Подключение внешних файлов

При разработке программ на PHP, возможно, какую-ту часть кода мы захотим использовать одновременно в других файлах с кодом PHP. В этом случае отдельные части кода можно распределить по отдельным файлам. Это позволить не писать один и тот же код по сто раз на сотнях скриптов, а будет достаточно подключить файл с кодом PHP. Кроме того, если потребуется изменить поведение подключаемого кода, достаточно будет изменить код в подключаемом файле.

Для подключения файлов PHP предоставляет ряд возможностей.

Инструкция include

Инструкция include подключает в программу внешний файл с кодом php. Так, для примера определим файл welcome.php:

<?php

function welcome($name)

{

    echo "Hello, $name";

}

?>

Здесь определена функция welcome, которая в качестве параметра принимает условное имя и использут его для вывода приветствия.

Теперь подключим данный файл в нашу программу, которая определена в другом файле в той же папке:

<?php

include "welcome.php";

$name = "Tom";

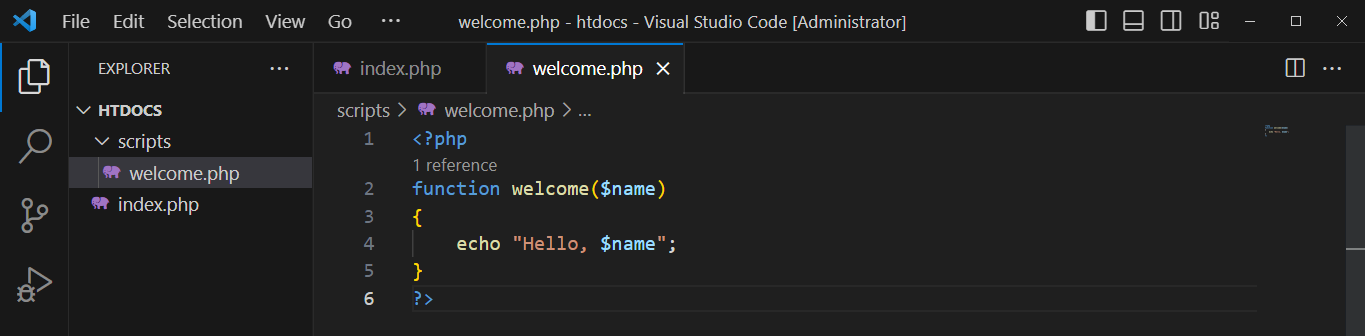
welcome($name);

?>

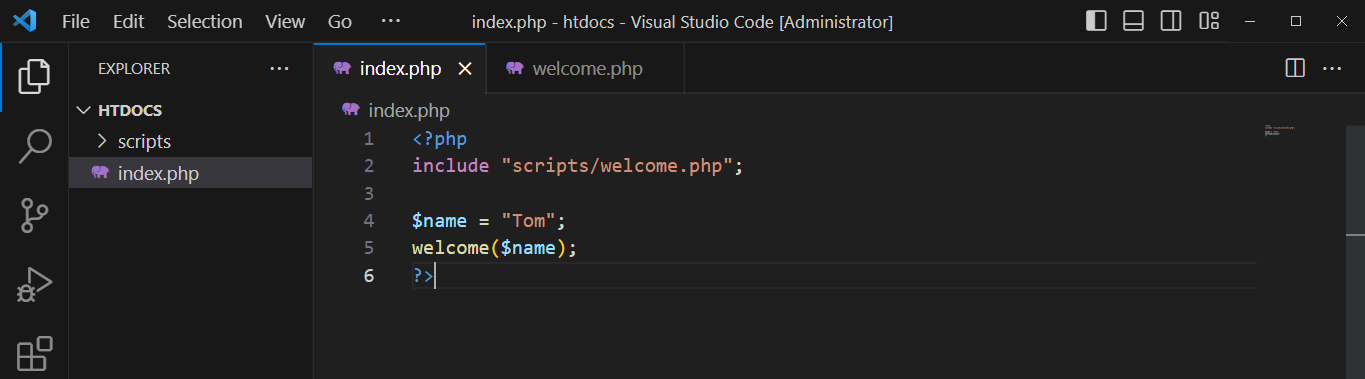
В место определения инструкции include будет вставляться весь код из файла welcome.php. При этом вставка файла должна происходить до использования функции, определенной в этом файле. При этом в данном случае файл welcome.php и файл, в который он подключается, располагаются в одной папке.

Конструкция include может использовать как относительные, так и абсолютные пути. Например, выше использовался относительный путь. Или, к примеру, если мы имеем следующую структуру

Welcom.php



Index.php

То чтобы подключить файл welcome.php из папки scripts, в файле index.php необходимо использовать следующий относительный путь:

include "scripts/welcome.php";

Если файл welcome.php располагается по полному пути C:\localhost\scripts\welcome.php, то также можно было бы использовать абсолютный - полный путь:

include "C:\localhost\scripts\welcome.php";

Инструкция include\_once

Использование инструкции include имеет недостатки. Так, мы можем в разных местах кода неумышленно подключить один и тот же файл, что при выполнении кода вызовет ошибки.

Чтобы исключить повторное подключение файла, вместо инструкции include следует применять инструкцию include\_once

<?php

include\_once "welcome.php";

$name = "Sam";

welcome($name);

?>

И теперь, если мы подключим этот же файл с помощью include\_once еще где-нибудь ниже, то это подключение будет проигнорировано, так как файл уже подключен в программу.

Инструкции require и require\_once

Действие инструкции require подобно инструкции include: она также подключает внешний файл, вставляя в программу его содержимое. Только теперь, если данный файл не будет найден, действие программы прекратится (инструкция include в этом случае выбрасывает предупреждение):

<?php

require "welcome.php";

?>

И также если у нас в коде встретятся несколько инструкций require, которые подключают один и тот же файл, то интерпретатор выдаст ошибку. И также чтобы избежать данной ситуации, следует использовать инструкцию require\_once:

<?php

require\_once "welcome.php";

?>

Функция spl\_autoload\_register

В больших приложениях количество подключаемых файлов может быть довольно большим. Однако встроенная функция spl\_autoload\_register() в определенных ситуациях позволяет избежать большого количества инклудов. В качестве параметра она принимает функцию автозагрузки. Эта функция автоматически вызывается, когда в программе начинает использоваться неизвестный класс или интерфейс. И функция автозагруки пытается загрузить этот класс или интерфейс. В качестве параметра функция автозагрузки принимает название класса или интерфейса, которые надо загрузить.

Например, пусть у нас будет файл Person.php, в котором располагается класс Person:

<?php

class Person

{

    private $name, $age;

    function \_\_construct($name, $age)

    {

        $this->name = $name;

        $this->age = $age;

    }

    function printInfo()

    {

        echo "Name: $this->name<br>Age: $this->age";

    }

}

?>

Обращаю внимание, что название файла соответствует названию класса.

Используем функцию автозагрузки для подключения подобного класса:

<?php

function my\_autoloader($class) {

    echo "Вызов функции автозагрузки<br>";

    include $class . ".php";

}

spl\_autoload\_register("my\_autoloader");

$tom = new Person("Tom", 25);

$tom->printInfo();

?>

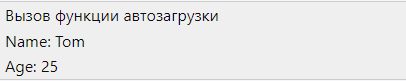
Функция spl\_autoload\_register() в качестве параметра принимает название функции автозагрузки - в данном случае это функция my\_autoloader(). В качестве параметра она принимает название класса. Например, в данном случае используется класс Person, который в этом скрипте не определен. И когда программа встретит использование данного класса, она вызовет функцию my\_autoloader(), в качестве параметра $class передаст в нее название класса Person.

Все действия функции автозагрузки мы определяем сами. В данном случае с помощью echo выводится некоторое диагностическое сообщение и подключается файл класса:

<?php

include $class . ".php";

При этом в данном случае неважно какой класс, главное, чтобы он хранился в одноименном файле с расширением .php. В этом случае программа выведет следующее:



Пространства имен

Пространства имен позволяют избежать конфликта имен и сгруппировать функционал. Внутри пространства имен могут быть размещены классы, интерфейсы, функции и константы.

Если какая-та конструкция (например, класс или функция) определена вне любого пространства имен, то считается, что она расположена в глобальном пространстве имен.

Определение пространства имен

Для создания пространства имен применяется директива namespace, после которой идет название пространства имен:

<?php

namespace base;

class Person

{

    private $name;

    function \_\_construct($name) { $this->name = $name; }

}

В данном случае определено пространство имен "base". Обычно названия пространств указываются в нижнем регистре. Все, что расположено ниже этой директивы, относится к пространству имен "base". То есть класс Person принадлежит пространству имен base.

Стоит учитывать, что определение пространства имен должно быть расположено выше любого другого кода или разметки html. Например:

<?php

namespace base;

class Person

{

    public $name;

    function \_\_construct($name) { $this->name = $name; }

}

?>

<!DOCTYPE html>

<html>

<head>

<title>Namespaces</title>

<meta charset="utf-8" />

</head>

<body>

<?php

$tom = new Person("Tom");

echo $tom->name;

?>

</body>

</html>

Обращение к пространству имен

Для обращения к конструкциям из пространства имен перед названием конструкции через слеш указывается ее пространство имен. Например, у нас есть файл Person.php:

<?php

namespace base;

class Person

{

    public $name;

    function \_\_construct($name) { $this->name = $name; }

}

?>

Теперь в другом файле подключим этот файл и обратимся к классу Person:

<?php

namespace work;

include "Person.php";

$tom = new \base\Person("Tom");

echo $tom->name;

Здесь определено другое пространство имен - "work". Сначала в нем подключается файл "Person.php". Затем создаем объект класса Person.

Если конструкции одного пространства имен используются в другом пространстве имен (в том числе в глобальном пространстве имен), то перед названием конструкции указывается название ее пространства имен. Так, поскольку класс Person расположен в другом пространстве имен - "base", то перед названием класса указываем его полное имя с учетом его пространства имен:

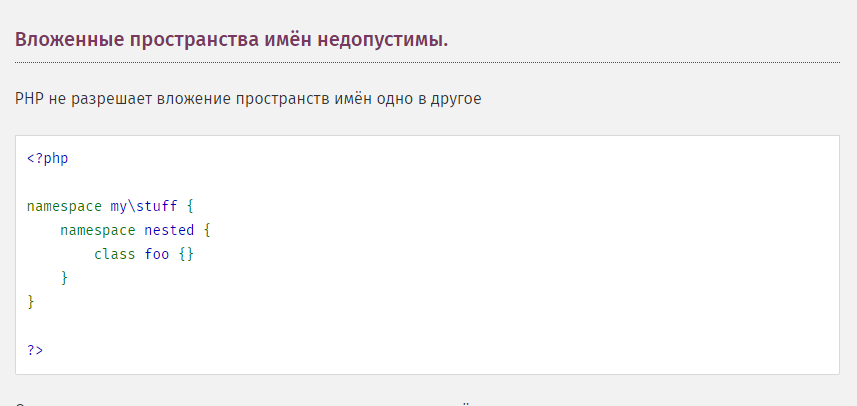
<?php

$tom = new \base\Person("Tom");

Обратите внимание, что сначала идет слеш, потом название пространства имен и потом через слеш название класса (\base\Person).

Без указания пространства имен мы можем использовать конструкцию только в том же пространстве имен, в котором она определена.

Вложенные пространства имен



<https://www.php.net/manual/ru/language.namespaces.faq.php>

Одни пространства имен могут содержать другие. Например: (Error)

<?php

namespace base\classes;

class Person

{

    public $name;

    function \_\_construct($name) { $this->name = $name; }

}

Здесь класс Person определен в пространстве имен base\classes\. То есть теперь для обращения к классу Person в другом пространстве имен надо указывать \base\classes\Person:

<?php

namespace work;

include "Person.php";

$tom = new \base\classes\Person("Tom");

echo $tom->name;

Псевдонимы

Если в одном пространстве имен используется какой-то класс из другого пространства имен, то писать каждый раз полное имя класса с учетом его пространства имен может быть утомительно, кроме того, перегружает код. Например:

<?php

namespace work;

include "Person.php";

$tom = new \base\classes\Person("Tom");

echo $tom->name . "<br>";

$bob = new \base\classes\Person("Bob");

echo $bob->name;

В этом случае мы можем использовать псевдонимы, которые задаются в виде

<?php

use полное\_имя\_класса as псевдоним;

Конструкция use импортирует класс из другого пространства имен, а оператор as устанавливает для него псевдоним.

Так, сократим предыдущий код с помощью псевдонимов:

<?php

namespace work;

include "Person.php";

// установка псевдонима

use \base\classes\Person as User;

$tom = new User("Tom");

echo $tom->name . "<br>";

$bob = new User("Bob");

echo $bob->name;

В данном случае для класса \base\classes\Person установлен псевдоним User, соответственно теперь для обращения к классу Person мы можем использовать его псевдоним User. Результат тот же, но кода меньше.

Можно использовать только конструкцию use без указания псевдонима. Тогда класс можно будет исользовать по его непосредственному имени:

<?php

namespace work;

include "Person.php";

use \base\classes\Person;

$tom = new Person("Tom");

echo $tom->name . "<br>";

$bob = new Person("Bob");

echo $bob->name;

Подключение множества классов

Подобным образом можно подключать сразу несколько классов. Например, пусть в файле Person.php имеются следующие классы:

<?php

namespace base\classes;

class Person

{

    public $name;

    function \_\_construct($name) { $this->name = $name; }

}

class Employee extends Person { }

Подключение классов Person и Employee:

<?php

namespace work;

include "Person.php";

use \base\classes\Person as User, \base\classes\Employee as Employee;

$tom = new User("Tom");

echo $tom->name . "<br>";

$sam = new Employee("Sam");

echo $sam->name;

Определение псевдонимов можно сократить:

<?php

use \base\classes\{Person as User, Employee as Employee};

Подключение констант и функций

Для подключения констант применяется инструкция use const, а для подключения функций - use function.

Например, определение файла Person.php:

<?php

namespace base\classes;

const adminName = "Odmen";

function printPerson($person){

    echo $person->name . "<br>";

}

class Person

{

    public $name;

    function \_\_construct($name) { $this->name = $name; }

}

Подключим класс Person вместе с константой adminName и функцией printPerson:

<?php

namespace work;

include "Person.php";

use \base\classes\Person;

use const \base\classes\adminName;

use function \base\classes\printPerson;

$tom = new Person(adminName);

printPerson($tom);  // Odmen

Типизация данных

В отличие от ряда языков программирования в PHP при определении переменных или параметров функций можно не указывать тип данных. Однако в принципе PHP в ряде ситуаций - при определении свойств классов, параметров или возвращаемого значения функций - позволяет указать тип данных. Установка типа данных позволит избежать ситуаций, когда в программу будут переданы данные не тех типов, которые ожидалась разработчиком.

Например:

<?php

function isPositive($number)

{

    return $number > 0;

}

$result = isPositive("-Youdontknowwhoiam");

if($result) echo "положительное";

else echo "отрицательное или равно нулю";

В данном случае функция isPositive() очевидно ожидает, что в качестве параметра будет передано число, чтобы установить, больше оно нуля или нет. Тем не менее при вызове функции мы можем передать в нее произвольное значение. Чтобы оградиться от подобным ситуаций необходимо явным образом указать, что функция может принимать только число, то есть типизировать параметр функции.

Определение типа

Какие определения типов могут использоваться при типизации:

bool: допустимые значения true и false

float: значение должно число с плавающей точкой

int: значение должно представлять целое число

string: значение должно представлять строку

mixed: любое значение

callable: значение должно представлять функцию

array: значение должно представлять массив

iterable: значение должно представлять массив или класс, который реализует интерфейс Traversable. Применяется при переборе в цикле foreach

Имя класса: объект должен представлять данный класс или его производные классы

Имя интерфейса: объект должен представлять класс, который реализует данный интерфейс

Self: объект должен представлять тот же класс или его производный класс. Может использоваться только внутри класса.

parent: объект должен представлять родительский класс данного класса. Может использоваться только внутри класса.

Типизация параметров функции

При типизации параметров тип указывается перед названием параметра:

<?php

function isPositive(int $number)

{

    return $number > 0;

}

$result1 = isPositive(25);                // норм - 25 число

$result2 = isPositive("25");              // норм - PHP может преобразовать значение в число

$result3 = isPositive("-Youdontknowwhoiam"); // Ошибка TypeError

В данном случае параметр $number должен представлять тип int, то есть целое число. Поэтому при вызове функции мы должны передать в функцию целочисленное значение. Если будет передано значение другого типа, то PHP попытается преобразовать значение. В некоторых случаях такое преобразование можно завершится успешно:

<?php

$result2 = isPositive("25");

В других случаях прееобразование может завершится неудачно, и программа завершит выполнение с ошибкой TypeError:

<?php

$result3 = isPositive("-Youdontknowwhoiam");

Другой пример:

<?php

function sum(array $numbers, callable $condition)

{

    $result = 0;

    foreach($numbers as $number){

        if($condition($number))

        {

            $result += $number;

        }

    }

    return $result;

}

$isPositive = function($n){ return $n > 0;};

$myNumbers = [-2, -1, 0, 1, 2, 3, 4, 5];

$positiveSum = sum($myNumbers, $isPositive);

echo $positiveSum;  // 15

В данном случае параметры функции должный представлять массив и другую функцию (тип callable). В качестве функции можно передать анонимную функцию.

Подробнее про callable: <https://www.php.net/manual/ru/language.types.callable.php>

Типизация возвращаемого значения

Для установки типа возвращаемого из функции значения после списка параметров указывается двоеточие : и после него тип данных:

<?php

function isPositive (int $number) : bool

{

    return $number > 0;

}

$result = isPositive(34);

В данном случае функция isPositive должна возвращать значение типа bool, то есть true или false.

Другой пример - возвращение функции:

<?php

function select($n): callable{

   switch($n){

       case 1: return function($a, $b) {return $a + $b;};

       case 2: return function($a, $b) {return $a - $b;};

       case 3: return function($a, $b) {return $a \* $b;};

       default: return function($a, $b) {return $a . " " . $b;};

   }

}

$selection = select(2);

echo $selection(4,5);   // -1

Особо стоит отметить ключевое слово static, добавленное в PHP 8, которое применяется, если надо возвратить из метода класса объект этого же класса:

<?php

class Node{

   function generate() : static{

       return new Node();

   }

}

$node1 = new Node();

$node2 = $node1->generate();

Типизация свойств

В качестве типа свойств может применяться любой тип кроме callable:

<?php

class Person{

   public $name;

   public int $age;

}

$tom = new Person();

$tom->name = "Tom";

$tom->age = 36;      // корректное значение

echo $tom->age;      // 36

$tom->age = "36";    // корректное значение, так как PHP может преобразовать в число

echo $tom->age;      // 36

$tom->age = "thirty-eight";  // некорректное значение, возникнет ошибка TypeError

echo $tom->age;

В данном случае явным образом определено, что свойство $age представляет именно тип int, то есть целое число. Соответственно этому свойству мы сможем присвоить только целое число.

Стоит учитывать, что свойство, для которого не указан тип данных, по умолчанию имеет значение null. Тогда как свойство, для которого указан тип, неинициализировано, то есть не имеет никакого конкретного значения.

<?php

class Person{

   public $name;       // равно null

   public int $age;    // неинициализировано

}

Соответственно если нетипизированное свойство мы сможем использовать, то при попытке обратиться к типизированному, но неинициализиованному свойству программа завершит выполнение ошибкой:

<?php

$tom = new Person();

echo $tom->name;     // норм - null

echo $tom->age;      // ошибка - свойство неинициализировано

Тип Union

В PHP 8 был добавлен тип union или объединение, который по сути представляет объединение типов, разделенных вертикальной чертой |. Например, мы хотим написать функцию сложения чисел, и чтобы в функцию можно было передавать только числа. Однако числа в PHP предствлены двумя типами - int и float. Чтобы не создавать по функции для каждого типа, применим объединения:

<?php

function sum(int|float $n1, int|float $n2,): int|float

{

    return $n1 + $n2;

}

echo sum(4, 5);         // 9

echo "<br>";

echo sum(2.5, 3.7);     // 6.2

В данном случае мы говорим, что параметры $n1 и $n2 могут представлять как тип int, так и тип float. Аналогично возвращаемое значение также может представлять либо int, либо float.

Работа со строками

Строки играют большую роль, задачи на парсинг строк довольно часто встречаются в PHP , поэтому рассмотрим некоторые базовые функции работы со строками.

Строки представляют некоторый текст, который заключен в одинарные или двойные кавычки:

<?php

$str1 = "Hello";

$str2 = 'Welcome';

Но также PHP позволяет определить строку и без кавычек с помощью оператора <<<, за которым следует метка завершения строки:

<?php

$s = <<< LABEL

Hello World!

Bye World..

LABEL;

echo $s;

В данном случае "LABEL" - это название метки, которая указывает на начало и конец строки. Название метки произвольное. То есть все, что расположено между <<< LABEL и LABEL;, будет представлять сроку $s. В частности, браузер выведет нам следующее:



И в данном случае, как и в обычной строке мы можем использовать переменные:

<?php

$name = "Tom";

$age = 36;

$s = <<< USER

Name = $name

Age = $age

USER;

echo $s;

Обращение к символам строки

PHP позволяет обращаться к отдельным символам строки как к элементам массива по числовому индексу:

<?php

$str = "Hello Tom";

echo $str[0];// получим первый символ - H

$str[1] = "i";  // поменяем второй символ

echo "<br>" . $str . "<br>";    // Hillo Tom

Индексация символов, как и в массиве, начинается с нуля.

Специальные функции и расширение mbstring

Кроме некоторых базовых возможностей по работе со строками PHP также предоставляет ряд специальных функций. Для некоторых из них требуется подключить специальное расширение - mbstring.dll. Для подключения этого расширения откроем файл конфигурации PHP - php.ini и найдем в нем следующую строку:

<?php

;extension=mbstring

Раскомментируем ее, убрав точку с запятой.

<?php

extension=mbstring

Также убедимся, что у нас указан путь к папке расширений.

Найдем в файле php.ini строку:

<?php

;extension\_dir = "ext"

Эта строка указывает на каталог с подключаемыми расширениями для PHP. По умолчанию все расширения располагаются в папке ext. Раскомментируем эту строку, убрав точку с запятой, и укажем полный путь к папке ext. Так, в моем случае папка c php расположена в корне диска C (ОС Windows), соответственно я указываю путь "C:\php\ext":

<?php

extension\_dir = "C:\php\ext"

И так как у нас файл php.ini изменился, перезапустим веб-сервер Apache.

Подключение расширения mbstring позволит задействовать некоторые дополнительные функции для работы со строками, которые в стандартном наборе PHP отсутствуют (такие функции обычно начинаются с префикса mb\_).

Функции strpos() и mb\_strpos()

Функция strpos($str, $search) возвращает позицию подстроки или символа $search в строке $str или значение false, если строка $str не содержит подстроки $search:

<?php

$input = "This is the end";

$search = "is";

$position = strpos($input, $search); // 2

if($position!==false)

{

    echo "Позиция подстроки '$search' в строке '$input': $position";

}

При использовании этой функции надо учитывать, что индексация символов в строке начинается с нуля, поэтому позиция символа "T" будет равна 0. Поэтому сравнение $position!=false будет работать некорректно, ведь false и 0 при сравнении и приведении к общему типу будут представлять одно и то же значение. Поэтому в данном случае корректно использовать только операцию эквивалентности: $position!==false или $position===false.

Теперь применим функцию на другом примере:

<?php

$input = "Мама мыла раму";

$search = "мы";

$position = strpos($input, $search); // 9

Неожиданно, но результатом функции будет число 9. Хотя мы видим, что истинная позиция подстроки "мы" в исходной строке равна 5.

Все дело в том, что некоторые строковые функции не всегда корректно обрабатывают кириллические символы, и для них лучше использовать другую функцию - mb\_strpos():

<?php

$position = mb\_strpos($input, $search); // 5

Функция strrpos()

Функция strrpos() во многом аналогична функции strpos(), только ищет позицию не первого, а последнего вхождения подстроки в строку:

<?php

$input = "This is the end";

$search = "is";

$position = strrpos($input, $search); // 5

Но опять же данная функция не совсем корректно работает с кириллическими символами, поэтому нам надо использовать ее аналог - mb\_strrpos():

<?php

$position = mb\_strrpos($input, $search);

Функция trim()

Функция trim($str) удаляет из строки начальные и конечные пробелы, а также управляющие символы \n, \r, \t:

<?php

$input = "  Мама мыла раму  ";

$input = trim($input);

Изменение регистра

Для перевода строки в нижний регистр используется функция strtolower:

<?php

$input = "The World is Mine";

$input = strtolower($input);

Для перевода в нижний регистр строки с кириллическими символами можно использовать функцию mb\_strtolower:

<?php

$input = mb\_strtolower($input);

Для перевода строки в верхний регистр примеяются функции strtoupper()/mb\_ strtoupper(), которые работают аналогично.

Функция strlen()

Функция strlen() возвращает длину строки, то есть количество символов в ней:

<?php

$input = "Hello world";

$num = strlen($input);

echo $num;

Функция strlen() также некорректно работает с кириллицей, поэтому в этом случае лучше применять функцию mb\_strlen():

<?php

$input = "Мама мыла раму";

$num = mb\_strlen($input);

echo $num;

Получение подстроки

Применяя функцию substr($str, $start [, $length]), можно получить из одной строки ее определенную часть. Данная функция обрезает строку $str, начиная c символа в позиции $start до конца строки. С помощью дополнительного необязательного параметра $length можно задать количество вырезаемых символов.

<?php

$input = "The world is mine!";

$subinput1 = substr($input, 2);

$subinput2 = substr($input, 2, 6);

echo $subinput1;

echo "<br>";

echo $subinput2;

Так как данная функция некорректно работает с кириллицей, то вместо нее следует применять функцию mb\_substr(), которая действует аналогично:

<?php

$input = "Мама мыла раму";

$subinput1 = mb\_substr($input, 2);

$subinput2 = mb\_substr($input, 2, 6);

Замена подстрок

Для замены определенной части строки применяется функция str\_replace($old, $new, $input). Эта функция заменяет в строке $input все вхождения подстроки $old на подстроку $new с учетом регистра:

<?php

$input = "Мама мыла раму";

$input = str\_replace("мы", "ши", $input);

echo $input;

Основной набор функций для работы со строками: <https://www.php.net/manual/ru/ref.strings.php>

Регулярные выражения

Регулярные выражения в PHP (с англ. RegEx или regex) представляют собой последовательность символов, описывающих специальные шаблоны, которые применяются для различных вариантов обработки текста: проверки, поиска и замены того или иного фрагмента, состоящего из букв, цифр или любых других символов, в том числе специальных символов и символов пунктуации.

Синтаксис регулярных выражений

Регулярное выражение представляет собой строку, которая всегда начинается с символа разделителя, за ним следует шаблон регулярного выражения, затем еще один символ разделителя и, наконец, необязятельный список модификаторов.

<?php

$exp = "/wm-school/i";

В приведенном выше примере, в качестве символа разделителя используется слэш /, wm-school — это шаблон, по которому идет поиск, а символ i, расположенный после второго разделителя — это модификатор, позволяющий вести поиск без учета регистра.

В качестве символа разделителя может быть любой символ, кроме буквы, цифры, обратной косой черты \ или пробела. Самый распространенный разделитель — это слэш /, но если ваш шаблон содержит слэш, удобно выбрать другие разделители, такие как # или ~.

Функции регулярных выражений

PHP предоставляет программистам множество полезных функций, позволяющих использовать регулярные выражения. Рассмотрим некоторые функции, которые являются одними из наиболее часто используемых:

|  |  |
| --- | --- |
| **Функция** | **Определение** |
| preg\_match() | Эта функция ищет конкретный образец в некоторой строке. Он возвращает 1 (true), если шаблон существует, и 0 (false) в противном случае. |
| preg\_match\_all() | Эта функция ищет все вхождения шаблона в строке. Она возвращает количество найденных совпадений с шаблоном в строке, или 0 — если вхождений нет. Функция удобна для поиска и замены. |
| ereg\_replace() | Эта функция ищет определенный шаблон строки и возвращает новую строку, в которой совпадающие шаблоны были заменены другой строкой. |
| eregi\_replace() | Функция ведет себя как *ereg\_replace()* при условии, что поиск шаблона не чувствителен к регистру. |
| preg\_replace() | Эта функция ведет себя как функция *ereg\_replace()* при условии, что регулярные выражения могут использоваться как в шаблоне так и в строках замены. |
| preg\_split() | Функция ведет себя как функция PHP *split()*. Он разбивает строку на регулярные выражения в качестве параметров. |
| preg\_grep() | Эта функция ищет все элементы, которые соответствуют шаблону регулярного выражения, и возвращает выходной массив. |
| preg\_quote() | Эта функция принимает строку и кавычки перед каждым символом, который соответствует регулярному выражению. |
| ereg() | Эта функция ищет строку, заданную шаблоном, и возвращает TRUE, если она найдена, иначе возвращает FALSE. |
| eregi() | Эта функция ведет себя как функция *ereg()* при условии, что поиск не чувствителен к регистру. |

Примечание:

По умолчанию регулярные выражения чувствительны к регистру.

В PHP есть разница между строками внутри одинарных кавычек и строками внутри двойных кавычек. Первые обрабатываются буквально, тогда как для строк внутри двойных кавычек печатается содержимое переменных, а не просто выводятся их имена.

Таблица регулярных шаблонов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **[abc]** | Один из символов a, b, c | **.** | Любой символ | **(...)** | Группировка (группа) |
| **[^abc]** | Любой символ кроме a, b, c | **\s** | Пробельный символ (в т.ч. табуляция) | **(a|b)** | a или b |
| **[a-z]** | Любой символ из диапазона | **\S** | Любой не пробельный символ | **a?** | Одна a или пусто |
| **[a-zА-Я]** | Любой символ из диапазонов | **\d** | Любая цифра | **a\*** | Ноль и более раз a |
| **^** | Начало строки | **\D** | Не цифра | **a+** | Один и более раз a |
| **$** | Конец строки | **\w** | Буква (буквы, цифры, подчеркивание) | **a{3}** | 3 раза a |
| **\A** | Начало текста | **\W** | Не буква (специальные символы) | **a{3,}** | 3 и более раз a |
| **\z** | Конец текста | **\b** | Граница слова (начало или конец) | **a{3,5}** | От 3 до 5 раз a |
| **\n** | Символ новой строки в Unix | **\r\n** | Символ новой строки в Windows | **\t** | Символ табуляции |
|  |  | | | | |
| **\pL** | Буква (в т.ч. русская) в кодировке utf-8, используется с модификатором **u** | | | | |

**Модификаторы**

|  |  |
| --- | --- |
| **i** | Регистронезависимость. Например "/a/i" ищет и a, и A. |
| **U** | Инвертирует "жадность" (по умолчанию жадный, т.е. пытается захватить как можно большую строку, подходящую по условию). |
| **u** | Использование кодировки utf-8 (для поиска русского текста например). |
| **m** | Многострочный поиск. |
| **s** | Символ . (точка) соответствует и переводу строки. |
| **x** | Игнорировать пробелы. В этом случае пробелы нужно экранировать обратным слэшем \. |

Функция preg\_match()

Функция preg\_match() выполняет проверку на соответствие регулярному выражению.

Пример. Поиск подстроки "php" в строке без учета регистра:

<?php

$str = "PHP - это язык веб-сценариев.";

$pattern = "/php/i";

if (preg\_match($pattern, $str)) {

  echo "Вхождение найдено.";

} else {

  echo "Вхождение не найдено.";

}

Результат выполнения кода:

Вхождение найдено.

В примере выше символ "i" после закрывающего ограничителя шаблона означает регистронезависимый поиск, поэтому вхождение будет найдено.

Функция preg\_match\_all()

Функция preg\_match\_all() выполняет глобальный поиск шаблона в строке.

В примере регулярное выражение используется для подсчета числа вхождений «ain» в строку без учета регистра:

<?php

$str = "The rain in England falls mainly on the plains.";

$pattern = "/ain/i";

echo preg\_match\_all($pattern, $str);

?>

Результат выполнения кода:

3

Функция preg\_replace()

Функция preg\_replace() выполняет поиск и замену по регулярному выражению.

В следующем функция выполняет поиск в строке совпадений с шаблоном pattern и заменяет их на replacement:

<?php

$string = "Выпускной вечер в 2021 г";

$pattern = "/2021/i";

$replacement = "2022";

echo preg\_replace($pattern, $replacement, $string);

?>

Результат выполнения кода:

Выпускной вечер в 2022 г

Группировка

Группы (подмаски) в регулярных выражениях делаются с помощью метасимвола группировки ().

Например в выражении xyz+ знак плюс (+) относится только к букве z и это выражение ищет слова типа xyz, xyzz, xyzzz. Но если поставить скобки x(yz)+ то квантифиактор (+) относится уже к последовательности yz и регулярка ищет слова xyz, xyzyz, xyzyzyz.

<?php

$str = "Hello, boy Bananan!";

$pattern = "/Ba(na){2}n/";

echo preg\_match($pattern, $str);

?>

Результат выполнения кода:

1

Практика регулярные выражения

1. Напишите сценарий PHP, удаляющий последнее слово из строки.

Пример строки: Быстрая рыжая лиса

Ожидаемый результат: Быстрая рыжая

<?php

$str1 = 'Быстрая рыжая лиса';

echo preg\_replace('/\W\w+\s\*(\W\*)$/iu', '$1', $str1)."\n";

?>

Синтаксис

<?php

mixed preg\_replace ( mixed pattern, mixed replacement, mixed subject [, int limit] )

?>

Синтаксис

<?php

mixed preg\_replace ( mixed pattern, mixed replacement, mixed subject [, int limit] )

?>

Выполняет поиск в строке *subject* совпадений с шаблоном *pattern* и заменяет их на *replacement*. В случае, если параметр *limit* указан, будет произведена замена *limit* вхождений шаблона; в случае, если *limit* опущен либо равняется -1, будут заменены все вхождения шаблона.

**Расшифровка к упражнению:**

После ограничителя идут модификаторы "iu", которые изменяют поведение функции preg\_replace. В нашем случае:

* "i" означает отключение чувствительности к регистру искомого символа
* "u" означает, что нужно включить в функции preg\_replace поддержку UTF-8 кодировки. Иначе кириллица будет обрабатываться неправильно

|  |  |
| --- | --- |
| **\W** | - НЕ алфавитно-цифровой или '\_' символ |
| **\w+** | - слово |
| **\w** | - алфавитно-цифровой или '\_' символ |
| **\s** | - любой пробельный символ |
| **\*** | - повторяется 0 или большее число раз |
| **$** | - конец строки |

1. Напишите сценарий PHP, который проверяет, содержит ли строка другую строку.

<?php

$pattern = '/[^\w]лиса\s/';

if (preg\_match($pattern, 'Быстрая коричневая лиса прыгает через ленивую собаку'))

  {

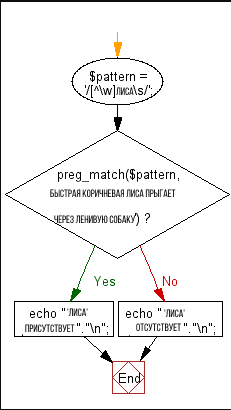
  echo "'лиса' присутствует"."\n";

  }

  else

  echo "'лиса' отсутствует"."\n";

?>



|  |  |
| --- | --- |
| **\** | - считать следующий метасимвол как обычный символ. |
| **^** | - начало строки |
| **\w** | - все буквенно и цифровые символы (многие руководства почему-то не договаривают про цифровые символы) |
| **\s** | - любой пробельный символ |
| **[]** | - класс символов |

1. Напишите сценарий PHP, удаляющий пробелы из строки.

Пример строки: Быстрая " " рыжая лиса

Ожидаемый результат: Быстрая""рыжаялиса

<?php

$str1 = 'Быстрая " "   рыжая лиса';

echo preg\_replace('/\s+/', '', $str1)."\n";

?>

1. Напишите сценарий PHP для удаления нечисловых символов, кроме запятой и точки.

Пример строки: $123,34.00A

Ожидаемый результат: 12,334.00

<?php

$str1 = "$12,334.00A";

echo preg\_replace("/[^0-9,.]/", "", $str1)."\n";

?>

[^0-9,.] — любой символ кроме цифр, запятой и точки

1. Напишите сценарий PHP для удаления символов новой строки \n из строки.

Пример строки: *Мерцай, мерцай, звездочка,\nКак мне интересно, кто ты. \nВверху над миром, так высоко, \nПодобно алмазу в небе.*  
Ожидаемый результат: *Мерцай, мерцай, маленькая звездочка, Как мне интересно, кто ты. Вверху над миром так высоко, Подобно алмазу в небе.*

<?php

$str = "Мерцай, мерцай, звездочка,\nКак мне интересно, кто ты. \nВверху над миром, так высоко, \nПодобно алмазу в небе.";

echo preg\_replace('/\s+/', ' ', trim($str))."\n";

?>

### Синтаксис

<?php

string trim ( string str [, string charlist] )

?>

Эта функция возвращает строку str с удаленными из начала и конца строки пробелами. Если второй параметр не передан, **trim()** удаляет следующие символы:

* " " (ASCII 32 (0x20)), символ пробела.
* "\t" (ASCII 9 (0x09)), символ табуляции.
* "\n" (ASCII 10 (0x0A)), символ перевода строки.
* "\r" (ASCII 13 (0x0D)), символ возврата каретки.
* "\0" (ASCII 0 (0x00)), NUL-байт.
* "\x0B" (ASCII 11 (0x0B)), вертикальная табуляция.

1. Напишите сценарий PHP для извлечения текста (в скобках) из строки.

Пример строки: *Быстрая рыжая [лиса]*  
Ожидаемый результат: *лиса*

<?php

$my\_text = 'Быстрая рыжая [лиса].';

preg\_match('#\[(.\*?)\]#', $my\_text, $match);

print $match[1]."\n";

?>

При использовании любой PCRE функции необходимо заключать шаблон в разделители. Разделителем может быть любой символ не являющийся буквой, цифрой, обратной косой чертой или каким-либо пробельным символом.

Часто используемыми разделителями являются косые черты (/), знаки решётки (#) и тильды (~). Ниже представлены примеры шаблонов с корректными разделителями.

1. Напишите сценарий PHP для удаления всех символов из строки, кроме a-z A-Z 0-9 или " ".

Пример строки: *abcde$ddfd @abcd )der]*  
Ожидаемый результат: *abcdeddfd abcd der*

<?php

$string = 'abcde$ddfd @abcd )der]';

echo 'Старая строка : '.$string.'<br>';

$newstr = preg\_replace("/[^A-Za-z0-9 ]/", '', $string);

echo 'Новая строка : '.$newstr."\n";

?>

Работа с cookie

Cookie (куки) представляют небольшие наборы данных (не более 4 кБайт), с помощью которых веб-сайт может сохранить на компьютере пользователя любую информацию. С помощью куки можно отслеживать активность пользователя на сайте: залогинен пользователь на сайте или нет, отслеживать историю его визитов и т.д.

Сохранение cookie

Для установки куки на компьютере пользователя используется функция setcookie(). Она должна вызываться перед тем, как будет отправлен ответ пользователю. Эта функция имеет следующее определение:

<?php

bool setcookie(string $name, string $value, int $expire,

string $path, string $domain, bool $secure, bool $httponly);

Функция setcookie() может принимать следующие параметры:

name: имя cookie, которое будет использоваться для доступа к его значению

value: значение или содержимое cookie - любой алфавитно-цифровой текст не более 4 кБайт

expire (необязательный параметр): срок действия в секундах, после которого cookie уничтожаются. Если данный параметр не установлен или равен 0, то уничтожение cookie происходит после закрытия браузера. Обычно устанавливается относительно текущего времени, которое можно получить с помощью функции time()

path (необязательный параметр): путь к каталогу на сервере, для которого будут доступны cookie. Если задать "/", cookie будут доступны для всего сайта. Если задать, например, "/mydir/", cookie будут доступны только из каталога /mydir/ и всех его подкаталогов. По умолчанию значением является текущий каталог, в котором устанавливаются cookie.

domain (необязательный параметр): задает домен, для которого будут доступны cookie. Если это домен второго уровня, например, localhost.com, то cookie доступны для всего сайта localhost.com, в том числе и для его поддоменов типа blog.localhost.com.

Если задан поддомен blog.localhost.com, то cookie доступны только внутри этого поддомена.

secure (необязательный параметр): указывает на то, что значение cookie должно передаваться по протоколу HTTPS. Если задано true, cookie от клиента будет передано на сервер, только если установлено защищенное соединение. По умолчанию равно false.

httponly (необязательный параметр): если равно true, cookie будут доступны только через http протокол. То есть cookie в этом случае не будут доступны скриптовым языкам, например, JavaScript. По умолчанию параметр равен false

Сохраним cookie:

<?php

$name = "Tom";

$age = 36;

setcookie("name", $name);

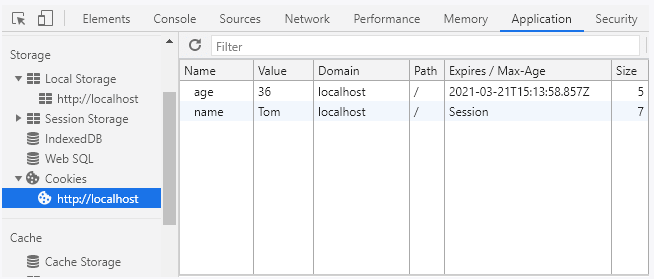
setcookie("age", $age, time() + 3600);  // срок действия - 1 час (3600 секунд)

echo "Куки установлены";

?>

Здесь устанавливаются две куки: "name" и "age". Первая куки уничтожается после закрытия браузера, а вторая - через 3600 секунд, то есть через час.

При необходимости мы можем увидеть сохраненные куки в браузере с помощью инструментов разработчика. Например, вид куки в Google Chrome:



В cookie можно сохранить любую информацию, но не стоит сохранять важные с точки зрения безопасности данные, например, пароли. А если и сохранять какую-то важную информацию, то следует хранить ее в зашифрованном виде.

Получение cookie

Чтобы получить cookie, можно использовать глобальный ассоциативный массив $\_COOKIE, например, $\_COOKIE["name"]. Так, получим ранее сохраненные куки:

<?php

if (isset($\_COOKIE["name"])) echo "Name: " . $\_COOKIE["name"] . "<br>";

if (isset($\_COOKIE["age"])) echo "Age: " . $\_COOKIE["age"] . "<br>";

?>

Сохранение массивов в cookie

Сохранение в куки массивов имеет некоторые особенности. Например, сохраним следующий массив:

<?php

setcookie("lang[1]", "PHP");

setcookie("lang[2]", "C#");

setcookie("lang[3]", "Java");

?>

Теперь получим его и выведем на страницу:

<?php

if (isset($\_COOKIE["lang"])) {

   foreach ($\_COOKIE["lang"] as $name => $value) {

       $name = htmlspecialchars($name);

       $value = htmlspecialchars($value);

       echo "$name. $value <br />";

   }

}

?>

Удаление cookie

Для удаления cookie достаточно в качестве срока действия указать какое-либо время в прошлом:

<?php

setcookie ("name", "", time() - 3600);

?>

Сессии

Сессии представляют набор переменных, которые хранятся на сервере (либо часть на сервере, а часть - в cookie браузера) и которые относятся только к текущему пользователю. В какой-то степени сессии являются альтернативой кукам в плане сохранения данных о пользователе.

Для запуска сессии необходимо вызвать функцию session\_start(). Она должна вызываться до отправки ответа пользователю:

<?php session\_start(); ?>

При запуске сессии с помощью функции session\_start(), если пользователь первый раз заходит на сайт, PHP назначает ему уникальный идентификатор сессии. Этот идентификатор с помощью cookie, которые по умолчанию называются "PHPSESSID", сохраняется в браузере пользователя. С помощью этого идентификатора пользователь ассоциируется с данными сессии. Если для пользователя уже установлена сессия, то данная функция продлевает текущую сессию вместо установки новой.

С помощью специальных функций мы можем получить идентификатор сессии:

<?php

session\_start();

echo session\_id(); // идентификатор сессии

echo session\_name();  // имя - PHPSESSID

То же значение мы могли бы получить, обратившись к cookie напрямую:

<?php

echo $\_COOKIE["PHPSESSID"];

Затем для сохранения или получения данных в сессии надо использовать глобальный ассоциативный массив $\_SESSION. Сохранение переменной в сессии:

<?php

$\_SESSION["имя\_переменной"] = значение;

Получение сохраненного значения:

<?php

$переменная = $\_SESSION["имя\_переменной"];

Сохранение данных в сессии

Запустим сессию и сохраним в ней значения:

<?php

session\_start();

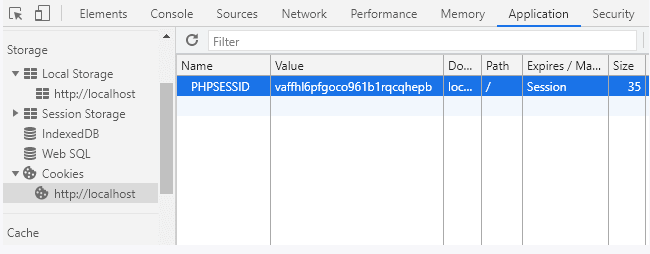
$\_SESSION["name"] = "Sam";

$\_SESSION["age"] = 41;

echo "Данные сохранены в сессии";

?>

После установки сессии в браузере мы сможем заметить установку специальной куки, которая по умолчанию называется "PHPSESSID":



Получение данных сессии

Теперь получим эти значения и выведем на страницу:

<?php

session\_start();

if (isset($\_SESSION["name"]) && isset($\_SESSION["age"]))

{

    $name = $\_SESSION["name"];

    $age = $\_SESSION["age"];

    echo "Name: $name <br> Age: $age";

}

?>

Удаление данных сессии

Сессия уничтожается с закрытием браузера, однако мы также можем программно удалить либо какие-то отдельные, либо все данные сессии.

Для удаления одной переменной из сессии применяется функция unset():

<?php

session\_start();

unset($\_SESSION["age"]);    // удаляем из сессии переменную "age"

?>

Удалить все данные сессии можно с помощью функции session\_destroy():

<?php

session\_start();

session\_destroy();

?>

Куки и сессии используются именно для авторизации и регистрации пользователей, мы реализуем такую систему в следующих встречах, когда будем изучать mysql

Пример:

<?php

session\_start();

function getUserDatabase() {

    return [

        1 => ['id' => 1, 'login' => 'Alex', 'password' => 12345],

        2 => ['id' => 2, 'login' => 'Bob', 'password' => 11111]

    ];

}

function checkSession() {

    return isset($\_SESSION['user\_id']);

}

function checkCookieAndLogin() {

    if(isset($\_COOKIE['user\_id'])) {

        $userId = (int)$\_COOKIE['user\_id'];

        $users = getUserDatabase();

        if(isset($users[$userId])) {

            $\_SESSION['user\_id'] = $userId;

            return true;

        }

    }

    return false;

}

if(!checkSession() && !checkCookieAndLogin()) {

    $users = getUserDatabase();

    $randUserId = array\_rand($users);

    $\_SESSION['user\_id'] = $randUserId;

    setcookie('user\_id', $randUserId);

}

echo "Welcome your id: ".$\_SESSION['user\_id'];

?>

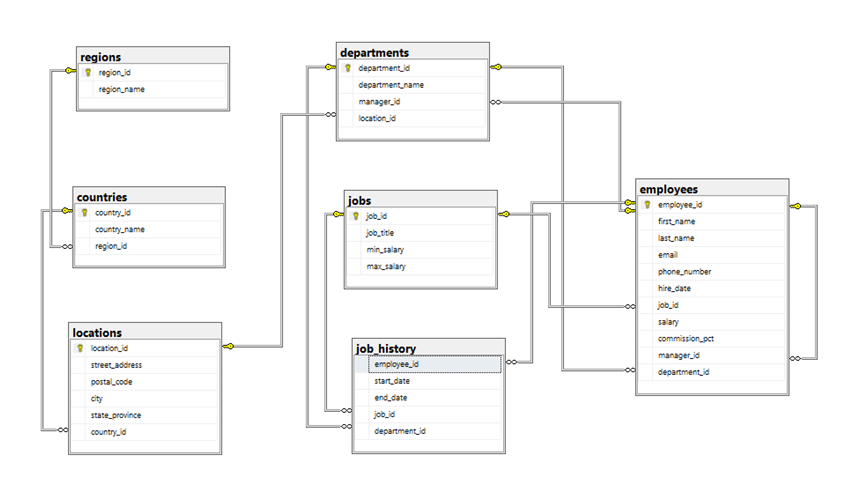
Материалы к уроку:

* 1. <https://www.php.net/manual/ru/function.include.php>
  2. <https://www.php.net/manual/en/language.namespaces.php>
  3. Полный набор функций для работы со строками <https://www.php.net/manual/en/ref.strings.php>
  4. <https://www.php.net/manual/ru/ref.pcre.php>
  5. <https://www.php.net/manual/ru/features.cookies.php>
  6. <https://www.php.net/manual/ru/book.session.php>
  7. Book <https://drive.google.com/file/d/1QrZIPGf5UFVajoLiYnwaX4Z83SiPrkkl/view?usp=sharing>

Домашнее задание:

1. Реализуйте систему авторизации пользователя на сайте. При первом посещение вебсайта у пользователя отображается функционал войти на сайт или регистрация, но если пользователь вводит правильный логин и пароль в формы, то при последующих входах на веб сайт у него есть только кнопка выйти, если он нажимает выйти, то снова у него на экране будет войти на сайт или регистрация.
2. Создайте нужные классы для компании google диаграмма ниже на скрине:

Реализуйте конструкторы, getters, setters, toString для каждого класса:



Реализовать систему CRUD

Create

Read

Update

Delete

Сделайте массивы объектов и их инициализацию данными, данные можно придумать или взять с интернета например топ 100 имен в USA

Выполнить ряд задач ниже для этих массивов, данные отобразить с помощью html/css/js в веб браузере:

1. Получить список с информацией обо всех сотрудниках
2. Получить список всех сотрудников с именем 'David'
3. Получить список всех сотрудников с job\_id равным 'IT\_PROG'
4. Получить список всех сотрудников из 50го отдела (department\_id) с зарплатой(salary), большей 4000
5. Получить список всех сотрудников из 20го и из 30го отдела (department\_id)
6. Получить список всех сотрудников у которых последняя буква в имени равна 'a'
7. Получить список всех сотрудников из 50го и из 80го отдела (department\_id) у которых есть бонус (значение в колонке commission\_pct не пустое)
8. Получить список всех сотрудников у которых в имени содержатся минимум 2 буквы 'n'
9. Получить список всех сотрудников у которых зарплата находится в промежутке от 8000 до 9000 (включительно)
10. Получить список всех сотрудников у которых в имени содержится символ '%'
11. Получить список всех ID менеджеров
12. Получить список всех сотрудников у которых длина имени больше 10 букв
13. Получить список всех сотрудников у которых в имени есть буква 'b' (без учета регистра)
14. Получить список всех сотрудников у которых в имени содержатся минимум 2 буквы 'a'
15. Получить список всех сотрудников зарплата которых кратна 1000
16. Получить первое 3х значное число телефонного номера сотрудника если его номер в формате ХХХ.ХХХ.ХХХХ
17. Получить первое слово из имени департамента для тех у кого в названии больше одного слова
18. Получить имена сотрудников без первой и последней буквы в имени
19. Получить список всех сотрудников у которых последняя буква в имени равна 'm' и длинной имени большей 5ти
20. Получить город в котором сотрудники в сумме зарабатывают меньше всех.
21. Получить список сотрудников у которых менеджер получает зарплату больше 15000.
22. Показать все департаменты в которых нет ни одного сотрудника
23. Показать всех сотрудников которые не являются менеджерами
24. Показать всех менеджеров которые имеют в подчинении больше 6ти сотрудников
25. Показать сотрудников которые работают в департаменте IT

