Le 28.03.2022

Module PHP

https://www.w3schools.com/php/php_syntax.asp

Pour exécuter du code PHP, on doit se trouver dans le répertoire www (environnement serveur géré par Apache), le code PHP sera interprété et le résultat généré (le plus souvent du HTML dans une architecture MVC) est envoyé au client.
Pour écrire du code PHP, on doit mettre les <? PHP et ?>
On peut alterner entre PHP et HTML

```
<? PHP
Echo "<p>Mon 1er cours PHP</p>";
?>

<h1>Test de PHP</h1>

<? PHP
Echo "<p>Bonne chance</p>";
?>
```

Les variables
En PHP les variables commencent par le signe $
Pour afficher le contenu d'une variable, on peut utiliser var_dump (var), cette fonction est utile pour le débogage.

```
$nom="HIDRI";
Var_dump($nom);
```

Si on veut afficher le contenu de la variable, on utilise echo

```
$nom = "HIDRI";
$prénom = "Raye";
Echo $nom. " " . $prénom;
```

NB : pour la concaténation, on utilise le caractère. (Point).

Les constantes
https://www.w3schools.com/php/php_constants.asp

En dehors d'une classe, les constantes sont définies comme suit :

```
Define ("NOM", valeur);
```

NB : par convention, les noms des constantes sont toujours en maj.

```
Define ("VILLE", "Calais");

$département = "Pas-de-Calais";
Define ("DEPARTEMENT", $département);

Echo VILLE." -- ". DEPARTEMENT."<Br/>";
```

Les commentaires

```
// une seule ligne

# une seule ligne

/*
Bloc de commentaires
Multi-lignes
*/
```

Types de données

https://www.w3schools.com/php/php_datatypes.asp

Les différents types de données gérés par PHP :

- **String**
- **Integer**
- **Float**
- **Booléen**
- **Array**
- **Object**
- NULL
- Resource

Les chaines de caractères

https://www.w3schools.com/php/php_string.asp

```
Exp de fonctions de chaines de caractères :
echo strlen("Calais"); // 6
echo str_word_count("Pas de calais"); // 3

Liste complete des fonctions de chaines de caractères :
https://www.w3schools.com/php/php_ref_string.asp
```

Les entiers

https://www.w3schools.com/php/php_numbers.asp

```
En PHP, on a deux types de nombres : entiers (int ou integer) et les decimaux
(float), et ils sont considérés comme deux types différents.
Transtypage de variables (casting en anglais) : (type)$var;
Exemples de transtypage de nombres :
Cast int to string
<?php
```

```php
$x = 23456;
$string_cast = (string)$x;
var_dump($string_cast);

echo "<br>";

Cast string to int
$x = "23456.768";
$int_cast = (int)$x;
Echo $int_cast;

//les langages faiblement typés, le transtypage est à utiliser avec prudence
```

Les fonctions mathématiques

https://www.w3schools.com/php/php_math.asp

Php propose un ensemble de fonctions mathématiques pour traiter les données numériques.

Il faut accorder une attention particulière aux fonctions d'arrondissement surtout quand on travaille dans le domaine scientifique ou financier, car toute valeur décimale compte.
Liste complete des fonctions mathématiques

https://www.w3schools.com/php/php_ref_math.asp

Les tableaux

https:*www.w3schools.com/php/php_arrays.asp*
Déclaration d'un tab en PHP

Syntaxe 1

```php
$tab = array (1,2,3,4,5);
var_dump($tab);

// syntaxe 2
$tab2 =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var_dump($tab2);
La syntaxe 2 est la plus recommandée.
En PHP, il existe trois types de tableaux :
 •    Tableaux indexés - Tableaux avec un index numérique
(tab et tab2, dans l'exemple precédent), les index commencent à 0.
$tab[0]; // 1

$eleves = ["Paul", "Rémy", "Amir", "Benjamin", "olivier"];
```

```php
$eleves[3]; // Benjamin  // acces
$eleves[3] = "Adrien";  // modif
•    Tableaux associatifs - Tableaux avec des clés nommées
 Déclaration de tab associatif (clé => valeur)
$age = ["Peter" => 35, "Ben" => 37, "Joe" => 43];
var_dump($age["Peter"]);  // 35 :  acces
$age["Peter"] = 30;       //    : modif
// NB : les clés sont sensibles à la casse.
// •    Tableaux multidimensionnels - Tableaux contenant un ou plusieurs
// tableaux


//NB : les clés sont sensibles à la casse.
//•    Tableaux multidimensionnels - Tableaux contenant un ou plusieurs
tableaux
// tab indexé qui contient des tab imbriqués (4) qui sont indexés eux aussi
$cars = array (
      array ("Volvo", 22, 18),
      array ("BMW", 15, 13),
      array("Saab", 5, 2),
      array("Land Rover", 17, 15)
);
var_dump($cars[0][0]); // volvo
$cars[0][0] = "Audi"; //    modif
// tab indexé, contient 4 cases dont chacune contient un tab associatif
$voitures = array(
  array("marque" => "Volvo", "vendus" => 22, "reste" => 18),
  array("marque" => "BMW", "vendus" => 15, "reste" => 13),
  array("marque" => "Saab", "vendus" => 5, "reste" => 2),
  array("marque" => "Land Rover", "vendus" => 17, "reste" => 15)
);
// afficher la quantité vendue de la marque Land Rover
echo $voitures[3]["vendus"];

// on a vendu 20 voitures supplémentaires de la marque saab
$voitures[2]["vendus"] += 20;  // eq $voitures[2]["vendus"] =
$voitures[2]["vendus"] +  20;
echo $voitures[2]["vendus"]; //
//Liste complete des fonctions de tableaux
https://www.w3schools.com/php/php_ref_array.asp
```

```
Les dates
https://www.w3schools.com/php/php_date.asp
```

Le 29-03-2022

https://www.php.net/datetime.format

PHP propose des fonctions qui permettent de gérer les dates.
date(format,timestamp) : cette fonction permet de créer une date formatée en fonction du paramètre format (obligatoire) et du param timestamp (optionnel).

En absence du param timestamp, la fonction date retourna la date et l'heure actuelle (celle du serveur qui peut être différente de celle du client).

Les formats de date :

- *d - Représente le jour du mois (01 à 31)*
- *m - Représente un mois (01 à 12)*
- *Y - Représente une année (en quatre chiffres)*
- *l (minuscule 'L') - Représente le jour de la semaine*
- *H - Format 24 heures d'une heure (00 à 23)*
- *h - format 12 heures d'une heure avec des zéros non significatifs (01 à 12)*
- *i - Minutes avec des zéros non significatifs (00 à 59)*
- *s - Secondes avec des zéros non significatifs (00 à 59)*
- *a - Minuscule Ante meridiem et Post meridiem (am ou pm)*

```php
echo "Aujourd'hui, on est le " . date("Y-m-d h:i:s") . "<br>";
```

mktime(hour, minute, second, month, day, year) : permet de définir un timestamp en fonction des param de date et heure qui lui sont passés.

On peut chercher le timestamp d'une date donnée sans passer par la date elle-même strtotime(time, now);

```php
echo (strtotime("+1 week") . "<br>");
```

Autres exemples :
https://www.w3schools.com/php/func_date_strtotime.asp

On peut ajuster l'horaire de l'application en fonction du fuseau horaire de l'utilisateur en utilisant date_default_timezone_set('continent/ville') :

```php
date_default_timezone_set('Europe/Paris');
```

Liste des fuseaux horaires :
https://www.php.net/manual/fr/timezones.php

Liste complète des fonctions de date et heure

https://www.w3schools.com/php/php_ref_date.asp

Les var superglobales

Les variables superglobales PHP sont :

- $GLOBAUX
- $_SERVEUR

- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

Les param GET sont envoyés dans l'URL et sont transmises en clair

http://localhost/calais/cours.php?nom=HIDRI&prenom=ryan

Alors que les param POST sont envoyés séparément et non visibles dans l'URL.

NB : si on passe par un formulaire, la méthode POST est la plus recommandée, il faut la mettre explicitement dans l'attribut action de la balise form (si on ne met rien, par défaut c'est GET)

Exemple de code de transmission de données via un formulaire

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
    Nom: <input type="text" name="nom"> <br><br>
    prenom: <input type="text" name="prenom">
    <input type="submit">
</form>

<?php
echo $_SERVER["REQUEST_METHOD"] . "  --  ";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    echo $_POST['prenom'] . " " . $_POST['nom'];
}
?>
```

Les cookies

https://www.w3schools.com/php/php_cookies.asp

Pour créer un cookie, on utilise setcookie("nom","valeur", durée, "chemin_de_visibilité") et une fois pour toutes

Cookie qui est visible sur tt le site (/) pendant un an
une fois le coolie créé, il sera visible dans tous les scripts du chemin et pendant la période qui ont été spécifiés lors de la création

```
// tester l'existance du cookie (on utilise la fonction isset)
```

```php
if (isset($_COOKIE['theme'])) {
  echo $_COOKIE['theme'];
} else {
  echo "cookie inexistant";
}
```

Pour modifier un cookie, on utilise la meme fonction que la création en gardant le même nom lui introduisant de nouvelles valeurs.

```php
setcookie("theme", "bleu", time() + (86400 * 365), "/");
```

Pour supprimer un cookie, il suffit d'utiliser la fonction

```php
unset(nom_du_cookie)
unset($_COOKIE['theme']);
```

Les sessions

Pour démarrer une session, on utilise

```php
session_start();
```

NB : La fonction session_start()doit être la toute première chose dans votre document, avant toute balise HTML.

```php
//pour démarrer une session, on utilise
session_start();
//NB : La fonction  session_start()doit être la toute première chose dans
votre document. Avant toute balise HTML.
// création de var de session mail
$_SESSION["mail"] = "ryan@gmail.com";

// modif de var de session
$_SESSION["mail"] = "admin@gmail.com";

//tester  l'existance d'une var de session
if( isset($_SESSION ['mail'])) {
  echo $_SESSION["mail"];
}
//Vider la session de toutes ses var
session_unset();
//détruire la session
session_destroy();
```

Les opérateurs

Il existe différents types d'opérateurs qui permettent de réaliser les opérations sur les variables et les valeurs

- Opérateurs arithmétiques
- Opérateurs d'affectation
- Opérateurs de comparaison
- Opérateurs d'incrémentation/décrémentation

- Opérateurs logiques
- Opérateurs de chaîne
- Opérateurs de tableau
- Opérateurs d'affectation conditionnelle

Liste complète des opérateurs de chaque type
https://www.w3schools.com/php/php_operators.asp

print et echo
https://www.w3schools.com/php/php_echo_print.asp

echo et print permettent d'afficher des informations sur l'écran. Elles admettent une chaine de caractère à afficher ou le contenu d'une variable.

```php
// La chaine peut contenir du HTML.
echo "<h2>PHP est fantastique!</h2>";
// La concaténation se fait avec le symbole . (point) et on peut concaténer des var entre elles ou des var et des chaines.
$ch1 = "Bonjour";
$ch2 = "Monsieur";

echo $ch1." ".$ch2;
```

if / else

https://www.w3schools.com/php/php_if_else.asp

```php
//l'instruction if permet de faire un traitement donné si la condition est vraie.
$x=10;
if($x == 10) {
  echo "la var 'x' a la bonne valeur ";
}
//On peut utiliser plusieurs cas possibles avec les traitements correspondants
$x=10;
$y = 20;
if($x == $y) {
  echo "x est = à y ";
}
elseif ($x > $y) {
  echo "x est > à y ";
}
else {
  echo "x est < à y ";
}
```

Le 30.03.2022

Switch

https://www.w3schools.com/php/php_switch.asp

L'instruction switch sert à exécuter des traitements correspondants à différents cas possibles (traitement par cas) selon la valeur d'une var.

Syntaxe générale

```
switch (var) {
  case val1:
    // trait 1
    break;
  case val2:
    // trait 2
    break;
  case val3:
    // trait 3
    break;
    etc ...
  default:
    // trait si aucun des cas précèdent ne correspond
}
```

Exp :

```
 // 2 : moto / 4 : voiture / 6 : camion
$nbRoues = 4;

switch ($nbRoues) {
  case 2:
    echo "Moto";
    break;
  case 4:
    echo "Voiture";
    break;
  case 6:
    echo "Camion";
    break;

  default:
    echo "Type inconnu";
}
```

Autre variante de switch

```
$x=  90;

switch (true) {
  case $x>=0 && $x<=10:
    echo "intervalle [0-10]";
```

```
    break;
  case $x >= 11 && $x <= 20:
    echo "intervalle [11-20]";
    break;
  case $x >= 21 && $x <= 30:
    echo "intervalle [21-30]";
    break;

  default:
    echo "intervalle [31-100]";
}
```

boucles

https://www.w3schools.com/php/php_looping.asp

1- boucle while

```
// initialisation
while ( condition ) {
  // trait

  // incrémentation
}
//NB : l'initialisation se fait à l'extérieur de la boucle et avant.
//L'incrémentation se fait à l'intérieur de la boucle et en dernier lieu.

Exp :
// initialisation
$i = 1;
//condition
while ($i <= 5) {
  // trait
  echo $i . "<br>";

  // incrementation
  $i++;
}
```

2- do … while

```
//Cette boucle permet d'exécuter le code au moins une fois, puis vérifie la condition, si elle est vraie, on continue à boucler sinon on quitte la boucle.
$i = 6;

do {
  echo $i . "<br>";
  $i++;
```

```
} while ($i <= 5);
```

Dans l'exemple précèdent, la boucle est exécutée puis le test de la condition est fait (la condition n'est pas vérifiée, donc on quitte)(la boucle est exécutée au moins une fois malgré que la condition n'est pas respectée dès le départ.

NB : le test de la condition se fait après le traitement

La boucle est exécutée au moins une fois.

Comparatif entre while et do … while

	Verif condition	Nb itérations
while	Avant le trt	0,N
do ...while	Apres le trt	1,N (au moins 1 fois)

Exp :

```
echo "avant while<br>";
$i = 6;
while ($i <= 5) {
  echo $i . "<br>";
  $i++;
}
echo "après while<br><br>";

echo "avant do while<br>";
do {
  echo $i . "<br>";
  $i++;
} while ($i <= 5);
echo "après do while<br>";
```

3- la boucle for

```
//Syntaxe
for (initialisation; condition; incrementation) {
  // trait
}
Exp :
for ($i=1; $i<=10; $i++) {
 echo $i."<br>";
}
```

4- boucle foreach

Elle boucle obligatoirement sur un tableau

Elle parcoure tous les éléments du tableau dans l'ordre, et pour chacun, elle effectue son traitement.

Syntaxe : tab indexé

```
foreach ($tab as $element) {
  // trait
}
Exemple :
```

```php
$couleurs = ["rouge", "vert", "bleu", "jaune"];

foreach ($couleurs as $couleur) {
  echo "$couleur <br>";
}
//Syntaxe : tab associatif
foreach ($tab as $cle => $valeur) {
  // trait
}


//Exp
$ages = ["Peter" => "35", "Ben" => "37", "Joe" => "43"];
foreach ($ages as $nom => $age) {
  echo "$nom : $age<br>";
}
```

** les instructions break et continue

Break : permet d'interrompre l'exécution d'une boucle à une condition donnée.

Continue : permet la reprise de la boucle à l'itération suivante si on rencontre une condition donnée.

Exemple avec break

```php
// afficher 1 - 2 - 3
for ($x = 1; $x < 10; $x++) {
  if ($x == 4) {
    break;
  }
  echo "nombre : $x <br>";
}
```

Exemple avec continue

```php
// nombres de 1 à 10 SAUF le 4
for ($x = 1; $x <= 10; $x++) {
  if ($x == 4) {
    continue;
  }
  echo "nombre : $x <br>";
}
```

NB : dans l'exemple ci-dessus, lorsqu'on arrive à une itération où x == 4, on arrête cette itération, puis la boucle continue avec l'itération suivante.

Les fonctions

https://www.w3schools.com/php/php_functions.asp

syntaxe :

```php
function nom(param, param2, ...,  paramN) {
  // trait
```

```php
}
NB : les param ne sont pas forcément obligatoires.
//Une fonction peut retourner ou pas une valeur, si c'est le cas, il faut utiliser le mot clé return.
//Une fonction n'est exécutée qu'au moment de SON APPEL (l'implémentation à elle seule ne suffit pas).
// l'implémentation
function salutation()
{
  echo "Bonjour monsieur";
}
// exécution
salutation ();
//on peut passer des valeurs par défaut aux paramètres, lors de l'appel de la fonction, les param par defaut peuvent etre omis.
// fonction avec valeur par defaut
function afficherAge($age = 30)
{
  echo "votre age est $age";
}

afficherAge(); // affiche 30 - la veleur par defaut dans l'implémentation
afficherAge(60); // affiche 60, la valeur par defaut est écrasée par la valeur passée en param
//NB : les param par defaut doivent etre mis à la fin de la liste des param.
//Typage des param
// param sans typage (strict_types desactivé)
function addNumbers( $a, $b)
{
  return $a + $b;
}
// resultat 15 + notice d'erreur car il s'attend à avoir un nombre alors qu'il
// recoit une chaine,  tout de meme il fait le calcul en convertissant "10 days" en entier 10 et s'il
//n'arrive pas à le convertir il le considere comme 0
echo addNumbers(5, "10 days"); //  15 (5 + 10)
echo addNumbers(5, "dd 10 days"); //  5 ( 5 + 0)

// param avec typage (strict_types desactivé)
function addNumbers2(int $a, int $b)
{
  return $a + $b;
}
// resultat 15 + notice d'erreur car il s'attend à avoir un nombre alors qu'il
// recoit une chaine,  tout de meme il fait le calcul en convertissant "10 days" en entier 10 et s'il
//n'arrive pas à le convertir il déclenche une erreur
echo addNumbers2(5, "10 days"); // 15 +  notice
echo addNumbers2(5, "dd10 days"); // erreur fatale
```

```php
//NB : si on veut activer le transtypage strict, il faut que
declare(strict_types=1) soit la toute 1ere instruction
declare(strict_types=1);
// param avec typage  (strict_types activé)
function addNumbers3(int $a, int $b)
{
  return $a + $b;
}
// erreur fatal avec arret de script (aucun résultat)
 echo addNumbers3(5, "10 days"); // erreur fatale
 echo addNumbers2(5, "dd10 days");  // erreur fatale
//Typage du retour
//On peut typer la valeur du retour, si ce type n'est pas respecté, ca
declenche une erreur fatale
declare(strict_types=1); // strict types activé
function addNumbers(float $a, float $b): float
{
  return $a + $b;
  // return "az" . ($a + $b); // erreur fatale : retourne une chaine au lieu
d'un float attendu
}
echo addNumbers(1.2, 5.2);

//Param par référence
Il est possible de passer une var par référence comme param à une fonction,
dans ce cas, la modif de la valeur de cette var est persistante meme apres
avoir quitté la fonction.
//passage de param par valeur
function ajoutValeur($val)
{
  $val += 5;
}
$a = 2;
echo $a . " : avant fonction <br>";
ajoutValeur($a); //
echo $a . " : apres fonction <br>"; // 2

echo "<hr>";

// passage de param par reference
function ajoutReference(&$ref)
{
  $ref += 5;
}
$b = 7;
echo $b." : avant fonction <br>";
ajoutReference($b);
echo $b . " : apres fonction <br>"; // 12
//----------------
```

ex de PHP et algo

EX 1 : écrire une fonction qui affiche la table de multiplication au format suivant :

//multiplication de 1 : 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 – 10 (1er message en gras, pas de – apres le chiffre 10).

```php
function multiplication()
{
  for ($i = 1; $i <= 10; $i++) {
    echo "<b>Table de multiplication de " . $i . " : </b>";
    for ($j = 1; $j <= 10; $j++) {
      echo $i * $j;
      echo $j === 10 ? "<br>" : " - ";
    }
  }
}

multiplication();
```

Ex 2 : écrire une fonction qui permet d'afficher une phrase donnée sous forme de mot par ligne

```php
function motLigne($phrase)
{
 return str_replace(" ", "<br/>", $phrase);
}
$chaine= "Lorem ipsum dolor sit amet consectetur, adipisicing elit.";
echo motLigne($chaine);
```

ex 3 : meme ex que le précédent, mais sans utiliser str_replace

```php
//1ere solution
function motLigne2 ($phrase) {
  $chaine="";
  $tab = explode(" ", $phrase);
  for($i=0;$i<count($tab);$i++) {
    $chaine.= $tab[$i]."<br>";
  }
  return $chaine;
}
$ch="Lorem ipsum dolor sit amet consectetur adipisicing elit. Commodi asperiores ";
echo motLigne2($ch);
//solution meilleure

function motLigne2 ($phrase) {
  return implode("<br>", explode(" ", $phrase));
```

```php
}

//le code ci-dessus est équivalent à

// function motLigne2($phrase)
// {
//   $tab = explode(" ", $phrase);
//   $chaine = implode("<br>", $tab);
//   return $chaine;
// }


ou bien
function motLigne2($phrase)
{
  return preg_replace('/ /',"<br>", $phrase  );
}

Ex 4 : même ex mais sans utiliser aucune fonction prédéfinie

//Exceptionnellement on peut utiliser stlen()

function motLigne3($phrase)
{
  $ch = "";
  for ($i = 0; $i < strlen($phrase); $i++) {
    $ch .= $phrase[$i] === ' ' ? "<br>" :  $phrase[$i];
  }
  return $ch;
}


$ch = "Lorem ipsum dolor sit amet consectetur adipisicing elit. Commodi
asperiores veritatis temporibus libero quaerat";
echo motLigne3($ch);

Ex 5 : calculer la somme des nombres d'un tab (admettons que la taille du tab
> 0)

function somme($tab)
{
  return array_sum($tab);
}
$nombres = [1,2, 3, 4, 5, 6, 7, 8, 9, 10];
echo somme($nombres);


Ex 6 : même ex sans utiliser la fonction array_sum()
```

```php
function somme2($tab)
{
  $somme = 0;
  for ($i = 0; $i < count($tab); $i++) {
    $somme += $tab[$i];
  }
  return $somme;
}

$nombres = [1,2,3,4,5,6,7,8,9,10];
echo somme2($nombres);
```

Exercices complémentaires

Ex 7 : calculer la somme des éléments d'un tab pour un intervalle donné (position inf et position sup, on utilise les positions naturelles).
// Si le tab est vide ou position inf > position sup, on retourne false.
Ex 8 : dans un tournoi, on a un certain nombre d'équipes, chacune doit confronter toutes les autres une fois.
// Exp : on a 3 équipes A, B et C, il faut avoir les combinaisons AB, AC, BC sans les doublons comme CA BA ou CB

Correction

Ex :7

```php
function sommeIntervalle($tab, $inf, $sup)
{

  // cond non respectées : Si le tab est vide ou position inf > position sup,
  //on retourne false. (deux conditions avec meme trt)
  if ((count($tab) === 0)  || ($inf > $sup)) {
    return false;
  }

  // cond respectées
  else {
    // parcours du tab
    $somme = 0;
    for ($i = $inf - 1; $i < $sup; $i++) {
      $somme += $tab[$i];
    }
    return $somme;
  }
}
```

```php
$nombres = [1, 2, 3, 4,     5, 6, 7, 8,       9, 10];
echo sommeIntervalle($nombres, 4, 6) === false ? "erreur" :
sommeIntervalle($nombres, 4, 6);  //15
// affiche erreur en cas de tab vide ou 1er nombre > 2eme nombre

2eme solution (meilleure)

function sommeIntervalle($tab, $inf, $sup)
{
  if (empty($tab)  || ($inf > $sup)) {
    return false;
  }

  $sousTab = array_slice($tab, $inf - 1, $sup - $inf + 1);
  return  array_sum($sousTab);

  //return array_sum(array_slice($tab, $inf - 1, $sup - $inf + 1));
}

$nombres = [1, 2, 3, 4,     5, 6, 7, 8,       9, 10];
echo sommeIntervalle($nombres, 3, 4) === false ? "erreur" :
sommeIntervalle($nombres, 3, 4);
```

```php
Correction ex 8

Solution 1
function equipes($tab)
{
  $combinaisons = [];
  for ($i = 0; $i < count($tab); $i++) {

    for ($j = 0; $j < count($tab); $j++) {
      //
      if ( ( $tab[$i]=== $tab [$j]) || (in_array($tab [$j]. $tab[$i],
$combinaisons) ) )   {
        continue;
      }
      array_push($combinaisons, $tab[$i] . $tab[$j]);
    }

  }
  return $combinaisons;
}

$equipes = ["A", "B", "C", "D", "E"];
var_dump(equipes($equipes));
```

Solution 2 :

```php
$tableau = ["A", "B", "C","E","F","D"];

function combinaison($tab)

{
    $tab3 = " ";
    for ($i = 0; $i < count($tab) - 1; $i++) {
        for ($j = $i + 1; $j < count($tab); $j++) {
            $tab3 .= "<br>" . $tab[$i] . $tab[$j];
        }
    }

    return $tab3;
}


print_r(combinaison($tableau));
```

json et php

On peut convertir un tab php en json et inversement en utilisant respectivement les fonctions json_encode() et json_decode().

Encodage (PHP \JSON)

Encoder un tab indexé en json

```php
$voitures = ["Volvo", "BMW", "Toyota"];
echo json_encode($voitures);// ==> donne le tab json suivant
["Volvo","BMW","Toyota"]
```

Encoder un tab associatif en json

```php
$age = array("Peter" => 35, "Ben" => 37, "Joe" => 43);
echo json_encode($age); // donne l'objet json {"Peter":35,"Ben":37,"Joe":43}
```

Décodage (JSON \PHP)

Json_decode() :*permet de decoder un objet json en tab associatif ou en objet php.*

NB : La fonction json_decode() renvoie un objet par défaut. La fonction json_decode() a un deuxième paramètre, et lorsqu'elle est définie sur true, les objets JSON sont décodés dans des tableaux associatifs.

```php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$tab = json_decode($jsonobj, true);
echo $tab['Peter'];

echo "<br>";
```

```php
$obj = json_decode($jsonobj);
echo $obj->Peter;

Parcourir un objet (exactement comme un tab associatif)
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
$obj = json_decode($jsonobj);

foreach ($obj as $key => $value) {
  echo $key . " => " . $value . "<br>";
}
```

Les exceptions

Une exception est un objet qui décrit une erreur ou un comportement inattendu d'un script PHP.

Syntaxe

```php
try {
  // code qui peut lever une exception
} catch(Exception $e) {
  // code exécuté quand une exception est levée
}
```

Exp :

```php
function division($divise, $diviseur)
{
  if ($diviseur == 0) {
    throw new Exception("Division par zero", 1);
  }
  return $divise / $diviseur;
}

try {
  Echo division(5, 0);
}
catch (Exception $e) {
  $code = $e->getCode();
  $message = $e->getMessage();
  $file = $e->getFile();
  $line = $e->getLine();
  Echo "Eception thrown in $file on line $line: [Code $code]
  $message";
}
```

Dans l'implémentation de la fonction division, si on détecte une division par 0, et pour éviter le déclenchement d'une erreur fatale, on préfère traiter nous-mêmes et à notre façon cette erreur (on déclenche une exception throw new Exception("Division by zéro", 1); (on lui attribue le code 1, un code de notre choix pour faciliter le debugge, et un message d'erreur ("Division par zéro").

Lors de l'appel de la fonction, et pour surveiller le déclenchement de l'exception, je l'entoure par un bloc try. Si une exception se déclenche dans ce bloc, c'est celui de catch qui prend la relève pour la traiter (dans celui-ci, on peut mettre le traitement qu'on veut).

les include

https://www.w3schools.com/php/php_includes.asp

Pour inclure d'autres fichiers dans un fichier PHP, on utilise les fonctions include *et* require.
En cas d'échec d'inclusion :
- avec include () : un avertissement (E_WARNING) s'affiche et le script continue à s'exécuter.
- avec require () : *une erreur fatale survient et le script s'arrête.*
Pour inclure un fichier une seule fois, on utilise includeonce () et require once (), *c'est très pratique surtout quand il s'agit des initialisations. On privilégie toujours les require_* once ().

```
Ex :
Require_once "header. PHP";
Echo "<h1>Bienvenue </h1>";
```

Le contenu du fichier header sera inclus dans le fichier qui contient l'instruction require_once ()

PHP filters

https://www.w3schools.com/php/php_filter.asp

//les fonctions filter permettent de faire une validation des variables en fonction d'un type et d'un format donné. Pour ce faire, on utilise la fonction filter_var().
Syntaxe :

```
filter_var($var, FORMAT);

exp :

$int = 100;
var_dump(filter_var($int, FILTER_VALIDATE_INT));

//NB : si la donnée est conforme au format spécifié, filter_var() retourne la valeur de la variable, sinon false, pour pouvoir bien tester la validation, il faut toujours se référer à une valeur différente de false.
//Exp de test de validité

$int = 100;
```

```php
if (filter_var($int, FILTER_VALIDATE_INT) !== false) {
  echo ("Integer is valid");
} else {
  echo ("Integer is not valid");
}

//Exception pour la vaeur 0 (zero)
//La valeur 0 est considérée comme false, donc il faut lui faire un test exceptionnel pour le prendre en considération

$int = 0;

if (filter_var($int, FILTER_VALIDATE_INT) === 0
||     filter_var($int,FILTER_VALIDATE_INT) !== false)
{
    echo ("Integer is valid");
  } else {
    echo ("Integer is not valid");
  }

//PHP met à disposition plusieurs formats qui sont sous forme de FILTERVALIDATE<suffixe> tels que FILTER_VALIDATE_INT et FILTER_VALIDATE_IP. Le bout de code ci-dessous, une fois exécuté, nous donne la liste des suffixes des formats
<table>
  <tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
  </tr>
  <? PHP

  foreach (filter_list() as $id => $filter) {
    echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) .
'</td></tr>';
  }
  ?>
</table>
//NB : avant de valider les données, on peut les nettoyer pour éviter les caractères indésirables ou l'injection de code avec le format FILTER SANITIZE<suffixe>
//Exp :
  $email = "john.doe@example.com";

  // nettoie la chaine de tt caractere non valide dans un mail
  $email = filter_var($email, FILTER_SANITIZE_EMAIL);

  // Valider le format e-mail de la chaine nettoyée
  if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
```

```php
    echo ("$email is a valid email address");
  } else {
    echo ("$email is not a valid email address");
  }
```

Les regex

```php
//les regex cherchent à vérifier la conformité d'une chaine de caractères à un pattern (ou modèle ou masque) spécifique.
//Pour utiliser les regex on dispose de 3 fonctions.
preg_match() //: permet de vérifier si une chaine est conforme au pattern ou pas, elle retourne 1 si oui, 0 sinon.
Syntaxe :
preg_match($pattern, $chaine);


//syntaxe des patterns : admet des délimiteurs, des symboles spéciaux tels que / / ou # #

Ex :
$pattern = "/[a-z]/i";
$pattern = "#[a-z]#i";
//Symboles des patterns :
//- symbole de groupement [] : la chaine doit etre conforme aux symboles du groupe entre []
Ex :
$str = "salut";
$pattern = "/[abc]/";
var_dump (preg_match($pattern, $str)); // 1
//la chaine contient des caractères du groupe abc
$str = "slt";
$pattern = "/[abc]/";
var_dump (preg_match($pattern, $str));  // 0
//la chaine ne contient pas un caractère du groupe abc
//NB : je peux spécifier aussi des intervalles de données
$pattern = "/[a-b]/"; // intervalle de a à z minuscules
$pattern = "/[A-Z]/"; // intervalle de A à Z majuscules
$pattern = "/[0-9]/"; // chiffres de 0 à 9
NB : on peut combiner les intervalles
$pattern = "/[a-zA-Z0-9]/"; // les lettres min et maj et les chiffres
//- Pattern sans délimiteur de début et de fin : la chaine doit contenir au moins l'un des symboles du groupe
$str = "salut";
$pattern = "/[abc]/";
var_dump (preg_match($pattern, $str)); // 1 : la chaine contient au moins l'un des symboles //du groupe
```

```php
$str = "slt";
$pattern = "/[abc]/";
var_dump (preg_match($pattern, $str)); // 0 : la chaine ne contient aucun

symbole du groupe
//- Symboles de debut et de fin ^ $ (délimiteur de début et de fin ) : ^ indique le début de la chaine, et $ indique sa fin. La chaine doit STRICTEMENT respecter le pattern à la lettre (en nombre et en ordre).
$str = "az";
$pattern = "/^[abc]z$/";
var_dump (preg_match($pattern, $str)); // 1 car ca commence par un sysmbole du groupe abc et suivi par z
$str = "dz"; //  0 :ne prend pas car ne commence par a, b ou c
$str = "za"; // 0 : doit respecter l'ordre, z en denier lieu
$str = "ac"; // 0 : ne finit pas par z
- symbole de négation ^ : la chaine ne doit pas contenir le symbole précédé par ^
$str = "da";
$pattern = "/^[^abc][a-z]$/";
var_dump(preg_match($pattern, $str)); // 1 : ne commence par aucun des symboles abc et suivi d'un caractère du groupe a-z
$str = "cf"; // 0 : car ça commence par un symbole du groupe abc
- symbole OU logique | : une expression ou une autre rend la chaine valide :
$str = "d";
$pattern = "/^([^abc])|([^0-9])$/";
var_dump(preg_match($pattern, $str)); // 1
([^abc])  |    ([^0-9]) : cette expression est pour une chaine qui ne commence pas par a,b ou c ni un chiffre

$str = "5"; // 1 valide par l'expression [^abc]

- symbole générique . (point) : n'importe quel caractère
$str = "+";
$pattern = "/^.$/";
var_dump(preg_match($pattern, $str)); //1
$str = "5"; //1
$str = "f"; //1
$str = "D"; //1

Méta caractères des patterns

\d : correspond à n'importe quel caractère numérique. Identique à [0-9]
$str = "7";
$pattern = "/^\d$/";
\D : Correspond à tout caractère non numérique. Identique à [^0-9] (le contraire de \d)
$str = "a";
$pattern = "/^\D$/";
var_dump(preg_match($pattern, $str)); // 1
```

```php
$str = "9";
$pattern = "/^\D$/";
var_dump(preg_match($pattern, $str)); // 0 : car 9 est un chiffre
```

\s : Correspond à n'importe quel caractère d'espacement (espace, tabulation, saut de ligne ou retour chariot). Identique à [\t\n\r]

\S : le contraire de \s

\w : Correspond à n'importe quel caractère de mot (défini comme a à z, A à Z, 0 à 9 et le trait de soulignement). Identique à [a-zA-Z_0-9]

\W : le contraire de \w

Quantificateurs des patterns

+ : 1, N fois

```php
$str = "a";
$pattern = "/^[a-z]+$/";
var_dump(preg_match($pattern, $str)); // 1 : au moins une fois
$str = "abc";  // 1 : plusieurs fois
$str = "";  // 0 : chaine vide
```

* : 0, N fois

```php
$str = "a";
$pattern = "/^[a-z]*$/";
var_dump(preg_match($pattern, $str)); // 1 : au moins une fois

$str = "abc"; // 1 : plusieurs fois
$str = ""; // 1 : aucune fois
```

? : 0,1 fois

```php
$str = "a";
$pattern = "/^[a-z]?$/";
var_dump(preg_match($pattern, $str)); // 1 :  une seule fois

$str = ""; // 1 : aucune fois
$str = "ab"; // 0 : plusieurs fois
```

{n} : n est un nombre positif, ca se répète n fois (ni plus ni moins)

```php
$str = "abc";
$pattern = "/^[a-z]{3}$/";
var_dump(preg_match($pattern, $str)); // 1 :  3 fois

$str = "abcd"; // 0 : autre que 3
```

{n, m} : n et m des nombres positifs, ça se répète entre n et m fois (ni plus ni moins)

```php
$str = "a";
$pattern = "/^[a-z]{1,2}$/";
var_dump(preg_match($pattern, $str)); // 1 :  1 fois
```

```php
$str = "ab"; // 1 : 2 fois
$str = "abc"; // 0 :  sup à 2 fois
$str = ""; // 0 :  inf à 1 fois

{n,} : n : nombre positif, ça se répète au moins n fois (pas moins, plus oui)
$str = "abc";
$pattern = "/^[a-z]{3,}$/";
var_dump(preg_match($pattern, $str)); // 1 :  3 fois minimum

$str = "ab"; // 0 : 2 fois <3
$str = "abcd"; // 1 :  sup à 3 fois

Echappement des caractères spéciaux
NB : Si votre expression doit rechercher l'un des caractères spéciaux, vous pouvez utiliser une barre oblique inverse ( \ ) pour les échapper. Par exemple, pour rechercher un ou plusieurs points d'interrogation, vous pouvez utiliser l'expression suivante : $pattern = '/?+/';
$str = "???";
$pattern = "/^\?+$/";
var_dump(preg_match($pattern, $str)); // 1 :  contient ? plus qu'une fois
//==> le caractère ? a une signification particulière (caractère spécial), pour le prendre comme un caractère normal, il faut le précéder par le caractère d'échappement \
```

Rque : quand il s'agit de délimiteur de début et de fin, il faut respecter l'ordre et la quantité.

```php
Ex 1 : valider un num de tel en france

$str = "0705812367";
$pattern = "/^0[67][0-9]{8}$/";
var_dump(preg_match($pattern, $str));
```

Série sur les regex

```php
Ex 1 : valider un num de tel portable en france au format 0612345678

$pattern = "/^0[67][0-9]{8}$/";
$pattern = "/^0[67]\d{8}$/";
//étapes de réalisation de cette regex
//mettre le format sous formes de petites parties
```

```
//0==> 0
//6 ou 7  [67]
//8 chiffres   [0-9]{8}

//et si on veut faire l'oération inverse : décortiquer une regex
//    /^    0 [67]   [0-9]{8}    $/
//1ere caractere 0
//caractere soit 6 soit 7
//un caractere de 0-9 qui se repete 8 fois
Ex 2 : Même chose que l'ex précédent au format 06 12 34 56 78

$pattern = "/^0[67]( \d{2}){4}$/";

Explication :
// ^   0 [67]   (   \d {2}  ){4}   $

// 0
// 6 ou 7
// (espace suivi de 2 chiffres) 4 fois


Ex 3 : Faire une regex qui valide les deux formats



$pattern = "/^0[67]( ?\d{2}){4}$/";

$pattern = "/^0    [67]     (  espace?  \d{2}  )   {4}    $/";
/*                          expression 1 :
                             L'espace peut exister UNE SEULE FOIS ou pas
                             deux chiffres
0
6 ou 7
[
  L'espace peut exister UNE SEULE FOIS ou pas
  deux chiffres
] qui se repere 4 fois */


Ex 4 : valider un num fixe en France 01 à 05 et 08 et 09 avec ou sans espaces

$pattern = "/^0[1-589]( ?\d{2}){4}$/";

Ex 5 : valider un num de tel fixe ou portable en France avec ou sans espace
$pattern = "/^0[1-9]( ?\d{2}){4}$/";


Ex 6 : valider un num français au format international au format
+33(0)612345678
```

```
//le num peut etre fixe ou mobile, avec ou sans espaces
$pattern = "/^\+33\(0\)[1-9]( ?\d{2}){4}$/";

Ex 7 : meme ex que le precedent mais au format +33(0)612345678 ou
0033(0)612345678
//(fixe ou mobile, avec ou sans espaces)
$pattern = "/^(\+|00)33\(0\)[1-9]( ?\d{2}){4}$/";

Ex 8 : valider un code postal en France

$pattern = "/^(([0-8][1-9])|([1-9][0-57-9]) )\d{3}$/";

$pattern = "/^(0[1-9]|[1-8]\d|9[0-57-9])\d{3}$/";

Ex 9 : valider un id des chiffres qui commence de 1 à n (n nombre positif)
$pattern = "/^[1-9][0-9]*$/";

Ex 10 : valider la regex de la référence d'une salle dans un bâtiment au
format bloc-étage-salle comme A-01-S10, sachant que :
    - le bâtiment contient 4 blocs, le 1er commence par A
    - chaque bloc contient 8 étages
    - chaque étage contient 10 salles
    NB : on a un RDC  + 8 étages (RDC : étage 00)
$pattern = "/^[A-D]\-0[0-8]\-S(0[1-9]|10)$/";

Ex 11 : expliquer la regex suivante

"/^[\w\-\s]{1,}-fab-[a-zA-Z\s]{1,}-ref-[\w]{1,}$/"

//A quoi elle peut servir

/*
1- expression 1 :  [\w\-\s]{1,}
  \w : les lettre maj, min, chiffres, _,
  le - (tiret de 6)
  \s : espace, tabulation, retour à la ligne
se repete de 1 à n fois

2-  \-fab\- : -fab- en dur

3- expression 3 : [a-zA-Z\s]{1,}
  les lettres min ou maj ou espace
se repete de 1 à n fois

4-  \-ref\- :  -ref- en dur

5- expression 5 : [\w]{1,}
  \w : les lettre maj, min, chiffres, _,
se repete de 1 à n fois
```

```
*/
```

Lien utile : https://regex101.com/

Noté bien :

```
Pour valider les chaines qui contiennent des caractères non latins (utf-8)
voir doc : https://www.php.net/manual/fr/regexp.reference.unicode.php
$pattern = "/^[\p{L}\s]{2,}$/u";
```