Б. Панайотов

# лекции по
# ПРОГРАМИРАНЕ НА PHP

## 1. Принципи на работа в Интернет.

Езикът PHP е разработен като инструмент за решаване на чисто практически задачи. Неговият създател е Размус Лердорф.

### *1.1. Протоколи за предаване на данни*

Както всяка компютърна мрежа, Интернет е основан на множество компютри, свързани един с друг с кабели, чрез спътникови канали и др. За предаването/приемането на данни предаващите/приемащите страни трябва да се придържат към множество съглашения., позволяващи строго да се регламентира предаването на даните, а така също и че приемането ще се извърши успешно. Такъв набор от правила се нарича протокол на предаването. Протоколът позволява на системите, взаимодействащи в рамките на Интернет, да обменят данни в най-удобна за тях форма.

#### 1.1.1. Протокол TCP/IP

За различни цели съществуват различни протоколи. В Web-програмирането се използват протокол TCP (Transmission Control Protocol – Протокол за управлението предаването на данни), и по-точно протокол HTTP (Hypertext Transfer Protocol – Протокол за предаване на хипетекст), базиращ се на TCP. Протокол HTTP работи на браузъри и Web-сървъри.

В Интернет основно се използва протокол TCP, който в своята работа използва протокол IP, базиращ се на услуги, предоставени от други протоколи. Протоколи TCP и IP са толкова силно свързани, че е прието те да се обединяват в една група под името семейство TCP/IP (в него се включва също и протокол UDP).

Характерни са следните особености на протокол TCP/IP:
- Гарантирана коректна доставка на данни до местоназначението, разбира се, ако изобщо е възможно. Даже, ако връзката е ненадеждна, загубените фрагменти от данни се изпращат отново и отново, докато цялата информация не бъде предадена.
- Предаваната информация се представя във вид на поток, наподобяващ обмяна на файлове, в практически всички операционни системи.
- TCP/IP е устроен така, че има способността да избира оптималния маршрут за разпространение на сигнала между предаващата и приемащата страна, даже ако сигналът преминава през стотици междинни компютри. Протоколът избира път, по който данните може да бъдат предадени за минимално време, основан на статистическа информация за работата на Мрежата и т.н. таблици за маршрутизация.
- Предаването на данни се разбива на фрагменти – пакети, които се доставят в местоназначението отделно. Различните пакети може да следват различни маршрути в Интернет, но за всички тях е гарантирана правилното сглобяване в местоназначението в определен ред.

В Web-програмирането рядко се работи директно с TCP/IP, обикновено се използват езици от по-високо ниво, служещи за обмен на информация между сървърите и браузърите.

## *1.2. Адресация в мрежа*

Компютрите в Интернет са много. Идентифицирането на конкретна система или програма, желаещи да обменят данни с Интернет се извършва от TCP/IP с помощта на IP-адреса и номер на порта.

### 1.2.1. IP-адрес

Произволен компютър, включен в Интернет и желаещ обмяна на информация, е длъжен да има уникално име, или IP-адрес. IP-адресът изглежда примерно така:

```
127.12.232.56
```

Както се вижда, това са четири 8-разрядни числа (принадлежащи на интервала от 0 до 255 включително), съединени с точки. Не всички числа са допустими в записа на IP-адреса: някои се използват за служебни цели (например, адрес 127.0.0.1 е отделен за обръщение към локалната машина). Директната работа с IP-адресите е неудобна. За целта се използват имена на домейни

### 1.2.2. Имена на домейни

По-лесно е да се запомни символно име, отколкото набор числа. За да се облекчи обикновената ползвателска работа с Интернет се използва системата DNS (Domain Name System – Система имена на домейни). DNS е разпределена база от данни, способна да преобразува имената на домейни в техните IP-адреси.

При използване на DNS произволен компютър в Интернет може да има не само IP-адрес, но така също и символично име. То изглежда примерно така:

```
www.somehost.mtu.du
```

Това е набор от думи (с произволен брой), съединени с точки. Всяко такова съчетание от думи се нарича домейн от N-то ниво (например, `du` – домейн от първо ниво, mtu.du – домейн от второ ниво, somehost.mtu.du – домейн от трето ниво и т.н.).

Пълното DNS-име не изглежда така: в неговия край задължително стои точка, например:

```
www.somehost.mtu.du.
```

Именно този запис е правилният, но браузърите и другите програми често позволяват да се пропуска завършващата точка. В приетата по-горе терминология, тази точка се нарича домейн от нулево ниво или коренов домейн.

Един IP-адрес може да съответства на няколко имена на домейни. Всяко от тях води до едно и също място – единствен IP-адрес. Благодарение на протокол HTTP 1.1 Web-сървърът, установен на компютър и отговарящ на заявки, е способен да разбере кое име на домейн е въвел потребителя, и спрямо това да реагира, даже ако на неговия IP-адрес да съответстват няколко имена на домейни.

При условие, че на едно DNS-име са съпоставени няколко, различни IP-адреса, то DNS избира този адрес, който се намира най-близо до потребителя, или който най-дълго не е използван.

### 1.2.3. Търсене по име на домейн

За начало името на домейна се преобразува от специални DNS-сървъри, разположени по целия свят, в IP-адрес. Нека клиентът да зададе въпрос за определяне на IP-адрес на машината `www.somehost.bg.`. За да се извършва заявка за т.н. коренов домейн с име "`.`". Заявката съдържа команда: върни IP-адреса на машината, на която е разположена информация за домейн `bg.`. Когато се получи IP-адрес, то се извършва аналогично обръщение за определяне на адреса, съответстващ на домейн `somehost` вътре в домейн `bg` вътре в коренов домейн "`.`". Накрая, се запитва за IP-адреса на поддомейн `www` в домейн `somehost.bg.`.

В системата съществува йерархия: всеки домейн знае имената на всички свои поддомейни, а те на своите поддомейни и т.н. При изменение на домейн на някое ниво, за това са длъжни да разберат всички родителски домейни, за което съществуват специални протоколи за синхронизация.

За да не се получи пренатоварване на Мрежата, всички машини кешират информация за DNS-заявките, обръщайки се към коренов домейн или домейни от първо ниво – `bg`, `com` и др. Технологията позволява да се намали натоварването на DNS-сървърите в Интернет. В някои случаи е възможно получаването на грешни данни, при условие че междувременно е сменен IP-адреса на съответния хост.

### 1.2.4. Порт

Основата на протокол TCP е процесът – програма, стартирана на компютър в Интернет. Именно между процесите, а не между машините, се извършва обмен на данни в термините на протокол TCP.

Нека на някоя машина да се изпълнява програма (Клиент), която иска, чрез Интернет, да се свърже с друга програма (Сървър) на друга машина в Мрежата. За това се изпълняват следните условия:
- програмите се договарят, за това, как ще се идентифицират една с друга;
- програмата Сървър е длъжна да се намира в режим очакване, че някой ще се свърже с нея.

Всъщност, програмата Сървър се обръща към драйвера TCP като му съобщава, че тя ще използва за обмен на данни с Клиенти някой идентификатор, или порт, цяло число в диапазона от 0 до 65 535 (именно такива числа може да се съхраняват в клетки от паметта с размер 2 байта). TCP регистрира това в своите вътрешни таблици – подразбира се, че никоя друга програма не е заела нужния порт (в последния случай се регистрира грешка). След това Сървърът преминава в режим на очакване постъпване на заявки към този порт. Това означава, че произволен Клиент, който иска да пристъпи към диалог със Сървъра, трябва да знае номера на порта. В противен случай TCP-съединението е невъзможно.

Клиентът трябва да знае:
- IP-адреса, на който е стартиран Сървърът;
- номера на порта, които използва Сървърът.

Тази информация е достатъчна, затова Клиентът изпраща към драйвера на TCP команда за съединение с машината, разположена на зададения адрес с указания номер на порт. Тъй като сървърът е готов, той откликва и установява съединение. Сървърът отправя съобщение към

Клиента, че е готов за обмен на данни. TCP-драйверът на Клиента непосредствено преди установяване на съединението избира незает порт от списъка на свободните портове в дадения момент на клиентската машина и му присвоява процес Клиент. След това TCP-драйверът информира Сървъра за номера на порта в първото съобщение за искането за съединение. Чрез изпращането на три такива съобщения между Клиента и Сървъра се установява логически канал на връзка между тях.

Клиентът може да предава данни на Сървъра, с помощта на системните функции в канала, а Сървърът да приема данни, четейки ги от канала.

## *1.3. Терминология*

С изясняването на основните термини, свързани със същността на Интернет се избягват разногласията в по-нататъшните редове.

### 1.3.1. Сървър.

Сървърът е произволен компютър в Интернет, който позволява други машини да го използват в качеството на посредник при предаване на данни. Всички сървъри участват в йерархията при търсене на компютър по IP-адрес, на много от тях се складира всякаква информация, достъпна или не. Сървърът, това е именно машинната част, а не логическата част от Мрежата, той може да има няколко различни IP-адреса, така че може да изглежда в Интернет като няколко независими системи.

Отличителна черта на сървъра се явява това, че той използва само един-единствен TCP/IP-стек, т.е. на има само един екземпляр драйвери от TCP/IP протокола.

За термина сървър има и още едно съвършено друго определение – това е програма (процес) обработващ заявки на клиенти. Например, приложение обслужващо потребители на WWW, се нарича Web-сървър.

### 1.3.2. Възел.

Произволен компютър, свързан с Интернет, имащ свой уникален IP-адрес, се нарича възел. Няма адрес, няма възел. Възелът, съвсем не е задължително да е сървър (например, клиент, включен с модем към доставчик). От логическа гледна точка, Интернет е множество от възли, всеки, от които може да е свързан с произволен друг възел. На една система може да бъдат разположени няколко възела, ако тя има няколко IP-адреса. Например, един възел може да се занимава с пощата, втори – обслужващ WWW, а на трети да работи DNS-сървър.

### 1.3.3. Порт

Порт се нарича число, което идентифицира програмата, желаеща да приема данни от Интернет. Портът е втората съставна на TCP адресацията. Произволна програма, цяляща предаване на данни на друга програма, трябва да знае номера на порта, към който е закрепена последната. Например, традиционно за Web-сървъра се отделя порт с номер 80.

### 1.3.4. Мрежов демон

Мрежовият демон е сървърна програма, която обслужва различните потребители, които могат да се включат към нея. Типичен пример е Web-сървърът, а така също FTP-сървърът и Telnet-сървърът.

### 1.3.5. Провайдер

Провайдерът (доставчик) е организация, имаща модемни входове, към които може да се включват потребители за достъп до Интернет.

### 1.3.6. Хост

Произволен възел в Интернет се явява и хост. Но хост не е задължително да е отделен възел, ако това е виртуален хост. Хостът има собствено уникално име на домейн. Понякога (за да не се повтаря) хостовете се наричат сървъри. Хостът се отличава от възела по това, че той може да бъде виртуален.

### 1.3.7. Виртуален хост

Виртуалният хост е хост, който няма уникален IP-адрес в Мрежата, но е достъпен по някакъв допълнителен адрес (например, негово име на домейн). В последно време в Интернет постоянно нараства броя на виртуалните хостове, което е свързано с повсеместното разпространение на протокол HTTP 1.1.

### 1.3.8. Хостинг-провайдер (хостер)

Това е организация, която може да създава хостове (виртуални и обикновени) в Интернет и да ги продава на различни клиенти. Предлагат се услуги по регистрация на име на домейн в Интернет и др.

### 1.3.9. Хостинг

Хостингът е абонамент за дисково пространство и трафик на даден сървър, който предлага определени услуги за Интернет. Ако искате вашият сайт да е достъпен през цялото време и отвсякъде, е нужно той да бъде инсталиран на специален компютър, който да е непрекъснато включен и постоянно свързан с Мрежата. Този компютър се нарича Web-сървър или Интернет-сървър. Хостинг-услугите се предлагат от хостинг-провайдери.

Операционната система на хостинг-сървъра е Linux, поради своята надеждност. Дисковото пространство е размерът в мегабайти (гигабайти), което ще бъде на разположение на потребителя. В дисковото пространство се включва пространството за съхраняване на пощата към домейна. Месечният трафик зависи от размера и от посещаемостта на сайта. Броят разрешени e-mail адреси зависи от персонала на фирмата, която наема хостинга. Поддържаните сървърни технологии са програмните езици, с помощта на които се изгражда потребителския сайт – PHP, Perl и др. Възможна е връзката с бази от данни

### 1.3.10. Сайт

Сайтът е част от логическото пространство на хост, състоящо се от един или няколко HTML-страници (HTML-документи). Хостът може да съдържа множество сайтове, разпределени в различни категории.

### 1.3.11. HTML-документ

HTML-документът е файл, съдържащ данни в HTML формат.

### 1.3.12. Страница (или HTML-страница)

Адресируемата в Интернет минимална единица текстова информация на службата World Wide Web, се нарича HTML-страница. HTML-страницата може да се намира на Web-сървър и да се изобразява в браузър. Езикът HTML (Hypertext Markup Language – Език за размяна на

хипетекст) позволява да се поставят в страница връзки към други страници. Щракайки с бутона на мишката на полето за връзка може да се преминава към един или друг документ.

### 1.3.13. Web-програмиране

Web-програмирането е разработката на програми, които се занимават с динамично формиране на HTML-страници при потребителски заявки. Всичко останало (в това число, администриране на сървъри и др.) няма отношение към Web-програмирането. За успешна работа на Web-програмиста се изисква наличен, правилно конфигуриран и работещ хостинг.

## *1.4. World Wide Web u URL*

World Wide Web, Web или WWW е най-популярната служба в Интернет. Повечето сървъри в Мрежата поддържат WWW и свързания с нея протокол на предаване HTTP (Hypertext Transfer Protocol – Протокол за предаване на хипетекст). WWW позволява да се организират на хостове сайтове – хранилища на текстова или друга информация, която може да бъде преглеждана в интерактивен режим.

Адресът набиран в браузър се нарича URL (Universal Resource Locator – Универсален идентификато на ресурси) и означава в действителност нищо друго освен адрес.

Всеки Web-сайт съхранява множество документи и е нужен механизъм, който позволява да избирането на конкретен документ вътре в указания хост.

В общия случай URL изглежда примерно така:

```
http://www.somehost.com:80/path/to/document.html
```

### 1.4.1. Протокол

Частта от URL, която указва на бразузъра, какъв протокол от високо ниво да се използва при обмен на данни с Web-сървъра е `http://`. Обикновено това е HTTP, но могат да се поддържат и други протоколи, например, HTTPS, който позволява да се предава информация в специален зашифрован вид.

### 1.4.2. Име на хост

След името на протокола е името на възела, на който се намира страницата, за която е направено запитването – `www.somehost.com`. Това може да бъде не само име на домейн, но и неговия IP-адрес.

### 1.4.3. Порт

Веднага след името на порта чрез двоеточие може да следва името на порта. За протокол HTTP стандартният номер на порт е 80 (или 81). Именно това значение се използва в браузер, ако явно не е указан номер на порт. Портът се идентифицира с постоянно работеща програма на сървър, в частност, порт 80 се свързва с Web-сървъра, който осъществява обработката на HTTP-заявките на клиентите и доставя нужните документи.

### 1.4.4. Път към страница

Път към файла на страницата, в случая е `/path/to/document.html`. Разширението html е прието да се поставя на Web-страниците. HTML е език, на който се задават разположения на текст, фигури, връзки и др. Освен това в html често се срещат и други формати данни: gif, jpg- за изображения, cgi, pl- за скриптове (програми стартиращи се на сървъри) и др.

# 2. Възможности на PHP

## *2.1. CGI*

CGI (Common Gateway Interface – Общ шлюзов интерфейс) се явява стандарт, който е предназначен за създаване на сървърни приложения, работещи по протокол HTTP. Такива приложения (шлюзове или CGI-програми) се стартират на сървъра в реалновременен режим. Сървърът предава заявките на потребителите на CGI-програма, която ги обработва и връща резултат от своята работа в браузъра на потребителя. По този начин потребителят получава динамична информация, която може да се изменя в резултат на различни фактори. Самият шлюз (CGI-скрипт) може да бъде написан на различни езици за програмиране – C/C++, Fortran, Perl, TCL, UNIX Shell, Visual Basic, Python и др.

## *2.2. Области на използване*

PHP е платформено-независим език за програмиране. PHP се използва за създаване на скриптове, работещи от страна на сървъра. PHP е способен да решава задачи, каквито решават други CGI-скриптове, в това число да обработва данни от html-форми, динамично да генерира html-страници и др. Основно PHP се използва в три области: създаване на сървърни приложения , създаване на скриптове за команден ред и създаване на графични приложения.

### 2.2.1. Сървърни приложения

Обикновено PHP се използва за създаване на скриптове от този род. За такива цели е нужен PHP-парсер (обработчик на php-скриптове) и Web-сървър за обработка на скриптове, браузер за преглеждане на резултатите от работата на скрипта и, разбира се, текстов редактор за написване на самия php-код.

### 2.2.2. Скриптове за команден ред

Скриптовете за команден ред се изпълняват направо в командно-редовия интерпретатор. Те са независими от Web-сървъра и браузъра на конкретната машина. Необходим е PHP-парсер. Този вид PHP-скриптове се използват за решаване на обикновени обработки на текст.

### 2.2.3. Графични приложения

Графичните приложения се изпълняват от страната на клиента. Използва се специален инструмент, т.н. PHP-GTK, който е разширение на PHP.

## *2.3. Инсталиране и настройка на програмното осигуряване*

За решаване на задачи с помощта на технологията Клиент-Сървър изисква PHP за създаване на скриптове, които да се обработват на сървър. Необходимо е инсталирането на Web-сървър и интепретатор на PHP. В качеството на сървър се използва, например Apache, като най-популярен сред Web-разработчиците. За преглеждане на резултатите се използва Web-браузър, например Internet Explorer.

За избягване на трудните за извършване настройки е възможно използването на готови приложни програми инсталиращи програми, като EasyPHP, XAMPP, WAMPP и др.

Локалният сървър Apache се разполага на хост `localhost`. Използваното URL може да бъде `http://localhost/list.1.php`. Изглежда като отваряне на обикновена web-страница. Присвоеното разширение php е затова, сървърът да може да разбере, че е нужно да се използва PHP-интерпретатор за обработка на документа.

# 3. Основен синтаксис.

## *3.1. Вграждане на PHP в HTML*

### 3.1.1. Използване на PHP тагове

PHP-скриптът се вгражда в HTML-код. PHP-интерпретаторът узнава, че се използва код на PHP, когато той е заграден в таговете <?php и ?>.

```
<?php

/*код на програмата

?>
```

Например:

```
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>
        <?php
            echo "<p>Здравей, аз съм PHP скрипт!</p>";
        ?>
    </body>
</html>
```

или

```
<?php

echo “Здравей свят!”;

?>
```

Всички грешки в PHP се показват по подразбиране в прозореца на браузъра и намирането им при внимателно търсене и определен опит не е трудно. Интерпретаторът подсказва номера на реда, в който е регистрирана грешка.

PHP-кодът не се вижда, тъй като PHP интерпретаторът обработва скрипта и заменя този код с неговия резултат. Това означава, че от PHP всъщност се получава чист HTML-код, който може да се покаже във всеки браузер.

### 3.1.2. Стилове на PHP таговете

---

## 3.2. Разделяне на инструкциите.

Програмата е набор от инструкции (команди). Отделянето на инструкциите една от друга се извършва с използването на специални символи – разделители. В PHP операторите се разделят един от друг с точка със запетая – (;).

Затварящият таг (?>) се подразбира като завършек на инструкциите, с други думи, ако след инструкциите следва затварящ таг (?>), то може да не се използва точка със запетая в края на инструкциите. Например, програмният код

```
<?php

echo(“Добре дошли в света на PHP”);

?>
```

е еквивалентен на кода

```
<?php echo(“Добре дошли в света на PHP”) ?>
```

В първия случай се използва точка със запетая в края на инструкцията, а във втория – не.

Разделители като интервали, табулации и нови линии се игнорират.

## 3.3. Използване на коментари.

Коментарите са важни за документирането на изходния текст и поясняване на кода. Едноредовите коментари започват с // или #, а многоредовите коментари започват с /* и завършват с */.

```
<?php

echo "Казвам се Петър ";
  // Това е едноредов коментар
  // в стил C++

echo "Фамилията ми е Иванов ";
/* Това е многоредов коментар.
Тук може да бъдат написани няколко реда.
При изпълнение на програмата,
всичко написано тук ще бъде игнорирано. */

echo "Аз програмирам на PHP";
# Това е също едноредов коментар.

?>
```

Коментари от типа (//) и (#) могат да се включват в коментари от типа /*...*/.

# 4. Променливи, константи, изрази.

Променливите – това са области от оперативната памет, достъпът до които се осъществява по тяхното име. Всички данни, с които работят програмите се запазват във вид на променливи.
В PHP типът на променливата се установява не от програмиста, а се определя автоматично по време на изпълнение на програмата в зависимост от контекста, в който се използва променливата.

## *4.1. Имена на променливи*

### 4.1.1. Идентификатори

Имената на променливите са идентификатори, също както имената на функциите и класовете. Идентификаторите може да бъдат с произволна дължина и да съдържат букви, цифри, символ долна черта (_) и символ ($). Те не могат да започват с цифра.

Всички променливи в PHP имат имена, които започват със знак ($), например $my_var. По този начин те лесно се отличават от останалия код.

Имената на променливите са чувствителни към малките и големите букви, т.е. $my_var, $My_var и $MY_VAR са различни променливи. Изключение от това правило са имената на функции – за тях тази разлика не се прави.

Име на променлива може да съвпада с име на функция. Подобна практика трябва да се избягва.

В официалната документация е написано, че името на променливата може да се състои не само от латински букви и цифри, но така също от символи, чиито код е по-голям от 127. Това означава, че променливите могат да съдържат букви от българската азбука. Въпреки това, не се препоръчва използването на българоезични имена на променливите, заради различните кодировки на кирилицата.

Ако е нужно да се укаже тип на променливата, то може да се използва инструкция cast в произволна функция settype().

### 4.1.2. Присвояване на стойност на променливи

На променлива се присвоява стойност чрез оператора за присвояване (=).

## *4.2. Типове данни*

Типът на променливата сочи типа данни, които тя съхранява.

### 4.2.1. Тип integer

Това е целочислен тип, обикновено с дължина 32 бита (от -2 147 483 648 до 2 147 483 647). Целите числа може да бъдат задавани в десетична, осмична или шестнадесетична бройна система, с предшестващ знак (+) или (-).

Използването на осмична бройна система се извършва с прибавяне на префикс числото (0) (нула), за използването на шестнадесетична бройна система е нужно пред числото да се постави префикс (0x).

```
<?php
```

```
// десетично число
$a = 456;

//отрицателно число
$b = -456;

//осмично число (еквивалентно
//на 25 в десетичната бройна система)
$c = 031;

//шестнадесетично число (еквивалентно
//на 177 в десетичната бройна система)
$d = 0xB1;

?>
```

### 4.2.2. Тип double

Реални числа (числа с двойна точност) може да бъдат определяни с помощта на следващите редове:

```
<?php

$a = 4.567;
$b = 4.5e7;
$c = 9E+2;

?>
```

### 4.2.3. Тип string

Тип string се използва за работа с низове от знаци. Низовете са с произволна дължина. Те може да съдържат и нулеви символи.

Еднозначната идентификация на променливите дава възможност за използване на променливите непосредствено в низове. Например:

```
$name = "Иван";
$age = 23;
echo "$name е $age годишен.";
```

След изпълнение на примера се получава низ `"Иван е 23 годишен."`

Низовете се задават с използването на единични или двойни кавички.

Ако низът е заграден в двойни кавички (`"`), то променливите участват със своята стойност. Възможно е указването на специални символи (`\n`) – нов ред, (`\\`) - символ (\), (`\"`) – двойна кавичка, (`\$`) - знак ($).

Вторият способ за задаване на низове е използването на единични кавички (').  Тогава в низа може да се използват само символи (\\) и (\'). Променливите в такива низове не се обработват.

Обединение на низове се извършва с оператор точка (.).

```
<?php

/* низова променлива str */
$str = "Това е низ";

/* добавя на друг низ към нея */
$str = $str . " с допълнителен текст";

/* още един начин за добавяне */
$str .= " и низ в края. \n";

/* този низ завършва с текст '<p>Число: 9</p>' */
$num = 9;
$str = "<p>Число: $num</p>";

/* това е '<p>Число: $num</p>' */
$num = 9;
$str = '<p>Число: $num</p>' ;

/* получаване на първи символ от низ */
$str = 'Това е тест.';
$first = $str [0] ;

/* получаване на последен символ от низ */
$str = 'Това е все още тест';
$last = $str [strlen ($str) -1] ;

?>
```

Един низ може да се оцени числово, ако съдържа символ (.), (e) или (E) – като реално число и в противен случай като цяло число. Значението се определя от началната част на низа. Ако низът започва с допустими числови данни, те се използват в качеството на значение, в противен случай ще бъде оценен като 0.

Допустимите числови данни са незадължителния знак, след който следват една или няколко цифри (може да присъства и десетичната точка), а след тях незадължителнтата експонента. Експонентата, това е (e) или (E), следвано от една или няколко цифри.

```
$my_var = 5 + "12.4";       //$my_var е реално число (17.4)
$my_var = 1 + "-1.3e3";     //$my_var е реално число (-1299)
$my_var = 7 + "Асен-1.3E3"; //$my_var е цяло число (7)
$my_var = 8 + "Никола";     //$my_var е цяло число (8)
$my_var = 3 + "12 Димитър"; //$my_var е цяло число (15)
$my_var = "18.1 книги " + 7;//$my_var е реално число (25.1)
$my_var = "6.0" + 9.0;      //$my_var е реално число (15)
```

```
$my_var = "Иван" + "Силвия";//$my_var е цяло число (0)
```

Примерите може да се тестват, допълвайки всеки низ със следния ред:

```
echo "\$my_var == $my_var; тип " . gettype($my_var) . "<br>\n";
```

### 4.2.4. Тип array

Масивът е променлива, която съхранява множество или поредица от стойности. Един масив може да има много елементи. Всеки елемент може да съдържа единична стойност (например текст или число) или друг масив. Масив съдържащ други масиви се нарича многомерен масив.

Определянето на масив се извършва с конструкция `array()` или непосредствено задавайки стойността на неговите елементи.

```
array ([key0] => value0, [key1] => value1, ...)
```

Езиковата конструкция `array()` приема като параметри двойки ключ => стойност, разделени със запетаи. Символът (=>) установява съответствието между ключа и стойността. Ключът може да бъде както цяло число, така и низ. Числовите ключове на масива се наричат индекси. Индексирането на масив в PHP започва от нула. Стойностите на елементите от масив може да се получат като след името на масива в квадратни се запише ключа на търсения елемент.

```
<?php

$students = array ("KI" => "Програмиране на PHP", 10 => true);

echo $students["KI"];
//изписва се "Програмиране на PHP"

echo $students[10];
//изписва се 1

?>
```

Ако елементът ключ не е зададен, то за ключ се избира максималният числов ключ в масива, увеличен с единица. Ако този максимален елемент е отрицателен, то следващият ключ от масива ще е нула (0).

```
<?php

// масивите arry1 и arry2 са еквивалентни
$arry1 = array(8 => 12, 88, 56, "d" => 12);
$arry2 = array(8 => 12, 9 => 88, 10 => 56, "d" => 12);

echo $arry1["d"]; //изписва 12

?>
```

Използването на TRUE и FALSE като ключове довежда до съответното им преобразуване в единица (1) и нула (0) от тип integer. Като ключ може да се използва NULL – вместо ключ се получава празен низ. Използването на празен низ като ключ е допустимо след записването му в двойни кавички. Като ключове не се използват масиви и обекти.

Задаването на масив със скобите ([) и (]) се извършва като вътре в тях се записва ключа, например `$student["MI"]`. Указването на нов ключ и стойност, например `$student["SA"] = "Методи за транслация"`, добавя нов елемент в масива. Ако не се укаже ключ, а само се присвои стойност, например, `$student[] = "Теория на графите"`, то новия елемент ще има числов ключ, с единица по-голям от максималния съществуващ. При условие, че масив, в който се добавя елемент, несъществува, то този масив се създава. Промяната на стойността на съществуващ елемент от масив се извършва като по неговия ключ се присвои нова стойност.

```
<?php

$arry1 = array(8 => 12, 88, 56, "d" => 12);
$arry2 = array(8 => 12, 9 => 88, 10 => 56, "d" => 12);

$arry1["c"] = "Мария";      //добавя нов елемент
$arry2[8] = "програмиране"; //променя стойността на елемент по ключ
$arry2[] = "hash";// добавяне в масива на нов елемент с ключ 11

?>
```

Изтриването на елемент от масив се извършва с използване на фунцията `unset()`.

```
<?php

$students = array ("KI" => "Програмиране на PHP", 10 => true);

//добавя елемент с ключ 11
$students[] = "PHP – стъпка по стъпка";
//еквивалентно на $students[11] = "PHP – стъпка по стъпка";

unset($students[11]); //изтрива се елемент с ключ 11

unset($students); //изтрива се целия масив

?>
```

Максималният числов ключ при добавянето на елемент в масив с използване на прази скоби (`[]`) се търси сред ключовете съществуващи в масива при последното му преиндексиране. Числовото преиндексиране на масив може да се извърши автоматично с използване на функцията `array_values()`.

```
<?php

$lecturers = array ("Панайотов", "Николов", "Кирилов");
//ключовете не са указани, затова те ще бъдат 0, 1, 2
```

```
print_r($lecturers); //извеждане на масива (ключове и стойности)

unset($lecturers[0]);
unset($lecturers[1]);
unset($lecturers[2]);

print_r($lecturers); //извеждане на масива (ключове и стойности)

$lecturers[] = "Симеонов"; //елементът се добавя с ключ 3, а не 0

print_r($lecturers); //извеждане на масива (ключове и стойности)

$lecturers = array_values($lecturers); //числово преиндексиране на
масив

$lecturers[] = "Петров"; //елементът се добавя с ключ 1

print_r($lecturers); //извеждане на масива (ключове и стойности)

?>
```

### 4.2.5. Тип object

Обектните променливи се използват за съхраняване на инстанции на класове. За достъп до методите на обекта се използва оператор (`->`). Създаването на нов обект се извършва с израз `new`.

### 4.2.6. Тип boolean

Булевият (логически) тип е тип за променливи, съдържащи стойност `true` (истина) и `false` (неистина).

```
<?php

$uspeh = TRUE;

?>
```

### 4.2.7. Тип NULL

Променливи от тип `NULL` са тези, на които не е зададена стойност, които са били унищожени с помощта на `unset()` или им е присвоена стойност `NULL`.

### 4.2.8. Тип resource

Стойност от тип ресурс (връзки към външни ресурси) връщат някои вградени функции (например, функциите за работа с бази от данни).

### 4.2.9. Променливи от тип променлива

---

## 4.3. Константи

### 4.3.1. Дефиниране на константи

Стойностите, съхранявани в променлива, може да се променят. За съхраняване на постоянни величини, чиято стойност не се променя в хода на изпълнение на скрипта, се използват константи. Константата не се унищожава след нейното деклариране. Константите се използват с техните имена без префикс (`$`). Константите се дефинират с помощта на функция `define()`, която има следния синтаксис:

```
define("ИМЕ_НА_КОНСТАНТА",
       "стойност_на_константа"
      [нечувствителност_към_малки/големи_букви])
```

По подразбиране константите са чувствителни към малки/големи букви. Това може да се промени чрез задаване на стойност TRUE на аргумента `нечувствителност_към_малки/големи_букви`. Всички имена на константи се пишат с големи букви, за да се различават от променливите. За получаване стойностите на константите се използва функция constant() с параметър име на константа.

<?php

```
//определяне на константи PI и STR
define("PI", 3.146);
define("STR", "Това е низ");

$a = 9.50*sin(2*PI/9)+6;
echo "Това е числото PI";   //изписва се "Това е числото PI"
echo "Това е числото ".PI;  //изписва се "Това е числото 3.146"

echo STR;

?>
```

### 4.3.2. Предопределени константи

Предопределените константи са установени от самия PHP-интерпретатор. Те могат да се прегледат с командата `phpinfo()`. Командата извежда и допълнителна информация.

## 4.4. Област на видимост на променливите

Различават се четири области на видимост на променливите:
- Вградените суперглобални променливи са видими навсякъде в PHP-скрипта;
- Декларираните като глобални променливи в скрипта са видими навсякъде в този скрипт, освен в неговите функции;
- Променливи, използвани във функция, са локални за тази функция;
- Декларираните като глобални във функция (с конструкция `global`) сочат към глобални променливи със същите имена.

```
<?php

$a = 2;
```

```
$b = 3;

function sum(){

global $a, $b, $d;
$d = $a + $b;

}

sum();
echo $d;
echo $a;
echo $b;
//изписва се "523"
?>
```

Скриптовите суперглобални променливи са следните:
- $GLOBALS – масив от всички глобални променливи;
- $_SERVER – масив от променливи от обкръжението на сървъра;
- $_GET – масив от променливите, подадени на скрипта по метода GET;
- $_POST – масив от променливите, подадени на скрипта по метода POST;
- $_COOKIE – масив от cookie променливи;
- $_FILES - масив от променливи, свързани с качването на файлове (upload);
- $_ENV – масив от променливи от обкръжението;
- $_REQUEST – масив от всички подадени от потребителя променливи (GET, POST и COOKIE);
- $_SESSION – масив от сесийните променливи.

# 5. Оператори

Операторите са символите, които позволяват изпълнението на различни действия с променливи, константи и изрази. Израз е всеки валиден запис на език за програмиране, който има произволна стойност. Променливите и константите са най-простите форми на изрази.

## *5.1. Аритметични оператори*

Аритметичните оператори са математическите оператори.

```
+   Събиране    $a + $b
-   Изваждане   $a - $b
*   Умножение   $a * $b
/   Деление     $a / $b
%   Остатък     $a % $b
```

Всеки оператор позволява съхраняването на резултата от операцията. Например:

```
$c = $a + $b
```

## 5.2. Оператор за работа с низове

Използва се операторът за съединение (конкатенация) на низове (.).

```
$a = "Това е ";
$b = "низ!";
$c = $a.$b;
```

Променливата $c съдържа низа "Това е низ!";

## 5.3. Оператор за присвояване

Операторът за присвояване (=) се чете "приема стойност". Например,

```
$a = 5;
```

се чете "$a приема стойност пет".

### 5.3.1. Връщане на стойност от операция за присвояване

Използва се правилото, че стойността на целия израз за присвояване е стойността, която приема левия операнд.

## 5.4. Комбинирани оператори за присвояване

Комбинираните оператори за присвояване съществуват за всяка аритметична операция, както и за операцията по конкатенация на низове.

| Оператор | Употреба | Еквивалентен |
|---|---|---|
| += | $a += $b | $a = $a + $b |
| -= | $a -= $b | $a = $a - $b |
| *= | $a *= $b | $a = $a * $b |
| /= | $a /= $b | $a = $a / $b |
| %= | $a %= $b | $a = $a % $b |
| .= | $a .= $b | $a = $a . $b |

### 5.4.1. Пре- и пост-инкремент, пре- и пост-декремент

Тези оператори инкрементират (увеличават с единица) и декрементират (намаляват с единица) стойността на променлива, преди или след като се върне стойността на променливата.

| Означение | Име | Описание |
|---|---|---|
| ++$a | Пре-инкремент | Увеличава $a с единица и връща $a |
| $a++ | Пост-инкремент | Връща $a, после увеличава $a с единица |
| --$a | Пре-декремент | Намалява $a с единица и връща $a |
| $a-- | Пост-декремент | Връща $a, после намалява $a с единица |

### 5.4.2. Указател

Операторът за указател се отбелязва със знак & (амперсанд). Той позволява няколко променливи да получат достъп до паметта, съдържаща стойността.

```
$a = 3;
```

```
$b = &$a;
$a = 5; //сега променливите $a и $b са 5
```

Указателят $b може да бъде премахнат с:

unset($b);

Това не указва влияние върху $a или върху стойността й.

## 5.5. Оператори за сравнение

Операторите за сравнение позволяват сравнението на две стойности и ако условието е изпълнено, връщат TRUE, а ако не е – FALSE. Стойност TRUE или FALSE се връща независимо от типовете на участващите аргументи.

| Означение | Име | Описание |
|---|---|---|
| == | Равно на | $a == $b |
| === | Идентично на | $a === $b |
| != | Различно от | $a != $b |
| <> | Различно от | $a <> $b |
| < | По-малко от | $a < $b |
| > | По-голямо от | $a > $b |
| <= | По-малко или равно на | $a <= $b |
| >= | По-голямо или равно на | $a >= $b |

## 5.6. Логически оператори

Логическите оператори се използват за комбиниране на резултатите от логически условия. PHP поддържа следните логически операции:

| Оператор | Име | Употреба | Резултат |
|---|---|---|---|
| ! | НЕ | !$a | Връща истина, ако $b е FALSE, и обратно |
| && | И | $a && $b | Връща TRUE, ако $a и $b са TRUE, иначе връща FALSE |
| \|\| | ИЛИ | $a \|\| $b | Връща TRUE, ако или $a е TRUE, или $b е TRUE, или и двете са TRUE, иначе връща FALSE |
| and | И | $a and $b | Като &&, но с по-нисък приоритет |
| or | ИЛИ | $a or $b | Като \|\|, но с по-нисък приоритет |

## 5.7. Побитови оператори

---

## 5.8. Други оператори

Операторът "запетая" (,) се използва като разделител на аргументите на функция или елементите на списък.

Операторите `new` и (`->`) се използват съответно за създаване на инстанция на клас и за достъп до атрибутите на клас.

Операторът (`[]`) позволява достъп до отделните елементи на масив. При някои масиви се използва и операторът (`=>`).

### 5.8.1. Троичен оператор

---

### 5.8.2. Оператор за потискане на съобщения за грешки

---

### 5.8.3. Оператор за изпълнение

---

## *5.9 Условни оператори*

### 5.9.1 Блокове от код

При използване на условни и циклични оператори се изпълняват няколко инструкции. Групирането им се извършва със скобите (`{`) и (`}`).

### 5.9.2 Оператор if

При вземане на решение с конструкцията if се проверява дали дадено условие е истина или лъжа. Ако условието е TRUE, се изпълнява следващия блок от код. Условията в `if`-конструкцията са заградени със скоби `()`.

```
if (логически_израз) блок_за_изпълнение
```

Във FALSE се преобразуват следните стойности: логическа стойност FALSE, цялото число нула (`0`), реалното число нула (`0.0`), празен низ и низ `"0"`, масив без елементи, празен обект, тип NULL; всички останали стойности се преобразуват в стойност TRUE.

```
$sum = 12;
if ($sum > 10){
    echo "Променливата има стойност по-голяма от 5";
    $sum = 11;
    echo "Всъщност, нейната стойност е равна на $sum";
}
```

### 5.9.3 Оператор else

Оператор `else` позволява определянето на алтернативно действие, което да се извърши, когато оценката на `логически_израз` в оператор `if` е `FALSE`.

```
if (логически_израз) блок_за_изпълнение1
    else блок_за_изпълнение2
```

Възможна е следната употреба на конструкцията `if...else`:

```
$sum = 12;
if ($sum > 10){
    echo "Променливата има стойност $sum";
}
    else
    echo "Променливата има стойност по-малка или равна на 10";
```

### 5.9.4 Оператор `elseif`

При вземане на решения с повече възможности се използват т.н. каскадни тестове, при които се използва конструкция `elsif`, която е комбинация между `else` и `if`. Изпълнява се само една конструкция `elseif`, при условие че `логически_израз` в оператор `if` е FALSE.

```
$sum = 25;
if ($sum < 10)
    echo "Отстъпка 0 лв.";
    elseif ($sum < 20)
        echo "Отстъпка 2 лв.";
    elseif ($sum < 30)
        echo "Отстъпка 3 лв.";
    elseif ($sum < 40)
        echo "Отстъпка 4 лв.";
```

### 5.9.5 Оператор `switch`

Операторът `switch` може да се използва като алтернатива на `if` за избор на опция от списък с възможности. Операторът switch има следния синтаксис:

```
switch (израз или променлива){
case израз1: блок_от_оператори1
    break;
case израз2: блок_от_оператори2
    break;
...
case изразN: блок_от_операториN
    break;
default: блок_от_оператори
}
```

където `израз1`, `израз2`, ..., `изразN` са константни изрази с различни стойности. Незадължителният вариант `default` се изпълнява в случай, че никой от останалите варианти не е изпълнен. Употребата на оператори break предотвратява изпълнението на инструкциите, следващи вътре в оператор switch, и продължават с изпълнението на инструкцията, намираща се след оператора.

Семантиката на оператор switch е следната: първо се намира стойността на switch-израза. Получената константа се сравнява последователно със стойностите на израз1, израз2, ..., изразN. При съвпадение, се изпълняват операторите на съответния вариант до срещане на опеератор break. В противен случай, ако участва default-вариант, се изпълнява блока от операторите, който му съответства, до достигане на break и в случай, че не участва такъв – не следват никакви действия от оператора switch.

```
$chisla = array ("нула", "едно", "две");
$zaprint = 2;
switch ($zaprint){
case 0: echo "$chisla[0]";break;
case 1: echo "$chisla[1]";break;
case 2: echo "$chisla[2]";break;
default: echo "неразпознато";break;
}
```

## *5.10. Оператори за цикъл*

Операторите за цикъл се използват за изпълнение на повтарящи се действия. В PHP се работи с четири оператора за цикъл: while, do...while, for и foreach.

### 5.10.1. Оператор while

Операторът while има следния синтаксис:

```
while (условие) { блок_от_оператори}
```

Пресмята се стойността `условие`. Ако тя е FALSE, изпълнението на оператор while завършва без да се е изпълнило тялото му нито веднъж. В противен случай, изпълнението на `блок_от_оператори` и пресмятането на стойността `условие` се повтарят докато `условие` е TRUE. Когато `условие` стане FALSE, изпълнението на while завършва.

```
$i = 1;
while ($i <= 10){
    echo $i++."<br>\n";
}
```

### 5.10.2. Оператор do...while

Използва се за реализиране на произволни циклични процеси. Има следния синтаксис:

```
do{
блок_от_оператори
}
while (условие);
```

В `do...while` се изпълнява тялото на цикъла, след което се пресмята стойността на условие. Ако то е FALSE, изпълнението на оператора завършва. В противен случай се повтарят действията: изпълнение на тялото и пресмятане стойността `условие`, докато стойността на `условие` е TRUE.

```
//извежда числата от 1 до 10
$i = 1;
do{
    echo $i++."<br>\n";
}
while ($i < 11);
```

### 5.10.3. Оператор for

С цикълът for се получава най-компактен код. Той има следния синтаксис:

```
for (инициализация; условие; корекция)
блок_от_оператори
```

Израз инициализация се изпълнява еднократно в началото. Чрез него се установява начална стойност на брояча. Условието се проверява преди всяка итерация и ако резултатът е FALSE, цикълът приключва, без да се изпълни блок_от_оператори. В противен случай, се изпълняват операторите от частта корекция, които променят стойността на брояча и се изпълнява блок_от_оператори.

```
echo("<select name=\"num_players\">\n");

for ($i = 0; $i <= 5; ++$i) {
echo("<option value=\"$i\">$i</option>\n");
}

echo( "</select>\n " ) ;
```

### 5.10.4. Оператор foreach

Оператор foreach предоставя удобен начин за обхождане стойностите на масив. Той има следния синтаксис:

```
foreach ($array as $value) блок_от_оператори
```

или

```
foreach ($array as $key => $value) блок_от_оператори
```

В първата форма на запис се изпълнява цикъл по всички елементи на масива, зададен с променливата $array. На всяка итерация стойността на текущия елемент на масива се записва в променливата $value, и вътрешният брояч на масива се премества с единица напред. Вътре в блок_от_оператори стойността на текущия елемент може да се получи с помощта на променлива $value. Изпълнението на блок_от_оператори се извършва толкова пъти колкото са елементите на масива.

Във втория запис, като допълнителнение на първия, в променлива $key се записва ключа на текущия елемент, която може да се използва в блок_от_оператори.

```
$names = array ("Б. Панайотов", "А. Асенов", "П. Петров");
echo "<ul>";
foreach ($names as $printnames)
    echo "<li> $printnames";
echo "</ul>";
```

## *5.11. Оператори за преход*

Към тази група оператор принадлежат операторите break и continue. Те предават управлението безусловно в някаква точка на програмата.

### 5.11.1. Оператор break

Този оператор завършва изпълнението на текущия цикъл: while, while...do, switch, for, foreach. Може да се използва с числов параметър, който показва колко управляващи структури трябва да се завършат.

### 5.11.2. Оператор continue

Оператор continue позволява прекратяването на текущата итерация на цикъл и започването на нова итерация. Пропускат се останалите оператори в блок_от_оператори.

```
$names = array ("Б. Панайотов", "А. Асенов", "П. Петров");
echo "<ul>";
foreach ($names as $printnames){
    if ($printnames == "А. Асенов") continue;
    echo "<li> $printnames";
}
echo "</ul>";
```

В оператор switch оператор continue работи като оператор break. Ако continue се намира вътре в цикъла и е необходимо започване на нова итерация, следва да се използва `continue 2`.

## *5.12. Оператори за включване*

Операторите include и require позволяват включването на външни файлове в текущо изпълнявания скрипт.

### 5.12.1.Оператор include

Операторът позволява включването на код, съдържащ се в указания файл, и изпълняването му толкова пъти, колкото пъти се среща в програмата този оператор. Използва се следния синтаксис:

```
include 'име_на_файл';
include $file_name;
include ("име_на_файл");
```

файл exter.info

```
<?php
$names = array ("Б. Панайотов", "А. Асенов", "П. Петров");
?>
```

файл main.php

```
<?php

include "exter.info";

echo "<ul>";
foreach ($names as $printnames){
    echo "<li> $printnames";
}
echo "</ul>";
```

```
?>
```

### 5.12.2.Оператор require

Този оператор има действие като оператор include, с изключение на това, че при грешка в в оператора изпълнението на скрипта се преустановява.

# 6.HTML-форми и PHP

HTML-формите в web-сайтовете се нужни за регистрация на потребителите при вход, за обратна връзка, при електронна търговия и др. Даже на обикновените сайтове се използват формите на HTML.

Разполагането на данните във формулярите се извършва с HTML, а за обработката на тези данни е възможно използването на PHP. Не е необходимо извършването на синтактически анализ.

## *6.1.Създаване на примерна HTML-форма*

Следващият пример, файл formata.html, създава примерна HTML-форма, която изисква от потребителя въвеждането на собствено име, фамилно име, e-mail адрес и коментари. Добавени са и два бутона: Изчисти и Изпрати. Формата ще се обработва от файл obrabotka.php.

```
//formata.html

<HTML>
<HEAD>
<TITLE>Изпрати коментар!</TITLE>
</HEAD>
<BODY>
<FORM ACTION="obrabotka.php" METHOD=POST>

Собствено име<INPUT TYPE=TEXT NAME="fn" SIZE=20> <BR>
      Фамилия<INPUT TYPE=TEXT NAME="ln" SIZE=40> <BR>
 E-mail-адрес<INPUT TYPE=TEXT NAME="email" SIZE=60> <BR>
Коментари    <TEXTAREA NAME= "comments" ROWS=5 COLS=40> </TEXTAREA>
<BR>

<INPUT TYPE=RESET NAME="RESET" VALUE="Изчисти!"> <BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Изпрати!">

</FORM>
</BODY>
</HTML>
```

Атрибут ACTION трябва правилно да указва съществуващ на сървъра файл, иначе формата няма да се обработи. В този случай, файл obrabotka.php се намира в една папка със страницата formata.html.

## *6.2. Методи за заявка*

Атрибутът METHOD указва на сървъра как да бъдат предадени данните от формата към обработващия скрипт. За атрибутът METHOD има два варианта: GET и POST.

### 6.2.1. Метод GET

Метод GET връща ресурс. Може да се използва заявка, за да се добави допълнителна информация. Заявкат изпраща цялата събрана информация като част от URL-адреса. Например, при използване на метод GET потребителят ще види следния запис в браузъра си:
`http://127.0.0.1/obrabotka.php?FirstName=Borislav&LastName=Stoyanov&Email=abc@yahoo.com&Comments=bez+komentar&SUBMIT=Submit`

Метод GET спада към безопасните методи, защото няма постоянни странични ефекти върху сървъра. Интервалите се интерпретират като знак (+).

### 6.2.2. Метод POST

Метод POST изпраща скрито от потребителя данни към сървъра в тялото на HTTP-заявката. Например, с метод POST потребителят ще види само записа: `http://127.0.0.1/obrabotka.php`.

Методът POST спада към опасните методи, тъй като той има странични ефекти върху сървъра.

### 6.2.3. GET срещу POST

Методът GET се използва, ако са изпълнени всички от следните условия:
- Ако заявката на практика служи за откриване на ресурс, а данните в HTML-формата се използват за да подпомогнат търсенето;
- Резултатите от заявката нямат трайни странични ефекти;
- Ако общият размер на събраните данни и на имената на полетата <input> в HTML-формата е по-малък от 1024 символа.

Методът POST се използва, ако е изпълнено някое от следните условия:
- Резултатите от заявката имат трайни странични ефекти – например, добавяне на нов ред в базите от данни;
- Има вероятност данните, събирани във формуляра, да произведат дълъг URL, ако се използва метода GET;
- Изпращаните данни използват кодиране, различно от ASCII таблицата.

## *6.3. Получаване на данни от HTML-форма в PHP*

Съществуват различни начини за достъп до получаваните по протокол HTTP данни. Възможно е към имената от формата да се използват като променливи, например, ако е предадено FirsName=Borislav, то в скрипта се появява променлива $FirstName със стойност Borislav. Краткият стил е удобен, но изисква включена опцията register_global в конфигурационния файл php.ini. Този стил позволява допускането на грешки, които могат да доведат до несигурен код.

Ако е нужно да се различават начините за предаване на данни, то може да се използват асоциативните масиви $HTTP_POST_VARS и $HTTP_GET_VARS, ключовете на които са имената на предаваните променливи, а стойностите им са съответните стойности на тези

променливи. Например, ако двойката FirstName=Borislav е предадена по метод POST, то $HTTP_POST_VARS["fn"] = "Borislav".

```
//obrabotka.php

<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
       <title>Отговор на въвеждането!</title>
</head>
<body>

<?php

$fn = $HTTP_POST_VARS["fn"];
$ln = $HTTP_POST_VARS["ln"];
$email = $HTTP_POST_VARS["email"];
$comments = $HTTP_POST_VARS["comments"];

echo "Здравей $fn $ln <br>";
echo "Имате e-mail: $email <br>";
echo "Коментар: $comments";

?>
</body>
</html>
```

# 7.Функции

Функциите са основните структурни единици, които изграждат програмите на PHP. Всяка функция се състои от множество от оператори, оформени подходящо за да се използват като обобщено действие или операция. След като една функция бъде дефинирана, тя може да бъде изпълнявана многократно за различни входни данни.

## *7.1. Дефинирани от потребителя функции*

Функциите се обявяват с помощта на оператор function. Ако функцията приема аргументи, то те се обявяват като променливи във функцията:

```
function име_на_функцията (парам1, парам2, ..., парамN){
    Тяло на функцията
    return "стойност, връщана от функцията";
}
```

Ако функцията приема аргументи, то те се обявяват като променливи в името на фунцията. Променлива, съдържаща стойност, предадена чрез аргумент, се нарича параметър. Параметрите се отделят един от друг със запетая (,). Оператор return връща стойност на този оператор, който е извикал функцията. Не всички функции връщат стойност. Оператор return може да се използва за прекъсване изпълнението на функция без връщане на стойнсот, подобно на оператор break. Следва функция за преобразуване на инчове в сантиметри:

```
function InchToCm($Inches){
    return $Inches * 2.54;
}

echo (InchToCm(5)); // извежда 12.7
```

Функцията приема само един аргумент, числото за инчовете. В този пример 5 е аргумент, а $Inches е параметър.

## *7.2. Предаване на променливи на функции*

По подразбиране аргументите се предават по стойност. Това означава, че параметърът съдържа само копие от стойността. Ако стойността на параметъра се промени във функцията, то това не оказва влияние на променливите извън функцията:

```
function doublenumber($number){
    $number *= 2;
    return $number;
}

$cena = 70.0;
echo(doublenumber($cena));
echo($cena);
```

### 7.2.1. Предаване на аргументи чрез указател

В този случай всички промени в променливата във вътрешността на функцията ще повлияят на оригиналната променлива.

```
function doublenumber(&$number){
    $number *= 2;
    return $number;
}

$cena = 70.0;
echo(doublenumber($cena));
echo($cena);
```

Като параметър е предаден указател към променливата, а не самата стойност.

### 7.2.2. Аргументни стойности по подразбиране

PHP позволява дефинирането на функции с подразбиращи се стойности на аргументите. Такива аргументи се записват след всички останали аргументи във функцията. Стойността по подразбиране се задава в списъка с аргументи посредством оператор за присвояване (=).

```
function save_data($name, $position, $organisation="неизвестна"){
  echo "Потребител : $name <br>";
  echo "Длъжност: $position <br>";
  echo "Месторабота: $organisation <hr>";
}
save_data("Б. Панайотов", "лектор", "Университет");
```

```
save_data("А. Асенов", "програмист");
```

## 7.3. Извикване на функции

Функциите, които не изискват параметри се извикват с тяхното име, например:

```
printbold(){
    echo "<b>";
}

printbold();
```

Извикване на функция с параметри става като данните или името на променливата, съдържаща в себе си данните се поставят в скоби след името на функцията:

```
име_на_функция('параметър');
```

В този случай параметърът е низ, съдържащ единствено думата параметър. Следните изисквания също са коректни, в зависимост от функцията:

```
име_на_функция(4);
име_на_функция(9.99);
име_на_функция($variable);
```

В последния ред `$variable` може да е PHP променлива от произволен тип, включително и масив.

Декларацията на една функция се състои от типа за връщане, име и списък с аргументи – какво представлява всеки от тях и от какъв тип данни трябва да бъде. Тези три елемента се наричат прототип на функция. Следва прототип на функция fgets():

```
string fgets(int fp [, int length])
```

Тази функция се използва за четене на ред от файл. Важно е правилното интерпретиране на определените в прототипа спецификации. В този случай думата string преди името на функцията означава, че тази функция ще върне низ (стринг) с размер дължината на реда – 1 байт, прочетена от файла, указан чрез fp. Четенето се прекратява при достигане на length – 1 байт, символ newline или EOF. Параметрите fp и length са integer.

Квадратните скоби около length показват, че този параметър е незадължителен. На незадължителните параметри може да се дават стойности или да се игнорират, в който случай ще се използват техните стойности по подразбиране. Стойността по подразбиране на length=1024 байта.

При възникване на грешка се връща FALSE. Указателят на файла трябва да бъде коректен и трябва да указва файл, успешно отворен с функция fopen(), popen() или fsockopen().

От прототипа на тази функция следва, че следният фрагмент от код е валидно извикване на fopen():

```
<?php

$fp = fopen("danni.txt", "r");
while (!feof($fp)){
   $s = fgets($fp);
   echo $s;
}

?>
```

## 8.

031203083500.djvu
Mazurkevich 8.3 pdf
Koterov Samouchitel – Konstanti
Мазуркевич
орлов
котеров