mesmo resultado.



1.5 - O que é um algoritmo

Algoritmo é uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita de bolo, uma sequência de instruções que busca uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, logo, devem ser claras e precisas.

Os algoritmos servem como modelo para programas, pois sua linguagem é intermediária a linguagem humana e as linguagens de programação, são então uma boa ferramenta na validação da lógica de tarefas a serem automatizadas. Os algoritmos servem para representar a solução de qualquer problema, mas no caso do processamento de dados, eles devem seguir as regras básicas de programação para que sejam compatíveis com as linguagens de programação. Por esse motivo, os algoritmos são independentes das linguagens. Ao contrário de uma linguagem de programação, não existe um formalismo rígido de como deve ser escrito o algoritmo. O algoritmo deve ser fácil de se interpretar e fácil de codificar, ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação a qual for definir.

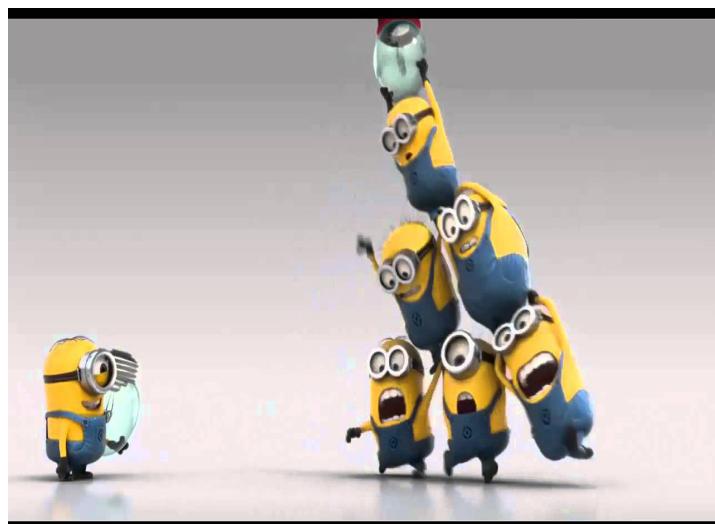
Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um DVD player, aparelhos celulares, televisores, que explicam passo a passo como operar, programar, gravar um evento, listar suas fotos,etc.

Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas.

Por exemplo:

"Trocar uma lâmpada".

- Pegar uma escada
- Pegar uma lâmpada nova.
- Subir na escada.
- Retirar a lâmpada velha.
- Colocar a lâmpada nova.
- Descer da escada.
- Guardar a escada.
- Colocar a lâmpada velha no lixo.
- Ligar o interruptor para testar.



"Somar dois números quaisquer".

- Escreva o primeiro número no retângulo A.
- Escreva o segundo número no retângulo B.
- Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C.

1.6 - Porque é importante construir um algoritmo

Construir um bom algoritmo é fundamental na formação de profissionais de informática ou computação e como tal, proporcionar condições para essa aprendizagem é um constante desafio aos professores e profissionais da área. Um algorítimo tem um

papel importante, pois ele determina de que formas podemos resolver um determinado problema, um programa de computador por exemplo, é desenvolvido com muita lógica e principalmente um conjunto de algoritmos. Outra importância da construção dos algoritmos é que uma vez concebida uma solução algorítmica para um problema, esta pode ser traduzida para qualquer linguagem de programação e ser agregada das funcionalidades disponíveis nos diversos ambientes; costumamos denominar esse processo de codificação.

1.7 - Maneiras de Representar um algoritmo

Existem diversas formas de representação de algoritmos, mas não há um consenso com relação a melhor delas para ser aplicada na resolução do problema proposto. Algumas formas de representação de algoritmos tratam os problemas apenas em nível lógico, abstraindo-se de detalhes de implementação muitas vezes relacionados com alguma linguagem de programação específica.

Por outro lado existem formas de representação de algoritmos que possuem uma maior riqueza de detalhes e muitas vezes acabam por obscurecer as ideias principais do algoritmo, dificultando seu entendimento. Sendo assim temos as seguintes formas:

- **Descrição Narrativa;**
- **Fluxograma Convencional;**
- **Pseudocódigo** (também conhecido como Português Estruturado ou “Portugol”).

1.7.1 - Descrição Narrativa

Nesta forma de representação os algoritmos são expressos diretamente em linguagem natural. Como exemplo, observe os algoritmos seguintes:

Receita de bolo:

Misture os ingredientes
Unte a forma com manteiga
Despeje a mistura na forma
Se houver coco ralado
então despeje sobre a mistura
Leve a forma ao forno
Enquanto não corar
deixe a forma no forno
Retire do forno
Deixe esfriar

Troca de um pneu furado:

Afrouxar ligeiramente as porcas
Suspender o carro
Retirar as porcas e o pneu
Colocar o pneu reserva
Apertar as porcas
Abaixar o carro
Dar o aperto final nas porcas

Esta representação é pouco usada na prática porque o uso da linguagem natural muitas vezes dá oportunidade a mas interpretações, ambiguidades e imprecisões.

Por exemplo, a instrução afrouxar ligeiramente as “porcas” no algoritmo da troca de pneus esta sujeita a interpretações diferentes por pessoas distintas. Uma instrução mais precisa seria: “afrouxar a porca, girando-a 30 graus no sentido anti-horário”.

Por mais simples que seja um algoritmo narrativo, pode haver uma grande quantidade de detalhes, que por sua vez segue um conjunto de regras dentro da sequência a qual pertencem.

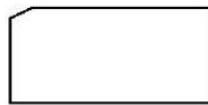
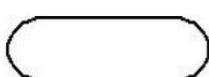
1.7.2 - Fluxograma Convencional

É uma representação gráfica de algoritmos onde formas geométricas diferentes implicam ações (instruções, comandos) distintos. Tal propriedade facilita o entendimento das ideias contidas nos algoritmos e justifica sua popularidade. Esta forma é aproximadamente intermediária a descrição narrativa e ao pseudocódigo (subitem seguinte), pois é menos imprecisa que a primeira e, no entanto, não se preocupa com detalhes de implementação do programa, como o tipo das variáveis usadas. Nota-se que os fluxogramas convencionais preocupam-se com detalhes de nível físico da implementação do algoritmo. Por exemplo, figuras geométricas diferentes são dotadas para representar operações de saída de dados realizadas em dispositivos distintos.

Existem diversas ferramentas para criação de fluxograma como Jude, Microsoft Visio, Dia (usado no linux), dentre outras, cada uma delas tem uma vantagem em particular. Trabalharemos com o editor de diagrama chamado "Dia", onde podemos facilmente instalar no Linux e sua licença é de livre acesso. O "BrOffice" também traz estes desenhos pronto para uso.

Observe as principais formas geométrica usadas para representação de Fluxogramas de algoritmos:

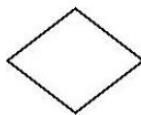
Início ou fim do fluxograma de dados Operação de entrada de dados Operação de saída



Operação de atribuição

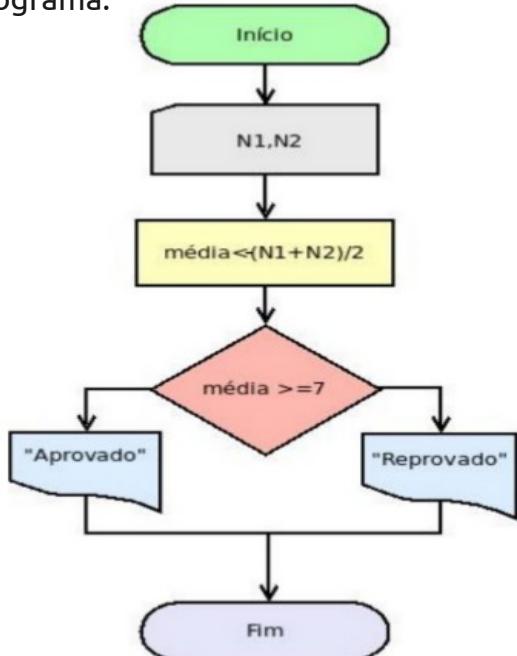


Decisão

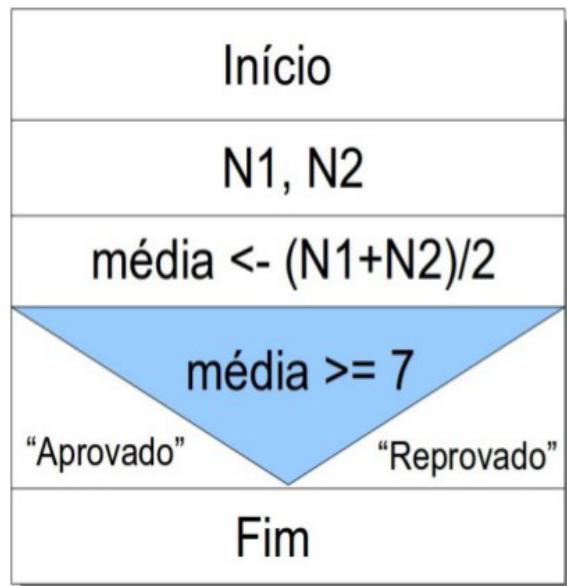


Um fluxograma se resume a um único símbolo inicial por onde a execução do algoritmo começa, e um ou mais símbolos finais, que são pontos onde a execução do algoritmo se encerra. Partindo do símbolo inicial, há sempre um único caminho orientado a ser seguido, representando a existência de uma única sequência de execução das instruções. Isto pode ser melhor visualizado pelo fato de que, apesar de vários caminhos poderem convergir para uma mesma figura do diagrama, há sempre um único caminho saindo desta. Exceções a esta regra são os símbolos finais, dos quais não há nenhum fluxo saindo, e os símbolos de decisão, de onde pode haver mais de um caminho de saída (usualmente dois caminhos), representando uma bifurcação no fluxo.

O exemplo abaixo representa um fluxograma convencional, mostrando a representação do algoritmo de cálculo da média de um aluno sob a forma de um fluxograma.



A imagem abaixo é uma representação no diagrama de Chapin. Não abordaremos esse diagrama, nessa apostila



1.7.3 - Pseudocódigo

Esta forma de representação de algoritmos é rica em detalhes, como a definição dos tipos das variáveis usadas no algoritmo. Por assemelha-se bastante a forma em que os programas são escritos, encontra muita aceitação. Na verdade, esta representação é suficientemente geral para permitir a tradução de um algoritmo nela representado para uma linguagem de programação específica seja praticamente direta. A forma geral da representação de um algoritmo na forma de pseudocódigo está ao lado:

Algoritmo é uma palavra que indica o início da definição de um algoritmo em forma de pseudocódigo.

```

algoritmo "nome" {1. denominação do programa}
var
{2. declaração e classificação das variáveis}
{3. declaração de constantes}
inicio {4. Início do bloco principal}
{5. inicialização das variáveis}
{6. solicitação de entrada de dados ao usuário}
{7. entrada de dados}
{8. processamento/cálculos}
{9. saída de informações}
fimalgoritmo {10. final do bloco principal}
  
```

I – Nome do programa: é um nome simbólico dado ao algoritmo com a finalidade de distingui-los dos demais.

II – Var: consiste em uma porção opcional onde são declaradas as variáveis globais usadas no algoritmo principal;

III – Início e Fim: são respectivamente as palavras que delimitam o início e o término do conjunto de instruções do corpo do algoritmo. O algoritmo do cálculo da média de um aluno, na forma de um pseudocódigo, fica da seguinte forma no visualg:

```
1 algoritmo "Calcular Média"
2 var
3   N1,N2,MÉDIA :real
4 inicio
5   escreval("escreva 2 numeros")
6   leia(N1,N2)
7   MÉDIA <- (N1+N2)/2
8   se(MÉDIA >=7) entao
9     escreval("aprovado")
10    senao
11      escreval("reprovado")
12    fimse
13 fimalgoritmo
```

1.8 – Exercícios

1º)Crie uma sequência lógica para sacar dinheiro em um caixa eletrônico.

2º)Crie uma sequência lógica para determinar se um numero é par ou ímpar.

3º)Crie um fluxograma para as questões um e dois.

4º)dado o fluxograma abaixo determine:

C = capital,ex: R\$ 1.000,00

T = tempo em anos, ex: 2 anos

I = Taxa de juros,ex: 5%

O usuário entra com esses valores e sabe quanto é calculado em juros simples.

a)quais as entradas de dados.

b)qual o processo que está sendo executado.

c)qual será a saída de informação se o usuário digitar 1025, 3, 12.

5º) Resolva as seguintes lógicas:

I Um time de futebol ganhou 8 jogos mais do que perdeu e empatou 3 jogos menos do que ganhou, em 31 partidas jogadas. Quantas partidas o time venceu?

II Uma pessoa supõe que seu relógio está 5 minutos atrasado, mas, na verdade, ele está 10 minutos adiantado. Essa pessoa que chega para um encontro marcado, julgando estar 15 minutos atrasada em relação ao horário combinado, chegou, na realidade, na hora certa em quantos minutos atrasados?

III Três músicos, João, Antônio e Francisco, tocam harpa, violino e piano. Contudo, não se sabe quem toca o quê. Sabe-se que o Antônio não é o pianista. Mas o pianista ensaia sozinho à Terça. O João ensaia com o Violoncelista às Quintas. Quem toca o quê?

IV – Existem três mulheres, Dona Rosa, Dona Branca, Dona Lilas, Dona branca diz: as cores de vestido que estamos usando combinam com nossos nomes, mas não estamos usando a cor do próprio nome, a pessoa que está de lilas diz: é verdade, quis as cores que cada uma está usando?

6º) Supondo que você tenha planejado no seu dia posterior ir para o colégio, após isso pagar uma conta no banco, após isso ir trabalhar, crie um algoritmo o mais detalhado possível que possa satisfazer todo o planejamento citado:

7ª) Você está numa cela onde existem duas portas, cada uma vigiada por um guarda. Existe uma porta que dá para a liberdade, e outra para a morte. Você está livre para escolher a porta que quiser e por ela sair. Poderá fazer apenas uma pergunta a um dos dois guardas que vigiam as portas. Um dos guardas sempre fala a verdade, e o outro sempre mente e você não sabe quem é o mentiroso e quem fala a verdade. Que pergunta você faria?

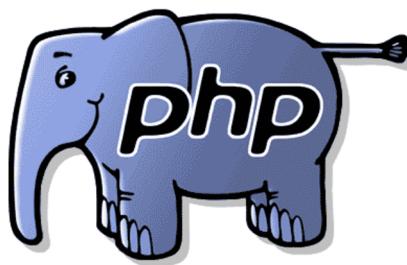
8ª) Você é prisioneiro de uma tribo indígena. Esta tribo conhece todos os segredos do Universo e portanto sabem de tudo. Você está para receber sua sentença de morte. O cacique o desafia: "Faça uma afirmação qualquer. Se o que você falar for mentira você morrerá na fogueira, se falar uma verdade você será afogado. Se não pudermos definir sua afirmação como verdade ou mentira, nós te libertaremos. O que você diria?

9º) Faça um algoritmo para imprimir um documento. Descreva com detalhes.

10º) Faça um algoritmo para colocar um quadro em uma parede. Descreva com detalhes.

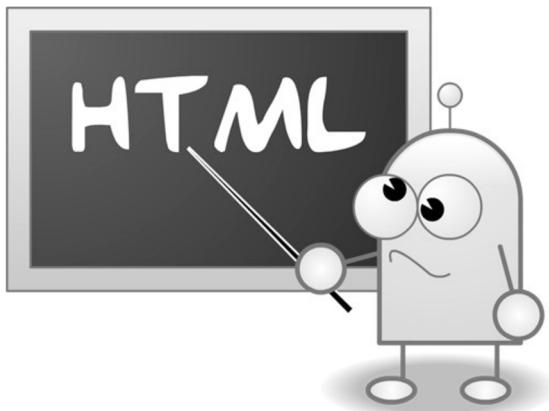
Capítulo 2 – Introdução ao PHP

Neste capítulo você conhecerá como surgiu essa poderosa linguagem. E aprenderá o básico para se trabalhar com esta linguagem.



2.1 - O que é PHP e um pouco da história

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 5.6.0, 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 deixaram 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, as 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.

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.

E como essa suprema linguagem surgiu? Quem foi o JEDI que a inventou?

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.

2.2 - O que precisa pra desenvolver em PHP – Instalação

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 explicar um passo a passo de todo o processo de instalação e configuração desses recursos no tópico seguinte.

2.2.1 – 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 é 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 nenhum LAMP instalado e configurado.

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:



It works!

This is the default web page for this server.

The web server software is running but no content has been added, yet.

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 . 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, ..., www100 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/>

2.2.2 – Instalação PHP

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 servidor 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
```

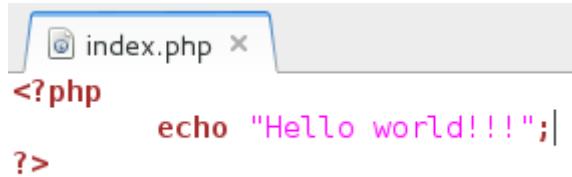
Renomeie o arquivo “index.html” para outro nome:

```
# mv index.html qualquer_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 criaremos um novo arquivo.

```
# gedit index.php
```

Após executar esse comando, digite o seguinte código no editor de texto gedit:



```
<?php  
echo "Hello world!!!";  
?>
```

Nesse momento estamos criando uma aplicação PHP que chama o comando `echo "Hello World!!!"` que tem como finalidade mostrar a frase "Hello World!!!". Lembre-se que toda operação em PHP estará dentro da Tag especial: "`<?php ?>`" Abra o Browser e digite <http://localhost/>, se tudo estiver correto, observaremos a seguinte tela:



Hello world!!!

2.3 - Características do 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 → Arquivo PHP contendo uma classe(veremos o conceito de classe mais adiante).
 - *.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.

2.4 - Saída de Dados

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: "<marquee>PHP is the Power! </marquee>", será mostrado no navegador apenas a mensagem "PHP is the Power!" passando de um lado pro outro, 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**.

2.4.1 - echo

É um comando que imprime uma ou mais variáveis ou textos, onde os mesmos são colocados em aspas simples '' ou duplas “ ”.

Sintaxe:

```
echo 'J', 'A', 'H';
```

Resultado:

JAH

2.4.2 - print

É uma função que imprime um texto na tela.

Sintaxe:

```
print ("viva o GNU/Linux");
```

Resultado:

viva o GNU/Linux

Observe um exemplo com o uso do **echo** e do **print**:

```
index.php
1 <?php
2     echo '<i>UTD / SECITECE</i><br>';
3     print ("PHP & MySQL");
4 ?>
```

UTD / SECITECE
PHP & MySQL

2.5- Comentários

Usamos os comentários para nos ajudar a lembrar de partes do código ou para orientar outros programadores ao algoritmo que está sendo escrito.

Para comentar uma única linha nos códigos PHP podemos utilizar tanto o separador “//” como também “#”, observe o exemplo abaixo:

```
index.php
1 <?php
2     //isto é um comentário
3     # e isto também...
4     # ambos serão desconsiderados pelo interpretador.
5 ?>
```

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:

```

index.php
1 <?php
2     /*
3         Script usado para testar
4         a funcionalidade do código...
5
6         echo 1+1;
7         echo '<br>';
8         echo "está funfando";
9     */
10 ?>

```



```

index.php
1 <?php
2     /**
3         * Documentação do código
4         * @author Alessandro Feitoza
5         * @version 1.0
6         * @package controls
7         * @subpackage class
8     */
9 ?>

```

2.6 – Estrutura de Dados

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.

2.6.1 – Tipos 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", "9930002", "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 em breve como esses valores serão armazenados e interpretados pelo PHP.

2.6.3 - Exercícios

1^a) Como podemos definir PHP e qual a sua relação com HTML?

2^a) Descreva um exemplo estrutural relacionando tag's HTML e delimitações PHP.

3^a) Quais aplicativos básicos podemos instalar para criarmos um servidor de páginas em PHP?

4^a) Em relação ao apache, qual a sua principal finalidade?

5^a) Quais as principais extensões de arquivos PHP e diga a diferença entre elas?

6^a) Crie dois exemplos de código PHP usando seus delimitadores.

7^a) Qual as finalidades de usar comentários dentro de um código-fonte?

8^a) Cite os principais comandos de saída usados em PHP .

9^a) Qual a diferença entre echo e print?

10^a) Observe o seguinte código e diga qual o item correto:

```
index.php
1 <?php
2 //página web.
3 echo '<h1>Olá, essa é sua primeira página!<h1>';
4 echo '<h2>Responda!</h2>' print 'O que é PHP?';
5 print 'Qual sua finalidade?'; echo '<h4>Resposta: item a!</h4>';
6 ?>
```

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.

2.6.3.1 - Práticos

11^a) 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.

12^a) 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.

2.7 - 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.

“Quando eu vejo um pássaro que caminha como um pato, nada como um pato e granya como um pato, eu chamo aquele pássaro de pato.”



James Whitcomb Riley

Para criar uma variável em PHP precisamos atribuir-lhe um nome de identificação, sempre procedido pelo caractere cifrão (\$). Observe o exemplo:

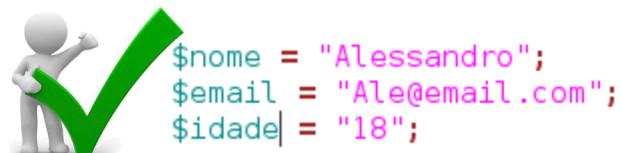
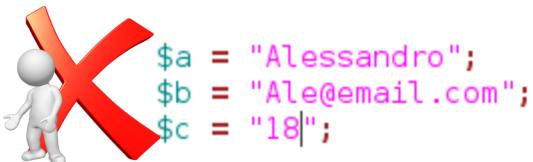
```
index.php
1 <?php
2     $nome = "Alessandro";
3     $sobre_nome = "Feitoza";
4     echo "$nome $sobre_nome";
5 ?>
```

Para imprimirmos as duas variáveis usamos aspas duplas no comando "echo", no exemplo ao lado temos a saída ao lado:

Alessandro Feitoza

2.7.1 – Algumas regras sobre variáveis

→ 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.



→ Nunca inicie a nomenclatura de variáveis com números .



→ Nunca utilize espaço em branco no meio do identificado da variável.



→ Nunca utilize caracteres especiais(! @ # ^ & * / | [] { }) na nomenclatura das variáveis.



→ Evite criar variáveis com mais de 15 caracteres em virtude da clareza do código-fonte.



```
$matricula_do_aluno_mais_inteligente_do_mundo;
```



```
$matricula_foda;
```

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 cífrão (\$) 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:

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 = 5
6  echo $b; // resultado = 10
7  ?>
8

```

```

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.

2.7.2 – 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.

2.7.2.1 - Boolean(lógico)

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 em breve, para testar a variável.

Observe:

```
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 .

2.7.2.2 – Int (íntero)

São os números que pertencem ao conjunto dos números inteiros, abrangendo valores negativos, positivos e o zero, tratam-se de valores decimais.

```
$n1 = 0;
$n2 = 42;
$n3 = -24;|
```

2.7.2.3 – Float(Real)

Os números do tipo float são números com casas decimais, onde a vírgula é substituída por um ponto(já que nos EUA não se usa "," para separar casas decimais). Vale salientar que os números do tipo float também podem ser negativos, positivos, ou zero.

Exemplo:

```
$n1 = 0.5;
$n2 = -4.2;
$n3 = 24.9765;
$n4 = 10.2e24
```

2.7.2.4 – String(cadeia de caracteres)

Uma string é uma cadeia de caracteres alfanuméricos. Para declará-las podemos utilizar aspas simples (' ') ou aspas duplas ("").

Exemplo:

```
$curso = "PHP - OO & MySQL Relacional";
$duracao = '120 horas';|
```

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:

```
$string = "\"Linux\", porque nós amamos a \\Liberdade\\\";
```

Resultado:

"Linux", porque nós amamos a \Liberdade\

2.7.2.5 – Array(Matrizes)

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. Veremos mais sobre arrays logo a seguir.

2.7.2.6 – Object(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 a bíblia do programador PHP no site <http://php.net> e pesquise sobre object.

2.7.2.7 – Resource(Recurso)

Recurso (resource) é uma tipo de dado 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. Entenderemos mais sobre esse tipo de dado quando estudarmos ***functions***.

2.7.2.8 - Null(Nulo)

Quando atribuímos um valor do tipo null (nulo) a uma variável estamos determinando que a mesma não possui valor nenhum, e que seu único valor é nulo. Exemplo:

```
$rapadura = null;
```

2.7.3 - Conversão de Variáveis

PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos de 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
<i>(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.

Exemplos:

Convertendo de ponto flutuante para inteiro.

```

1 | 1<?php
2 | 2 $a =(int)(48.56 + 145 + 15 - 0.21);
3 | 3 echo $a;
4 | 4 ?>
5 |

```

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.

2.7.4 - Exercícios

1^a) Qual a principal finalidade de uma variável?

2^a) O que significa tipagem automática.

3^a) Cite algumas dicas importantes na nomenclatura de variáveis:

4^a) Das variáveis abaixo, quais possuem nomenclaturas válidas.

\$a__b; \$a_1_; \$_início;
\$@nome; \$val_!; \$nome;
\$a_|_; \$#valor; \$palavra;
\$tele#; \$123; \$__=_;
\$VALOR_MAIOR; \$____; \$all;

Resposta:

5^a) 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;  
    ....  
?>
```

6^a) Quais os tipos de variáveis que podemos citar em PHP ?

7^a) Como podemos distinguir um tipo de variável de outro, uma vez que a tipagem é feita de forma automática em PHP ?

8^a) 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.

2.8 - Constantes

Constantes são espaços não memória que ao serem inicializados não podem ter seu valor alterado, ou seja, diferente das variáveis que podemos mudar seu valor e tipo a todo momento, nas constantes o valor e o tipo são perpétuos e inalteráveis.

2.8.1 – 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.

2.8.2 – 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: **define("NOME_DA_CONSTANTE", "valor da constante")**

Exemplo:

```
define("LINK_DO_SITE", "php.net");  
echo LINK_DO_SITE;
```

Resultado: php.net

O nome de uma constante tem quase 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.

2.8.3 - Exercícios

1^a) Qual a principal finalidade de um constante e como elas são definidas em PHP ?

2^a) Em que momentos precisamos converter uma variável de um tipo em outro?

3^a) Quais os typecasting usados em PHP ?

4^a) Crie uma constante com o comando **define** e imprima com o comando **print()**;

5^a) 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.

2.9 - Arrays

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 com um array real, lista (vetor), hashtable (que é uma implementação de mapa), dicionário, e 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.

**2.9.1 – Criando array Unidimensional**

Arrays são acessados mediante uma posição, como um índice numérico. Para criar um array pode-se utilizar a função **array('chave' => "valor", ...)**.

Também conhecidos como vetores, os arrays unidimensionais recebem este nome por terem apenas uma dimensão, no caso, cada elemento do array fica em um espaço definido por um índice.

Exemplo:

```
$alunos = array(0=>"Chiquim", 1=>"Zezim", 2=>"Joãozim", 4=>"Tizim");
ou
$alunos = array("Chiquim", "Zezim", "Joãozim", "Tizim");
```

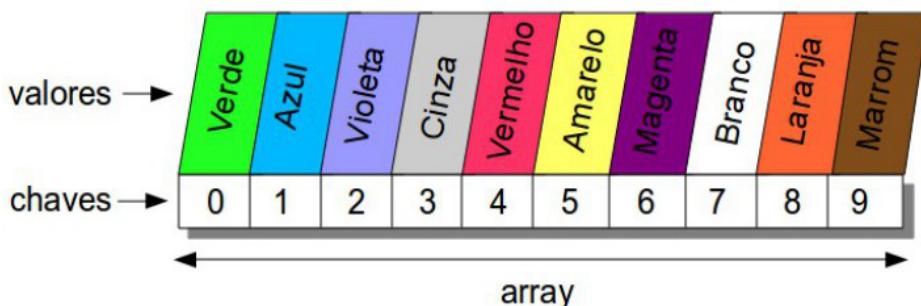
Nessa sintaxe temos duas formas de declarar uma variável do tipo array. Onde a chave ou índice podem ser de forma automática como no primeiro exemplo, ou manual como no segundo. Outro detalhe importante é que: todo array começa pela chave ou índice de número 0, quando o mesmo não é declarado.

Também temos outras formas de criar um array, onde simplesmente podemos adicionar valores conforme a sintaxe abaixo:

<pre>\$alunos[0] = "Chiquim"; \$alunos[1] = "Zezim"; \$alunos[2] = "Joãozim"; \$alunos[3] = "Tizim";</pre>	ou	<pre>\$alunos[] = "Chiquim"; \$alunos[] = "Zezim"; \$alunos[] = "Joãozim"; \$alunos[] = "Tizim";</pre>
--	----	--

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(0=>"Verde", 1=>"Azul", 2=>"Violeta", 3=>"Cinza", 4=>"Vermelho",
5=>"Amarelo", 6=>"Magenta", 7=>"Branco", 8=>"Laranja", 9=>"Marrom");
```

Veja como mostrar os elementos do array acima:

```
echo $cores[0];
echo $cores[1]; Dessa forma para acessar o array. Basta determinar o nome do
echo $cores[2]; array e qual a chave, onde cada chave tem um valor já
... determinado. Resultará em um erro o uso de uma chave errada.
echo $cores[9];
```

2.9.2 - Criando Array Associativo

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:

```
$dados = array('nome'=>"Alessandro", 'email'=>"Ale@email.com", 'senha'=>"rapadura");
```

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 (loop), mas em outros casos sempre é bom utilizar arrays associativos.

Outra forma de iniciarmos um array associativo é adicionar valores conforme abaixo:

```
$dados[ 'nome' ] = "Alessandro Feitoza";
$dados[ 'rua' ] = "Barca Velha";
$dados[ 'bairro' ] = "Quintino Cunha";
$dados[ 'cidade' ] = "Fortaleza";
```

A imagem abaixo representa o exemplo anterior:

nome	Alessandro Feitoza
rua	Barca Velha
bairro	Quintino Cunha
cidade	Fortaleza

Umas das vantagens do array associativo `$dados['nome'] = "Alessandro Feitoza";` é quando fazemos o acesso ao array, `$dados['rua'] = "Barca Velha";` onde temos de forma clara e `$dados['bairro'] = "Quintino Cunha";` compreensível o valor que aquela chave `$dados['cidade'] = "Fortaleza";` pode conter. Como por exemplo nome, onde só vai existir o nome de pessoas.

```
echo $dados[ 'nome' ];
echo $dados[ 'rua' ];
```

Veja ao lado um exemplo de acesso ao `echo $dados['bairro'];` valores armazenados em um array dessa `echo $dados['cidade'];` natureza.

2.9.3 - 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
Linhas	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 quantidade de chaves (índices), onde cada um dos índices representa uma dimensão.

Observe o código da representação ao lado.

Código:

```

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  ?>

```

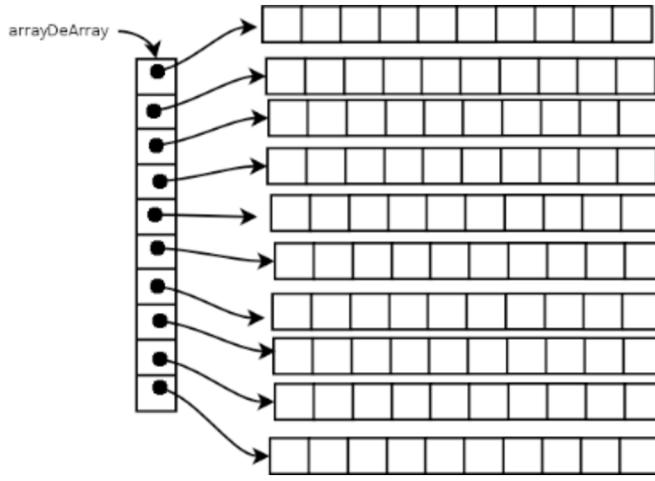
Outra forma de iniciar o array:

```

1  <?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 colocarmos 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['indice1']['indice2']...['indice_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.

2.9.4 - Exercícios

1^a) O que é um array, e qual a sua principal finalidade?

2^a) Declare um array chamado “nomes” com 8 posições, e grave nomes de pessoas que você conhece em cada uma delas. E responda:

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

echo nomes[3];

b) O que acontece se chamarmos uma posição que não existe no array, exemplo:

nomes[15];

3^a) O que é um array associativo, dê exemplos:

4^a) O que é um array multidimensional?

5^a) Crie um array “palavras” multidimensional 5x3 com os valores da tabela abaixo, e responda:

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

- a) Crie um código PHP onde com os valores do array possa ser impresso na tela com a frase “oi, tudo bem com você?”.
- b) 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 ?
- c) Construa um código PHP para mostrar uma resposta(que está no array) para a pergunta do item a.

2.10 - Operadores

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

2.10.1 – Operadores de String

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 próximo exemplo:

```

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 ?>

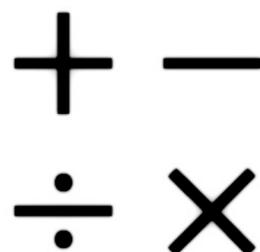
```

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.

2.10.2 - Operadores Matemáticos

São os operadores para manipular dados numéricos, ou seja, int's, float's, e double's. Veremos isso em 3(três) sub-tópicos:

- Aritméticos
- Atribuição
- Descremento e Incremento



2.10.2.1 – Operadores 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

```

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 $\$b+5$.

2.10.2.2 - 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 ao lado:

```

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 :

```

$a = "Dia "; #inicia $a com Dia
$b = "Bom "; #inicia $b com Bom
$b .= $a.= "turma"; #concatena primeiro $a,logo depois $b
echo $b; #imprime na tela.

```

Resultado: `Bom dia turma`

Podemos definir uma sequência com duas concatenações, onde `$a = "Dia"."turma"` e projeto logo após temos `$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 <?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.

<pre> 1 <?php 2 \$a = 8; \$b = 2; // inicializa duas variáveis 3 # multiplica o valor de \$a(8) por 3 e atribui. 4 echo (\$a *= 3)."
" ; 5 # divide o valor de \$a(24) por 2 e atribui. 6 echo (\$a /= 2)."
" ; 7 # resto da divisão de \$a(12) por 5 e atribui. 8 echo (\$a %= 5)."
" ; 9 // os pontos "." concatena com "
" . 10 ?> 11 </pre>	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.

2.10.2.3 - Operadores de Incremento e Decremento

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

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

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 $a(2), depois decrementa 1.
8  print($a); // imprime $a(1)
9  ?>
10

```

Nesse exemplo temos uma forma aplicada do uso de decremneto 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
<code>++\$a</code>	<code>\$a = \$a + 1</code>	<code>\$a += 1</code>
<code>- -\$a</code>	<code>\$a = \$a - 1</code>	<code>\$a -= 1</code>

2.10.3 – 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
<code>==</code>	Igual. Resulta em TRUE se as expressões forem iguais.
<code>====</code>	Idêntico. Resulta em TRUE se as iguais e do mesmo tipo de dados.
<code>!= ou <></code>	Diferente. Resulta verdadeiro se as variáveis foram diferentes.
<code><</code>	Menor ou menor que. Resulta TRUE se a primeira expressão for menor.
<code>></code>	Maior ou maior que. Resulta TRUE se a primeira expressão for maior.
<code><=</code>	Menor ou igual. Resulta TRUE se a primeira expressão for menor ou igual.
<code>>=</code>	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:

```

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"; // falso
10 echo $a <= $c ? "verdadeiro" : "falso"; // verdadeiro
11 echo $a >= $c ? "verdadeiro" : "falso"; // falso
12
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 `$c` um tipo ponto flutuante, já o comparador “`==`” compara somente os valores onde 45 é igual a 45.0 retornando verdadeiro. Também podemos usar o operador “`!==`” onde tem a função semelhante ao operador “`!=`”, mas retorna **TRUE** se os tipos forem diferentes. Se a variável for do tipo booleano, podemos compará-los assim: `$a == TRUE`, `$a == FALSE`

2.10.4 – 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 outras proposições lógicas simples. Observe:

>>> Operador <<<	>>> Função <<<
Não	Negação
E	Conjunção
Ou	Disjunção

Com isso podemos construir a Tabela verdade onde trabalhamos com todas as possibilidades combinatórias 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 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.



Dica: or e and tem precedência 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 precedência maior:

```
$a = 10 > 2;      // verdadeiro, pois 10 é maior que 2.
$b = 12 >= 12;    // verdadeiro, pois 12 é igual a 12.
$c = FALSE;       // inicia com FALSE(tipo booleano).
echo ($b and $a) ? "sim" : "não" ; //sim, pois $a e $b são verdadeiros.
echo ($b or $c) ? "sim" : "não" ; //sim, pois $b é verdadeiro.
echo ($b xor $a) ? "sim" : "não" ; //não, pois $b e $a são verdadeiros.
```

Em outro exemplo temos os operadores lógicos mais comuns:

```
$a = 10 > 2;      // verdadeiro, pois 10 é maior que 2.
$b = 12 >= 12;    // verdadeiro, pois 12 é igual a 12.
$c = FALSE;       // inicia com FALSE(tipo booleano).
echo (!$c)        ? "sim" : "não" ; //sim, pois $c é falso.
echo ($a && $c)   ? "sim" : "não" ; //não, pois $c é falso.
echo ($b || $c)   ? "sim" : "não" ; //sim, pois $b é verdadeiro.
```

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.

```
$b = 0;
($a = 2 or $b = 3);
```

O segundo **echo** mostra 5 e 3, pois tanto a primeira quanto a segunda precisam ser executadas.

```
echo $a.", ".$b;
($a = 5 and $b = 3);
echo $a.", ".$b;
```

2.10.5 – 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:

```
echo 5 + 2 * 6; // resultado: 17
```

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 deseje realizar a operação com o operador + para só em seguida realizar a operação com o operador *, temos que fazer conforme o exemplo abaixo:

```
echo (5 + 2) * 6; // resultado: 42
```

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 :

```
echo (5 + 2) * (6-3); // resultado: 21
```

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

Seguindo a ordem de procedência temos:

$$(5) * (6) / (16 ((7)*2)) \ggg 5 * 6 / (16 (14)) \ggg 5 * 6 / 2 \ggg 30 / 2$$

Resultado : 15

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

2.10.6 – Exercícios

- 1^a) Qual a finalidade dos operadores de strings?
- 2^a) Quais os operadores de decremento e incremento? Cite alguns exemplos:
- 3^a) Qual a finalidade do operador aritmético %(módulo)?
- 4^a) Cite os operadores relacionais, mostre alguns exemplos.
- 5^a) Quais operadores lógicos ou booleanos?
- 6^a) Quais os operadores de atribuição?
- 7^a) Qual a sintaxe do uso de ternário e cite um exemplo?
- 8^a) Quais os operadores utilizados e o resultado final do código abaixo:

```
$a = 10;
$b = 12.5;
$c = $a + $b;
print($a > $b? "verdadeiro" : "falso");
print($c >= $b? "verdadeiro" : "falso");
```

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

$$\$a = 8*5-3+4/2+19\%5/2+1;$$

- 10^a) Faça testes com os operadores `$var1 = 2.2564;`
relacionais substituindo o operador `>` do `$var2 = 2.2635;`
código-fonte ao lado. `print($var1 > $var2 ? "sim" : "não");`

- 11^a) Usando o operador de String `.` para montar a seguinte frase ao lado:

```
$a = "de";
$b = "é um";
$c = "comunicação";
$d = "a";
$e = "internet";
$e = "meio";
print(.....);
```

- 12^a) 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");
```

2.11 - Estruturas de Controle

As estruturas de controle de dado são responsáveis pela manipulação dos dados conforme seja necessário na realização de um processamento. Cada instrução programável possui uma lógica de operação e estabelece uma sequencia de ações a serem efetuadas ou verificadas. Quando estivermos criando os algoritmos serão necessárias as manipulações de dados respeitando a sequencia lógica de cada comando, buscando sempre alcançar os objetivos almejados pelos algoritmos. Basicamente as estruturas de controle de dados são de três tipos principais:

- **Sequencial:** conjunto de comandos separados por ponto e vírgula (;) que são executados em uma sequência linear de cima para baixo.
- **Seleção:** a partir de um teste condicional, uma instrução, ou um conjunto de instruções, podem ser executados ou não, dependendo exatamente do resultado do teste efetuado.
- **Repetição:** uma instrução ou o conjunto de instruções que será executado repetidamente, de acordo com o resultado de um teste condicional.

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.

2.11.1 – Estrutura Sequencial

A estrutura sequencial é a mais convencional entre todas as possíveis, pois ela consiste na execução de uma instrução de cada vez, onde o encerramento da primeira instrução permite o acionamento da instrução seguinte, respeitando a ordem de cima para baixo.

Exemplo:

```
$nome = "Alessandro";
$ano_nascimento = 1995;
$ano_atual = 2014;

$idade = $ano_atual - $ano_nascimento;

echo "Nome: $nome<br>";
echo "Idade: $idade<br>";
```

2.11.2 – Estruturas de Seleção/Decisão

Um bloco de estrutura de seleção/decisão consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como **if**, **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.

2.11.2.1 - Simples(IF)

Essa estrutura condicional está entre uma das 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:}**”.



```
$a = 0;
if($a>2)
    echo "comando1";
    echo "comando2";
```

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.

```
$a = 0;
if($a>2){ // início do bloco de código.
    echo "comando1";
    echo "comando2";
} // fim do bloco de código.
```

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.

2.11.2.2 - Composta(IF-ELSE)

Esta estrutura é bem semelhante a anterior a única diferença é que com **if-else** temos duas possibilidades de saída, onde encontramos algumas regras:



- Só um bloco pode ser executado: ou do **if** ou o do **else**.
- Os dois blocos não poderão ser executados juntos.
- Não existe a possibilidade de não ser executado nenhum dos dois blocos.

A tradução de **if-else** é “**SE – SENÃO**”.

Sintaxe:

<i>if(condição){ bloco de códigos; }else{ bloco de códigos; }</i>	<i>se(condição) então{ bloco de códigos; }senão então{ bloco de códigos; }</i>
---	--

Exemplo:

```
$idade = 18;

if($idade >= 18){
    echo "De maior";
} else{
    echo "De menor";
}
```

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.

2.11.2.3 - Encadeada(IF-ELSE-IF-IF)

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 **IF-ELSEIF-ELSE**, Para facilitar o entendimento de uma estrutura do tipo:

<pre>\$numero = 0; if(\$numero > 0){ echo "Positivo"; } else{ if(\$numero < 0){ echo "Negativo"; } else{ echo "Nulo"; } }</pre>	<p>ou</p> <pre>\$numero = 0; if(\$numero > 0){ echo "Positivo"; } elseif(\$numero < 0){ echo "Negativo"; } else{ echo "Nulo"; }</pre>
--	--

2.11.2.4 - Múltipla Escolha(SWITCH-CASE)

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 **IF-ELSE-ELSEIF**. O comando **SWITCH-CASE** é uma estrutura que simula uma bateria de testes sobre uma variável. Frequentemente é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um desses valores.

<pre>\$numero = 2; switch(\$numero){ case 1: echo "Um"; break; case 2: echo "dois"; break; case 3: echo "três"; break; default: echo "número inválido"; }</pre>	<pre>\$numero = 2; escolha(\$numero) então{ caso 1: echo "Um"; interrompa; caso 2: echo "dois"; interrompa; caso 3: echo "três"; interrompa; padrão: echo "número inválido";}</pre>
---	---

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 parar/interromper o bloco de execução, nesse caso ele interrompe a estrutura assim que algum **case** for executado. Veremos mais sobre **break** no decorrer da apostila.

Mas o que acontece se não entrar em nenhum case?

A resposta é bem simples, nenhum dos blocos seria executado, 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).



No exemplo anterior se a variável **\$numero** não tivesse um valor satisfatório aos cases, o bloco do **default** seria executado, nesse caso, mostrando “número inválido”.

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

```
$numero = "quarenta e dois";

switch($numero){
    case "um":
        echo 1;
        break;
    case "dois":
        echo 2;
        break;
    case "quarenta e dois":
        echo 42;
        break
    default:
        echo "número inválido";
}

Resultado: número inválido
```

2.11.2.5 - Exercícios

1º) Faça um script em PHP que receba um número representado por uma variável e verifique se este número é par ou ímpar e imprima a mensagem na tela.

2º) Crie um script baseando-se na idade de uma pessoa:

Idade	Resultado
Menos que 12	Criança
Entre 12 e 18	Adolescente
Entre 18 e 60	Adulto
Mais que 60	Idoso

3ª) 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.

2.11.3 - Estruturas de Repetição

As estruturas de Repetição, como o próprio nome já diz, servem para repetir um determinado bloco de código, ou seja, gerar um laço de repetição(loop). Muitas vezes nos deparamos com scripts onde um mesmo código se repete muitas vezes, por exemplo, um campo select-option de formulários do html que serve para escolher o “ano de nascimento” que varia de 1900 a 2014, todos sabemos que um para cada número teríamos que criar uma **option** e isso daria muito trabalho, pois seriam 115 options, nesse caso poderíamos usar uma estrutura de repetição para repetir um determinado código 115 vezes, que no caso seria a criação de cada option.

Não só no PHP, mas em outras linguagens temos 4 estruturas de repetição:

- Repetição com teste condicional no início.
- Repetição com teste condicional no final.
- Repetição com variável de controle
- Estrutura para percorrer arrays



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 onde vai determinar, ou automática onde quem determina o fluxo de execução do código são as funções do PHP, funções estas já existentes.

Para trabalharmos com essa contagem de quantas vezes o laço deve se repetir, usaremos **incremento** ou **decremento** de uma variável conforme vimos antes.



Dica: Toda estrutura de repetição requer 3 elementos:

- de onde vai? (**início**) – variável de início
- pra onde vai? (**término**) – condição
- como vai? (**passo**) – incremento ou decremento

2.11.3.1 - Condição no Início(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... }**”.

while (condição){ comandos; }	enquanto(condição) faça{ comandos }
-------------------------------------	---

Observe o exemplo abaixo:

```
$n = 0;

//enquanto $n for menor que 10...
while($n < 10){
    echo $n; //comando a ser repetido...

    $n++; //incrementando o $n

}//finaliza o bloco...
```

Resultado: 123456789

O código acima irá contar de 0 até 9, utilizando a dica dada antes temos:

- **De onde vai?** De 0(zero)
- **Pra onde vai?** Até 9(nove)
- **Como vai?** De 1 em 1

Perceba que no código acima o laço irá rodar 10 vezes, e a cada rodada a variável \$n será incrementada em 1, fazendo assim com que o código tenha um final garantido.



Cuidado: Ao trabalhar com loop's (laço de repetição), pois caso a condição 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.

2.11.3.2 - Condição no Final(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{ comandos; }while(condição);	faça{ comandos }enquanto(condição);
---------------------------------------	---

Exemplo:

```
$n = 0; //de onde vai

do{
    echo "PHP é o Poder!<br>";
    $n++; //como vai
}while($n < 5); //pra onde vai
```

O código resultará na frase “PHP é o Poder!” sendo mostrada 5 vezes.

2.11.3.3 - Variável de Controle(FOR)

Outra estrutura de repetição 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.

<code>for(inicialização; condição; incremento){ comandos; }</code>	Para <i>inicio</i> até <i>término</i> de <i>passo</i> { 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 possibilhamos 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.

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  ?>
10
```

Ambos exemplos geram o mesmo

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 ?>
13
```

resultado:123456789

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.

```
$a = 1;           //inicia a variável.
for( ; $a < 10 ; ){ //condição.
    echo $a;      //comando.
    $a++;         //incremento.
}
```

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

2.11.3.4 – Exercícios

- 1º) Faça um script que conte de 1 até 100.
- 2º) Faça um script que imprima na tela números de 3 em 3 entre 0 e 99, ex: 0,3,6,9...
- 3º) Faça um script que conte de -1 até -100.
- 4º) Faça um script que imprima na tela somente números pares de 2 até 20.
- 5º) Faça um script que receba duas variáveis \$a e \$b, logo após imprima os números de intervalos entre eles. Ex: a=5 ,b = 11, imprime : 5,6,7,8,9,10,11.

2.11.3.5 - Percorrer Arrays(Foreach)

O **foreach** é um laço de repetição para interação em array's ou matrizes. Trata-se de um **for** mais especializado que compõe um vetor ou matriz em cada um de seus elementos por meio de sua cláusula **AS**.

<code>foreach(array as indice => valor){ comandos; }</code>	Para cada elemento do array pegue indice e valor { comandos; }
---	---

Exemplo:

```
$dados[ 'nome' ] = "Alessandro Feitoza";
$dados[ 'email' ] = "Ale@email.com";
$dados[ 'fone' ] = "8888-8888";
$dados[ 'senha' ] = "qwe123";

foreach($dados as $indice=>$valor){
    echo "$indice = $valor<br>";
}
```

Resultado:

```
nome = Alessandro Feitoza
email = Ale@email.com
fone = 8888-8888
senha = qwe123
```

Outro exemplo:

```
$pessoas = array("Alessandro", "Anthony", "Rachel", "Dioul");

foreach($pessoas as $valor){
    echo "$valor - |";
}
```

Resultado:

Alessandro - Anthony - Rachel - Dioul -

Perceba que no primeiro exemplo selecionamos em cada elemento do array o índice e o valor, e mostramos ambos, já no segundo exemplo selecionamos apenas o valor de cada elemento do array, mas se quiséssemos poderíamos também selecionar o índice, no caso seria: 0, 1, 2, e 3.

2.11.4 - Break

Um comando bastante 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.

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:

```
$a = 1;           // inicia a variável.
for( ; ; ){
    echo "-".$a; // imprime na tela.
    if($a == 10) // condição.
        break;    // break, para o laço.
    $a++;         // incremento
}
```

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 quantidade de níveis que podem existir em um ***break***, observe o exemplo:

```

for($a = 1;$a < 5;$a++){           // primeiro laço
    echo $a." - ----- laço<br>";
}

for($b = 1;$b < 5;$b++){           // segundo laço
    echo $b." - segundo for<br>";
    if($a == 2)                   // condição
        break 2;                  // break
}
}

```

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.

2.11.5 - 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:

```

for ($i = 0; $i < 20; $i++) { //laço de repetição.
    if ($i % 2) continue; //continue.
    echo $i.",";
}

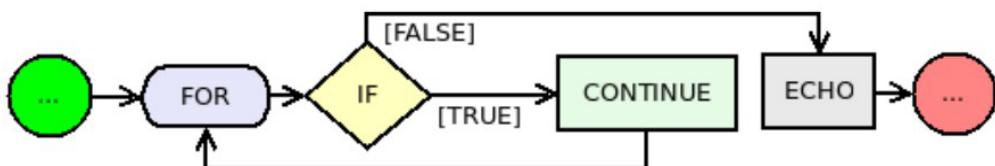
```

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 **\$i** 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:



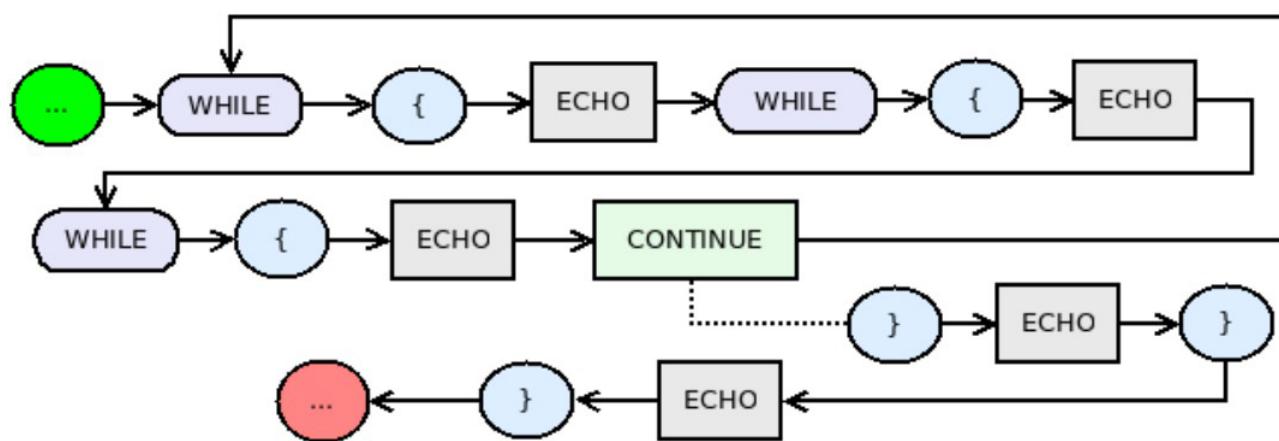
Assim como o **break**, também podemos definir em qual nível queremos que a execução continue. Veja o exemplo abaixo:

Resultado:	
\$i = 0;	--primeiro
while (\$i++ < 5) { //laço de 5 interações, 3º nível.	-segundo
echo "---primeiro ;	-terceiro
while (1) { //laço infinito, 2º nível.	--primeiro
echo "-segundo ;	-segundo
while (1) { //laço infinito, 1º nível.	-terceiro
echo "-terceiro ;	--primeiro
continue 3; //continua do 3º nível.	-segundo
}	-terceiro
echo "Esta saída não é Alcançada. ;	--primeiro
}	-segundo
echo "Esta saída não é Alcançada. ;	-terceiro
}	--primeiro

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:



2.11.6 - Exercícios

- 1^a) Qual a principal finalidade de uma estrutura de controle?
- 2^a) Qual a principal finalidade de uma estrutura de repetição?
- 3^a) 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".
- 4^a) Com o comando IF e ELSE crie um código que determine se uma expressão é verdadeira ou falsa.
- 5^a) Qual a finalidade da estrutura de controle SWITCH e cite um exemplo onde comparamos uma opção com 4 casos diferente?
- 6^a) Crie um contador de 1 até 20 usando a estrutura de repetição WHILE.
- 7^a) Crie um contador de 1 até 100 usando DO WHILE.
- 8^a) Crie um contador de 100 até 1 usando FOR.
- 9^a) Qual a fonalidade de um FOREACH?
- 10^a) Crie um código onde podemos para a execução de um laço infinito com o uso de BREAK.
- 11^a) Como podemos determinar o uso de CONTINUE e qual a sua aplicação prática em PHP .
- 12^a) 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 `if($valor % 2 == 0){`
responsável por compara os valores `echo $valor."é um numero par";`
passado a ele e imprimir os pares e ímpares. `}else{`
Exemplo: `echo $valor."é um numero impar";`
`}`
 - 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.

Capítulo 3 – Programando em PHP

- 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 informações.
- Veremos também como manipular arquivos externos e como trabalhar com mais um arquivo ao mesmo tempo.



3.1 - Interação PHP – HTML

A interação do PHP com o HTML é principalmente com formulários, pois a principal forma de entrada de dados no PHP é com formulários HTML. Veremos a seguir uma revisão do html, e logo após como enviar e receber esses dados com o PHP.

3.1.1 - Revisão HTML - Formulários

Podemos definir fomulário como um conjunto 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.

3.1.1.1 - 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:

```
<form action="" method="">  
  
</form>
```

- **Action:** Para qual arquivo irão os dados.
- **Method:** Poder ser “POST” ou “GET”, como veremos adiante.

Os elementos do formulário serão preenchidos com os componentes input onde a tag é `<input>` conforme o exemplo abaixo:

```
<input name="" type="" value="">
```

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:** o nome do campo, esse atributo é obrigatório, já que o PHP deve saber o nome do campo para pegar seu valor.

→ **value:** este campo é o valor do campo, o valor é normalmente informado pelo usuário, mas em alguns casos esse valor já deve ser pré-definido.

→ **type:** o tipo do campo, esse atributo também é obrigatório, mas se ficar em branco o padrão é do tipo "text".

Observe a tabela abaixo com a definição dos tipos que podemos atribuir ao elemento type do input.

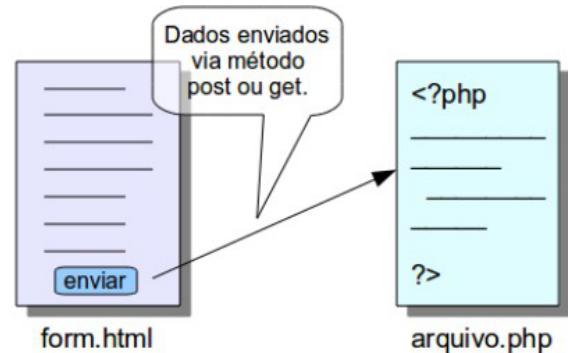
Type	Descrição	Exemplo
Text	Elemento utilizado para entrada de texto simples, é um dos mais utilizados.	<input type="text" value="texto..."/>
Email	Elemento utilizado para entrada de e-mails.	<input type="text" value="email..."/>
Senha	Elemento utilizado para entrada de senhas.	<input type="password" value="....."/>
Checkbox	Utilizado para exibir opções de verificação, muito utilizado para perguntas booleanas.	<input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/>
Radio	Exibe botões para seleção exclusiva, utilizado para seleção de apenas um item.	M <input checked="" type="radio"/> F <input type="radio"/>
Color	Elemento utilizado para seleção de cores.	<input type="color" value="#FF0000"/>
Date	Elemento utilizado para seleção de data.	<input type="text" value="dd/mm/aaaa"/> <input type="button" value="▼"/> <input type="button" value="▲"/>
Number	Elemento utilizado para entrada de números.	<input type="text" value="42"/> <input type="button" value="▼"/> <input type="button" value="▲"/>
Time	Elemento utilizado para seleção de horas.	<input type="text" value="04 : 20X"/> <input type="button" value="▼"/> <input type="button" value="▲"/>

File	Elemento utilizado para anexar arquivos	<input type="button" value="Selecionar arquivo..."/>
Hidden	Elemento utilizado para esconder campos	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="Botão"/>
Reset	Utilizado para limpar todos os campo do formulário, voltando ao valor inicial.	<input type="button" value="Limpar"/>
Submit	Botão usado para submeter os dados do formulário no servidor, ele envia as informações de acordo com as o action e methos informado na tag <form>	<input type="button" value="Enviar"/>
Select option	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).	<input type="button" value="-- Selecione --"/>
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.	<input type="text" value="Digite aqui..."/>

3.1.2 - Recebendo dados de Formulários(\$_GET - \$_POST)

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, processa a informação devolvendo resultado ou não. Observe ao lado:



3.1.2.1 – 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 URL de onde o formulário vai ser enviado e envia, separando o endereço da URL dos dados do formulário por um "?" (ponto de interrogação) e "&".

Quando você busca algo no DuckDuckGo, 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 DuckDuckGo.

Abaixo temos um exemplo de uma URL de busca do DuckDuckGo. Observe:



Podemos notar a passagem de um campo chamado "q" e o que foi digitado na busca, "php".

Recebendo dados via \$_GET:

Agora trabalharemos com o arquivo PHP , onde podemos resgatar os valores enviados pelo método \$_GET, que no caso é um tipo de array super global(estudaremos mais adiante), sua sintaxe é a seguinte:

\$_GET['nome_da_campo'] → retorna o valor passado pelo campo.

\$_GET → retorna um array com todos os valores 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:

URL:

```
<a href="gravar.php?pagina=3&conteudo=4">Informação</a>
```

Código:

```
//imprime os valores específicos
echo $_GET['pagina'].'<br>';
echo $_GET['conteudo'].'<br>';

//monta array com todos os valores enviados
$valores = $_GET;

foreach($valores as $campo=>$valor){
    echo "$campo = $valor<br>";
}
```

Resultado:

3

4

pagina = 3

conteudo = 4

Exemplo com formulário:

Formulário:

```
<form method="get" action="gravar.php">
    Login: <input name="login" type="text">
    Senha: <input name="senha" type="password">
        <input type="submit" value="Logar">
</form>
```

Código:

```
echo $_GET['login']."<br>";
echo $_GET['senha']."<br>";
```

Resultado: *será o que for digitado nos campos...*

3.1.2.2 – 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:

Formulário:

```
<form method="post" action="gravar.php">
    Login: <input name="login" type="text">
    Senha: <input name="senha" type="password">
        <input type="submit" value="Logar">
</form>
```

Código:

```
echo $_POST['login']."<br>";
echo $_POST['senha']."<br>";
```

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).

3.1.3 - Exercícios

1º) Crie um formulário HTML com os seguintes campos:

“nome”, “endereço”, “email”, “senha”, e o botão “enviar”.

2º) Utilize o método Get para visualizar os dados do array na URL do navegador ao clicar no botão enviar.

3º) Qual a diferença do método POST e GET?

4º) Como podemos receber dados via método GET? Exemplifique.

5º) Como podemos receber dados via método POST?. Exemplifique.

6º) 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 aquivo dados.php deverá conter cinco variáveis, cada uma para determinado campo, exemplo:

`$nome = $_POST['nome'];`

Os valores deverão ser impresso na tela.

3.2 - Especialidades do PHP

Aqui abordaremos as functions do PHP, e algumas especialidades da linguagem.

3.2.1 - Functions

Quando queremos um código funcional para determinado fim, como 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 reaproveitamento de código, deixando as funcionalidades mais legíveis.

3.2.1.1 – 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_funcao($param_1, $param_2, $param_3){  
    //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:

```
// declaração da função:  
function calculo_IMC($peso, $altura){  
  
    return $peso/($altura*$altura);  
  
}  
// fazendo a chamada da função:  
echo calculo_IMC( 62 , 1.75 );
```

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.

```
// declaração da função:  
function imprime($texto){  
  
    echo "<h1>$texto</h1>";  
  
}  
// fazendo a chamada da função:  
imprime("Testando a função!!");
```

Resultado:

Testando a função

3.2.1.2 - 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:

```
function Teste() {  
    $a = 25; // variável local  
}  
Teste(); /* ativa a função */  
echo $a; /* não imprime */
```

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.

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:

```
$a = 0;          /* escopo global */
function Teste() {
    global $a; /* variável global */
    $a = 25;
}
Teste();        /* ativa a função */
echo $a;        /* imprime 25 na tela*/ 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 última execução. Usamos o operador **static** para declararmos a variável. Exemplo:

Resultado: 10, 20, 30,

```
function Teste() {
    static $a; // variável statica
    $a += 10; // atribuição +10
    echo $a.",";
}
Teste(); // 1º chamada da função
Teste(); // 2º chamada da função
Teste(); // 3º chamada da função
```

Observe que o valor é mantido e a cada chamada é acrescentado +10, caso não exista o **static** o resultado seria: 10,10,10, .

3.2.1.3 - 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:

```
function valores($variavel , $valor) {
    $variavel += $valor;
}
$a = 23;
valores($a,26);
print $a; #Resultado: 23
```

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.

```
function valores( &$variavel , $valor)
{
    $variavel += $valor;
}
$a = 23;
valores($a,26);
print $a; #Resultado: 49
```

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 exemplo ao lado:

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 $23 + 26 = 49$.

3.2.1.4 - 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:

```
function Teste($caso) {
    switch($caso){
        case 1:
            return "opção 1"; break;
        case 2:
            return "opção 2"; break;
        case 3:
            return "opção 3"; break;
        default:
            return null;
    }
}
echo Teste(2);
```

```
function Soma($n1, $n2) {
    return $n1+$n2;
}
function Texto(){
    return "O resultado é: ";
}
echo Texto().Soma(115,24.5);
```

Resultado: 139.5

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

3.2.1.5 - Recursão

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.
Exemplo:

```
function Fatorial($numero) {
    if($numero == 1) // Fatorial encerra quando o numero é igual a 1.
        return $numero;
    else
        // Nesse retorno fazemos a chamda da função, passando o numero - 1.
        return $numero* Fatorial($numero-1);
}
echo Fatorial(5);
```

Resultado: 120

3.2.1.6 - Funções nativas

O PHP possui uma ampla lista de funções com várias possibilidades de uso, tais como: manipulação de strings, de números, de arrays, e até mesmo de funções.

Não seria viável abordar todas essas funções nesta apostila, principalmente porque existem muitas, e a cada versão surgem novas e outras vão se tornando obsoletas, para conhecer as funções nativas do PHP acesse a bíblia do programador PHP: http://www.php.net/manual/pt_BR/funcref.php.

Mas vale saber que para executar uma função basta invocá-la pelo nome e se for necessário passe os parâmetros.

```
$array = array(1, 2, 3, 4, 12, 42, 54);
//conta o numero de elementos no array.
$n_elementos = count($array);
```

3.2.1.7 – Exercícios

- 1^a) Diga com suas palavras uma definição para função, e como podemos declará-la em PHP.
- 2^a) Qual a diferença de variáveis globais para variáveis locais e como podemos defini-las em PHP?
- 3^a) O que é um parâmetro, e quais os tipos de parâmetros em PHP?
- 4^a) Quais as funções que podemos usar para criarmos uma função onde seus parâmetros são passados pro argumentos variáveis?
- 5^a) O que é um valor de retorno e qual o comando usado quando queremos retornar algo dentro de uma função?
- 6^a) O que é recursão?
- 7^o) Crie uma função que determine se um numero é par ou ímpar. E faça uma chamada dessa função imprimindo o resultado.
- 8^o) Crie uma função que calcule a factorial de um número.
- 9^o) 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.

3.2.2 - Arrays Super Globais

Variáveis ou arrays super globais, são estruturas nativas da linguagem PHP que tem funcionalidades específicas, e que como o nome já diz são globais, e podem ser acessadas de qualquer lugar do código. Vale lembrar que já conhecemos 2 arrays super globais: `$_GET` e `$_POST`.

3.2.2.1 - `$_COOKIES`

Cookies são mecanismos para armazenar e consultar informações. Eles são armazenados na máquina do cliente que acessa o 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 trata-se

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");
```

Na tabela abaixo temos a descrição de cada atributo.

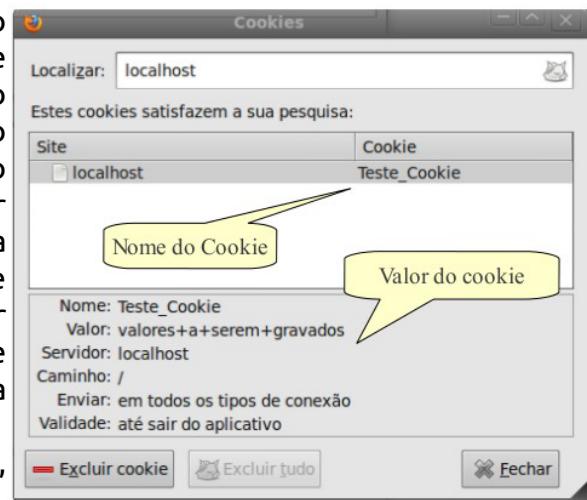
Atributo	Descrição
nome_do_cookie	É o nome que, posteriormente, se tornará a variável e o que o servirá de referência para indicar o cookie.
seu_valor	É o valor que a variável possuirá. Esse valor pode ser de todos os tipos.
tempo_de_vida	É o tempo, em segundos, que o cookie existirá no computador do 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:

```
$valor = "valores a serem gravados";
setcookie ( "Teste_Cookie" , $valor );
```

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 máquina. Se tudo ocorrer corretamente deverá aparecer a seguinte tela:

Veja que outras informações como caminho, enviar, e validade não foram especificados,



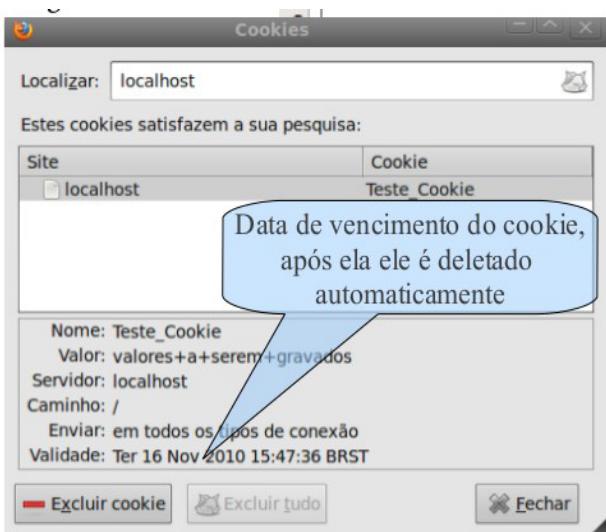
porém podemos determiná-los na hora da criação do cookie dentro do código php. Mostraremos agora um código onde atribuímos o tempo de vida do cookie, para isso devemos capturar o tempo com a função time() atual e somá-lo ao tempo que queremos em segundos, isso faz com que o cookie exista na máquina do cliente de acordo com a quantidade de tempo determinado pelo programador, observe um exemplo onde atribuirmos mais esse parâmetro à função setcookie.

Exemplo:

```
$valor = "valores a serem gravados";
$tempo = time() + 3600;
setcookie ("Teste Cookie" , $valor , $tempo);
```

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:

O novo resultado é o seguinte:



```
$tempo = time()+(3600*24*7);
```

Esse cookie tem seu tempo de vida de 7 dias, pois $3600 \text{ segundos} = 1 \text{ hora}$, $24 \text{ horas} = 1 \text{ dia}$ e $7 * \text{horas de um dia resulta em 7 dias}$.

Exemplo:

```
$valor = "valores a serem gravados";
$tempo = time() +(3600*24*7);
setcookie ("Teste Cookie" , $valor , $tempo);
```

Validade:

Validade: Sex 19 Nov 2010 18:55:29 BRST

Acessando um cookie:

Para acessarmos o valor gravado em um cookie é bem simples, basta utilizar usar o array super global `$_COOKIE`.

Sintaxe:

§ COOKIE`coloque aqui o nome do cookie`

```
Exemplo: $valor = "valores a serem gravados";
$tempo = time()+(3600*24*7); // tempo de vida
setcookie ("Teste_Cookie", $valor, $tempo);
echo $COOKIE['Teste_Cookie']; // imprime
```

Resultado: valores a serem gravados

Observe agora um exemplo `$val or=1;`

Observe agora um exemplo de um código utilizado para contar as visitas de um site usando cookie:

```
$valor=1;  
$tempo = time()+(3600*24*7); // tempo de vida  
$valor = $_COOKIE['contador']+1; // inicia a contagem  
setcookie ("contador" , $valor, $tempo); // grava o valor  
echo $_COOKIE['contador']." visitas no site!"; // imprime
```

O resultado é de acordo com a quantidade de vezes que o cliente entrou no site ou atualizou o mesmo.

3.2.2.2 - \$_SERVER

O array super global **\$_SERVER** possui muitas informações do servidor, por exemplo caminho do servidor, nome do servidor, endereço, browser, SO, dentre outros.

Exemplo:

```
echo $_SERVER['SERVER_NAME'];
```

Resultado: *localhost*

Para mais informações sobre esta variável acesse:

http://www.php.net/manual/pt_BR/reserved.variables.server.php

3.2.2.3 - \$_SESSION

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:

session_start();

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:

`session_cache_expire(5);` 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.

Com o array global **\$_SESSION** podemos gravar valores na sessão, veja um exemplo:

```
session_start();
$var = "Sessão Criada!";
$_SESSION['minha_sessao'] = $var;
```

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:

```
session_start();
echo $_SESSION['minha_sessao'];
```

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 a função **session_start()**, para trazermos ao arquivo todas as regras usadas em sessão no PHP.

```
session_start();
if(isset($_SESSION['minha_sessao'])) {
    echo "sessão ativa!";
} else {
    echo "sessão não existe!";
}
```

Ao lado temos um exemplo com o uso da função **isset()**, que verifica se uma variável existe ou não, retornando um valor booleano(true ou false):

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 criada pelo usuário, a outra forma é desalocarmos a sessão criada com o uso da função **unset()**.

Uso de **session_destroy()**:

```
session_start();
session_destroy();
```

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:

```
session_start();
// Destroi a sessão especificada.
unset($_SESSION['minha_sessao']);
```

Dessa forma desalocamos(destruimos) a sessão “minha_sessao”, porém se existirem outras, elas ainda continuarão ativas.

3.2.2.4 - \$GLOBALS

Este é um array superglobal, ou global automática também. Isto simplesmente significa que ela está disponível em todos escopos pelo script. Não há necessidade de fazer **global \$variable;** para acessá-la dentro de uma função ou método.

É também um array associativo contendo referências para todas as variáveis que estão atualmente definidas no escopo global do script. O nome das variáveis são chaves do array.

Exemplo:

```
$GLOBALS['nome'] = "Rapadura";

function teste(){
    echo $GLOBALS['nome'];
}

teste();
```

Resultado: Rapadura

3.2.2.5 – Outros Super Globais

Para conhecer outros arrays superglobais acesse:

http://www.php.net/manual/pt_BR/language.variables.superglobals.php

3.2.2.6 - Exercícios

1º) Qual a finalidade da variável global **`$_SERVER['HTTP_USER_AGENT']`**?

2º) Crie um **cookie** gravando nele o seu nome, logo após abra o Firefox e exiba o valor gravado.

3º) Crie um arquivo chamado “`criar_sessao.php`”, utilize comando PHP para cria uma seção com a durabilidade de 3 minutos e adicione o valor “sessão ativa”.

4º) Crie um novo arquivo chamado “`ler_sessao.php`” para verificar se a sessão criada na questão 3 existe ou não. Utilize o comando **`$_SESSION`** para ler o valor nela contido e imprima na tela.

3.2.3 - Manipulação de Arquivos e Diretórios

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.

3.2.3.1 - 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 konsole ou terminal do seu sistema(Linux). Digite:

`chmod 777 /var/www`

O comando **`chmod 777`** dá 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.

```
1  <?php
2
3  $a = fopen("meu_arquivo.txt", "w");
4
5  ?>
```

Veja um exemplo do uso do comando **fopen**:

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 executamos esse comando teremos um novo arquivo com o nome *meu_arquivo.txt*.

3.2.3.2 - 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 | 1 <?php
2 | 2
3 | 3 $a = fopen("meu_arquivo.txt","w");
4 | 4 $texto = "cidade: Fortaleza.\nrua: Pedro Pereira.";
5 | 5 fwrite($a, $texto);
6 | 6
7 | 7 ?>

```

Resultado:

meu_arquivo.txt
cidade: Fortaleza.
rua: Pedro Pereira.

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 | 1 <?php
2 | 2
3 | 3 $a = fopen("meu_arquivo.txt","w");
4 | 4 $textol = "Apostila de PHP.";
5 | 5 $texto2 = "Apostila de HTML.";
6 | 6 $texto3 = "Apostila de JAVA.";
7 | 7 fwrite($a, $textol."\n");
8 | 8 fwrite($a, $texto2."\n");
9 | 9 fwrite($a, $texto3."\n");
10 |
11 | 11 ?>

```

Resultado:

meu_arquivo.txt
Apostila de PHP
Apostila de HTML
Apostila de JAVA

No exemplo, fizemos a chamada do comando **fwrite** três vezes e escrevemos a cada chamada um valor diferente concatenando com “\n”.

3.2.3.3 - 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**

sintaxe:

fclose("identificador")

```

1 | 1 <?php
2 | 2 $a = fopen("meu_arquivo.txt","w");
3 | 3 fwrite($a,"escreva seu conteúdo aqui!!!");
4 | 4 // fecha o arquivo
5 | 5 fclose($a);
6 | 6 ?>

```

exemplo:

Toda vez que abrimos um arquivo com **fopen**, devemos fechá-lo com o comando **fclose** conforme o exemplo ao lado.

3.2.3.4 - 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**.

Este comando 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.

Sintaxe:

```
$array = file("string_do_arquivo")
```

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 | 1<?php
2 | 2
3 | 3 $conteudo = file("meu_arquivo.txt");
4 | 4 echo $conteudo[0]."<br>";
5 | 5 echo $conteudo[1]."<br>";
6 |
7 | ?>
```

Apostila de PHP
Apostila de HTML

Resultado:

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). Oura forma é percorrer o array usando um foreach(), dessa forma podemos ler todas as linhas existentes no arquivo, veja:

Exemplo:

```
1 | 1<?php
2 | 2
3 | 3 $conteudo = file("meu_arquivo.txt");
4 | 4 foreach($conteudo as $valor)
5 | 5 echo $valor."<br>";
6 |
7 | ?>
```

Resultado:

Apostila de PHP
Apostila de HTML
Apostila de JAVA

3.2.3.5 - 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:

/home/aluno/meu_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.

3.2.3.6 - 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 → é definido como o caminho com o nome do diretório, ou somente o nome.

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:

```
rmdir("nome_diretório");
```

Observe o exemplo envolvendo os três comandos abaixo:

```

1  <?php
2
3  // cria um diretório.
4  $a = mkdir("minha_pasta", 0777);
5  if($a)
6      echo "pasta criada.";
7  else
8      echo "erro!";
9  // retorna o diretório onde a pasta foi criada.
10 echo "<br>em ".getcwd()."<br>";
11 // apaga o diretório.
12 $a = rmdir("minha_pasta");
13 if($a)
14     echo "pasta apagada.";
15 else
16     echo "erro!";
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.

3.2.3.7 – Exercícios

1º) O que é manipulação de arquivos?

```
<?php  
    $arquivo = fopen("all.txt", "w");  
    fwrite($arquivo, "oi tudo bem!");  
    fclose($arquivo);  
?>
```

2º) Observe o código-fonte abaixo:

- a) Que tipo de arquivo é gerado na linha 2 e aonde o mesmo é criado?
- b) Que o parâmetro “w” da linha 2 do código representa?
- c) Qual a finalidade do comando fwrite da linha 3 do código?
- d) 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

3º) Grave uma mensagem dentro do arquivo criado.

4º) Com base no arquivo criado, utilize o comando fwrite para ler o mesmo imprimindo na tela do navegador o conteúdo do arquivo.

5º) 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.

6º) Crie uma cópia do arquivo renomeando o novo arquivo para “palavras.txt”.

7º) Agora apague o arquivo “frases.txt” com o comando *unlink*.

8º) 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 .

9º) Crie um código que imprima na tela todo o caminho de pastas onde se localiza o arquivo “palavras.txt”.

3.2.4 - 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:

3.2.4.1 - INCLUDE

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 | <?php
2 |
3 | echo "<h1>Bem vindo ao Site</h1>";
4 |
5 | ?>
6 |

```

código do arquivo *index.php*

```

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.

3.2.4.2 - REQUIRE

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';**

3.2.4.3 - INCLUDE_ONCE

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, evitara sobreposição de arquivo.

Sintaxe: **include_once 'nome_do_arquivo.php';**

3.2.4.4 - REQUIRE_ONCE

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';**

Capítulo 4 – PHP Orientado à Objetos

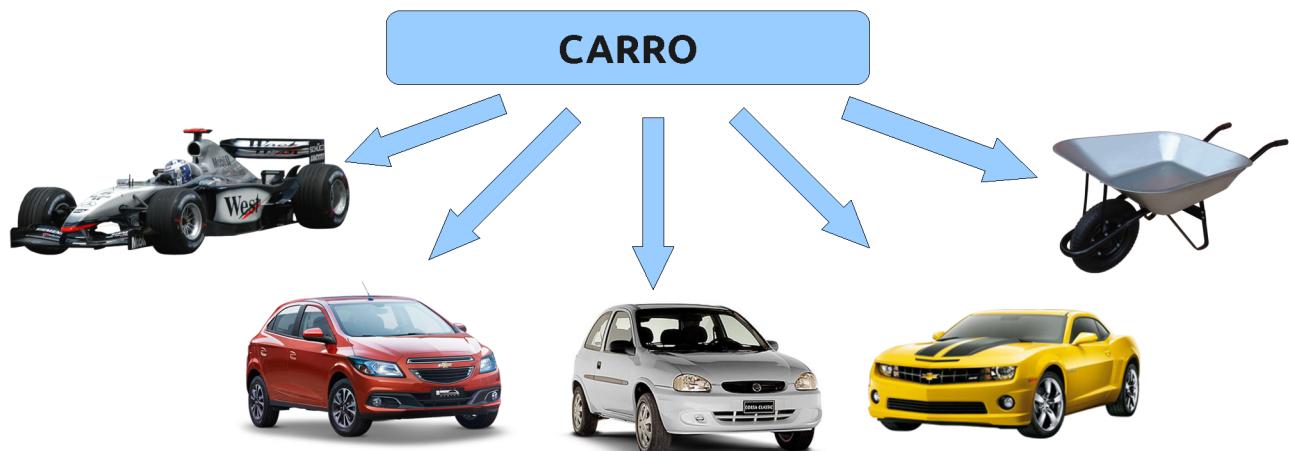
- A Orientação a Objetos é um paradigma que representa toda uma filosofia para construção de sistemas. Em vez de construir um sistema formado por um conjunto de procedimentos e variáveis nem sempre agrupadas de acordo com o contexto, como se fazia em linguagens estruturadas (Cobol, Clipper, Pascal), na orientação a objetos utilizamos uma ótica mais próxima do mundo real. Lidamos com objetos, estruturas que já conhecemos do nosso dia a dia e sobre as quais possuímos maior compreensão.



4.1 - Introdução a OO

Iremos aprender a trabalhar com a linguagem PHP Orientada a Objetos. Aconselhamos que estude antes algoritmos básicos de lógica de Programação em PHP e construção HTML/CSS, pois teremos que está afiados nos conceitos estruturais para continuar a nossa caminhada em assuntos mais aprofundados.

Programação Orientada a Objetos, apartir de agora chamaremos de “POO”, constitui em um conjunto de regras de programação. Estudamos anteriormente como programar de forma estruturada, iniciávamos a execução do código e finalizávamos todos os processos na mesma página. Com a Programação Orientada a Objetos podemos usar algoritmos já feitos por alguém, ou criar novos algoritmos interagindo com os que já foram desenvolvidos por alguém com objetivo de resolver problemas mais complexos gerando um novo produto final. Vamos fazer uma comparação no o dia a dia para entendermos melhor.



Perceba que todos os modelos acima são carros, porém possuem características diferentes, por exemplo: tração, modelo, número de portas, tamanho do aro, etc. É aí que entra parte da POO, pois todos esse são objetos do tipo carro, o que mudam são os detalhes, ou seja, os atributos de cada um.

4.1.1 - Como funciona? Pra que serve? Em que melhora?

A POO, é uma metodologia que os programadores utilizam com objetivo de atingir alguns requisitos importantes na construção de um bom programa ou site como:

- **Organizar o código:** Com o nosso código dividido, podemos facilmente dar manutenção ou encontrar algum erro com mais facilidade.
- **Reutilização do código:** Não precisamos reescrever todo o código, quando necessitarmos dele novamente.
- **Manter Padrões:** Utilização de um mesmo padrão conceitual durante todo o processo de criação do código.

Até o final do curso iremos aprender e trabalhar nessa metodologia. Então, bastante atenção nesse e nos próximos capítulos. Para programar orientado a objeto precisamos dividir o nosso projeto em classes específicas. Classes essas que estarão em ***.class.php**. Antes de criarmos uma classe em PHP , vamos ver alguns conceitos importantes para que possamos estruturar uma classe com mais segurança.

4.1.2 - Classes

Como estamos trabalhando com POO, o nosso objetivo final é construir um Objeto que tenha suas características. O Objeto é criado a partir de uma “Classe”. Ela irá servir como modelo para criação do Objeto. Vamos fazer uma comparação do dia-a-dia para entendermos melhor.

Uma casa não é construído de qualquer forma, o pedreiro irá construir a casa de acordo com as instruções da planta. Como tamanho do banheiro, quantos quartos, onde estará a porta de entrada. A planta da casa, seria uma “Classe” que serve como referência na construção dessa e de outras casa que são os “Objetos”.



= Objeto

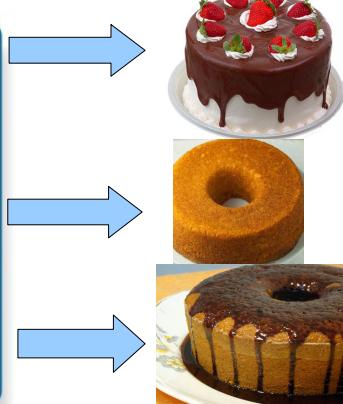


= Classe

Outro exemplo:

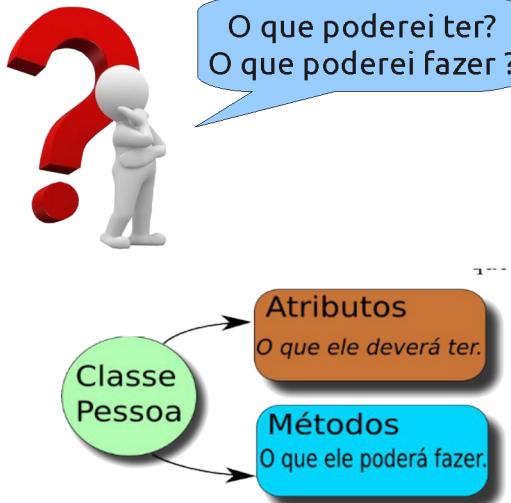
Receita de bolo:

Misture os ingredientes
Unte a forma com manteiga
Despeje a mistura na forma
Se houver coco ralado
então despeje sobre a mistura
Leve a forma ao forno
Enquanto não corar
deixe a forma no forno
Retire do forno
Deixe esfriar



Da mesma forma, a receita de bolo é a “classe” e os bolos são os “objetos”. E também da mesma forma a mesma receita serve como base para criação de vários tipos de bolo.

Iremos aprender, como criar uma “Classe”, ou seja, referência para futuros “Objetos”. Imagine que queremos criar um “Pessoa”. Para poder criar uma “Pessoa” teremos antes que fazer 2 perguntas básicas.

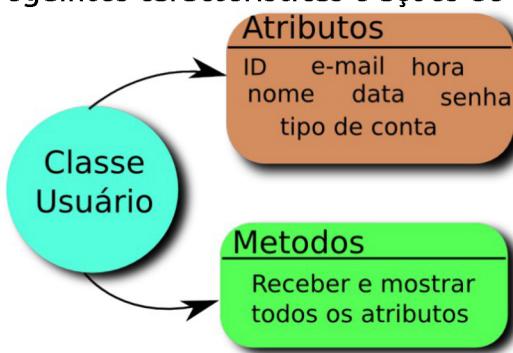


→ Anote tudo que uma pessoa pode ter, como nome, data de nascimento, RG, CPF, etc. Para a resposta da primeira pergunta, nós temos o que chamamos de “**Atributos da classe Pessoa**”.

→ Anote Tudo que uma pessoa pode fazer, ou seja, suas ações, como acordar, dormir, falar o seu nome, etc. Para a resposta da segunda pergunta nós temos o que chamamos de “**Métodos da classe Pessoa**”.

Exemplo:

Imagine que iremos criar um usuário para o nosso site, seguindo a mesma linha de raciocínio acima temos as seguintes características e ações de uma classe “**Usuário**”:



Exercício rápido:

- Crie com papel e caneta, uma classe para: Pessoa, Carro, bicicleta e computador.
- Para cada classe criada, faça a pergunta: “O que minha classe deverá ter?” “O que ela poderá fazer?”.
- Organize as informações em atributos e métodos e esteja pronto, para implementar em código para o nosso próximo tópico.

4.1.3 – Construindo uma classe em PHP

Antes de criarmos uma classe, vamos organizar essa classe dentro de um diretório dentro de um **projeto** e chamá-la de “**class**”. Na pasta “class”, criaremos uma “**Classe PHP**”. E iremos chamá-la de “**usuarios.class.php**”.

```
POO
  ↓
    class
      usuarios.class.php
```

Com a classe criada, temos que criar os atributos e métodos.

```
class Usuarios{
}
```

4.1.3.1 - Atributos

Nesse momento precisamos criar as variáveis que farão parte da estrutura básica da classe usuário. O **encapsulamento** (que veremos adiante) “**private**” foi usado para proteger essas variáveis. Lembrando que dependendo da situação, podemos deixá-las como “**protected**”, ou “**public**”. O encapsulamento também deverá ser usado nos métodos, veremos a seguir.

```
private $id;
private $usuario;
private $senha;
private $email;
private $tipo_conta;
```

4.1.3.2 - Métodos

Para o nosso usuário, teremos também que usar um método especial, chamado de “**método mágico**”. Esses métodos têm nomes e funções específicas e começam com “**__**” (2 underlines). O método mágico que iremos trabalhar se chama: **construct**. Este método é um **construtor** da classe, ou seja, a nossa classe só poderá existir se satisfazer plenamente a chamada do **class** Usuários{

```
private $id;
private $usuario;
private $senha;
private $email;
private $tipo_conta;

//métodos são funções!!!
public function __construct($novo_usuario, $nova_senha){
    $this->usuario = $novo_usuario;
    $this->senha = $nova_senha;
}
```



Atenção: Quando precisamos chamar os atributos dentro da classe usamos o “**\$this->**” assim o PHP irá fazer as devidas atribuições como mostra na imagem acima.

Veremos mais alguns métodos mágicos em breve.

4.1.4 – Métodos GET e SET

Com o nosso construtor feito, vamos para os demais métodos. Temos algumas nomenclaturas de bons costumes para métodos, veja uma delas na tabela abaixo:

Método	Descrição
Sets	Métodos para alterar atributos de uma classe
Gets	Métodos para retornar um valor de atributo da classe

4.1.4.1 - Método de Armazenamento de Atributos(SET)

Estes métodos são responsáveis por armazena valores nos atributos, por segurança os atributos estão encapsulados como “**private**” nos impedido acessá-los diretamente, então precisamos de métodos internos à classe para a realizar essa operação, veja um exemplo de como podemos alterar o atributo id da nossa classe.

```
public function set_id($novo_id){  
    $this->$id = $novo_id;  
}
```

Perceba que por padrão de nomenclatura, os métodos que tem essa função, começa com o termo “set” seguido pelo atributo que queremos alterar. Podemos receber outros objetos como parâmetro.

4.1.4.2 - Método de Retorno de Atributos(GET)

Os métodos podem realizar uma ação interna e podem influenciar em outros métodos através do seu Retorno. Ele pode representar os tipos primitivos de dados que conhecemos como: **int**, **string**, **double**, **float**, **char**, **boolean** por exemplo, um outro objeto, ou pode retornar **vazio**, simbolizado por “**void**”. Os métodos que tiver retorno, deverá ter dentro do seu corpo um “**return**” que irá representar o resultado final da minha operação. Veja o exemplo abaixo:

```
public function get_id(){  
    return $this->$id;  
}
```

Sabemos que o “**id**” é uma atributo privado do tipo numérico que não poderá ser retornado em outras classes ou páginas, então criamos um método “**public**” para verificar o valor desse atributo e retorná-lo para outras classes ou para a nossa página WEB.

Perceba que por padrão de nomenclatura, os métodos que tem essa função, começa com o termo “**get**” seguido pelo atributo que queremos alterar. Podemos também retornar outros objetos.

Exercício rápido:

- Lembra das classes criadas: Pessoa, Carro, bicicleta, Casa, Computador?
- Como seria essa implementação em código levando em consideração ao que foi anotado no exercícios passado?

4.1.4.3 – Incluindo os Métodos GET e SET

Incluindo na classe de usuário estes métodos, temos:

```

class Usuarios{
    private $id;
    private $usuario;
    private $senha;
    private $email;
    private $tipo_conta;

    //métodos são funções!!!
    public function __construct($novo_usuario, $nova_senha){
        $this->usuario = $novo_usuario;
        $this->senha = $nova_senha;
    }

    public function set_id($novo_id){
        $this->id = $novo_id;
    }
    public function get_id(){
        return $this->id;
    }

    public function set_email($novo_email){
        $this->email = $novo_email;
    }
    public function get_email(){
        return $this->email;
    }
}

```



Atenção: Perceba que o método “**set_id**”, pede um valor como parâmetro, esse parâmetro modifica o seu atributo. O Método “**get_id**” não irá precisar de parâmetro nenhum, pois sua função é apenas retornar o valor do atributo. Iremos ver nos próximos tópicos mais a fundo sobre esse assunto.

4.1.5 - Instanciando a classe criada

Agora que criamos a classe, vamos criar um arquivo chamado **testa_usuario.php** e construir um determinado usuário. Para usar a classe que está em uma outra página PHP, primeiro temos que incluir o arquivo. Podemos usar 4 métodos básicos(que já vimos):

- **include**: comando usado para inserir recursos externos à página.
- **require**: faz o mesmo procedimento que o **include**, porém se a página que for incluída der erro, toda a página não será aberta. Usamos esse método para proteger alguns dados dos usuários.
- **include_once**: faz o mesmo procedimento que o **include**, mas inclui “apenas” a página referenciada.
- **require_once**: faz o mesmo procedimento que o **require**, mas inclui “apenas” a página referenciada.

Estes métodos inclui na página código “php” ou páginas “html” por exemplo. Veja como

estanciar um objeto usuário na próxima página:

```
//incluindo a minha classe usuários em index.php
include_once 'class/usuarios.class.php';

//Construindo um novo usuário.
$primeiro_usuario = new Usuarios("Alessandro", "qwe123");
//Alterando o atributo "id" do meu novo usuário.
$primeiro_usuario->set_id(1);

//Imprimindo dados criados na tela do navegador.
echo "Meu ID é: ".$primeiro_usuario->get_id().'  
';
echo "Meu Usuário é: ".$primeiro_usuario->get_usuario().'  
';
echo "Minha Senha é: ".$primeiro_usuario->get_senha().'  
';
```



Atenção: Perceba que também usamos o sinal “->” para chamar métodos das classes.

Então finalmente entrando no navegador temos o seguinte resultado da imagem abaixo:

Meu ID é: 1
 Meu Usuário é: Alessandro
 Minha Senha é: qwe123

Podemos criar métodos que não sejam do tipo “**get/set**”. Podemos criar métodos para fazer qualquer operação com o que foi aprendido em todo o nosso curso de PHP , como inserir operações matemáticas, estruturas condicionais, estruturas de repetição, etc... Veja os exemplos abaixo:

```
//recebe array de numeros para a media
public function media($numeros){
    //pega total de numeros no array
    $n_numeros = count($numeros);
    ou
    //soma os numeros contidos no array
    $soma_total = array_sum($numeros);
    }

    //calculo da média...
    $media = $soma_total/$n_numeros;

    return $media;
}

public function ver_maioridade($idade){
    if($idade >= 18){
        echo "De maior!";
    } else{
        echo "De menor!";
    }
}
```



Parabéns jovem *padawan*, os princípios básicos para a criação de classes, atributos, métodos, construtores e como chamá-los está feito. Agora vamos em frente!

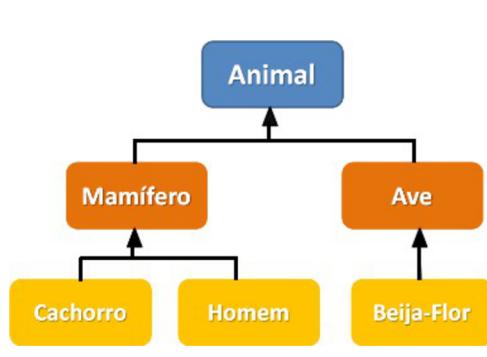
4.2 - 4 pilares da OO

Uma linguagem é caracterizada como Orientada a Objetos quando atende a estes quatro elementos que iremos ver.

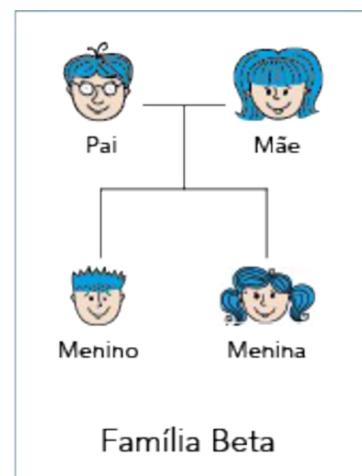
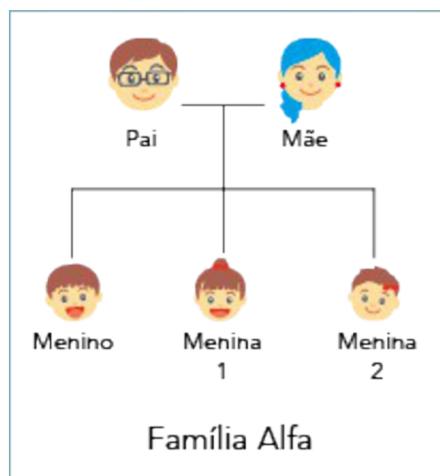
Quando se programa orientado a objetos alguns itens são essenciais para caracterizá-lo, a partir desses itens é que toda a lógica pode ser aplicada e consequentemente todas as vantagens dessa metodologia podem ser aproveitadas. Hoje vamos explicar cada uma dessas características que são tratadas como pilares.

4.2.1 - Herança

O uso de herança é muito importante na criação de classes orientadas a objetos, com ela você poderá criar uma nova classe utilizando uma outra classe que já existe. Este recurso lhe permite organizar melhor o seu código reutilizando código das classes herdadas, facilitando a manutenção do código e evitando reescrita de algoritmo. Agora, iremos aprender a gerar uma interação entre classes criadas e como utilizá-las.



Uma das grandes vantagens da programação orientada a objetos é justamente o reaproveitamento do código fonte, isso só é possível graças a herança. Com ela uma classe pode herdar as propriedades de uma classe mestra como por exemplo, a classe cachorro herda as propriedades da classe mamífero. Essa analogia indica que o cachorro tem características próprias, mas ele é um mamífero e compartilha das mesmas características que um gato, por exemplo, porém cada qual com suas peculiaridades;



As classes funcionam como uma família, os filhos herdam características dos seus pais, porém nada impede que eles também tenham suas características.

4.2.1.1 - Extends

Ok, usaremos a classe chamada “**Usuários**” que criamos nos exemplos anteriores, que contem suas características e ações. Porém se nós quisermos criar uma nova classe chamada “**Alunos**”, “**Funcionários**”, “**Clientes**” ou “**Professores**” teremos novamente que criar a estrutura de um usuário dentro de cada uma delas?

Para evitar reescrita de código, podemos fazer com que essas classe por exemplo herde todas as características de **usuários**. Neste nosso caso, a classe “**Usuários**” é a classe Pai e as demais classes que vão herdar características dela, serão classificadas como classes filhas.

```
<?php
    include_once 'usuarios.class.php';

    class Alunos extends Usuarios{

        //atributo que aluno tem, mas usuario nao tem
        private $matricula;

        public function __construct($usuario, $senha, $matricula){
            //acessando os metodos da classe pai...
            parent::__construct($usuario, $senha);

            $this->matricula = $matricula;
        }

        public function get_matricula(){
            return $this->matricula;
        }

    }

?>
```

Veja a imagem abaixo, de como seria uma Classe “**Alunos**” herdando características da classe pai que já criamos nos capítulos passados chamada “**Usuários**”.

Precisamos primeiro incluir a classe que será herdada com “**include_once**” em seguida usamos o “**extends**” para referenciar qual classe será herdada.

Sintaxe de herança em PHP:

class Nome_da_classe_a_ser_criada extends Nome_da_classe_a_ser_herdada

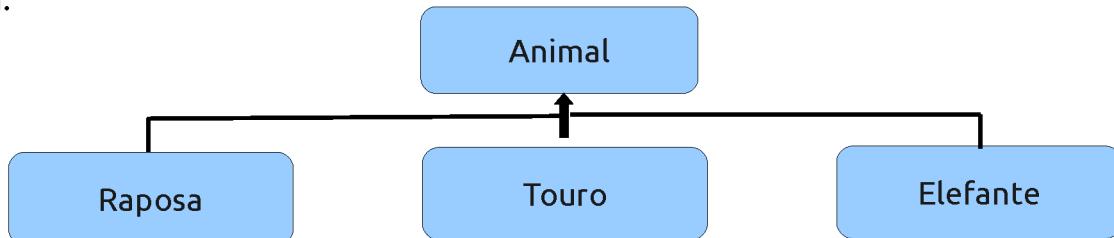
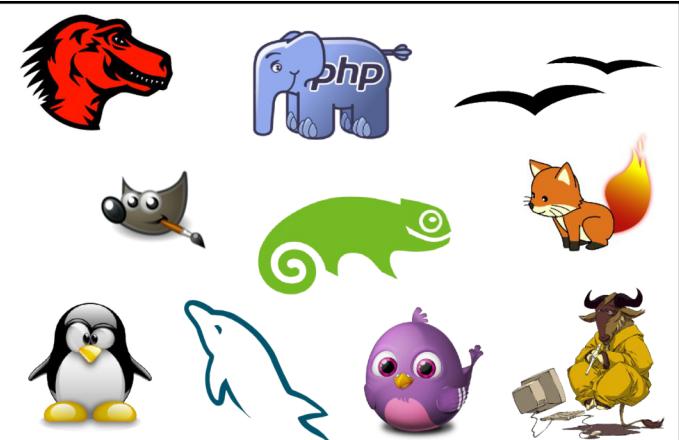
Além das características herdadas, a classe filha, também herda as necessidades da classe pai, no nosso exemplo, a classe “**Usuários**”, para existir, precisa de “**usuario**” e “**senha**”. Assim um aluno, para que possa existir, também precisará obrigatoriamente no mínimo essas características. Alimentamos essas dependência, chamando métodos da classe pai usando o comando: “**parent::**”. Com esse comando temos acesso a todos os métodos da classe pai, inclusive o construtor, então a qualquer momento temos acesso as informações da classe pai dentro da classe filha, claro obedecendo as regras de encapsulamento. Adicionamos ao Aluno um atributo “**matricula**” que será obrigatório para a existência dessa classe, e um método responsável por resgatar a matricula desse aluno.

Outro exemplo:

Para demonstrar como definir o relacionamento de herança, considera-se o contexto de ter que desenvolver um programa que simule o comportamento de vários tipos de animais como mostrado na figura a seguir em um determinado ambiente.

Inicialmente, apenas um grupo de animais estará presente neste ambiente e cada animal deve ser representado por um objeto. Além disso, os animais movem-se no ambiente ao seu modo e podem fazer um conjunto de coisas. O programa deverá prever que novos tipos de animais poderão ser incluídos no ambiente.

O primeiro passo é observar cada animal e definir o que cada objeto do tipo animal tem em comum no que diz respeito aos atributos e comportamentos (métodos). Deve-se definir também como os tipos de animais se relacionam. Inicialmente, o programa deve simular o comportamento dos animais ilustrados na figura, ou seja, de um pinguim, de um lobo, de um touro, de um elefante, de pássaros, de um dinossauro, de um camaleão, de um golfinho, e de uma raposa.



O segundo passo consiste em projetar a superclasse, ou seja, a classe que representa o estado e o comportamento em comum a todos os animais. Para este exemplo, como todos os objetos são animais, a superclasse foi denominada como *Animal* e as variáveis e métodos em comuns a todos os animais foram atribuídos a ela, como ilustrado na Figura.

Neste exemplo, cinco variáveis foram definidas: (o tipo de comida que o animal come), (o nível de fome do animal), (representação da altura e largura do espaço que o animal vagará ao seu redor) e (as coordenadas X e Y do animal no espaço). Além disso, quatro métodos foram definidos para definir o comportamento dos animais: (comportamento do animal ao fazer algum ruído), (comportamento 57 do animal ao comer), (comportamento do animal dormindo) e (comportamento do animal quando não está nem dormindo nem comendo, provavelmente vagueando no ambiente). O terceiro passo é o de decidir se alguma subclasse precisa de comportamentos (métodos) específicos ao seu tipo de subclasse.

Analizando a classe *Animal*, pode-se extrair que os métodos devem ter implementações diferentes em cada classe de animais, afinal, cada tipo de animal tem comportamento

distinto ao comer e fazer ruídos. Assim, esses métodos da superclasse devem ser sobrescritos nas subclasses, ou seja, redefinidos. Os métodos não foram escolhidos para serem redefinidos por achar que esses comportamentos podem ser generalizados a todos os tipos de animais.

Mais um exemplo:

Imaginem que vocês precisam criar um programa de gerencias de conta de Banco, todos os clientes terão um determinado tipo de conta, Conta poupança e Conta especial, veja no diagrama abaixo.

Neste caso temos a classe pai “**Conta**” também chamada de superclasse. Também temos 2 classes filhas chamadas também de subclasses (**CPoupança** e **CEspecial**) . Estas contas têm características em comum como, código, saldo e CPF, porém cada tipo de conta tem sua característica única, a Poupança tem uma data de Abertura, já a CEspecial tem um limite.

Exercício Rápido:

1º) Como seriam as classes mostradas no diagrama em PHP? Modele e monte essas estruturas organizadas de acordo com o que aprendeu.

4.2.2 - Encapsulamento

Encapsulamento é uma forma de proteção do nosso código, existem informações que queremos que algumas classes vejam, outras não. Imagine a sua conta de email. A conta do seu usuário poderá ser vista por outras pessoas, pois ele irá identificar o seu email dos demais, porém a sua senha é algo “**privado**” onde só você poderá ter acesso a ela. Da mesma forma que queremos mostrar ou proteger os nossos dados, uma classe também tem as mesmas necessidades e podemos defini-las através do encapsulamento.

→ **public**: ou público, qualquer classe pode ter acesso.

→ **private**: ou privado, apenas os métodos da própria classe podem manipular o atributo .

→ **protected**: ou protegido, pode ser acessado apenas pela própria classe ou pelas suas subclasses.

Por boas práticas de programação, os atributos da nossa classe **Usuários** estará como “**private**” .

4.2.3 - Polimorfismo

Usamos esse recurso quando queremos modificar métodos herdados. Para isso, temos que aprender o conceito e prática de **sobrescrita**(veremos logo-logo).

4.2.3.1 - Sobreescrita

Usamos sobreescrita de métodos quando temos que modificar um método herdado da classe pai, para sobreescrivar os métodos teremos que criar o mesmo método herdado na classe filha com as assinaturas(nome) iguais. Levando em consideração a classe Animal

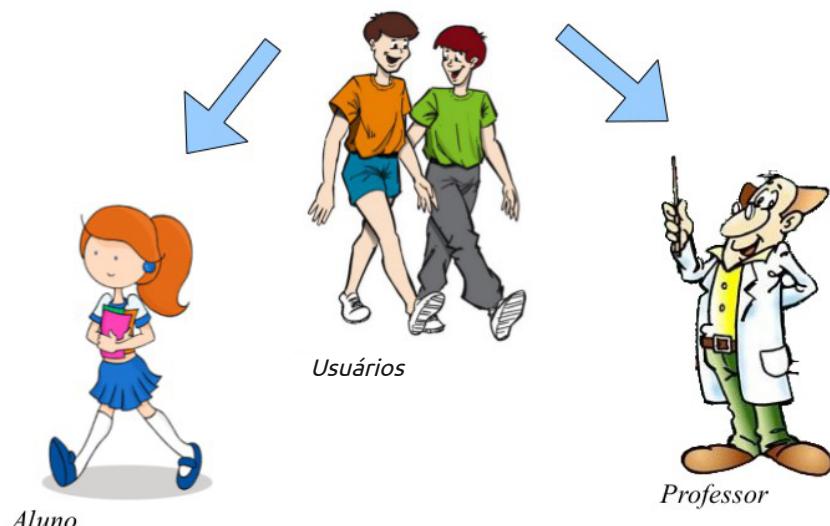
criada no tópico anterior, veja no exemplo a seguir:

```
public class Cachorro extends Animal{
    @Override
    public function comer(){
        echo "Ração...";}
}
```

No exemplo anterior, o cachorro já herda o método **comer**, porém o animal come qualquer alimento, mas este cachorro não come qualquer alimento, ele só come ração. O termo “**@Override**” no exemplo, não é obrigatório, apenas para avisar a IDE que este é um método de sobrescrita.

4.2.3.2 – Usando o Polimorfismo

Podemos dizer que polimorfismo é quando temos métodos distintos entre classes filhas que referenciam-se a um mesmo Pai. Veja o exemplo abaixo:



Note os usuários acima. Eles podem ter várias ações diferenciadas em um sistema, dependendo da situação temos que usar permissões apropriadas. Um professor também é um usuário, porém do tipo “**Professor**” e ele precisa de um tipo específico de permissão para acessar o sistema. O aluno também é um Usuário, porém ele precisa de permissões adequadas para assistir as web-aulas. Assim nós temos a classe “**Professor**” e “**Aluno**” herdando de “**Usuários**”, porém cada uma delas possuem métodos diferentes quando nos referimos ao modo de usar o sistema. Vamos ver como ficaria em código PHP o nosso Polimorfismo.

```
public class Usuarios{
    /* ... */
    public function logar(){
        echo "Login de Usuário qualquer";
    }
}

public class Professores{
    /* ... */
    public function logar(){
        echo "Login de Usuário Professor";
    }
}
```

Classe Pai

Classe filha de Usuários que sobrescreve o método logar().

```
public class Alunos{  
    /* ... */  
    public function logar(){  
        echo "Login de Usuário Aluno"; Classe filha de Usuários que sobrescreve o método logar().  
    }  
}
```

4.2.4 - Abstração

Esse pilar, como o próprio nome já diz, visa **abstrair** algo do mundo real e transforma-lo em um objeto na programação. Esse objeto será composto por uma identidade, propriedades e métodos. Dessa forma fica mais simples trazer para dentro da programação o problema que precisa ser resolvido ou o processo que precisará ser automatizado.

4.2.4.1 – Classe Abstrata

Para entender o conceito de classe abstrata, precisamos primeiro compreender melhor na o significado da palavra “**Abstrata**”. Refere-se a algo que não existe fisicamente, então se convertermos essa expressão em PHP podemos dizer que Classe Abstrata é uma classe que não pode ser um Objeto. Assim ela precisará ser herdada ou implementada. Vamos voltar ao nosso primeiro exemplo. Onde temos Usuários, Alunos e Professores. Imagine que nas regras de negócio da nosso meio acadêmico, não precisamos manipular diretamente um Objeto Usuário, porém essa classe é essencial para a formação de Alunos e Professor, Objetos importantes no nosso programa veja a seguir como tornar uma classe abstrata.

Apenas adicionamos o comando “**abstract**” antes da **abstract class** Usuários{ classe, assim podemos garantir que essa classe não poderá existir fisicamente. Veja o que acontece quando estanciamos essa nova classe:

```
$usuario = new Usuarios("Alessandro", "qwe123");  
$aluno = new Alunos("Anthony", "123456", "20140311");  
$professor = new Professor("Rapadura", "doce", "PHP");
```

O Programa irá encontrar um erro, pois a classe Usuários agora é uma classe abstrata e não poderá ser instanciada diretamente, porém se analisarmos a classe “Alunos”, podemos notar que não existe nenhum erro.

As classes abstratas, também poderão ser implementada como interface, tópico que iremos ver em breve.

4.3 – Incrementando a OO

Veremos agora mais alguns métodos e estruturas que podem ajudar no desenvolvimento com PHP Orientado a Objetos.

4.3.1 – Variáveis Estáticas

Variáveis estáticas são valores únicos para qualquer objeto de uma determinada classe. Isso quer dizer que se uma determinada variável estática for alterada em uma classe, essa variável terá o mesmo valor para todos os outros objetos da mesma classe. Normalmente usamos esse método todos os objetos de uma mesma classe tem uma variável que terá que ser idêntica. Imagine a seguinte situação.

A Classe “Usuários” poderá ter um atributo chamado: **onde_vivo = “Planeta Terra”**. Todos moramos em um planeta que conhecemos por “Planeta Terra”. Porém pesquisas recentes mostram que o nome do nosso planeta deveria ser chamado de “Planeta Água”, pois o fato de existir água, nos torna diferente dos demais planetas.

Se tivermos 300 objetos do tipo Usuário, como alterar **onde_vivo = “Planeta Água”** em todos os objetos em uma única instrução sem precisar percorrer todos eles?



É claro que não, basta criar uma variável estática no atributo **“onde_vivo”**, assim se alteramos em 1 dos objetos essa variável, todas elas sofrerão a mesma alteração. Veja como criá-lo:

```
class Usuarios{
    /* ... */

    private static $onde_vivo = "Planeta Água";
    /* ... */

    public function set_onde_vivo($onde_vivo){
        self::$onde_vivo = $onde_vivo;
    }

    public function get_onde_vivo($onde_vivo){
        return self::$onde_vivo
    }
}
```

Perceba que para criar uma variável estática, basta adicionarmos o termo **“static”** nela.

Para acessarmos as variáveis nos métodos precisamos usar o termo: **“self::”**.

Faça o teste instanciando mais de um objeto do tipo Usuários, chamando o método **“set_onde_vivo()”** em um dos objetos. Em seguida, procure ver nos demais objetos instanciados as alterações feitas pelo método **“get_onde_vivo()”**.

4.3.2 - Métodos Estáticos

Métodos estáticos são usados para criar classes que não precisam ser estanciados. Usamos esses métodos quando queremos fazer algumas de verificação de validade, CPF, RG, registros de login e senha, podemos usar também para estabelecer algumas sequência de *tags* desejáveis dentre várias outras funções, veja como construir uma fila de Artigos em uma página simples usando métodos estáticos:

```
class List_artigos{
    public static function separa_artigos($n_artigos){
        for($i = 1; $i <= $n_artigos; $i++){
            echo '<h2> Artigo '.$i.'</h2>';
            echo "Conteúdo<br>";
            echo '<hr>|';
        }
    }
}
```

Para criar um método estático, basta adicionarmos o termo “static” no método. Ao invés de instanciar essa classe como aprendemos antes, você poderá chamar diretamente com “**Nome_da_classe::nome_do_metodo**”. Neste exemplo, a classe é “**List_artigos**” então vamos chamá-lo com “**List_artigos::separa_artigos(numero)**” veja como implementá-lo:

```
include_once 'List_artigos.class.php';
```

```
List_artigos::separa_artigos(3);
```

O resultado será:

Artigo 1

Conteúdo

Artigo 2

Conteúdo

Artigo 3

Conteúdo



Cuidado: Métodos estáticos podem ter atributos, Contanto que estes atributos também sejam estáticos. Variáveis internas ao métodos poderão ser usadas sem problemas.

4.3.3 - Exercícios

1º) Imagine outras situações onde podemos criar uma variável estática e procure implementar em alguma classe criada ou em um classe nova que possa criar.

2º) No exemplo acima, faria alguma diferença se o encapsulamento da variável estática fosse “private”?

3º) Quais seriam as vantagens ou desvantagens quanto a segurança dessa informação encapsulada?

4º) Crie uma classe que contenha as informações em um vetor dos classes Criados e métodos que possam receber e Enviar essas informações em uma tabela.

4.3.4 – Métodos Mágicos

Os métodos Mágicos, são métodos que tem características especiais, é muito simples identificá-los, são métodos que tem 2 “underlines” na sua nomenclatura. Não é obrigatório o seu uso, porém estes métodos já trazem com eles algumas funcionalidades especiais, sem necessária mente precisarmos desenvolvê-las, iremos ver alguns métodos mágicos agora.

4.3.4.1 - construct

Esse método mágico é o mais conhecido de todos, vimos eles no capítulo anterior, vamos recapitular então: como o próprio nome já diz, “**construct**”, significa construção, esse método tem como função ser inicializado no momento da instancia de um objeto, vamos observar novamente a utilização desse método na classe Usuários:

```
public function __construct($usuario, $senha) {
    $this->usuario = $usuario;
    $this->senha = $senha;
}
```

Com a função “construct” implementada, qualquer classe só poderá existir se obedecer as regras de parâmetros desse método, no exemplo da classe usuário, a cada vez que eu instanciar um novo objeto usuário, eu terei obrigatoriamente que preencher um “**usuário**” e uma “**senha**”.

4.3.4.2 - destruct

Como o nome já diz, “**destruct**” significa eliminação, quando os objetos não são mais usados pelo algoritmo, podemos eliminá-los da memória, o método mágico “destruct” é responsável por realizar uma ação ao fim da sua existência, vamos criar esse método em uma classe **Alunos** para ver como funciona.

Neste algorítimo, pedimos para ele imprimir a mensagem: “**O Objeto foi finalizado**” quando o interpretador PHP deixar de usá-la.

```
class Alunos extends Usuarios{
    /* ... */

    public function app(){
        echo "Olá, sou um aluno!<br>";
    }

    public function __destruct(){
        echo "O Objeto foi finalizado";
    }
}
```

Assim, podemos instanciar ela e usar os seus métodos e ter a seguinte resposta no browser:

```
include_once 'usuarios.class.php';
include_once 'alunos.class.php';

$aluno = new Alunos();
$aluno->app();
```



Oi, sou um aluno!
O Objeto foi finalizado

4.3.4.3 - toString

Como o próprio nome já diz, “**toString**” significa “**tornar sequência de caracteres**”, esse método é responsável por representar um valor ao tentarmos ter como resposta o próprio objeto. Nos exemplos anteriores, retornamos valores ou atributos através de métodos **get**, porém podemos fazer com que o próprio objeto retorne um valor. Veja na Imagem a baixo a sua construção:

```
class Alunos extends Usuarios{
    private $matricula;

    public function __construct($usuario, $senha, $matricula){
        parent::__construct($usuario, $senha);
        $this->matricula = $matricula;
    }

    public function __toString(){
        return "sou Aluno: ".parent::get_usuario();
    }

    include_once 'usuarios.class.php';
    include_once 'alunos.class.php';

    $aluno = new Alunos("Alessandro", "qwe123", "201401");

    echo $aluno;
```

A diferença do método **__toString** para o método **get** é que não precisamos chamar um método para que possamos retornar esse valor, o próprio Objeto irá representar o valor que você desejar, veja como executá-lo:

sou Aluno: Alessandro

4.3.4.4 - invoke

Como o próprio nome já diz, “**invoke**”, significa “chamar/invocar”. Ele será chamado sempre quando o script tentar chamar o próprio objeto como uma função, veja sua construção na imagem abaixo:

```
class Alunos extends Usuarios{
    private $matricula;

    public function __construct($usuario, $senha, $matricula){
        parent::__construct($usuario, $senha);
        $this->matricula = $matricula;
    }

    public function __invoke($seu_nome){
        return "Olá $seu_nome <br> Usando objeto como método!";
    }
```

Olhe o uso e o Resultado:

```
include_once 'usuarios.class.php';
include_once 'alunos.class.php';

$aluno = new Alunos("Alessandro", "qwe123", "201401");

echo $aluno("Feitoza");
```



Olá Feitoza
Usando objeto como método!

Perceba que não precisamos usar o “**->**” para chamar nenhum método, o próprio objeto “**Aluno**” terá um método incorporado, conseguimos implementá-lo usando o Método mágico “**__invoke**”.

Existem outros métodos mágicos na API do PHP , continue pesquisando para estudos complementares. Veja mais sobre métodos mágicos em:

http://php.net/manual/pt_BR/language.oop5.magic.php

4.4.5 - Interface

Interface é uma espécie de superclasse 100% abstrata que define os métodos que uma subclasse deve suportar, mas não como esse suporte deve ser implementado. Isso porque ao criar uma interface, estará definindo um contrato com o que a classe pode fazer, sem mencionar nada sobre como o fará. Qualquer tipo de classe e de qualquer árvore de herança pode implementar uma interface. Usamos a função **implements** para implementar as interfaces em uma outra classe.

O que iremos construir são regras de convivência entre os alunos, nada melhor que usar as regras de bons modos. Iremos criar uma interface com métodos que contenham ações de boa convivência entre aluno, funcionários e professores na escola, as ações que iremos criar são basicamente 3: “Obrigado”, “Bom dia” e “Com licença”. Nesse momento estou querendo informar o PHP que qualquer classe que for implementada esse método, terá que ter obrigatoriamente esses 3 métodos obedecendo até mesmo os parâmetros que cada função pede.

```
interface bons_modos_aulas{
    public function obrigado();
    public function bom_dia();
    public function com_licenca($nome);
}
```

Mas onde está o corpo dos métodos? Assim estes métodos não estão fazendo nada!!!

A ideia é essa mesmo, não precisamos ainda dizer como será feito, esse procedimento é realizado apenas para lembrá-lo que você precisa implementar esses métodos.



Agora iremos implementar na classe “Alunos” que criamos anteriormente.

```
include_once 'bons_modos_aulas.php';

class Alunos extends Usuarios implements bons_modos_aulas{
```

Perceba que deu um erro na classe Alunos, isso acontece justamente porque a classe Alunos, agora, precisa dos 3 métodos: **obrigado**, **bom_dia** e **com_licenca**.

Vamos implementando os métodos exatamente como estão na Interface teremos:

```
include_once 'usuarios.class.php';
include_once 'bons_modos_aulas.php';

class Alunos extends Usuarios implements bons_modos_aulas{
    private $matricula;

    public function __construct($usuario, $senha, $matricula){
        parent::__construct($usuario, $senha);
        $this->matricula = $matricula;
    }

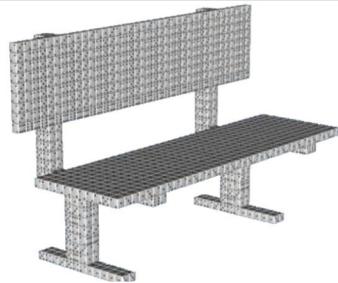
    public function obrigado(){
    }

    public function bom_dia(){
    }

    public function com_licenca(){
    }
}
```

Capítulo 5 – Banco de Dados(MySQL)

Os bancos de dados, são estruturas que guardam e manipulam dados para segurança e organização de sistemas, já que os dados ficam armazenados e a qualquer momento podem ser recuperados, e com isso podem ser geradas informações ao usuário.



5.1 - O que é o MYSQL?



O MySQL é um completo sistema de gerenciamento de bancos de dados relacional. Tem como características mais marcantes a estabilidade e agilidade. Foi criado por uma empresa sueca chamada TcX e vem sendo desenvolvido por mais de 10 anos. O seu intuito ao criá-lo foi de prover a seus clientes um produto altamente estável, rápido e seguro. Atualmente é um dos SGBDs (Sistema Gerenciador de Banco de Dados) mais utilizados na Internet. Várias linguagens de programação têm interface com este, como o PHP, Java (JDBC), Perl, TCL/TK, Python, C/C++, etc, e ainda ODBC.

Outro ponto forte é sua portabilidade. Existem versões para os mais diversos sistemas operacionais como Linux, FreeBSD, OpenBSD, NetBSD, Solaris, Windows 95/ 98/ NT/ 2000/ XP/ Vista/ Windows 7, HP-UX, AIX, etc.

Já vimos a instalação do MySQL no Linux no capítulo-1.2.3, onde instalamos também o PhpMyAdmin para administrarmos o banco de dados no modo gráfico e pelo navegador.

5.2 - Trabalhando com MySQL.

Para logarmos no mysql temos que abrir o Konsole(terminal) e digitar o seguinte comando:

mysql -u root -p

Nome do usuário.

-u → usuário, define o nome do usuário.

-p → password, pedido de senha.

Logo após esse comando uma senha será pedida. Digite a senha que foi definida na instalação do banco. Depois de logarmos visualizaremos a seguinte tela:

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> 
```

Agora estamos logado no gerenciador de banco de dados MySQL. Para sair do banco digite: **quit** ou **exit** e logo após der um **enter**.

Um banco de dados trabalha basicamente com tabelas, e cada tabela possui uma determinada quantidade de linhas(registros) e colunas(campos). As informações são

gravadas para futuramente serem inseridas, alteradas (atualizadas), ou deletadas.

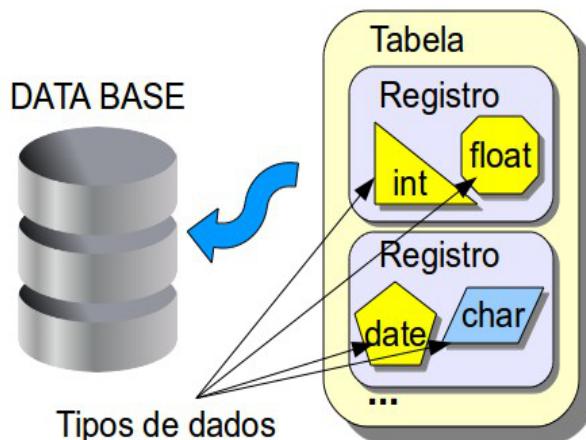
```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| agenda        |
| mysql          |
| nomes          |
| phpmyadmin    |
+-----+
5 rows in set (0.00 sec)
```

O primeiro passo é visualizarmos alguns bancos de dados que vem como padrão, para isso utilizamos comando **show databases**, veja ao lado:

Sempre utilize ";" para finalizar qualquer comando. Podemos perceber que na máquina que foi utilizado esse comando, foram encontrados cinco banco de dados.

5.3 - Estruturas de Dados

Os tipos de dados possuem diversas formas e tamanhos, permitindo ao programador criar tabelas específicas de acordo com suas necessidades. Podemos definir a hierarquia de banco de dados da seguinte forma: Banco de dados > Tabela > Registro > Tipo de dados, conforme a figura abaixo:



O MySQL provê um conjunto bem grande de tipos de dados, entre eles podemos citar os principais:

Tipo de dado	Descrição
CHAR(M)	strings de tamanho fixo entre 1 e 255 caracteres.
VARCHAR(M)	strings de tamanho flexível entre 1 e 255 caracteres.
VARCHAR	ocupa sempre o menor espaço possível, no entanto é 50% mais lento que o tipo CHAR.
INT(M) [Unsigned]	números inteiros entre -2147483648 e 2147483647. A opção "unsigned" pode ser usada na declaração mudando o intervalo para 0 e 4294967295 para inteiros não sinalizados.
FLOAT [(M,D)]	números decimais com D casas decimais.
DATE:	armazena informação relativa a datas. O formato default é 'YYYY-MM-DD' e as datas variam entre '0000-00-00' e '9999-12-31'. MySQL provê um poderoso conjunto de comandos para formatação e manipulação de datas.

TEXT/BLOB:	strings entre 255 e 65535 caracteres. A diferença entre TEXT e BLOB é que no primeiro o texto não é sensível ao caso e no segundo sim.
BIT ou BOOL	um número inteiro que pode ser 0(falso) ou 1(verdadeiro).
DOUBLE	número em vírgula flutuante de dupla precisão. Os valores permitidos vão desde -1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308

Existem outros tipos de dados, para mais detalhes consulte o site oficial do MYSQL (<http://www.mysql.com/>).

Abaixo temos exemplos dessas estruturas:

id	int(11)
nome	varchar(30)
data_nascimento	date
endereco	varchar(100)
cidade	varchar(50)
UF	char(2)

Tipos de dados e seus respectivos tamanhos

Nomes dos campos.

5.4 - Criando Banco e Tabelas

Agora que conhecemos algumas estruturas (tipos de dados), podemos começar a trabalhamos com alguns comandos, de acordo com a hierarquia criaremos o banco de dados e depois as tabelas:

5.4.1 - CREATE DATABASE

Após logar no MYSQL podemos cria qualquer base de dados com a seguinte sintaxe:

CREATE DATABASE 'nome_do_banco';

Exemplo:

```
mysql> CREATE DATABASE alunos;
Query OK, 1 row affected (0.02 sec)
mysql> 
```

Criamos então um banco de dados chamado alunos

Dica: Os comandos funcionam normalmente em minúsculo, mas por padrão é bom sempre utilizar os comando SQL em maiúsculo para diferenciá-los dos nomes e atribuições, facilitando assim a organização.

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| agenda |
| alunos |
| mysql |
| nomes |
| phpmyadmin |
+-----+
6 rows in set (0.00 sec)
```

Novamente com o comando SHOW DATABASE podemos visualizar todos os bancos de dados inclusive o que acabamos de criar:

Após criamos o banco de dados alunos, temos que conectarmo-nos a ele para iniciar qualquer operação. Para isso é utilizado comando **USE** com a seguinte sintaxe:

USE 'nome_do_banco';

Exemplo:

```
mysql> USE alunos;
Database changed
mysql> [REDACTED]
```

Com esse comando, todas as operações que forem realizadas afetarão de forma direta somente o banco de dados alunos. A mensagem *Database changed* significa banco de dados alterado.

5.4.2 - CREATE TABLE

Uma base de dados pode possuir várias tabelas, onde cada uma tem seu papel específico e estão relacionadas de forma direta ou não, para criarmos uma tabela usaremos a seguinte sintaxe:

```
CREATE TABLE 'nome_da_tabela' (
  'nome_dos_campos' tipo_de_dado(tamanho) 'comandos...',
  ...
  ...
);
```

Exemplo:

```
mysql> CREATE TABLE dados(
    -> id INT(11) NULL,
    -> nome VARCHAR(40) NULL,
    -> cidade VARCHAR(30) NULL,
    -> data DATE NULL
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> [REDACTED]
```

Nem todo tipo de dado tem um tamanho, como por exemplo o tipo DATE. Além disso o comando NULL significa que a informação inicial daquele campo é nulo.

Para visualizar detalhes da tabela criada utilize o comando **DESCRIBE 'nome_da_tabela'**, veja o exemplo:

Exemplo:

```
mysql> DESCRIBE dados;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| nome  | varchar(40)| YES  |     | NULL    |       |
| cidade | varchar(30)| YES  |     | NULL    |       |
| data  | date      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Para exibir as tabelas use o comando **SHOW TABLES**:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_alunos |
+-----+
| dados           |
+-----+
1 row in set (0.00 sec)
```

5.4.3 - DROP

Para apagar o banco de dados ou a tabela, usamos o comando **DROP** com a seguinte sintaxe:

```
DROP DATABASE 'nome_do_banco';
DROP TABLE 'nome_da_tabela';
```

//Usado para apagar o banco de dados
//Usado para apagar tabelas.

Exemplo:

```
mysql> DROP DATABASE nomes;
Query OK, 0 rows affected (0.00 sec)
```

Apaga o banco de dados **nomes**.

```
mysql> DROP TABLE dados;
Query OK, 0 rows affected (0.00 sec)
```

Apaga a tabela **dados**.

5.5 - Manipulando dados das tabelas

Para manipulação dos dados de uma tabela usamos os seguintes comandos:

5.5.1 - INSERT

Usado para inserir informações em uma tabela. Possui a seguinte sintaxe:

```
INSERT INTO nome_da_tabela(campo1,campo2,...,...)
VALUES(valor1,valor2,...,...);
```

O valor é inserido na tabela na mesma ordem que é definida os campos, ou seja, o campo1 receberá o valor1, o campo2 o valor2 e assim sucessivamente.

Exemplo:

```
mysql> INSERT INTO dados(id,nome,cidade,data) VALUES(1,"Alex Sousa", "Fortaleza", '2010-11-11');
Query OK, 1 row affected (0.00 sec)
```

5.5.2 - SELECT

Permite recuperar informações existentes nas tabelas. Sintaxe:

```
SELECT 'campo1','campo2',....,... FROM 'nome_da_tabela';
```

Exemplo:

Abaixo temos a exibição somente dos campos especificados após o SELECT. Caso queira visualizar todos os dados utilize asteriscos(*) conforme a segunda figura.

```
mysql> SELECT cidade,data FROM dados;
+-----+-----+
| cidade | data   |
+-----+-----+
| Fortaleza | 2010-11-11 |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM dados;
+-----+-----+-----+-----+
| id  | nome    | cidade  | data   |
+-----+-----+-----+-----+
| 1   | Alex Sousa | Fortaleza | 2010-11-11 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

5.5.3 - UPDATE

Usado para atualização dos dados, que altera os valores dos campos em uma tabela especificada com base em critérios específicos. Sintaxe:

```
UPDATE nome_da_tabela
SET campo1 = 'novo_valor',campo2 = 'novo_valor',....,...
WHERE critério;
```

Exemplo:

```
mysql> UPDATE dados SET id=2,nome='Alex da Silva' WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Imagine mais de 1000 registros e você precise alterar somente 1, nesse caso usamos o critério sendo definido com o id, ou seja, a alteração é feita aonde o id = 1, em nosso exemplo.

5.5.4 - DELETE

Remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE. Sintaxe:

DELETE FROM nome_da_tabela WHERE critério;

Exemplo:

```
mysql> DELETE FROM dados WHERE id = 3; Foi deletado a informação do id 3;
Query OK, 1 row affected (0.00 sec)
```

Obs.: muito cuidado quando usar o comando DELETE. Caso o critério saia errado por um erro do programador, poderá ser deletado todos os dados da tabela.

5.5.5 - COMANDOS ADICIONAIS

Vimos até agora alguns comandos principais, ainda existem outros comando que adicionado nos comando SELECT,UPDATE, DELETE forma uma conjunto de manipulações.

Comandos	Descrição
DISTINCT	Elimina linhas duplicadas na saída.
AS	um aliás para o nome da coluna.
FROM	Lista as tabelas na entrada.
WHERE	Critérios da seleção.
ORDER BY	Critério de ordenação das tabelas de saída. ASC ordem ascendente, DESC ordem descendente.
COUNT(*)	Usado no SELECT, conta quantos registros existem na tabela.
AND	Usado para dar continuidade a mais de um critério.
PRIMARY KEY	Chave primaria definida na criação da tabela, evitando duplicidade de dados em um mesmo campo.
FOREIGN KEY	Chave estrangeira definida na criação da tabela, usada para fazer ligações entre tabelas de um banco de dados relacional.

Dica: Para mais detalhes consulte materiais disponíveis na internet, livros, tutoriais, vídeo aulas e outras apostilas de MYSQL ou SQL.

5.6 - Trabalhando com PhpMyAdmin.

O phpMyadmin é basicamente um front-end para administração de bancos de dados MySQL.

Ele dispõe de uma série de recursos interessantes para administração do banco de dados. Com essa ferramenta é possível fazer desde atividades básicas de administração como criar, alterar, renomear tabelas, ou ainda fazer consultas usando SQL, como também gerenciar as conexões com o banco.

Vimos a sua instalação no capítulo 1.2.4, onde copiamos para a pasta var/www o seu conteúdo.

Para fazer o login na ferramenta abra o navegador e digite:

<http://localhost/phpmyadmin/>, aparecerá a seguinte tela:



Utilize o usuário e a senha que foi definida na instalação:

Exemplo:

Usuário: root

Senha: utd123456

Informações do servidor

Após o login visualizaremos a seguinte tela:

A imagem mostra a interface principal do phpMyAdmin. No topo, uma barra com links para "Banco de Dados", "SQL", "Status", "Variáveis", "Conjuntos de caracteres", "Engines", "Privilégios", "Replicação" e "Processos". Abaixo, uma barra "Ações" com links para "Alterar a senha" e "Sair". A seção central é intitulada "MySQL localhost" e contém uma barra "Interface" com "Linguagem - Language" (Português - Brazilian portuguese) e "Tema / Estilo" (Original). À direita, seções exibem informações sobre o servidor (Servidor: Localhost via UNIX socket, Versão do Servidor: 5.1.41-3ubuntu12.6, etc.), o Web server (Apache/2.2.14 (Ubuntu), etc.) e o próprio phpMyAdmin (Informações da versão: 3.3.2deb1, Documentação, Wiki, etc.). No lado esquerdo, uma lista mostra os bancos de dados disponíveis: agenda (3), alunos (1), information_schema (28), mysql (23) e phpmyadmin (8). Um link "Selecionar um Banco de Dados" aponta para a lista. Abaixo, uma barra "Lista de banco de dados" mostra a lista de bancos de dados.

Panel de criação de banco de dados.

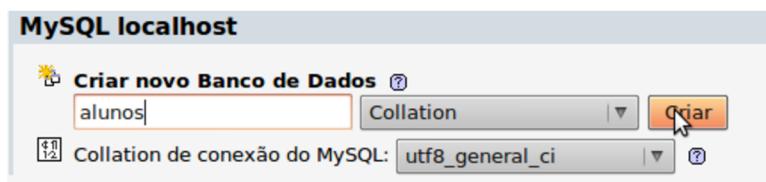
Acima temos a figura da página inicial do phpMyAdmin. No frame do lado esquerdo está a lista de todos os bancos de dados que existem no servidor. Clicando nesses bancos é possível navegar através das tabelas do banco escolhido e chegar até os campos desta tabela. Na página inicial do phpMyAdmin, mostrada acima, ainda é possível ver documentação e existe um box para criação de um novo banco de dados.

Todas as funcionalidades que utilizamos no Konsole(Terminal), anteriormente pode ser feito de forma mais simples e prática usando a interface gráficas do PhpMyAdmin.

Veremos agora algumas funcionalidades.

5.7 - Manipulando banco de dados no PhpMyadmin.

Podemos observar na tela inicial a opção para criarmos um banco. Insira um nome e clique em criar:



Após criarmos o banco de dados podemos observá-lo na lista de bancos ao lado esquerdo. Clique no banco que acabamos de criar.

Após clicarmos podemos **criar as tabelas** com a opção abaixo:

Após executarmos a tabela, visualizaremos uma série de informações detalhadas dos campos da tabela criada. Definimos então, os tipos de dados que a tabela vai suportar e seus respectivos tamanhos.

A tabela abaixo mostra os campos adicionados.

Campo	Tipo	tamanho	nulo	índice	Auto incremento
id	int	11		primary	x
nome	varchar	30	x		
rua	varchar	40	x		
bairro	varchar	40	x		
cidade	varchar	40	x		

Em **índice**, definimos o id como chave primária, onde os valores não podem ser duplicados, e também auto incremento, onde tem a finalidade de incrementar valores automaticamente sem precisar inseri-los de forma manual. Nulo significa que ao inserirmos um valor, os demais campos não tem a obrigação de serem preenchidos, com isso seu valor pode ficar null(nulo). Os demais itens do campo podem ser deixado do jeito que estão, agora é só clicar em Salvar

Dica: observe que a cada procedimento o comando a ser executado é exibido.

Tela de comandos:

A tabela `alunos`.`dados_pessoais` foi criada.

```
CREATE TABLE `alunos`.`dados_pessoais` (
  `id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  `nome` VARCHAR( 30 ) NULL ,
  `rua` VARCHAR( 40 ) NULL ,
  `bairro` VARCHAR( 40 ) NULL ,
  `cidade` VARCHAR( 40 ) NULL
) ENGINE = MYISAM ;
```

Após criarmos a tabela podemos **inserir** valores em seus campo, para isso entre no banco de dados criado e escolha a tabela conforme a figura abaixo:



Ao clicar aparecerá uma nova tela, escolha a opção inserir.

Opção para inserir dados.

Campo	Tipo	Collation	Atributos	Nulo	Padrão	Extra	Ação
<input type="checkbox"/> id	int(11)			Não	None	auto_increment	
<input type="checkbox"/> nome	varchar(30)	latin1_swedish_ci		Sim	NULL		
<input type="checkbox"/> rua	varchar(40)	latin1_swedish_ci		Sim	NULL		
<input type="checkbox"/> bairro	varchar(40)	latin1_swedish_ci		Sim	NULL		
<input type="checkbox"/> cidade	varchar(40)	latin1_swedish_ci		Sim	NULL		

↑ Marcar todos / Desmarcar todos Com marcados:

Visualização para impressão Ver relações Propor estrutura da tabela Adicionar 1 campo(s) No final da tabela No início da tabela Depois Executar

Ao entrar na opção inserir, será exibido a seguinte tela:

Campo	Tipo	Funções	Nulo	Valor	
id	int(11)				<input type="button" value="Executar"/>
nome	varchar(30)		<input checked="" type="checkbox"/>		
rua	varchar(40)		<input checked="" type="checkbox"/>		
bairro	varchar(40)		<input checked="" type="checkbox"/>		
cidade	varchar(40)		<input checked="" type="checkbox"/>		

Agora basta inserir os valores e executar em seguida, lembrando que não se deve preencher o campo id, pois ele é auto incremento.

Para alterar os valores dos campos entre novamente na tabela e observe a seguinte tela:

Campo	Tipo	Funções	Nulo	Valor	
id	int(11)			1	<input type="button" value="Executar"/>
nome	varchar(30)		<input type="checkbox"/>	Sara	
rua	varchar(40)		<input type="checkbox"/>	Rua D Pedro II	
bairro	varchar(40)		<input type="checkbox"/>	Maraponga	
cidade	varchar(40)		<input type="checkbox"/>	Fortaleza	

Ao clicar teremos a seguinte tela:

Agora basta editar os valores.

Até agora colocamos em prática os seguintes comandos:

INSERT → quando inserimos dados.

SELECT → quando visualizamos as informações da tabela.

UPDATE → quando editamos os valores adicionados.

DELETE → ao apagamos o campo da tabela.

Abaixo temos uma tabelas com algumas das principais funcionalidades do PhpMyAdmin.

Menu	Funcionalidade
Visualizar	Aqui é possível visualizar os registros da tabela e também modificar cada registro individualmente.
Estrutura	Nesta opção é possível ver os campos da tabela, também modificar estes campos e até mesmo excluí-los.
SQL	A opção SQL permite você digitar livremente um comando SQL diretamente numa caixa de texto.
Procurar	É possível realizar pesquisas nos registros da tabela. Este recurso é muito útil quando se deseja encontrar determinado registro em uma tabela muito extensa.
Inserir	Permite inserir um registro na tabela.
Exportar	Com a opção Export (Exportar), é possível gerar um arquivo texto com todo o conteúdo da tabela, juntamente com sua estrutura.
Importar	Importa um arquivo com comandos sql para dentro do banco.
Operações	Igual e do mesmo tipo permite realizar modificações na tabela, como renomear, alterar ordem, movê-la para outra database, alterar seu tipo e outras operações.
Privilégios	Área onde é feita a administração de usuários e suas permissões.
Designer	Mostra todas as tabelas e seus relacionamentos em diagramas.

5.8 – Exercícios

1º) Qual a definição que podemos atribuir ao conceito de banco de dados.

2º) Como podemos logar pelo Konsole no MYSQL.

3º) O que é estrutura de dados em MYSQL.

4º) Faça o seguinte exercício prático:

- a) crie uma tabela com os campos nome e sobrenome.
- b) após criar visualize a tabela.
- c) insira alguns valores na tabela, no caso nomes e sobrenomes de pessoas conhecidas.
- d) crie um SELECT para visualizar todas as informações gravadas no banco.
- e) agora atualize alguns dos campos com o comando UPDATE.

5º) Abra o PhpMyAdmin e visualize a tabela criada na questão 4.

6º) Qual a finalidade da ferramenta PhpMyadmin.

7º) Pesquise sobre banco de dados relacional e diga porque o MYSQL é considerado um banco de dados relacional.

8º) Pesquise e defina o que é chave primeira e chave estrangeira em banco de dados relacionais.

5.9 – Aprofundando no MySQL

Estudamos anteriormente a forma básica de selecionar ou consultar dados no banco de dados, vimos que para utilizar o comando SELECT, devemos indicar as tabelas que serão consultadas e as colunas que queremos recuperar.

No exemplo abaixo, as colunas nome e email são recuperadas da tabela Aluno.

```
1  SELECT nome, email FROM Aluno;
```

Recuperando as colunas nome e e-mail da tabela Aluno

Aluno	
nome	email
Rafael Cosentino	rafael.cosentino@k19.com.br
Jonas Hirata	jonas.hirata@k19.com.br

Resultado da consulta: SELECT nome, e-mail FROM Aluno

Estudamos também que quando todas as colunas devem ser recuperadas, é mais prático utilizar o caractere “*”. Veja o exemplo abaixo:

```
1  SELECT * FROM Aluno;
```

Recuperando todas as colunas da tabela Aluno

Aluno			
nome	email	telefone	altura
Rafael Cosentino	cosentino@k19.com.br	11 23873791	1.87
Jonas Hirata	hirata@k19.com.br	11 23873791	1.76

*Resultado da consulta: SELECT * FROM Aluno*

Agora vamos aprender a fazer consultas mais bem elaboradas. Os tópicos a seguir iram aprofundar esse assunto que é bastante presente no cotidiano do profissional que usa aplicações com banco de dados.

5.9.1 – Usando SELECT em duas ou mais tabelas

É possível recuperar colunas de várias tabelas. Nesse caso, os registros das tabelas consultadas são “cruzados”. Vamos fazer esse cruzamento entre a tabela Aluno e Professor.

A tabela professor tem as seguintes registros:

Professor	
nome	codigo
Marcelo Martins	1
Rafael Lobato	2

Observe a imagem seguinte:

```
1 SELECT * FROM Aluno, Professor;
```

Recuperando todas as colunas das tabelas Aluno e Professor

Aluno x Professor					
nome	email	telefone	altura	nome	codigo
Rafael Cosentino	cosentino@k19.com.br	11 23873791	1.87	Marcelo Martins	1
Rafael Cosentino	cosentino@k19.com.br	11 23873791	1.87	Rafael Lobato	2
Jonas Hirata	hirata@k19.com.br	11 23873791	1.76	Marcelo Martins	1
Jonas Hirata	hirata@k19.com.br	11 23873791	1.76	Rafael Lobato	2

*Resultado da consulta: SELECT * FROM Aluno, Professor*

Perceba que no resultado da Tabela 7.2 foi capturado todos os campos da tabela Aluno e da tabela Professor e que essas tabelas têm campos com nomes iguais. Caso façamos uma consulta dessa forma:

```
mysql> SELECT nome FROM Aluno, Professor ;
```

O MySQL ficará na dúvida de que tabela, Aluno ou Professor, será capturado os dados da coluna nome e irá retornar um erro desse tipo:

```
ERROR 1052 (23000): Column 'nome' in field list is ambiguous
```

Nesse caso, para recuperá-las, devemos eliminar a ambiguidade utilizando os nomes das tabelas junto ao campo que está gerando ambiguidade. Veja o exemplo abaixo:

```
1 SELECT Aluno.nome, Professor.nome FROM Aluno, Professor;
```

Recuperando as colunas nome das tabelas Aluno e Professor

Você pode usar essa forma do SELECT, estudada nesse tópico, não só com duas tabelas, como vimos acima, mas para cruzar dados de quantas tabelas você necessitar.

Aluno x Professor	
nome	nome
Rafael Cosentino	Marcelo Martins
Rafael Cosentino	Rafael Lobato
Jonas Hirata	Marcelo Martins
Jonas Hirata	Rafael Lobato

Resultado da consulta: SELECT Aluno.nome, Professor.nome FROM Aluno, Professor

5.9.2 – Usando o poder do WHERE

Os resultados de uma consulta podem ser filtrados através da cláusula WHERE. Veja o exemplo abaixo.

```
1 | SELECT * FROM Aluno WHERE altura > 1.80;
```

Aplicando o comando WHERE

Aluno			
nome	email	telefone	altura
Rafael Cosentino	cosentino@k19.com.br	11 23873791	1.87

Resultado da consulta: SELECT * FROM Aluno WHERE altura > 1.80

A consulta nos retornou todos os campos da tabela Aluno com os registros que atendem a seguinte condição: o dado armazenado no campo altura da tabela Aluno tem que ser maior que o valor 1.80. Dessa forma podemos fazer vários tipos de filtragens em nossas consultas.

Eis uma lista de algumas funções e operadores de comparação do MySQL Server que podem ser utilizados(as) com o comando WHERE:

5.9.2.1 - Comparação de igualdade “=”

```
1 | SELECT * FROM Aluno WHERE altura = 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura o valor 1.8

5.9.2.2 – Comparação de Diferença “!=” ou “<>”

```
1 | SELECT * FROM Aluno WHERE altura <> 1.8;
2 | SELECT * FROM Aluno WHERE altura != 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura um valor diferente de 1.8

5.9.2.3 – Comparação Menor ou igual “<=”

```
1 | SELECT * FROM Aluno WHERE altura <= 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura um valor menor ou igual a 1.8

5.9.2.4 – Comparação Menor “<”

```
1 | SELECT * FROM Aluno WHERE altura < 1.8;
```

Retorna todos os registros que tenham armazenado, no campo altura um valor menor que 1.8

5.9.2.5 – Comparação Maior ou igual “>=”

```
1 SELECT * FROM Aluno WHERE altura >= 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura um valor maior ou igual a 1.8

5.9.2.6 – Comparação Maior “>”

```
1 SELECT * FROM Aluno WHERE altura > 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura um valor maior que 1.8

5.9.2.7 – Comparação booleana “IS”

```
1 SELECT * FROM Aluno WHERE aprovado IS TRUE;
```

Retorna todos os registros que tenham armazenado no campo aprovado um valor diferente de 0;

5.9.2.8 – Comparação booleana “IS NOT”

```
1 SELECT * FROM Aluno WHERE aprovado IS NOT TRUE;
```

Retorna todos os registros que tenham armazenado no campo aprovado um valor igual a 0;

5.9.2.9 – Comparação booleana “IS NULL”

```
1 SELECT * FROM Aluno WHERE nome IS NULL;
```

Retorna todos os registros que tenham nada armazenado no campo nome;

5.9.2.9 – Comparação booleana “IS NOT NULL”

```
1 SELECT * FROM Aluno WHERE nome IS NOT NULL;
```

Retorna todos os registros que tenham algo armazenado no campo nome;

5.9.2.10 – Comparação estão entre “BETWEEN...AND”

```
1 SELECT * FROM Aluno WHERE altura BETWEEN 1.5 AND 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura valores entre 1.5 e 1.8;

5.9.2.11 – Comparação não estão entre “NOT BETWEEN... AND”

```
SELECT * FROM Aluno WHERE altura NOT BETWEEN 1.5 AND 1.8;
```

Retorna todos os registros que tenham armazenado no campo altura valores que não estejam entre 1.5 e 1.8;

5.9.2.12 – Comparação string inicial “LIKE”

```
1 SELECT * FROM Aluno WHERE nome LIKE 'Rafael%';
```

Retorna todos os registros que tenham armazenado, no campo nome, dados que iniciam com a String Rafael

5.9.2.13 – Comparação não é string inicial “NOT LIKE”

```
1 SELECT * FROM Aluno WHERE nome NOT LIKE 'Rafael%';
```

Retorna todos os registros que tenham armazenado, no campo nome, dados que não iniciam com a String Rafael

5.9.2.14 – Comparação “IN()”

```
1 SELECT * FROM Aluno WHERE altura IN (1.5, 1.6, 1.7, 1.8);
```

Retorna todos os registros que tenham armazenado, no campo altura, os valores iguais a 1.5, 1.6, 1.7 e 1.8

5.9.2.15 – Comparação “NOT IN()”

```
1 SELECT * FROM Aluno WHERE altura NOT IN (1.5, 1.6, 1.7, 1.8);
```

Retorna todos os registros que não tenham armazenado, no campo altura, os valores iguais a 1.5, 1.6, 1.7 e 1.8

Eis uma lista dos operadores lógicos do MySQL Server que também podem ser utilizados com o comando WHERE.

5.9.2.16 – Operador “NOT” ou “!”

```
1 SELECT * FROM Aluno WHERE NOT altura = 1.80;
2 SELECT * FROM Aluno WHERE !(altura = 1.80);
```

Retorna todos os registros que tenham armazenado, no campo altura, um valor diferente de 1.80

5.9.2.17 – Operador “AND” ou “&&”

condição 1

condição 2

```
1 SELECT * FROM Aluno WHERE altura < 1.8 AND nome LIKE 'Rafael%';
2 SELECT * FROM Aluno WHERE altura < 1.8 && nome LIKE 'Rafael%';
```

Retorna todos os registros que tenham armazenado, no campo altura, um valor menor que 1.80. E que tenha armazenado no campo nome um valor que inicie com a string Rafael, ou seja só serão capturados os registros que tiverem as duas condições verdadeiras.

5.9.2.18 – Operador “OU” ou “||”

```

1 SELECT * FROM Aluno WHERE altura < 1.8 OR nome LIKE 'Rafael%';
2 SELECT * FROM Aluno WHERE altura < 1.8 || nome LIKE 'Rafael%';

```

Retorna todos os registros que tenham armazenado, no campo altura, um valor menor que 1.80 **OU** que tenham armazenado no campo nome um valor que inicie com a string Rafael, ou seja só serão capturados os registros que tiverem pelo menos uma das condições verdadeiras.

5.9.2.19 – Operador “XOR”

```

1 SELECT * FROM Aluno WHERE altura < 1.8 XOR nome LIKE 'Rafael%';

```

Retorna todos os registros que tenham armazenado, no campo altura, um valor menor que 1.8 e não tenham armazenado no campo nome um texto que inicie com a string Rafael **OU** retorna todos os registros que tenham o valor armazenado no campo nome iniciando com a string Rafael e não tenham os valores armazenado no campo altura menores que 1.8, ou seja só serão capturados os registros que tiverem a condição 1 como verdadeira e a condição 2 como falsa, isso é recíproco.

5.9.3 - ALIAS

ALIAS é um recurso que te permite, dentro do seu comando SELECT, dar a uma tabela ou a uma coluna outro nome ou “apelido”, usando a palavrinha AS. Esta pode ser uma boa coisa a fazer se você tem os nomes de tabelas ou colunas muito extensos ou complexos. Um nome de alias poderia ser qualquer coisa, mas geralmente é curto.

Sintaxe para usar ALIAS em tabelas:

```

SELECT nome_do_campo
FROM nome_da_tabela1 AS apelido_tabela1, nome_da_tabela2 AS apelido_tabela2;

```

Sintaxe para usar ALIAS em campos:

```

SELECT nome_do_campo AS apelido_campo1, nome_do_campo2 AS apelido_campo2
FROM nome_da_tabela1;

```

Vamos fazer uma consulta MySQL sem utilizar o ALIAS e outra utilizando esse recurso. Vamos selecionar todos os professores do aluno **Rapadura Doce**, usaremos como referência a tabela **Aluno** e **Professor** usadas anteriormente .

→ Consulta sem usar o ALIAS

```

SELECT Aluno.nome, Professor.nome
FROM Aluno, Professor
WHERE Aluno.nome = 'Francisco Jonas';

```

→ Consulta com uso do ALIAS

```
SELECT a.nome, p.nome  
FROM Aluno AS a , Professor AS p  
WHERE a.nome = 'Francisco Jonas';
```

Como vimos no SELECT acima, ALIAS podem nos ajudar bastante nas consultas, tanto para escrever como para ler as instruções SQL.

Não é permitido utilizar apelido de coluna em uma cláusula WHERE, por exemplo:

```
SELECT a.telefone_aluno AS fone FROM Aluno AS a WHERE fone = '88111185';
```

Resulta nesse erro:

```
ERROR 1054 (42S22): Unknown column 'fone' in 'where clause'
```

Esse erro surgiu, pois tentamos utilizar fone, o apelido do campo telefone_aluno na condição da cláusula WHERE.

5.9.4 – Exercícios

1^a) Crie uma tabela Aluno com as colunas nome (VARCHAR(255)), email (VARCHAR(255)) telefone (VARCHAR(10)), altura (DECIMAL(3,2)) e aprovado (TINYINT(1)).

2^a) Insira alguns registros na tabela evitando valores repetidos.

3^a) Utilizando a cláusula WHERE refaça os exemplos criando uma consulta para cada tipo de operador.

Observação: nas consultas utilize valores que façam sentido de acordo com os valores que você inseriu na tabela.

5.9.5 – Funções de Agrupamento

O resultado de uma consulta pode ser processado e algumas informações podem ser obtidas. Por exemplo, podemos obter o valor máximo ou mínimo de uma coluna numérica. É possível contabilizar a quantidade de registros obtidos através de uma consulta. Também podemos calcular a soma ou a média de uma coluna numérica entre outras informações.

Eis uma lista com as principais funções de agrupamento do MySQL Server e a sintaxe para aplicá-las:

→ **COUNT:** A função COUNT() retorna o número de linhas que corresponde a um determinado critério.

```
1 SELECT COUNT(*) FROM Aluno;
```

→ **AVG:** Calcula a média de um conjunto de valores, onde pode ser uma coluna numérica ou uma expressão numérica passada como um parâmetro.

```
1 SELECT AVG(altura) FROM Aluno;
```

→ **SUM:** A função SUM() retorna a soma total de uma coluna numérica.

```
1 SELECT SUM(altura) FROM Aluno;
```

→ **MAX:** Retorna o maior valor NOT NULL de uma dada coluna. Retorna NULL se não houver colunas que atendam as condições de pesquisa.

```
1 SELECT MAX(altura) FROM Aluno;
```

→ **MIN:** Retorna o menor valor não NULL de uma certa coluna. Retorna NULL se não houver linhas que atendam à condição de pesquisa.

```
1 SELECT MIN(altura) FROM Aluno;
```

→ **VARIANCE:** calcula da variância de uma lista de valores;

```
1 SELECT VARIANCE(altura) FROM Aluno;
```

Observação: Variância é uma medida de dispersão. Busca avaliar o quanto os dados estão dispersos. Para fazer essa avaliação, define-se uma medida de referência: a média aritmética. Assim, a variância é uma medida que avalia o quanto os dados estão dispersos em relação à média aritmética. Ou seja, o quanto os dados estão afastados da média aritmética.

5.9.6 – GROUP BY

A instrução **GROUP BY** é usada em conjunto com as funções de agregação para agrupar o conjunto de resultados de uma ou mais colunas.

Sintaxe SQL GROUP BY :

```
SELECT nome_campo, funcao_agrupamento(nome_coluna)
      FROM nome_da_tabela
      [WHERE condicao]
GROUP BY nome_coluna;
```

Nós temos uma tabela chamada **Ordem**, com as ordens de pagamento de uma determinada empresa, ou seja uma empresa recebe pagamentos por serviços prestados a seus clientes, e isto é tomado nota na tabela abaixo:

O_ID	OrdemData	OrdemPreco	Cliente
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Agora queremos encontrar o valor total pago por cada cliente. Vamos ter de usar a instrução **GROUP BY** para agrupar os clientes e depois somar os valores de cada agrupamento usando a função de agregação **SUM** estudada no anteriormente.

Nós usamos a seguinte instrução SQL:

```
SELECT cliente, SUM(OrdemPreco)
      FROM Ordem
      GROUP BY cliente;
```

O resultado será parecido com esse:

Cliente	SUM (OrdemPreco)
Hansen	2000
Nilsen	1700
Jensen	2000

A primeira ação que o MySQL fez foi criar grupos. O critério e o comando para criar os grupos foi estabelecido pelo comando SQL(GROUP BY Cliente). O que esse trecho de código nos diz é:

Crie grupos com o seguinte critério, todos os registros que tenha no campo **Cliente** da tabela **Ordem** valores iguais formaram um grupo. Então seguindo esse critério vamos formar os grupos:

Grupo 1: Hansen

O_ID	OrdemData	OrdemPreco	Cliente
3	2008/11/08	1000	Hansen
4	2008/09/02	300	Hansen
1	2008/09/03	700	Hansen

Grupo 2: Nilsen

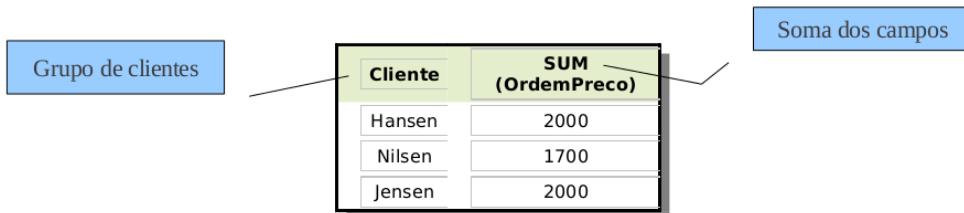
O_ID	OrdemData	OrdemPreco	Cliente
2	2008/10/23	1600	Nilsen
6	2008/10/04	100	Nilsen

Grupo 3: Jensen

O_ID	OrdemData	OrdemPreco	Cliente
5	2008/08/30	2000	Jensen

Agora que já temos nossos grupos formados vamos analisar a próxima parte da instrução MySQL. Agora, analisaremos o seguinte trecho de código: SUM (OrdemPreco). O que esse trecho da instrução faz é somar todos os valores do campo OrdemPreço da tabela **Grupo 1: Hansen** e guarda, depois soma todos os valores do campo OrdemPreço da tabela **Grupo 2: Nilsen** e guarda e faz isso também com a tabela **Grupo 3: Jensen**.

Todos os cálculos ficaram dentro de uma tabela que guarda informações do grupo, no nosso caso a tabela gerada será essa:



5.9.7 – DISTINCT

Resultados repetidos de uma consulta podem ser eliminados através do comando **DISTINCT**. Por exemplo, queremos obter uma lista das cidades onde os alunos nasceram da tabela Aluno(criada anteriormente).

```
1 SELECT DISTINCT(cidade) FROM Aluno;
```

5.9.8 – LIMIT

A quantidade de resultados de uma consulta pode ser limitada através do comando **LIMIT**. Na consulta abaixo, os 10 primeiros registros da tabela Aluno são recuperados. Se a quantidade de registros nessa tabela for inferior a 10, então todos os registros serão recuperados.

```
1 SELECT * FROM Aluno LIMIT 10;
```

Também podemos descartar os primeiros registros do resultado de uma consulta. Para isso, basta passar dois parâmetros para o comando LIMIT.

```
1 SELECT * FROM Aluno LIMIT 5, 10;
```

No exemplo acima, os 5 primeiros registros da tabela Aluno são descartados. O resultado dessa consulta conterá no máximo 10 registros a partir do sexto.

5.9.9 – Índices, Otimizando Consultas

O uso de índices torna a consulta mais rápida, da mesma forma que o índice de um livro auxilia na procura de um determinado capítulo. Com índices no banco de dados podemos aumentar a velocidade das minhas consultas em 100 vezes e em alguns casos muito mais do que isso. Os índices podem ser entendidos como o turbo do MySQL. São uma versão organizada de campos específicos de uma tabela e são usados para facilitar a consulta de registos. Sem índices, o MySQL necessita percorrer todos os regisitos de uma tabela para encontrar o registo pretendido, o que não acontece quando se usam índices. Neste caso, o MySQL consegue “saltar” diretamente para o registo pretendido.

Com o seu uso, as consultas de informações tornam-se mais rápidas e eficazes, poupando tempo de processamento. Para testar a funcionalidade dos índices vamos explicar o que vamos fazer aqui. Primeiro temos uma tabela **agenda**, com os campos **ID**, **ddd** e **número**, todos do tipo **integer**. Essa tabela é populada com 280 mil registros (o que é pouco, mas para um teste rápido já podemos perceber alguma diferença de performance). Em seguida duplicamos essa tabela, criamos índices em uma delas e na outra não. Logo após realizamos algumas consultas e comparamos a performance. Fácil né? Então...

Agora é a hora que todo mundo gosta... a hora de por a mão no código. Em primeiro lugar vamos criar duas tabelas idênticas no mysql:

```
CREATE TABLE agenda(
    id int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ddd int(3) NOT NULL,
    numero int(8) NOT NULL
);
```

```
CREATE TABLE agenda_indexada(
    id int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ddd int(3) NOT NULL,
    numero int(8) NOT NULL
);
```

Agora vamos criar um índice para a coluna número.

```
CREATE INDEX numero ON agenda_indexada(numero(4));
```

No script acima criamos um índice para o campo número, onde esse índice indexa de acordo com os 4 primeiros números, ou seja, o prefixo do telefone. Foi inserido em nossas tabelas agenda e agenda_indexada 280 mil registros, os quais serão usados para analisarmos o poder dos índices;

Em primeiro lugar, vamos buscar pelo número e ddd juntos:

```
SELECT * FROM agenda WHERE ddd=426 AND numero=43117459;
```

Ao testar isso em meu ambiente de produção obtive os seguintes resultados:

tabela agenda: 0.3857

tabela agenda_indexada: 0.0294

Nessa situação o índice aumentou a performance da sql em 13.11 vezes. Isso já faz uma diferença razoável em determinados sistemas. Mas ainda não é tudo. Vamos fazer agora uma consulta apenas pelo número:

```
SELECT * FROM agenda WHERE numero=43117459 LIMIT 5;
```

Quando a consulta é apenas em campos indexados o resultado é bem mais visível.

Veja só:

Tabela agenda: 0.4568

Tabela agenda_indexada: 0.0019

Nesse caso a consulta na tabela indexada foi 240.42 vezes mais rápido.

5.9.9.1 - Criando um índice

Para criar um índice, utilize a sintaxe apresentada a seguir.

```
CREATE [tipo_de_índice] INDEX nome_do_índice ON nome_da_tabela(coluna(tam))
```

Legenda:

[tipo_de_índice] → tipo do índice - não é obrigatório;

nome do índice → nome do índice que será criado, ou seja tabela onde será guardado as referencias aos registros;

nome_da_tabela → nome da tabela onde esta a coluna que você quer criar os índices;

coluna → nome da coluna que você quer criar os índices;

tam → número de dígitos do dado armazenado na coluna será tomado como referência para construir os índices;

Existem três tipos de índices que podem ser criados. São eles:

UNIQUE: Os valores dos índices não se repetem, fazendo com que este fato seja levado em conta em seu armazenamento.

FULLTEXT: Torna possível criar índices para campos de texto, inclusive para uma faixa de caracteres iniciais se for o caso (somente para o mecanismo de armazenamento e funcionamento MyISAM). Para criar um índice somente para os **n** primeiros caracteres de uma coluna do tipo VARCHAR, utilize (n) após o nome da coluna substituindo n pelo número de caracteres desejados.

SPATIAL: Suporta índices para os tipos de dados geométricos(somente para o mecanismo de armazenamento e funcionamento MyISAM). Nas próximas versões do MySQL, será possível optar pela ordem ascendente ou descendente na tabela de índices, contudo este recurso ainda não foi implementado.



Importante: Colunas do tipo BLOB ou TEXT somente podem ser transformadas em índices se seus mecanismos de armazenamento forem do tipo MyISAM ou InnoDB. Colunas que possuam valores nulos(NULL) somente poderão ser transformadas em índices nos motores armazenamento MyISAM, InnoDB ou MEMORY.

Dicas para Criação de Índices

→ Crie preferencialmente nas colunas que são inteiros (integer), aumenta o desempenho comparado aos valores varchar, valores de inteiro geralmente tem tamanho menor e são muito mais rápidos para comparar, consequentemente ocupará menos espaço na memória para o processamento.

→ Crie índices para as colunas de chave estrangeira que serão utilizadas nas cláusulas where.

→ Considere a criação baseando-se nas colunas frequentemente usadas nas cláusulas where, order by, group by.

- Não crie índices com possíveis valores duplicados. Ex: Sexo(Masculino ou Feminino).
- Crie índices ao unir duas tabelas na cláusula where (a.id = b.id), caso contrário sua consulta ficará lenta, e será cada vez mais demorada na medida em que o número de registros aumentarem em sua base de dados.
- Crie índices com várias colunas de sua tabela.

5.9.9.2 - Alterando um índice

Não existe um comando de alteração de índice no MySQL. Neste caso, para realizar este processo, é necessário excluir o índice em questão e recriá-lo com suas novas propriedades, as quais se deseja alterar.

5.9.9.3 - Removendo um índice

Para remover um índice, basta utilizar o comando **DROP INDEX**, como é mostrado a seguir:

```
DROP INDEX nome_do Índice ON nome_da_tabela
```

Observação:

Ao remover uma tabela do sistema, automaticamente seus índices serão excluídos.

5.10 - Relacionamentos

Os relacionamentos do banco de dados, são formas como as tabelas se comportam e se relacionam com as outras tabelas, compartilhamento de informações é a base, e melhoria da organização e filtragem dos dados é o resultado dessas relações.

5.10.1 - Unique

Em alguns casos nossas tabelas precisam ter a garantia de que uma determinada informação seja única dentre os registros. Por exemplo, uma tabela Cliente poderíamos ter uma coluna cpf para representar o número do CPF de um determinado cliente. Nesse caso seria interessante garantir que não sejam inseridos na tabela dois clientes com o mesmo CPF ou que um cliente não seja inserido duas vezes.

Para evitar esse tipo de problema poderíamos realizar uma consulta na tabela Cliente antes de fazermos a inserção afim de verificarmos se já existe algum cliente cadastrado com o CPF que desejamos inserir. Essa abordagem não seria tão ruim se as operações realizadas em um banco de dados não ocorressem de forma concorrente.

Como esse cenário é muito comum, geralmente os SGBD's disponibilizam formas de garantirmos a unicidade de um registro. No caso do MySQL, podemos utilizar a restrição UNIQUE. Nossa janela ficará da seguinte forma:

```
1 CREATE TABLE Cliente(  
2     nome VARCHAR(255),  
3     cpf VARCHAR(20) UNIQUE  
4 )  
5 ENGINE = InnoDB;
```

Também podemos apenas alterar uma coluna, caso a tabela já exista.

```
1 ALTER TABLE Cliente ADD UNIQUE (cpf);
```

Em uma tabela podemos ter quantas colunas com a restrição UNIQUE forem necessárias. Por exemplo, na tabela Aluno poderíamos ter a coluna primeiro_nome definida com a restrição UNIQUE e a coluna sexo sem a restrição. Ao tentarmos inserir um aluno do sexo masculino com o primeiro_nome Yuki poderemos ter um problema, pois em alguns países o nome Yuki pode ser usado tanto para homens quanto para mulheres. Nesse caso poderíamos definir a restrição UNIQUE em um índice composto pelas colunas primeiro_nome e sexo.

```
1 CREATE TABLE Aluno (  
2     id INT NOT NULL,  
3     primeiro_nome VARCHAR (255) NOT NULL,  
4     sexo VARCHAR (255) NOT NULL,  
5     UNIQUE INDEX(primeiro_nome, sexo)  
6 )  
7 ENGINE = InnoDB ;
```

Fica claro que no exemplo dado acima a nossa tabela Aluno permitiria, no máximo, a inserção de dois alunos com o primeiro nome Yuki: um do sexo masculino e outro do sexo feminino. Para resolver esse problema podemos adicionar outras colunas ao nosso índice composto que possui a restrição UNIQUE, por exemplo.

5.10.2 – Chaves Primárias

Já vimos que em alguns momentos as tabelas necessitam a garantia da unicidade de um registro. Em alguns casos isso não basta, pois além da unicidade precisamos garantir que o valor de uma coluna não seja nulo e que tal valor seja suficiente para identificar um registro. Para situações como essa devemos utilizar a restrição PRIMARY KEY (chave primária).

Uma chave primária deve conter valores únicos, não nulos e uma tabela pode conter apenas uma coluna como chave primária. É uma prática muito comum criarmos uma coluna com o nome id para armazenarmos um código de identificação do nosso registro dentro de uma tabela. Observe:

```
1 CREATE TABLE Cliente(
2     id INT NOT NULL,
3     cpf VARCHAR(20) UNIQUE,
4     nome VARCHAR(255),
5     PRIMARY KEY (id)
6 )
7 ENGINE = InnoDB;
```

5.10.3 – Chaves Compostas

O exemplo abaixo mostra como criar uma tabela com uma chave primária simples. Perceba que a chave primária foi criada no campo cidade.

```
CREATE TABLE cidades(
    cidade varchar(100) primary key,
    estado varchar(10),
    populacao int
);
```

Agora considere o exemplo abaixo que irá inserir os dados das cidades de Tururu/CE e Bahia/BA na tabela criada.

```
INSERT INTO cidades VALUES('Tururu','CE',185421);
INSERT INTO cidades VALUES('Tururu','BA',17216);
```

No exemplo acima teremos um erro, pois há uma violação da chave primária, já que o nome “Tururu” será duplicado e isto não é permitido. A solução para o problema acima é criarmos uma chave primária composta. É muito importante entendermos que não existe duas chaves primárias e sim chave primária composta.

A chave primária composta é aquela que é criada em dois campos e desta forma passa a utilizar a junção dos dados dos dois campos indicados para formar um valor único e assim aplicar o bloqueio de duplicidade.

No nosso exemplo, poderemos usar o campo estado junto com o campo cidade para formar uma chave composta. Desta forma Tururu/CE será diferente de Tururu/BA. Veja o exemplo abaixo:

```
CREATE TABLE cidades(
    cidade varchar(100),
    estado varchar(10),
    populacao int,
    CONSTRAINT pk_CE primary key(cidade,estado)
);
```

Constraint é o mesmo que restrição, já que a chave primária é uma restrição. pk_CE é o nome da restrição. Cidade e estado são os campos que usei para criar a chave composta.

Se executarmos novamente o exemplo do INSERT, como mostrado abaixo, não teremos erro já que a junção dos nomes das cidades com o estado gerou termos diferentes.

```
INSERT INTO cidades VALUES('Tururu','CE',185421);
INSERT INTO cidades VALUES('Tururu','BA',17216);
```

5.10.4 – Chaves Estrangeiras

Uma coluna com a restrição **FOREIGN KEY** (chave estrangeira) faz referência à uma chave primária definida em uma outra tabela. O uso das chaves estrangeiras nos traz alguns benefícios como prevenir que uma operação realizada no banco de dados possa corromper a relação entre duas tabelas ou que dados inválidos sejam inseridos em uma coluna com a restrição FOREIGN KEY.

```
1 CREATE TABLE Conta(
2     id INT NOT NULL,
3     numero INT UNIQUE,
4     saldo DECIMAL(14,2),
5     limite DECIMAL(14,2),
6     PRIMARY KEY (id),
7     FOREIGN KEY (banco_id) REFERENCES Banco(id)
8 )
9 ENGINE = InnoDB;
```

Por enquanto a definição de chave estrangeira pode parecer um tanto vaga, porém tópicos a seguir seu funcionamento e utilidade poderão ser observadas mais claramente.

5.10.5 – Relacionamento “One to One”

Suponha que nos foi dada a tarefa de modelar o banco de dados de uma rede social. Em algum momento iremos criar a tabela que guarda os registros de usuários e poderíamos chegar a algo semelhante a isso:

```
1 CREATE TABLE Usuario(
2     id INT NOT NULL,
3     nome VARCHAR(255),
4     nome_usuario VARCHAR(10),
5     senha VARCHAR(10),
6     email VARCHAR(100),
7     sexo TINYINT(1),
8     profissao VARCHAR(255),
9     onde_estudou VARCHAR(255),
10    hobbies VARCHAR(255),
11    gosto_musical VARCHAR(255),
12    PRIMARY KEY (id)
13 )
14 ENGINE = InnoDB;
```

Não há nada de errado com a nossa tabela Usuário, entretanto podemos dividir a tabela em duas: uma apenas para as informações pertinentes à conta do usuário na rede social e outra para suas informações pessoais(perfil).

No exemplo ao lado acabamos de definir um relacionamento One to One(um para um), no qual 1 usuário está para 1 perfil assim como 1 perfil está para 1 usuário.

Repare no uso da chave estrangeira id na tabela Perfil. A coluna id da tabela Perfil faz referência à coluna id da tabela Usuário e, por ser uma chave estrangeira, o MySQL não permitirá que um valor inválido (id inexistente de usuário) seja atribuído à coluna id da tabela Perfil. Sem a restrição FOREIGN KEY poderíamos atribuir qualquer número inteiro.

```

1 CREATE TABLE Usuario(
2   id INT NOT NULL,
3   nome_usuario VARCHAR(10),
4   senha VARCHAR(10),
5   email VARCHAR(100),
6   PRIMARY KEY (id)
7 )
8 ENGINE = InnoDB;
9
10 CREATE TABLE Perfil(
11   id INT NOT NULL,
12   nome VARCHAR(255),
13   sexo TINYINT(1),
14   profissao VARCHAR(255),
15   onde_estudou VARCHAR(255),
16   hobbies VARCHAR(255),
17   gosto_musical VARCHAR(255),
18   PRIMARY KEY (id),
19   FOREIGN KEY (id) REFERENCES Usuario(id)
20 )
21 ENGINE = InnoDB;
```

Ainda com relação à chave estrangeira, se tentarmos remover do banco de dados um usuário que tenha uma entrada relacionada à ele na tabela Perfil, o MySQL nos informará que a operação não é permitida. Para que possamos remover o usuário devemos primeiro remover o registro relacionado da tabela Perfil e em seguida remover o registro do usuário.

5.10.6 – Relacionamento “One to Many” ou “Many to One”

Para ilustrar o relacionamento One to Many ou Many to One, vamos usar o exemplo da conta bancária :

```

1 CREATE TABLE Conta(
2   id INT NOT NULL,
3   numero INT UNIQUE,
4   saldo DECIMAL(14,2),
5   limite DECIMAL(14,2),
6   banco_id INT,
7   PRIMARY KEY (id),
8   FOREIGN KEY (banco_id) REFERENCES Banco(id)
9 )
10 ENGINE = InnoDB;
```

No exemplo acima vimos apenas uma das pontas do relacionamento. Vamos ver como seria a outra ponta, ou seja, a tabela Banco:

```

1 CREATE TABLE Banco(
2   id INT NOT NULL,
3   nome VARCHAR(255),
4   endereco VARCHAR(255),
5   PRIMARY KEY (id)
6 )
7 ENGINE = InnoDB;
```

As tabelas Banco e Conta possuem um relacionamento One to Many, pois um banco pode possuir diversas (many) contas enquanto que uma conta pertence a um único (one) banco.

5.10.7 – Relacionamento “Many to Many”

As tabelas Banco e Conta possuem um relacionamento One to Many, pois um banco pode possuir diversas (many) contas enquanto que uma conta pertence a um único (one) banco.

```
1 CREATE TABLE Aluno(
2     id INT NOT NULL,
3     nome VARCHAR(255),
4     email VARCHAR(255),
5     data_nascimento DATETIME,
6     PRIMARY KEY (id)
7 )
8 ENGINE = InnoDB;
```

```
1 CREATE TABLE Turma(
2     id INT NOT NULL,
3     inicio DATETIME,
4     fim DATETIME,
5     observacoes LONGTEXT,
6     PRIMARY KEY (id)
7 )
8 ENGINE = InnoDB;
```

Definindo as colunas aluno_id e turma_id como chave primária composta garantimos que cada registro será único e não nulo. Além disso, como ambas colunas também são chaves estrangeiras não será possível inserir um id inválido tanto na coluna aluno_id quanto na coluna turma_id.

Parabéns, você concluiu o módulo de Banco de Dados, o próximo passo será integrar esse conhecimento com o PHP-OO.

Agora lembre-se: “Com grandes poderes, vem grandes responsabilidades”



Capítulo 6 – UML

Neste capítulo iremos aprender como instalar e criar uma diagramação de classes para a linguagem PHP. UML significa: “Unified Modeling Language”, ou seja, “Linguagem de Modelagem Unificada”. Com o UML, modelamos nosso código com o objetivo de documentar e visualizar melhor o projeto.



6.1 – Pra que serve a UML?

6.1.1 – Visualização

- A existência de um modelo visual facilita a comunicação e faz com que os membros de um grupo tenham a mesma idéia do sistema.
- Cada símbolo gráfico tem uma semântica bem definida.

6.1.2 - Especificação

- É uma ferramenta poderosa para a especificação de diferentes aspectos arquiteturais e de uso de um sistema.

6.1.3 - Construção

- Geração automática de código a partir do modelo visual.
- Geração do modelo visual a partir do código.
- Ambientes de desenvolvimento de software atuais permitem:
 - Movimentações em ambos sentidos;
 - Manutenção da consistência entre as duas visões.
 - Pode incluir artefatos como:

6.1.4 - Documentação

- **Deliverables:** (documentos como especificação de requisitos, especificações funcionais, planos de teste, etc.).
- Materiais que são importantes para controlar, medir, e refletir sobre um sistema durante o seu desenvolvimento e implantação.

6.1.5 – Razões para Modelar

- Comunicar a estrutura e o comportamento desejado de um Sistema.
- Visualizar e controlar a arquitetura de um Sistema.
- Para melhorar o nosso entendimento de um sistema e, assim, expor oportunidades para melhorias e reutilização.

6.2 – Tipos de Diagrama

Antes de começarmos a trabalhar com UML, precisamos saber quais os tipos de diagramas. Podemos desenvolver e qual a utilidade de cada uma dentro de um projeto.

6.2.1 – Diagrama de Atividade

- Um diagrama de estado especial, onde a maioria dos estados é estado de ação, e a maioria das transições é ativada por conclusão das ações nos estados de origem.
- O objetivo é estudar os fluxos dirigidos por processamento interno, descrevendo as atividades desempenhadas em uma operação.

6.2.2 – Diagrama de Sequência

- Apresenta a interação de tempo dos objetos que participam na interação.
- O diagrama de sequência mostra a colaboração dinâmica entre um número de objetos, e visa mostrar a sequência de mensagens enviadas entre objetos.

6.2.3 – Diagrama de Caso de Uso

- Casos de uso descrevem funcionalidades do sistema percebidas por atores externos.
- Um ator é uma pessoa (ou dispositivo, ou outro sistema) que interage com o sistema.

6.2.4 – Diagrama de Classe

- Denota a estrutura estática de um sistema. As classes representam coisas que são manipuladas por um sistema. Esse diagrama é considerado estático porque a estrutura de classes é válida em qualquer ponto do ciclo de vida do sistema.

Neste Curso aprenderemos como trabalhar com Diagrama de Classe, pois esse diagrama é bem próximo da criação e estruturação de classes e herança entre as classes criadas.

Para criar os métodos, lembre-se que para o diagrama, primeiro vem o nome da variável ou método, em seguida vem o seu tipo:

Exemplo: **nome - String**

Veja como criar uma classe simples com o tipo, Usuários:

Usuários	
Atributos	
id	Int
usuario	String
senha	String
e-mail	String
tipo_conta	String

Dessa forma podemos criar qualquer classe com os seus atributos e métodos de forma a poder visualizar melhor a estrutura do projeto.

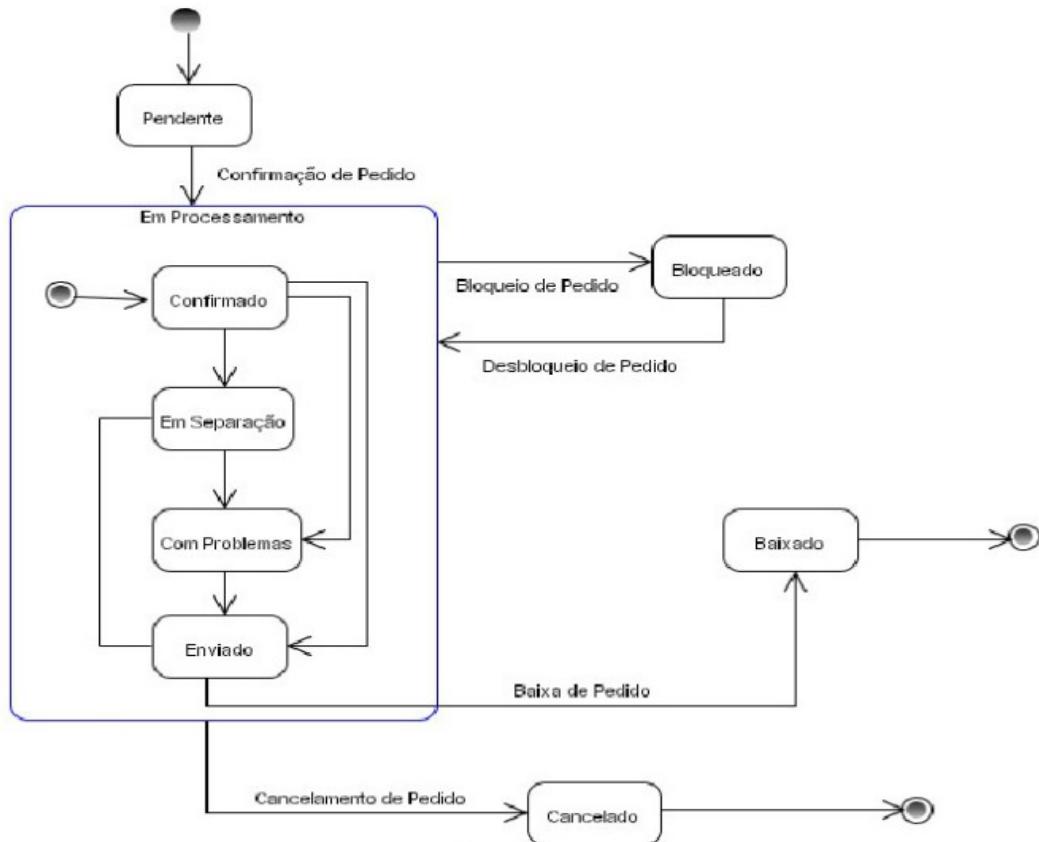
6.2.5 – Diagrama de Estado

O objetivo de um diagrama de estados é mostrar os possíveis estados de um objeto e os eventos que altera seu status. O estado de um objeto é o resultado das operações deste objeto num determinado momento.

Em geral, o objeto tem um estado inicial quando de sua criação, e em geral é alterado pela ocorrência de um determinado evento. Após ser acionado, o objeto realiza uma ou mais operações de forma que seu estado muda para uma outra condição. Num

determinado momento o objeto tem o seu estado alterado para um estado final, que pode ocorrer em diversas circunstâncias, indicando o fim das alternâncias de estado. Desta forma, através deste diagrama é possível visualizar os possíveis estados que um objeto pode ter, de forma a se prever seu comportamento.

No exemplo a seguir, mostro os estados possíveis para um pedido, os eventos que modificam seu estado e seu estado final.



6.3 - Documentação do Sistema

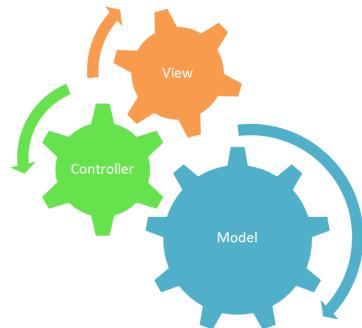
Este modelo pode ser utilizado para se discutir a visão do projeto de sistema com todos os envolvidos, desde os usuários-chave (e gerência) que irão se beneficiar do sistema até os elementos da equipe de desenvolvimento (programadores, analistas, testadores, etc).

Desta forma, o resultado final deverá ser um conjunto de diagramas e documentos avaliados por toda a equipe e em conformidade com a necessidade dos stakeholders. Um Controle de Versões poderá ser implementado, na medida em que se armazena (e identifica-se devidamente) os processos (e suas versões) e suas funcionalidades. Estes elementos irão mudar na medida em que se verifica a manutenção do sistema, de forma que é necessário identificar a versão dos mesmos.

*Tais itens (diagramas e documentos), requisitos (funcionais e não funcionais), documento visão e protótipos irão inicialmente compor a Documentação do Sistema.

Capítulo 7 – MVC

Neste capítulo iremos abordar um padrão de desenvolvimento universal, o MVC (Model/View/Controller). Este padrão serve basicamente para organizar e otimizar qualquer projeto



7.1 – O que é? Pra que serve?

Atualmente, muitos softwares e frameworks estão utilizando do padrão MVC para o desenvolvimento de seus aplicativos/sites. O MVC (Model, View e Controller) é uma arquitetura ou padrão que lhe permite dividir as funcionalidades de seu sistema/site em camadas, essa divisão é realizada para facilitar resolução de um problema maior.

Por isso, não fique preso a paradigmas, arquiteturas, padrões ou tecnologias, pois é de grande importância que você se atualize. E agora chegou a hora em que você irá entender o conceito e como funciona o “famoso” MVC.

Com o aumento da complexidade dos sistemas/sites desenvolvidos hoje, essa arquitetura tem como foco dividir um grande problema em vários problemas menores e de menor complexidade. Dessa forma, qualquer tipo de alterações em uma das camadas não interfere nas demais, facilitando a atualização de layouts, alteração nas regras de negócio e adição de novos recursos. Em caso de grandes projetos, o MVC facilita muito a divisão de tarefas entre a equipe.

Abaixo serão listadas algumas das vantagens em utilizar MVC em seus projetos:

- Facilita o reaproveitamento de código;
- Facilidade na manutenção e adição de recursos;
- Maior integração da equipe e/ou divisão de tarefas;
- Diversas tecnologias estão adotando essa arquitetura;
- Facilidade em manter o seu código sempre limpo;

7.2 - Organização

No MVC possuímos três camadas básicas, e cada uma delas, com suas características e funções bem definidas para facilitar a sua vida, caro programador.

7.2.1 - View

A visão (view) é responsável por tudo que o usuário final visualiza, toda a interface, informação, não importando sua fonte de origem, é exibida graças a camada de visão, no caso do PHP, as views são praticamente o HTML e tecnologias afins.

Exemplos de Views:

- Formulários
- Tabelas
- Listas

7.2.2 - Control

A Controladora (controller), como o nome já sugere, é responsável por controlar todo o fluxo de informação que passa pelo site/sistema. É na controladora que se decide "se", "o que", "quando" e "onde" deve funcionar. Define quais informações devem ser geradas, quais regras devem ser acionadas e para onde as informações devem ir, é na controladora que essas operações devem ser executadas. Em resumo, é a controladora que executa uma regra de negócio (modelo) e repassa a informação para a visualização (visão). Simples não?

7.2.3 - Model

O modelo (Model) é utilizado para manipular informações de forma mais detalhada, sendo recomendado que, sempre que possível, se utilize dos modelos para realizar consultas, cálculos e todas as regras de negócios do nosso site ou sistema. É o modelo que tem acesso a toda e qualquer informação sendo essa vinda de um banco de dados, arquivo XML, etc.

7.3 – Usando o MVC na prática

Vamos imaginar uma situação em que existe um site/sistema que precisa de login para acessar uma área restrita.

Com isso teremos nossa principal view, o formulário de login:

```
<form action="log.php" method="post">
    <input type="email" name="us_email">

    <input type="password" name="us_senha">
    <input type="submit" value="Entrar">
</form>
```



Com o formulário acima percebemos que os dados(email e senha) serão enviados para o arquivo **log.php**, que nesta aplicação será o Controller, a função do Controller será incluir um Model, e invocar algumas funções do mesmo. Observemos como será o Model:

```
function busca_log(){
    //recebendo dados do form
    $email = $_POST['us_email'];
    $senha = $_POST['us_senha'];

    //busca os dados no banco de dados
    //e retorna $usuario(com dados do usuario) quando a busca for true
    //ou retorna false, quando a busca não é concluída com sucesso.
    /* ... */
    if($busca != true){
        return false;
    }else{
        return $usuario;
    }
}
```

O controller irá incluir esse model, e invocar sua função, observer o Controller:

```
//inclui o model
include_once 'model.php';

//invoca a função de busca de dados
$log = busca_log();

//testa o resultado da função acima
if($log === false){ //se os dados não forem encontrados
    //inclui mensagem de erro
    include_once 'error.html';
}else{ //se os dados forem encontrados
    //cria sessão com os dados do usuário que vem no retorno
    session_start();
    $_SESSION['usuario'] = $log;

    //redireciona para a área restrita
    header("location: usuario.php");
}
```

Observe que o Controller avalia o retorno da função “busca_log” e se o retorno for “false” será mostrada uma mensagem de erro, mas caso o retorno não seja “false” será criada uma sessão com os dados do usuário logado, e redirecionado ao ambiente restrito do site/sistema.

Capítulo 8 – Interação PHP & MySQL

- Neste capítulo iremos abordar a forma de relação que o PHP faz com o MySQL, salientando que sendo Estruturada ou Orientada a Objetos, a ideia de manipulação do MySQL é a mesma.



A interação do PHP com o MySQL se dá por meio de funções nativas do PHP que possuem como principal característica o prefixo “mysql_” na nomenclatura das funções.

8.1 – Conexão com MySQL

No PHP a conexão com o Banco de Dados MySQL é bem simples, só requer o uso de 2 comandos, diferente do JAVA que precisa de um driver e várias linhas de código.

8.1.1 – mysql_connect ou mysql_pconnect

A conexão é realizada através da função **mysql_connect** ou **mysql_pconnect**, sendo que a única diferença entre as duas é que a primeira criar uma conexão temporária, e a segunda cria uma conexão permanente, mas a sintaxe das duas é a mesma, observe:

```
mysql_connect($servidor, $usuario, $senha);
```

```
mysql_pconnect($servidor, $usuario, $senha);
```

Podemos notar que para uma conexão ser realizada precisamos de 3 parâmetros:

- **Servidor:** é onde está instalado o banco de dados, no nosso caso será “localhost”.
- **Usuário:** é o usuário do banco de dados, no nosso caso é “root”.
- **Senha:** é a senha do usuário do banco de dados, no nosso caso, “utd123456”.

Lembrando que os 3 parâmetros devem ser passados sempre nesta ordem.

Essas funções retornam um id de verificação, ou seja, um valor que serve para identificar a conexão. Então utilizando a função **mysql_connect** ficaria assim:

```
$servidor = "localhost";
$usuario = "root";
$senha = "utd123456";

$conexao = mysql_connect($servidor, $usuario, $senha);
```

8.1.2 - mysql_select_db

Agora que já fizemos a conexão, precisamos selecionar o banco de dados que iremos usar, para isso precisamos de um id de conexão(criado anteriormente) e do nome do banco de dados, ou seja:

```
$servidor = "localhost";
$usuario = "root";
$senha = "utd123456";
$banco = "db_escola";

$conexao = mysql_connect($servidor, $usuario, $senha);

mysql_select_db($banco, $conexao);
```

Pronto agora além de conectarmo-nos ao MySQL, já selecionamos o banco de dados que iremos utilizar.

8.1.3 – Fechando conexão - mysql_close

Não é bom entrar dentro de casa e deixar a porta aberta não é? Pois podem entrar quem a gente não convidou, é por esse e outros motivos que existe o mysql_close, essa função tem o objetivo de encerrar uma conexão, ou seja, fechá-la.

Observe sua complexa sintaxe:

```
mysql_close();|
```

Simples não é? É interessante nunca se esquecer de encerrar a conexão ao terminar de resolver algum procedimento.

8.2 – Enviando querys - mysql_query

Para o envio de querys, você precisará utilizar a função “mysql_query”, mas além disso precisará também do mesmo id de conexão que usamos para selecionar o banco de dados.

A sintaxe é a seguinte:

```
mysql_query($query, $id_conexao);
```

8.2.1 - Consultas

Para realizar consultas é obrigatório que guardemos o retorno da função mysql_query em uma variável, pois já que o MySQL possivelmente retornará dados, temos que guardar esses dados. Observe:

```
$query = "SELECT * FROM `tb_alunos`";

$resultado = mysql_query($query, $conexao);
```

Nesse caso a variável \$resultado poderá ser “false” caso a tabela “tb_alunos” esteja em branco, ou poderá ser os dados de todos os registros da tabela.

8.2.2 – Demais querys

Para as demais querys, o mysql_query só retorna true ou false, e até mesmo pode ser usado sem variável para guardar seu retorno, por exemplo:

```
$query = "DELETE FROM `tb_professores` WHERE `status`='0'";  
mysql_query($query, $conexao);
```

O exemplo acima deleta os registros da tabela “tb_professores” que possuem o campos “status” igual a “0”(zero).

8.3 – Manipulando os Resultados

Depois de fazer algumas buscas é interessante saber usá-las, e poder usá-las. A função mysql_query envia a query e retorna dados, porém esses dados não são os dados puros, temos que precessá-los ainda, e fazemos isso através de algumas funções:

8.3.1 – mysql_result

Retorna o conteúdo de uma célula do resultado MySQL. Olhe a sintaxe:

```
mysql_result($resultado, $linha, $campo);
```

O argumento **\$resultado** é o retorno da função mysql_query, o argumento **\$linha**, é a linha do registro que quero obter dados, e o argumento **\$campo** pode ser o índice do campo, o nome do campo, ou a tabela ponto o nome do campo(tabela.campo). Se o nome da coluna usa apelido ('select foo as bar from...'), use o apelido ao invés do nome da coluna.

Por exemplo:

```
$nome = mysql_result($resultado, 0, 'nome');
```

O exemplo acima irá pegar o valor do campo “nome” da linha “0”(zero) dos registros da variável **\$resultado**(gerada de um mysql_query). Lembrando que já que a função mysql_result retorna dados, precisamos guardar esses dados.

8.3.2 - mysql_num_rows

Essa função conta o número de registro encontrados em uma busca, ela precisa do parâmetro **\$resultado** gerado pela mysql_query. Sintaxe:

```
$query = "SELECT * FROM `tb_cursos`";  
$resultado = mysql_query($query, $conexao);  
  
$n_registros = mysql_num_rows($resultado);
```

8.3.3 – mysql_fetch_array

Transforma o resultado da busca(mysql_query) em um array bidimensional do PHP, onde as linhas são cada registro, e as colunas são os campos da tabela.

Sintaxe:

```
$query = "SELECT * FROM `tb_disciplinas`";
$resultado = mysql_query($query, $conexao);

$disciplinas[] = mysql_fetch_array($resultado);
```

8.4 – Gerar Erros – mysql_error

Essa função tem o poder de mostrar na tela do navegador uma mensagem de erro caso tenha algum, ela aponta a linha e o provável motivo do erro.

Sintaxe:

```
mysql_error();|
```

8.5 – Outras Funções do MySQL

Para conhecer outras(muitas) funções para manipulação do MySQL acesse o link:

http://www.php.net/manual/pt_BR/function.mysql-result.php

Posfácio



Parabéns, você concluiu o curso para Padawan em PHP-OO & MySQL Relacional. Mas você não deve parar por aqui, você deve conhecer a verdade e dominar o mundo. O próximo passo é estudar algum framework(Zend, Cake, Yii, CodeIgniter, etc). Agora você tem duas opções:



Com a pílula azul você continuará na doce ilusão de que já é um programador fodástico, mas com a pílula vermelha você verá a amarga realidade que você é só mais um no meio do mundo, e que precisa estudar mais para poder se tornar o melhor dos melhores.

Mas antes de escolher, você deve entender que a maior parte dos programadores não está pronta para a verdade, e muitos são tão inertes, tão dependentes de sistemas que lutariam até mesmo para impedi-lo de se tornar um programador foderoso.



**GOVERNO DO
ESTADO DO CEARÁ**

*Secretaria da Ciência, Tecnologia
e Educação Superior*



PHP Orientado a Objetos & MySQL Relacional

