SI6

PHP CLI
Structures de stockage et de contrôle
I/O & Fonctions
REGEX de base

PHP

* CLI : Console Line Interface (client)
* CGI : Common Gateway Interface (server)
* PHP : Hypertext Preprocessor (acronyme récursif)

Propriétés

* Utilise PHP.exe (interpréteur PHP)
* Interpréteur : compile + exécute / ligne
* PHP (non typé) basé sur le langage C
* POO en PHP existe !
* PHP.exe pièce d'une pile AMP.
* Invoquer PHP.exe avec un chemin
* Travaille en console
* Utilise PHP.GTK pour affiche graphique

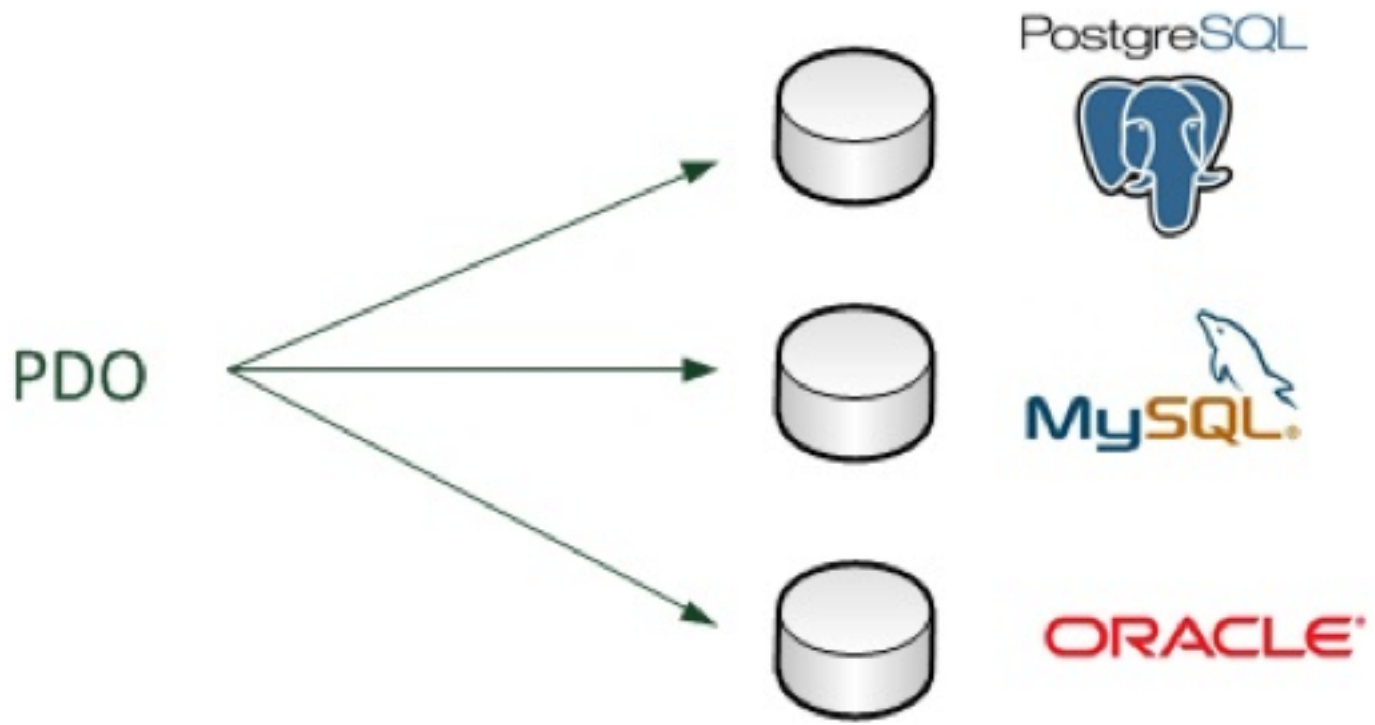
Configuration

* PHP.ini : configure l'interpréteur
* Chargement des extensions : MySQL, ..
* mysql_ : DEPRECATED
* mysqli_ : ACTIVATED
* PDO : permet la connexion à n'importe quelle base.

```
;extension=php_pdo_firebird.dll
;extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_odbc.dll
```

Configuration

* PDO : standard actuel en PHP (PHP Data Objects)



PHP CLI

* Attention dans les diapos qui suivent :
*
 : toutes les balises HTML
* $GET, $POST, $REQUEST : passage de paramètres

* NE FONCTIONNENT PAS EN CLI !

PHP CLI

* Sous Windows :
* Sans console :
  * Utiliser un lanceur .bat, et un script.php
* Avec console :
  * C:\...\php.exe –q script.php

PHP CLI

* Sous OSX, LINUX, UNIX

* #!/usr/bin/php
* Mettre une extension .sh ou .php ou autre
* Rendre le fichier exécutable (chmod 744 script.sh)
* Lancer le script : ./script.sh

Balises

* Pour signaler le code PHP :

```
<?php
// début du code
//...
//fin du code
?>
```

Commentaires

* Comme dans le langage C :

```
//une ligne de commentaires
/* Plusieurs
Ligne
De commentaires */
```

Paramètres

* $argc et $argv sont utilisables :

```php
<?php
  $name1 = $argv[1];
  echo $name1;
?>
```

```
C:\xampp\php\php.exe name.php Robby
```

```php
<?php
    var_dump($argc) //number of arguments passed
    var_dump($argv) //the arguments passed
?>
```

```
php script.php arg1 arg2 arg3
```

Paramètres

* Exemple :

```php
<?php
array_shift($argv);

if (empty($argv) || count($argv) != 3) {
  echo "Incorrect number of arguments\n";
}
else {
  $taskid = $argv[0];
  $file1 = file_get_contents($argv[1]);
  $file2 = fopen($argv[2],"w");
  echo $taskid . "\n" . $file1 . "\n" . $file2 . "\n";
}
```

I/O

* Affichage :
  * echo ‘’message \n”;
* Saisie :
  * $valeur=fgets(STDIN);

Inclusion

* Include(‘’monfich.php’’);

* Permet d’inclure un fichier externe au script

Les variables

* Une variable (structure de stockage) :
  * $nom=‘’Robert’’;
  * echo ‘’le nom $nom’’;
* Les variables ne sont pas typées, il n’y pas de déclaration de type

Opérations

* Exemples :

```php
<?php
$nombre = 2 + 4; // $nombre prend la valeur 6
$nombre = 5 - 1; // $nombre prend la valeur 4
$nombre = 3 * 5; // $nombre prend la valeur 15
$nombre = 10 / 2; // $nombre prend la valeur 5

// Allez on rajoute un peu de difficulté
$nombre = 3 * 5 + 1; // $nombre prend la valeur 16
$nombre = (1 + 2) * 2; // $nombre prend la valeur 6
?>
```

Opérations

* Exemples :

```
<?php
$nombre = 10 % 5; // $nombre prend la valeur 0 car la division
tombe juste
$nombre = 10 % 3; // $nombre prend la valeur 1 car il reste 1
?>
```

Structure de contrôle

* SI ... ALORS ... SINON ... FINSI :

```php
<?php
if ($autorisation_entrer)
{
    echo "Bienvenue petit Zéro. :o)";
}
else
{
    echo "T'as pas le droit d'entrer !";
}
?>
```

```php
<?php
$age = 8;

if ($age <= 12)
{
    echo "Salut gamin !";
}
?>
```

Structure de contrôle

* Opérateur de condition et booléen :

Symbole	Signification
==	Est égal à
>	Est supérieur à
<	Est inférieur à
>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

Mot-clé	Signification	Symbole équivalent
AND	Et	&&
OR	Ou	||

Structure de contrôle

* Exemple d'opérateur et condition :

```php
<?php
if ($age <= 12 AND $sexe == "garçon")
{
    echo "Bienvenue sur le site de Captain Mégakill !";
}
elseif ($age <= 12 AND $sexe == "fille")
{
    echo "C'est pas un site pour les filles ici, retourne jouer à la Barbie !";
}
?>
```

* SELON … cas :
* FINSELON

```php
<?php
$note = 10;

switch ($note) // on indique sur quelle variable on travaille
{
    case 0: // dans le cas où $note vaut 0
        echo "Tu es vraiment un gros Zér0 !!!";
    break;

    case 5: // dans le cas où $note vaut 5
        echo "Tu es très mauvais";
    break;

    case 7: // dans le cas où $note vaut 7
        echo "Tu es mauvais";
    break;

    case 10: // etc. etc.
        echo "Tu as pile poil la moyenne, c'est un peu juste…";
    break;

    case 12:
        echo "Tu es assez bon";
    break;

    case 16:
        echo "Tu te débrouilles très bien !";
    break;

    case 20:
        echo "Excellent travail, c'est parfait !";
    break;

    default:
        echo "Désolé, je n'ai pas de message à afficher pour cette
note";
}
?>
```

Structure de contrôle

* Ternaire (utilisée dans d'autres langages) :

```php
<?php
$age = 24;

if ($age >= 18)
{
 $majeur = true;
}
else
{
 $majeur = false;
}
?>
```

```php
<?php
$age = 24;

$majeur = ($age >= 18) ? true : false;
```

Structure de contrôle

* TANT QUE … FINTANTQUE :

```php
<?php
while ($continuer_boucle == true)
{
    // instructions à exécuter dans la boucle
}
?>
```

Structure de contrôle

* POUR ... FINPOUR :

```php
<?php
for ($nombre_de_lignes = 1; $nombre_de_lignes <= 100;
$nombre_de_lignes++)
{
    echo 'Ceci est la ligne n°' . $nombre_de_lignes . '<br />';
}
?>
```

Structure de contrôle

* REPETER … JUSQU'À :

```php
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

```php
<?php
do {
    if ($i < 5) {
        echo "i is not big enough";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
   echo "i is ok";

    /* process i */

} while (0);
?>
```

Les fonctions

* Longueur d'une chaîne :

```php
$longueur = strlen($phrase);
```

* Recherche et remplacement :

```php
$ma_variable = str_replace('b', 'p', 'bim bam boum');
```

* Mélanger les lettres :

```php
$chaine = str_shuffle($chaine);
```

* Ecrire en minuscule :

```php
$chaine = strtolower($chaine);
```

Les fonctions

* Les dates :

```php
$jour = date('d');
$mois = date('m');
$annee = date('Y');

$heure = date('H');
$minute = date('i');
```

```php
<?php
$annee = date('Y');
echo $annee;
?>
```

Les fonctions

* Ecrire une fonction : (
 = HTML !!)

```php
<?php
function DireBonjour($nom)
{
    echo 'Bonjour ' . $nom . ' !<br />';
}
```

* Utiliser la fonction : (pas de type de retour !)

```php
DireBonjour('Marie');
DireBonjour('Patrice');
DireBonjour('Edouard');
DireBonjour('Pascale');
DireBonjour('François');
DireBonjour('Benoît');
DireBonjour('Père Noël');
?>
```

Les fonctions

* Avec plusieurs paramètres et un retour :

```php
<?php
// Ci-dessous, la fonction qui calcule le volume du cône
function VolumeCone($rayon, $hauteur)
{
   $volume = $rayon * $rayon * 3.14 * $hauteur * (1/3); // calcul du volume
   return $volume; // indique la valeur à renvoyer, ici le volume
}
```

* Utilisation :

```php
$volume = VolumeCone(3, 1);
echo 'Le volume d\'un cône de rayon 3 et de hauteur 1 est de ' . $volume;
?>
```

Les tableaux

* Tableau à une dimension (structure de stockage) :

```php
<?php
// La fonction array permet de créer un array
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique',
'Benoît');
?>
```

* Utilisation (commence à ZERO) :

```php
<?php
$prenoms[0] = 'François';
$prenoms[1] = 'Michel';
$prenoms[2] = 'Nicole';
?>
```

Les tableaux

* Tableaux associatifs (clé/valeur) :

```php
<?php
// On crée notre array $coordonnees
$coordonnees = array (
    'prenom' => 'François',
    'nom' => 'Dupont',
    'adresse' => '3 Rue du Paradis',
    'ville' => 'Marseille');
?>
```

* Utilisation :

```php
$coordonnees['prenom'] = 'François';
$coordonnees['nom'] = 'Dupont';
$coordonnees['adresse'] = '3 Rue du Paradis';
$coordonnees['ville'] = 'Marseille';
?>
```

```php
<?php
echo $coordonnees['ville'];
?>
```

Les tableaux

* Parcours d'un tableau :

```php
<?php
// On crée notre array $prenoms
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique',
'Benoît');

// Puis on fait une boucle pour tout afficher :
for ($numero = 0; $numero < 5; $numero++)
{
    echo $prenoms[$numero] . '<br />'; // affichera $prenoms[0],
$prenoms[1] etc.
}
```

Les tableaux

* Parcours d'un tableau :

```php
<?php
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique',
'Benoît');

foreach($prenoms as $element)
{
    echo $element . '<br />'; // affichera $prenoms[0], $prenoms[1]
etc.
}
?>
```

```php
<?php
$coordonnees = array (
    'prenom' => 'François',
    'nom' => 'Dupont',
    'adresse' => '3 Rue du Paradis',
    'ville' => 'Marseille');

foreach($coordonnees as $element)
{
    echo $element . '<br />';
}
?>
```

Les tableaux

* Parcours et affichage clé/valeur :

```php
<?php
$coordonnees = array (
    'prenom' => 'François',
    'nom' => 'Dupont',
    'adresse' => '3 Rue du Paradis',
    'ville' => 'Marseille');

foreach($coordonnees as $cle => $element)
{
    echo '[' . $cle . '] vaut ' . $element . '<br />';
}
?>
```

Les tableaux

* Existence d'une CLE dans un tableau :

```php
<?php
$coordonnees = array (
    'prenom' => 'François',
    'nom' => 'Dupont',
    'adresse' => '3 Rue du Paradis',
    'ville' => 'Marseille');

if (array_key_exists('nom', $coordonnees))
{
    echo 'La clé "nom" se trouve dans les coordonnées !';
```

Les tableaux

* Existence d'une VALEUR dans un tableau :

```php
<?php
$fruits = array ('Banane', 'Pomme', 'Poire', 'Cerise', 'Fraise',
'Framboise');

if (in_array('Myrtille', $fruits))
{
    echo 'La valeur "Myrtille" se trouve dans les fruits !';
}

if (in_array('Cerise', $fruits))
{
    echo 'La valeur "Cerise" se trouve dans les fruits !';
}
?>
```

Les tableaux

* Récupérer la CLE d'une VALEUR (indice ici) :

```php
<?php
$fruits = array ('Banane', 'Pomme', 'Poire', 'Cerise', 'Fraise',
$position = array_search('Fraise', $fruits);
echo '"Fraise" se trouve en position ' . $position . '<br />';

$position = array_search('Banane', $fruits);
echo '"Banane" se trouve en position ' . $position;
?>
```

* Fonctionne aussi pour les tableaux associatifs (clé).

Les bases de données

* Connexion en PDO (new) :

```php
<?php
$bdd = new PDO('mysql:host=localhost;dbname=test', 'root', '');
?>
```

* Attention on est en objet ici (new) :

```php
<?php
try
{
 $bdd = new PDO('mysql:host=localhost;dbname=test', 'root', '');
}
catch (Exception $e)
{
        die('Erreur : ' . $e->getMessage());
}
?>
```

Les bases de données

* Une requête :

```php
<?php
$reponse = $bdd->query('SELECT * FROM jeux_video');
?>
```

Les bases de données

* Un exemple complet (ATTENTION aux
) :

```php
<?php
try
{
 $bdd = new PDO('mysql:host=localhost;dbname=test', 'root', '');
}
catch(Exception $e)
{
        die('Erreur : '.$e->getMessage());
}

$reponse = $bdd->query('SELECT nom FROM jeux_video');

while ($donnees = $reponse->fetch())
{
 echo $donnees['nom'] . '<br />';
}

$reponse->closeCursor();

?>
```

Les bases de données

* Pour la clause WHERE

```
<?php
$reponse = $bdd->query('SELECT nom FROM jeux_video WHERE
possesseur=\'Patrick\'');
?>
```

* Les requêtes préparées :

```
<?php
$req = $bdd->prepare('SELECT nom FROM jeux_video WHERE possesseur =
?');
$req->execute(array($_GET['possesseur']));
?>
```

Les bases de données

* Pour :
  * INSERT
  * UPDATE
  * DELETE

* Vous pouvez utiliser les requêtes préparées

Les expressions régulières

* REGEX : REGular EXpression
* Permet de faire des recherches dans des chaînes
* Permet de remplacer des valeurs (pas traité)
* Très complexe (au bout d'un moment)
* POSIX : standard PHP
* PCRE : plus rapide que POSIX (REGEX perl)

Expression régulière

* Recherche simple : (preg_match)

```php
<?php
if (preg_match("#guitare#", "J'aime jouer de la guitare."))
{
    echo 'VRAI';
}
else
{
    echo 'FAUX';
}
?>
```

Chaîne	Regex	Résultat
J'aime jouer de la guitare.	#guitare#	VRAI
J'aime jouer de la guitare.	#piano#	FAUX

Expression régulière

* Gestion de la casse dans une recherche (option i) : (preg_match)

Chaîne	Regex	Résultat
J'aime jouer de la guitare.	`#Guitare#`	FAUX
J'aime jouer de la guitare.	`#GUITARE#`	FAUX

Chaîne	Regex	Résultat
J'aime jouer de la guitare	`#Guitare#i`	VRAI
Vive la GUITARE !	`#guitare#i`	VRAI
Vive la GUITARE !	`#guitare#`	FAUX

Expression régulière

* Le OU dans une recherche (|) : (preg_match)

Chaîne	Regex	Résultat
J'aime jouer de la guitare.	`#guitare|piano#`	VRAI
J'aime jouer du piano.	`#guitare|piano#`	VRAI
J'aime jouer du banjo.	`#guitare|piano#`	FAUX
J'aime jouer du banjo.	`#guitare|piano|banjo#`	VRAI

Expression régulière

* Recherche en début (^) et fin de chaîne ($) : (preg_match)

Chaîne	Regex	Résultat
Bonjour petit zéro	#^Bonjour#	VRAI
Bonjour petit zéro	#zéro$#	VRAI
Bonjour petit zéro	#^zéro#	FAUX
Bonjour petit zéro !!!	#zéro$#	FAUX

Expression régulière

* Recherche avec une classe [ ] : (preg_match)

Chaîne	Regex	Résultat
La nuit, tous les chats sont gris	`#gr[aoi]s#`	VRAI
Berk, c'est trop gras comme nourriture	`#gr[aoi]s#`	VRAI
Berk, c'est trop gras comme nourriture	`#gr[aoi]s$#`	FAUX
Je suis un vrai zéro	`#[aeiouy]$#`	VRAI
Je suis un vrai zéro	`#^[aeiouy]#`	FAUX

* Intervalle dans une classe [-] : (preg_match)

Chaîne	Regex	Résultat
Cette phrase contient une lettre	`#[a-z]#`	VRAI
cette phrase ne comporte ni majuscule ni chiffre	`#[A-Z0-9]#`	FAUX
Je vis au 21^{e} siècle	`#^[0-9]#`	FAUX
Une balise de titre HTML	`#<h[1-6]>#`	VRAI

Expression régulière

* Dans une classe [ ], le caractère ^:
  * Exprime la négation : #[^0-9]#
  * Il n'exprime pas le début de la chaîne
  * En dehors de la chaîne : #^[^a-z]#
  * Il exprime le début de la chaîne

* Idem pour $: #[^aeiouy]$#
  * Négation pour la fin de chaîne.

Expression régulière

* Quantificateurs : ?, +, * : (preg_match)

- ? (point d'interrogation) : ce symbole indique que la lettre est facultative. **Elle peut y être 0 ou 1 fois**.
  Ainsi, `#a?#` reconnaît 0 ou 1 « a » ;
- + (signe plus) : la lettre est obligatoire. **Elle peut apparaître 1 ou plusieurs fois**.
  Ainsi, `#a+#` reconnaît « a », « aa », « aaa », « aaaa », etc. ;
- * (étoile) : la lettre est facultative. **Elle peut apparaître 0, 1 ou plusieurs fois**.
  Ainsi, `#a*#` reconnaît « a », « aa », « aaa », « aaaa », etc. Mais s'il n'y a pas de « a », ça fonctionne aussi !

* Exemple :

```
#bor?is#
```

Ce code reconnaîtra « boris » et « bois » !

Expression régulière

* Quantificateurs avec parenthèses () : (preg_match)

```
#Ay(ay)*#
```

Ce code reconnaîtra « Ay », « Ayay », « Ayayay », « Ayayayay »

Chaîne	Regex	Résultat
eeeee	#e+#	VRAI
ooo	#u?#	VRAI
magnifique	#[0-9]+#	FAUX
Yahooooooo	#^Yaho+$#	VRAI
Yahooooooo c'est génial !	#^Yaho+$#	FAUX
Blablablablabla	#^Bla(bla)*$#	VRAI

Expression régulière

Les accolades { } : (preg_match)

- `{3}` : si on met juste un nombre, cela veut dire que la lettre (ou le groupe de lettres s'il est entre parenthèses) doit être répétée **3 fois exactement**.
  `#a{3}#` fonctionne donc pour la chaîne « aaa ».
- `{3,5}` : ici, on a plusieurs possibilités. On peut avoir la lettre de **3 à 5 fois**.
  `#a{3,5}#` fonctionne pour « aaa », « aaaa », « aaaaa ».
- `{3,}` : si vous mettez une virgule, mais pas de 2e nombre, ça veut dire qu'il peut y en avoir jusqu'à l'infini. Ici, cela signifie « **3 fois ou plus** ».
  `#a{3,}` fonctionne pour « aaa », « aaaa », « aaaaa », « aaaaaa », etc. Je ne vais pas tous les écrire, ça serait un peu long.

Chaîne	Regex	Résultat
eeeee	`#e{2,}#`	VRAI
Blablablabla	`#^Bla(bla){4}$#`	FAUX
546781	`#^[0-9]{6}$#`	VRAI

Conclusion

* Les REGEX c'est très long (il faudrait un cours)
* Les fonctions PHP sont nombreuses.
* Le PHP est très documenté
* De nombreux exemples disponibles