# PHP : fonctions

**Achref El Mouelhi**

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

# Plan

# Plan

# PHP

## Une fonction ?

- Un bloc d'instructions réalisant une fonctionnalité bien déterminée.
- Peut être appelée plusieurs fois
- Déclaration d'une fonction $\not\Rightarrow$ son exécution
- Appel d'une fonction $\Rightarrow$ son exécution

# PHP

## Particularité du langage **PHP**

- Le nom d'une fonction n'est pas sensible à la casse.
- La surcharge des fonctions n'est pas autorisée.
- Depuis **PHP 7**, on peut typer les paramètres et la valeur de retour.

# PHP

## Une fonction en **PHP** peut

- retourner une seule valeur
- prendre 0 ou plusieurs paramètres
- appeler d'autres fonctions
- avoir comme paramètre le nom d'une autre fonction

# PHP

## Déclarer une fonction

```
function nom_fonction([les arguments])
{
  les instructions de la fonction
}
```

# PHP

## Déclarer une fonction

```
function nom_fonction([les arguments])
{
  les instructions de la fonction
}
```

## Exemple

```
function somme($a, $b)
{
    return $a + $b;
}
```

# PHP

## Déclarer une fonction

```
function nom_fonction([les arguments])
{
  les instructions de la fonction
}
```

## Exemple

```
function somme($a, $b)
{
    return $a + $b;
}
```



### Remarque

Le `return` permet de quitter la fonction. Toute instruction définie dans la fonction après ne sera pas exécutée.

# PHP

## Appeler une fonction

```php
$resultat = somme (1, 3);
echo $resultat;
/* affiche 4 */
```

# PHP

## Exercice

Écrire une fonction **PHP** nommée `maximum_2($a, $b)` qui retourne le maximum entre `$a` et `$b`.

# PHP

## Exercice

Écrire une fonction **PHP** nommée `maximum_2($a, $b)` qui retourne le maximum entre `$a` et `$b`.

## Exercice

Écrire une fonction **PHP** nommée `maximum_3($a, $b, $c)` qui retourne le maximum entre `$a`, `$b` et `$c` (vous pouvez utiliser la fonction `maximum_2`) .

# PHP

## Exercice

Écrire une fonction **PHP** qui permet de déterminer si un nombre passé en paramètre est premier. La fonction retourne un booléen.

# PHP

## Déclarer une fonction anonyme (appelée aussi closure ou fermeture)

```
$nom_variable = function ([les arguments])
{
  les instructions de la fonction
}
```

# PHP

## Déclarer une fonction anonyme (appelée aussi closure ou fermeture)

```
$nom_variable = function ([les arguments])
{
  les instructions de la fonction
}
```

## Exemple

```
$somme2 = function ($a, $b)
{
    return $a + $b;
};
```

# PHP

## Déclarer une fonction anonyme (appelée aussi closure ou fermeture)

```
$nom_variable = function ([les arguments])
{
  les instructions de la fonction
}
```

## Exemple

```
$somme2 = function ($a, $b)
{
    return $a + $b;
};
```

## Appeler une fonction anonyme

```
$resultat = $somme2 (1, 3);
echo $resultat;
/* affiche 4 */
```

**Appeler la fonction `somme` sans passer de paramètres ⇒ erreur fatale**

```php
echo(somme());
```

**Appeler la fonction `somme` sans passer de paramètres ⇒ erreur fatale**

```
echo(somme());
```

**Solution : affecter des valeurs par défaut aux paramètres**

```
function somme($a = 0, $b = 0)
{
    return $a + $b;
}
```

**Appeler la fonction `somme` sans passer de paramètres ⇒ erreur fatale**

```php
echo(somme());
```

**Solution : affecter des valeurs par défaut aux paramètres**

```php
function somme($a = 0, $b = 0)
{
    return $a + $b;
}
```

**Ainsi, on peut appeler la fonction avec 0, 1 ou deux paramètres**

```php
echo somme ();
/* affiche 0 */

echo somme (5);
/* affiche 5 */

echo somme (5, 3);
/* affiche 8 */
```

**Pour écrire une fonction qui accepte un nombre indéterminé de paramètres, on peut utiliser les fonctions `func_get_args()` et `func_num_args()`**

```php
function somme ()
{
    $arguments = func_get_args();
    $result = 0;
    for($i = 0; $i < func_num_args(); $i++) {
        $result += $arguments[$i];
    }
    return $result;
}
```

**Pour écrire une fonction qui accepte un nombre indéterminé de paramètres, on peut utiliser les fonctions `func_get_args()` et `func_num_args()`**

```php
function somme ()
{
    $arguments = func_get_args();
    $result = 0;
    for($i = 0; $i < func_num_args(); $i++) {
        $result += $arguments[$i];
    }
    return $result;
}
```

**Tous ces appels sont corrects**

```php
echo(somme(2, 3, 5));
/* affiche 10 */

echo(somme(2, 3));
/* affiche 5 */

echo(somme(2, 3, 5, 7));
/* affiche 17 */

echo(somme(2));
/* affiche 2 */
```

**Depuis PHP 5.6, on peut aussi utiliser le concept de décomposition avec l'opérateur . . .**

```
function somme (...$arguments)
{
    $result = 0;
    for($i = 0; $i < count($arguments) ; $i++) {
        $result += $arguments[$i];
    }
    return $result;
}
```

**Depuis PHP 5.6, on peut aussi utiliser le concept de décomposition avec l'opérateur . . .**

```
function somme (...$arguments)
{
    $result = 0;
    for($i = 0; $i < count($arguments) ; $i++) {
        $result += $arguments[$i];
    }
    return $result;
}
```

**Tous ces appels restent corrects**

```
echo(somme(2, 3, 5));
/* affiche 10 */

echo(somme(2, 3));
/* affiche 5 */

echo(somme(2, 3, 5, 7));
/* affiche 17 */

echo(somme(2));
/* affiche 2 */
```

# PHP

## Exercice

Écrire une fonction **PHP** qui permet de déterminer si le premier paramètre est divisible par tous les autres paramètres. Le nombre de paramètres est variable et la fonction retourne un booléen.



### Exemple

```
var_dump(est_divisible_par(10, 2, 3, 5)); // false
var_dump(est_divisible_par(10, 2, 5)); // true
```

# PHP

## Correction

```
function est_divisible_par($nombre, ...$diviseurs)
{
    foreach ($diviseurs as $diviseur) {
        if ($nombre % $diviseur != 0) {
            return false;
        }
    }
    return true;
}
```

# PHP

**Considérons la fonction suivante**

```
function say_hello($nom = "Doe", $prenom = "John", $age = 0)
{
    echo "<br>Hello $prenom $nom, you are $age years old.";
}
```

# PHP

**Considérons la fonction suivante**

```
function say_hello($nom = "Doe", $prenom = "John", $age = 0)
{
    echo "<br>Hello $prenom $nom, you are $age years old.";
}
```

**On peut appeler la fonction en respectant l'ordre des paramètres (paramètres positionnels)**

```
say_hello("Dupont", "Jean", 45);
// affiche  Hello Jean Dupont, you are 45 years old.

say_hello();
// affiche Hello John Doe, you are 0 years old.
```

# PHP

**Ou en nommant les paramètres (et sans avoir besoin de respecter l'ordre)**

```
say_hello("Wick");
// affiche Hello John Wick, you are 0 years old.

say_hello(nom: "Wick", age: 50);
// affiche Hello John Wick, you are 50 years old.

say_hello("Wick", age: 50);
// affiche Hello John Wick, you are 50 years old.

say_hello(prenom: "Sophie", age: 50, nom: "Wick");
// affiche Hello Sophie Wick, you are 50 years old.
```

# PHP

## Règles à respecter pour les paramètres nommés

- Pas de paramètres dupliqués : chaque paramètre doit être spécifié une seule fois.
- Si vous utilisez à la fois des arguments positionnels (sans nom) et des arguments nommés dans un appel de fonction, les arguments positionnels doivent être placés en premier, suivi des arguments nommés.

# PHP

## Exercice

Écrire une fonction qui permet de permuter le contenu de deux variables (de type entier par exemple) passées en paramètre

# PHP

## Exercice

Écrire une fonction qui permet de permuter le contenu de deux variables (de type entier par exemple) passées en paramètre

**Voici à quoi en pense la première fois**

```
function permuter($a, $b)
{
    $aux = $a;
    $a = $b;
    $b = $aux;
}
```

# PHP

**En testant ce code, c'est la surprise**

```php
$x = 2;
$y = 3;
permuter($x, $y);
echo $x, " ", $y;
/* affiche 2 3 */
```

# PHP

**En testant ce code, c'est la surprise**

```
$x = 2;
$y = 3;
permuter($x, $y);
echo $x, " ", $y;
/* affiche 2 3 */
```

**Et pourtant, si on vérifie dans la fonction, les valeurs ont bien été permutées**

```
function permuter($a, $b)
{
    $aux = $a;
    $a = $b;
    $b = $aux;
    echo $a, " ", $b;
    /* affiche 3 2 */
}
```

# PHP

## Passage par valeur

- Les fonctions, par définition, reçoivent une copie du contenu de la variable
- Elles ne manipulent pas la case de mémoire de la variable envoyée
- On appelle ce concept **passage par valeur**

# PHP

## Passage par valeur

- Les fonctions, par définition, reçoivent une copie du contenu de la variable
- Elles ne manipulent pas la case de mémoire de la variable envoyée
- On appelle ce concept **passage par valeur**



## Quelle solution ?

Passage par référence

# PHP

**Pour passer les paramètres par référence, on utilise &**

```
function permuter(&$a, &$b)
{
    $aux = $a;
    $a = $b;
    $b = $aux;
}
```

# PHP

**Pour passer les paramètres par référence, on utilise &**

```
function permuter(&$a, &$b)
{
    $aux = $a;
    $a = $b;
    $b = $aux;
}
```



**Pour tester**

```
$x = 2;
$y = 3;
permuter($x, $y);
echo $x, " ", $y;
/* affiche 3 2 */
```

# PHP

## Fonction de retour (ou de rappel, en anglais callback)

- fonction appelée comme un paramètre d'une deuxième fonction
- très utilisée avec les fonctions asynchrones

# PHP

## Fonction de retour (ou de rappel, en anglais callback)

- fonction appelée comme un paramètre d'une deuxième fonction
- très utilisée avec les fonctions asynchrones

**Considérons les deux fonctions suivantes**

```
function somme($a, $b)
{
    return $a + $b;
}

function produit($a, $b)
{
    return $a * $b;
}
```

# PHP

**Utilisons les fonctions précédentes comme callback d'une fonction** `operation()`

```
function operation($a, $b, $fonction)
{
    echo ($fonction($a, $b));
}
operation (3, 5, 'somme');
/* affiche 8 */
```

# PHP

**On peut aussi utiliser la fonction `is_callable` si un paramètre correspond à une fonction de rappel**

```
function operation($a, $b, $fonction)
{
    if (is_callable($fonction))
        echo ($fonction($a, $b));
    else
        echo "erreur";
}

operation (3, 5, 'somme');
/* affiche 8 */

operation (3, 5, 'division');
/* affiche erreur */
```

# PHP

**On peut aussi exécuter la fonction callback en appelant la fonction PHP** `call_user_func`

```
function operation($a, $b, $fonction)
{
    if (is_callable($fonction))
        echo call_user_func($fonction, $a, $b);
    else
        echo "erreur";
}

operation (3, 5, 'somme');
/* affiche 8 */

operation (3, 5, 'division');
/* affiche erreur */
```

# PHP

## Exercice

- Créer une fonction `composition` qui accepte 5 paramètres dont deux fonctions callback et qui retourne le résultat de la composition de ces deux callbacks.
- Par exemple :
  `composition (2, 3 , 6, 'somme', 'produit')` retourne 30 = (2 + 3) * 6

# PHP

## Quelques fonctions prédéfinies sur les tableaux utilisant les callback

- `array_map(callback, tableau)` : exécute la `callback` sur les éléments d'un tableau.
- `array_walk(tableau, callback)` : exécute la `callback` sur chaque élément du tableau et retourne toujours `true`.
- `array_filter(tableau, callback)` : filtre les éléments d'un tableau selon le prédicat de la `callback`.
- `array_reduce(tableau, callback, initial)` : applique itérativement la `callback` aux éléments du tableau array, de manière à réduire le tableau à une valeur simple.
- `array_find(tableau, callback)` [**PHP 8.4**] : retourne la clé du premier élément validant la `callback`.
- `array_find_key(tableau, callback)` [**PHP 8.4**] : retourne la clé du premier élément validant la `callback`.
- `array_all(tableau, callback)` [**PHP 8.4**] : vérifie si tous les éléments du tableau valident la `callback`.
- `array_any(tableau, callback)` [**PHP 8.4**] : vérifie qu'au moins un élément du tableau valide la `callback`.
- Liste complète des fonctions : `https://www.php.net/manual/fr/ref.array.php`

# PHP

## Exemple avec `array_map`

```php
$nombres = array(
    2,
    5,
    8
);

function carre($x)
{
    return $x * $x;
}

print_r(array_map("carre", $nombres));
/* affiche Array ([0] => 4[1] => 25[2] => 64 ) */
```

# PHP

**Exemple avec** `array_filter`

```php
$nombres = array(
    2,
    5,
    8
);

function est_pair($x)
{
    return $x % 2 == 0;
}

print_r(array_filter($nombres, "est_pair"));
/* affiche Array ([0] => 2[2] => 8 )  */
```

# PHP

**Exemple avec** `array_filter`

```php
$nombres = array(
    2,
    5,
    8
);

function est_pair($x)
{
    return $x % 2 == 0;
}

print_r(array_filter($nombres, "est_pair"));
/* affiche Array ([0] => 2[2] => 8 )  */
```

**Remarque**

Les indices des éléments filtrés sont ceux du tableau d'origine.

# PHP

## Pour filtrer et garder des indices consécutifs

```
$nombres = array(
    2,
    5,
    8
);

function est_pair($x)
{
    return $x % 2 == 0;
}

print_r([...array_filter($nombres, "est_pair")]);
/* affiche Array ([0] => 2[1] => 8 )  */
```

# PHP

## Exemple avec `array_reduce`

```
$nombres = array(
    2,
    5,
    8
);

function somme($x, $y)
{
    return $x + $y;
}

print_r(array_reduce($nombres, "somme", 0));
/* affiche 15  */
```

# PHP

**Étant donné le tableau suivant**

```
$marques = ["peugeot", "ford", "mercedes", "citroen", "audi"];
```



## Exercice

Écrire un script **PHP** qui permet de

1. remplacer les chaînes de caractère du tableau `marques` par leurs longueurs.
2. afficher les marques contenant un nombre pair de caractères.
3. calculer le nombre total de caractères présents dans `marques`.

# PHP

## Fonction variable

- une fonctionnalité qui permet d'appeler une fonction en utilisant une variable contenant son nom.
- si une variable contient le nom d'une fonction, vous pouvez l'appeler avec `()`.

# PHP

## Fonction variable

- une fonctionnalité qui permet d'appeler une fonction en utilisant une variable contenant son nom.
- si une variable contient le nom d'une fonction, vous pouvez l'appeler avec `()`.



### Exemple

```
function somme($a, $b)
{
    return $a + $b;
}

$sum = 'somme';
echo  $sum(2, 3);
// affiche 5
```

# PHP

**Remarque**

L'utilisation des fonctions variables peut poser des problèmes de sécurité si elle est utilisée avec des entrées (valeurs saisies par l'utilisateur) non contrôlées.

# PHP

## Fonctions fléchées (Arrow functions)

- Disponible depuis **PHP 7.4**
- Utilisant le mot-clé `fn` et l'opérateur =>
- Mono-instruction
- Avec un `return` implicite

# PHP

**Exemple avec** `array_map`

```php
$nombres = array(
    2,
    5,
    8
);

$tab = array_map(fn($elt) => $elt * $elt, $nombres);
print_r($tab);
/* affiche Array ([0] => 4[1] => 25[2] => 64 ) */
```

# PHP

**Exemple avec** `array_filter`

```php
$nombres = array(
    2,
    5,
    8
);

$tab = array_filter($nombres, fn($elt) => $elt % 2 == 0);
print_r($tab);
/* affiche Array ([0] => 2[2] => 8 )  */
```

# PHP

**Exemple avec** `array_reduce`

```php
$nombres = array(
    2,
    5,
    8
);

$tab = array_reduce($nombres, fn ($elt, $res) => $elt + $res,
   0);
print_r($tab);
/* affiche 15  */
```

# PHP

## Depuis **PHP 7**, on peut

- typer les paramètres
- typer la valeur de retour
- utiliser le typage strict
- ...

# PHP

## Typage de paramètres

- Disponible depuis **PHP 7**
- Permettant de filtrer les valeurs pour éviter à nos fonctions un comportement indésirable

# PHP

## Typage de paramètres

- Disponible depuis **PHP 7**
- Permettant de filtrer les valeurs pour éviter à nos fonctions un comportement indésirable



### Syntaxe

```
function nom_fonction([{type_variable $nom_variable
   }])
{
    // les instructions
}
```

# PHP

## Exemple

```
function somme(int $a, int $b)
{
    return $a + $b;
}
```

# PHP

## Exemple

```
function somme(int $a, int $b)
{
    return $a + $b;
}
```



## Appeler une fonction

```
$resultat = somme (1, 3);
console.log($resultat);
/* affiche 4 */
```

# PHP

## Typage de valeurs de retour

- Disponible depuis **PHP 7**
- Permettant de déterminer le type de la variable recevant la valeur de retour d'une fonction

# PHP

## Typage de valeurs de retour

- Disponible depuis **PHP 7**
- Permettant de déterminer le type de la variable recevant la valeur de retour d'une fonction



### Syntaxe

```
function nom_fonction([les paramètres]): type
{
    // les instructions
}
```

# PHP

## Exemple

```
function somme(int $a, int $b): int
{
    return $a + $b;
}
```

# PHP

## Exemple

```
function somme(int $a, int $b): int
{
    return $a + $b;
}
```



## L'appel ne change pas

```
$resultat = somme (1, 3);
console.log($resultat);
/* affiche 4 */
```

# PHP

## Types valides pour les fonctions

- `null`
- `int`
- `string`
- `bool`
- `float`
- `array`
- `resource` : un fichier ouvert, une connexion à une base de données, une image...
- `mixed` : équivalent au type union `objet|ressource|array|string|float|int|bool|null`
- `object` (à voir dans un prochain chapitre)
- `iterable` : paramètre itérable avec une structure itérative (`for`, `foreach`...)
- `callable` : fonction callback
- `void` : comme valeur de retour indique que la fonction n'a pas de valeur de retour, et comme paramètre indique que la fonction n'accepte pas de paramètre.

# PHP

## Union de type

Depuis **PHP 8**, un paramètre de fonction ou un attribut de classe peut avoir plusieurs types.

# PHP

## Union de type

Depuis **PHP 8**, un paramètre de fonction ou un attribut de classe peut avoir plusieurs types.

### Exemple

```
function somme(int|string $a, int|float $b): int|string|float
{
    if (gettype($a) == "integer") {
        return $a + $b;
    }
    return $a . $b;
}
```

# PHP

## Union de type

Depuis **PHP 8**, un paramètre de fonction ou un attribut de classe peut avoir plusieurs types.

### Exemple

```
function somme(int|string $a, int|float $b): int|string|float
{
    if (gettype($a) == "integer") {
        return $a + $b;
    }
    return $a . $b;
}
```

## Explication

Le premier paramètre peut être de type entier ou chaîne de caractères.

**Résultat de l'appel de la fonction `somme` avec différent type de paramètre**

```php
echo somme(3, 2.5);
// affiche 5.5

echo somme("a", 2.5);
// affiche a2.5
```

# PHP

## Considérons la fonction `somme` suivante

```
function somme(int $a, int $b): int
{
    return $a + $b;
}
```

# PHP

**Considérons la fonction `somme` suivante**

```php
function somme(int $a, int $b): int
{
    return $a + $b;
}
```

**Comme le montre l'exemple suivant, PHP essayera de convertir les valeurs s'ils ne sont pas de type entier**

```php
echo somme("1", 2.5);
/* affiche 3 */
```

## Question

Comment interdire l'exécution de la fonction si les types de valeurs ne correspondent pas aux types des paramètres de la fonction.

## Question

Comment interdire l'exécution de la fonction si les types de valeurs ne correspondent pas aux types des paramètres de la fonction.

## Solution

Utiliser le mode strict.

## Question

Comment interdire l'exécution de la fonction si les types de valeurs ne correspondent pas aux types des paramètres de la fonction.

## Solution

Utiliser le mode strict.

**Déclarer le mode strict au tout début du fichier**

```
<?php
declare(strict_types=1);
?>
```

## Question

Comment interdire l'exécution de la fonction si les types de valeurs ne correspondent pas aux types des paramètres de la fonction.

## Solution

Utiliser le mode strict.

**Déclarer le mode strict au tout début du fichier**

```
<?php
declare(strict_types=1);
?>
```

**Appeler une fonction avec des paramètres qui ne sont pas entiers implique désormais une erreur fatale**

```
echo somme("php", 2.5);
```

# PHP

**Considérons la fonction `dire_bonjour` suivante**

```
function dire_bonjour(string $nom): void
{
    if ($nom != null) {
        echo "bonjour $nom";
    } else {
        echo "bonjour doe";
    }
}
```

# PHP

**Considérons la fonction `dire_bonjour` suivante**

```
function dire_bonjour(string $nom): void
{
    if ($nom != null) {
        echo "bonjour $nom";
    } else {
        echo "bonjour doe";
    }
}
```

**Appeler cette fonction avec une valeur `NULL` implique une erreur fatale**

```
dire_bonjour(null);
```

# PHP

**Considérons la fonction `dire_bonjour` suivante**

```
function dire_bonjour(string $nom): void
{
    if ($nom != null) {
        echo "bonjour $nom";
    } else {
        echo "bonjour doe";
    }
}
```

**Appeler cette fonction avec une valeur `NULL` implique une erreur fatale**

```
dire_bonjour(null);
```

**Remarque**

Les types `null` et `string` sont incompatibles.

# PHP

## Question

Comment peut-on autoriser la valeur `null` pour les autres types ?

# PHP

## Question

Comment peut-on autoriser la valeur `null` pour les autres types ?

## Solution

En utilisant l'opérateur `?` pour rendre un type `nullable`.

# PHP

## Modifions la fonction `dire_bonjour`

```
function dire_bonjour(?string $nom): void
{
    if ($nom != null) {
        echo "bonjour $nom";
    } else {
        echo "bonjour Doe";
    }
}
```

# PHP

**Modifions la fonction** `dire_bonjour`

```
function dire_bonjour(?string $nom): void
{
    if ($nom != null) {
        echo "bonjour $nom";
    } else {
        echo "bonjour Doe";
    }
}
```



**Appeler** `dire_bonjour` **avec un paramètre** `NULL` **n'implique plus une erreur fatale**

```
dire_bonjour(null);
// affiche bonjour Doe
```

# PHP

**Exercice : qu'affiche le code suivant**

```php
function say_hello(?string $nom = 'Doe'): void
{
    echo "Hello $nom";
}

echo "\n" . say_hello();
echo "\n" . say_hello(null);
```

### Résultat

```
Hello Doe
Hello
```

# PHP

**Résultat**

```
Hello Doe
Hello
```



## Explication

La valeur par défaut du paramètre `$nom` n'a pas été utilisé car `null` est considérée comme une valeur valide.

# PHP

**Pour utiliser `Doe` à la place de la valeur `null`, on utilise `??`**

```
function say_hello(?string $nom = 'Doe'): void
{
    $nom = $nom ?? 'Doe';
    echo "Hello $nom";
}

echo "\n" . say_hello();
echo "\n" . say_hello(null);
```

# PHP

## Remarque

**PHP** ne prend pas en charge les alias de type, définis par l'utilisateur.

# PHP

## Fonction génératrice

- une fonction **PHP**
- contenant un générateur `yield` qui retourne autant de valeurs que nécessaire

# PHP

## Fonction génératrice

- une fonction **PHP**
- contenant un générateur `yield` qui retourne autant de valeurs que nécessaire



**Exemple d'une fonction qui retourne, à chaque appel, un nombre pair différent** $< 10$

```
function generateur_pair(int $seuil)
{
    for ($i = 0; $i <= $seuil; $i += 2) {
        yield $i;
    }
}
```

# PHP

**Pour appeler la fonction**

```
foreach (generateur_pair(10) as $value) {
    echo "$value ";
}
/* affiche 0 2 4 6 8 10 */
```

## Une fonction génératrice peut retourner aussi un couple (clé, valeur)

```
function carre(int $seuil) {
    for ($i = 0; $i <= $seuil; $i++)
    {
        yield $i => $i * $i;
    }
}
```

**Une fonction génératrice peut retourner aussi un couple (clé, valeur)**

```
function carre(int $seuil) {
    for ($i = 0; $i <= $seuil; $i++)
    {
        yield $i => $i * $i;
    }
}
```

**Pour appeler la fonction**

```
foreach (carre(5) as $key => $value) {
    echo "le carré de $key est $value<br>";
}

/* affiche
le carré de 0 est 0
le carré de 1 est 1
le carré de 2 est 4
le carré de 3 est 9
le carré de 4 est 16
le carré de 5 est 25
*/
```

# PHP

**Étant donnée la chaîne de caractères suivante**

```
$ma_chaine = "Hello les holoulos";
```

## Exercice

Écrire une fonction génératrice qui retourne chaque fois une voyelle de la chaîne `$ma_chaine` et sa position.

# PHP

## Correction

```
function detecteur_voyelle(string $str)
{
    $voyelles = "aeouiy";
    for ($i = 0; $i < strlen($str); $i++) {
        if (strpos($voyelles, $str[$i]) !== false)
            yield $i => $str[$i];
    }
}
foreach (detecteur_voyelle($ma_chaine) as $key => $value)
    {
    echo "position  $key : voyelle : $value<br>";
}
```

# PHP

**Étant donné le tableau à deux dimension suivant**

```php
$matrice = array(
    array(2, 3, 5, 7),
    array(1, 2, 5, 9, 5),
    array(4, 2, 9, 3, 1, 5)
);
```



## Exercice

Écrire une fonction génératrice qui retourne chaque fois les éléments pairs de la ligne suivante de la matrice.

# PHP

## Correction

```
function estPair(int $x): bool
{
    return $x % 2 == 0;
}
function pair_ligne_matrice(array $tab)
{
    foreach ($tab as $key => $value) {
        yield $key => array_filter($value, 'estPair');
    }
}
foreach (pair_ligne_matrice($matrice) as $key => $value)
  {
    echo "position  $key : élément.s pair.s : " . implode
       ("-", $value) . "<br>";
}
```

# PHP

**L'instruction `yield from` permet de retourner un nombre défini dans un tableau**

```
function generer_nombre()
{
    yield 0;
    yield from [1, 3, 6, 10];
    yield 100;
}
```

# PHP

**L'instruction `yield from` permet de retourner un nombre défini dans un tableau**

```
function generer_nombre()
{
    yield 0;
    yield from [1, 3, 6, 10];
    yield 100;
}
```



**Pour appeler la fonction**

```
foreach (generer_nombre() as  $value) {
    echo "$value ";
}

/* affiche 0 1 3 6 10 100 */
```

# PHP

## En PHP, une fonction utilise ses variables locales

```
function afficherNom()
{
    $nom = "doe";
    echo $nom;
}

$nom = "wick";
afficherNom();
// affiche doe
```

# PHP

**Une fonction peut accéder aux variables définies dans le contexte global en utilisant la variable super-globale `$GLOBALS` (tableau associatif)**

```
function afficherNom()
{
    $nom = "doe";
    echo $GLOBALS['nom'];
}

$nom = "wick";
afficherNom();
// affiche wick
```

# PHP

## PHP Math

**API** de **PHP CORE** contenant plusieurs fonctions permettant de réaliser des opérations mathématiques sur les nombres...

## Exemple de fonctions calculant l'arrondi

```php
echo round(2.1);
/* affiche 2 */

echo round(2.9);
/* retourne 3 */

echo ceil(2.1);
/* retourne 3 */

echo ceil(2.9);
/* retourne 3 */

echo floor(2.1);
/* retourne 2 */

echo floor(2.9);
/* retourne 2 */
```

# PHP

## Autres fonctions

```php
echo pow(2, 3);
/* retourne 8 */

echo sqrt(4);
/* retourne 2 */

echo abs(-2);
/* retourne 2 */

echo min(0, 1, 4, 2, -4, -5);
/* retourne -5 */

echo max(0, 1, 4, 2, -4, -5);
/* retourne 4 */
```

# PHP

**Pour la génération d'un nombre aléatoire**

```
echo rand(10, 100);
// retourne un nombre aléatoire compris entre 10 et 100
```

# PHP

**Pour la génération d'un nombre aléatoire**

```php
echo rand(10, 100);
// retourne un nombre aléatoire compris entre 10 et 100
```



**Pour la génération d'une valeur aléatoire à partir d'un tableau**

```php
$tableau = [10, 2, 3, 7, 4];

// Sélection aléatoire d'une clé
$cleAleatoire = array_rand($tableau);

// Accéder à la valeur correspondante
$valeurAleatoire = $tableau[$cleAleatoire];

// Affichage de la valeur aléatoire
echo $valeurAleatoire;
```

## Exercice

Écrire une fonction `equation_second_degre(a, b, c)` qui permet de résoudre une équation de second degré ($ax^2 + bx + c = 0$)

- $\Delta = b^2 - 4ac$
- Si $\Delta < 0$ : pas de solution
- Si $\Delta = 0$ : une solution $-b/2a$
- Si $\Delta > 0$ : deux solutions $(-b - \sqrt{\Delta})/2a$ et $(-b + \sqrt{\Delta})/2a$



### Résultat attendu

```
print_r(equation_second_degre(1, 1, -2));
// affiche Array([0] => -2 [1] => 1)

echo (equation_second_degre(1, 2, 1));
// affiche -1

echo (equation_second_degre(2, 3, 5));
// affiche pas de solution
```

# PHP

## PHP Date

**API** de **PHP CORE** contenant plusieurs fonctions permettant de travailler sur les dates...

# PHP

## Créer une date en **PHP**

- `date($format)` : retourne une chaîne formatée représentant la date et l'heure.
- `date_create()` et `new DateTime()` : retournent un objet `DateTime`, permettant des manipulations avancées.

# PHP

**Pour créer et retourner la date du jour (type de retour : `string`)**

```php
$format = "d/m/Y H:i:s";
$ma_date = date($format);
echo "\n" . $ma_date;
// affiche la date du jour selon le format passé en paramètre
```

# PHP

**Pour créer et retourner la date du jour (type de retour : `string`)**

```php
$format = "d/m/Y H:i:s";
$ma_date = date($format);
echo "\n" . $ma_date;
// affiche la date du jour selon le format passé en paramètre
```

**Pour avoir une heure respectant le fuseau horaire de Paris**

```php
date_default_timezone_set('Europe/Paris');
$ma_date = date($format);
echo "\n" . $ma_date;
// affiche la date du jour avec l'heure du fuseau horaire choisi
```

# PHP

## Quelques autres indices pour les dates

- `d` : le jour du mois (de 01 à 31)
- `D` : une représentation textuelle du jour (trois lettres)
- `j` : le jour du mois sans zéro (de 1 à 31)
- `w` : une représentation numérique du jour (0 pour dimanche, 6 pour samedi)
- `z` : le jour de l'année (de 0 à 365)
- `F` : une représentation textuelle complète du mois (January à December)
- `m` : une représentation numérique du mois (de 01 à 12)
- `M` : une représentation textuelle courte du mois (trois lettres)
- `n` : une représentation numérique du mois, sans le zéro au début (de 1 à 12)
- `Y` : une représentation numérique de l'année (4 chiffres)
- `y` : une représentation numérique de l'année (2 chiffres)

# PHP

## Quelques autres indices pour les heures

- `a` : `am` ou `pm` en minuscule
- `A` : `AM` ou `PM` en majuscule
- `g` : format d'heure de 1 à 12
- `G` : format d'heure de 0 à 23
- `h` : format d'heure de 01 à 12
- `H` : format d'heure de 00 à 23
- `i` : minutes avec un zéro au début (de 00 à 59)
- `s` : secondes avec un zéro au début (de 00 à 59)
- `u` : microsecondes (depuis PHP 5.2.2)
- `e` : fuseau horaire (exemples : UTC, GMT, Atlantic/Azores)
- `T` : abréviations du fuseau horaire (exemples : EST, MDT)

## Pour récupérer le mois actuel

```
echo date("m");
```

**Pour récupérer le mois actuel**

```php
echo date("m");
```

**Pour créer une date à partir d'une chaîne de caractères (type de retour : `string`)**

```php
$ma_date = date("d-m-Y H:i:s", strtotime("30-07-2015 10:30:11"));
echo $ma_date;
// affiche 30-07-2015 10:30:11
```

**Pour récupérer le mois actuel**

```php
echo date("m");
```

**Pour créer une date à partir d'une chaîne de caractères (type de retour : `string`)**

```php
$ma_date = date("d-m-Y H:i:s", strtotime("30-07-2015 10:30:11"));
echo $ma_date;
// affiche 30-07-2015 10:30:11
```

**Pour générer des dates dans le futur ou dans le passé (type de retour : `integer`)**

```php
$hier = mktime(0, 0, 0, date("m"), date("d") - 1, date("Y"));
echo date("d-M-Y", $hier);
// affiche 16-Mar-2020

$demain = mktime(0, 0, 0, date("m"), date("d") + 1, date("Y"));
echo date("d-M-Y", $demain);
// affiche 18-Mar-2020
```

# PHP

## Remarque

- `mktime` retourne un timestamp (entier) correspondant au nombre de secondes écoulées depuis le début de l'époque UNIX (01/01/1970 00 :00 :00 GMT) et le temps passé en paramètre.
- `time()` retourne le timestamp UNIX actuel

**Pour obtenir une date en français, on peut utiliser la classe `IntlDateFormatter` (`Intl` : Internationalisation))**

```php
$fmt = datefmt_create(
    'fr_FR',
    IntlDateFormatter::FULL,
    IntlDateFormatter::FULL,
    'Europe/Paris',
    IntlDateFormatter::GREGORIAN
);
echo datefmt_format($fmt, 0);
// jeudi 1 janvier 1970 à 01:00:00 heure normale d'Europe centrale
```

**Pour obtenir une date en français, on peut utiliser la classe `IntlDateFormatter` (`Intl` : Internationalisation))**

```php
$fmt = datefmt_create(
    'fr_FR',
    IntlDateFormatter::FULL,
    IntlDateFormatter::FULL,
    'Europe/Paris',
    IntlDateFormatter::GREGORIAN
);
echo datefmt_format($fmt, 0);
// jeudi 1 janvier 1970 à 01:00:00 heure normale d'Europe centrale
```



## Paramètres de la fonction `datefmt_create` dans l'ordre

- `$locale`
- `$dateType`
- `$timeType`
- `$timezone`
- `$calendar`
- `$pattern`

# PHP

**Pour avoir la date du jour en français**

```
$fmt = datefmt_create(
     'fr_FR',
     IntlDateFormatter::FULL,
     IntlDateFormatter::FULL,
     'Europe/Paris',
     IntlDateFormatter::GREGORIAN
);
echo datefmt_format($fmt, time());
// affiche la date du jour
```

# PHP

**Pour masquer la date (respectivement l'heure), mettez `NONE` pour le deuxième (resp. le troisième) paramètre**

```
$fmt = datefmt_create(
     'fr_FR',
     IntlDateFormatter::NONE,
     IntlDateFormatter::FULL,
     'Europe/Paris',
     IntlDateFormatter::GREGORIAN
);
echo datefmt_format($fmt, 0);
// affiche 01:00:00 heure normale d'Europe centrale
```

# PHP

## Autres constantes de `IntlDateFormatter`

- `NONE` pour masque
- `FULL` pour un affichage détaillé
  Exemple : `jeudi 1 janvier 1970 à 01:00:00 heure normale d'Europe centrale`
- `LONG` sans le jour pour la date et sans détails sur le fuseau horaire
  Exemple : `1 janvier 1970 à 01:00:00 UTC+1`
- `SHORT` pour afficher la date au format `jj/mm/aaaa` et date `hh:mm`
  Exemple : `01/01/1970 01:00`
- `MEDIUM` pour utiliser les abréviations
  Exemple : `1 janv. 1970, 01:00:00`

# PHP

**Pour avoir des mots comme (aujourd'hui, demain, hier...) dans la date on peut utiliser `RELATIVE_FULL` (uniquement pour `dateType`)**

```
$fmt = datefmt_create(
     'fr_FR',
     IntlDateFormatter::RELATIVE_FULL,
     IntlDateFormatter::FULL,
     'Europe/Paris',
     IntlDateFormatter::GREGORIAN
);
echo datefmt_format($fmt, time()) . '<br>';
// affiche aujourd'hui à 11:33:42 heure normale d'Europe
   centrale
```

# PHP

**On peut aussi spécifier un motif**

```
$fmt = datefmt_create(
     'fr_FR',
     IntlDateFormatter::RELATIVE_FULL,
     IntlDateFormatter::FULL,
     'Europe/Paris',
     IntlDateFormatter::GREGORIAN,
     'MM/dd/yyyy'
);
echo datefmt_format($fmt, time()) . '<br>';
// affiche 03/15/2022
```

# PHP

La fonction `strftime` est dépréciée depuis **PHP 8.1**.

# PHP

**Exemple avec `checkdate` qui retourne `true` si la date passée en paramètre est correcte, `false` sinon.**

```php
var_dump(checkdate(1, 31, -400));
/* affiche false */

var_dump(checkdate(2, 29, 2019));
/* affiche false */

var_dump(checkdate(2, 29, 2020));
/* affiche true */

var_dump(checkdate(30, 7, 2004));
/* affiche false */
```

# PHP

## Fonctions pour les dates (`DateTime`)

- `date_create($time, $timezone)` : crée et retourne un objet `DateTime` à partir d'une chaîne de caractères et, éventuellement, d'un fuseau horaire.
- `date_modify($datetime, $modifier)` : modifie un objet `DateTime` en fonction d'une expression en anglais (ex. `'+1 day'`).
- `date_format($datetime, $format)` : retourne une chaîne formatée à partir d'un objet `DateTime` et d'un format donné.
- `date_diff($datetime1, $datetime2)` : retourne la différence entre deux dates sous forme d'un objet `DateInterval`.
- `date_timestamp_get($datetime)` : retourne le timestamp d'un objet `DateTime` passé en paramètre.
- ...

# PHP

**Attention**

En **PHP**, les opérateurs +, −, < et > ne peuvent pas être utilisés directement avec les dates.
Utilisez les méthodes dédiées comme `date_diff()` ou `modify()`.

# PHP

**Exemple avec** `date_create` **et** `date_format` **(Type de retour :** `objet`**)**

```php
$date = date_create();
echo date_format($date, "d-m-Y H:i:s");
// affiche la date du jour avec un décalage d'une heure
```

# PHP

**Exemple avec `date_create` et `date_format` (Type de retour : `objet`)**

```
$date = date_create();
echo date_format($date, "d-m-Y H:i:s");
// affiche la date du jour avec un décalage d'une heure
```

**Pour avoir l'heure exacte, il faut préciser le fuseau horaire**

```
$date = date_create("", timezone_open("Europe/Paris"));
echo date_format($date, "d-m-Y H:i:s");
// affiche les date et heure exactes
```

# PHP

**Exemple avec `date_create` et `date_format` (Type de retour : `objet`)**

```
$date = date_create();
echo date_format($date, "d-m-Y H:i:s");
// affiche la date du jour avec un décalage d'une heure
```

**Pour avoir l'heure exacte, il faut préciser le fuseau horaire**

```
$date = date_create("", timezone_open("Europe/Paris"));
echo date_format($date, "d-m-Y H:i:s");
// affiche les date et heure exactes
```

**On peut aussi utiliser `date_create` pour créer une DateTime d'une autre date**

```
$date = date_create("2018-07-30 21:30:10");
// obligatoirement année-mois-jour
echo date_format($date, "d-m-Y H:i:s");
// affiche 30-07-2018 21:30:10
```

# PHP

## Exemple avec `date_diff`

```php
$date1 = date_create("2017-03-29");
$date2 = date_create("2017-07-30");
$diff = date_diff($date1, $date2);

echo ($diff->days);
/* affiche 123 */
```

# PHP

## Exemple avec `date_diff`

```php
$date1 = date_create("2017-03-29");
$date2 = date_create("2017-07-30");
$diff = date_diff($date1, $date2);

echo ($diff->days);
/* affiche 123 */
```



## Exemples avec `date_timestamp_get`

```php
$date = date_create("2018-01-01");
echo date_timestamp_get($date);
// affiche  1514764800
```

## Exemple avec `date_modify`

```php
$date = date_create("2018-01-01");
echo date_format($date,"d-m-Y H:i:s") . "<br>";
/* affiche 01-01-2018 00:00:00 */

date_modify($date,"+15 days");
echo date_format($date,"d-m-Y H:i:s") . "<br>";
/* affiche 16-01-2018 00:00:00 */

date_modify($date,"-1 week");
echo date_format($date,"d-m-Y H:i:s") . "<br>";
/* affiche 09-01-2018 00:00:00 */

date_modify($date,"+3 months");
echo date_format($date,"d-m-Y H:i:s") . "<br>";
/* affiche 09-04-2018 00:00:00 */

date_modify($date,"24 hours 1 minute");
echo date_format($date,"d-m-Y H:i:s") . "<br>";
/* affiche 10-04-2018 00:01:00 */
```

# PHP

**On peut aussi créer une DateTime en utilisant par la classe**

```php
$date = new DateTime('2008-07-30');
echo $date->format('d-m-Y H:i:s');
// affiche 30-07-2008 00:00:00
```

# PHP

**On peut aussi créer une DateTime en utilisant par la classe**

```
$date = new DateTime('2008-07-30');
echo $date->format('d-m-Y H:i:s');
// affiche 30-07-2008 00:00:00
```

**L'équivalent en programmation procédurale du code précédent**

```
$date = date_create('2008-07-30');
echo date_format($date, 'Y-m-d H:i:s');
// affiche 30-07-2008 00:00:00
```

# PHP

**On peut utiliser `date_diff` pour calculer la différence entre deux DateTime créée de deux manières différentes**

```php
$date1 = date_create("2017-03-29");
$date2 = new DateTime('2017-07-30');
$diff = date_diff($date1, $date2);

echo ($diff->days);
/* affiche 123 */
```

# PHP

**On peut utiliser `date_diff` pour calculer la différence entre deux DateTime créée de deux manières différentes**

```php
$date1 = date_create("2017-03-29");
$date2 = new DateTime('2017-07-30');
$diff = date_diff($date1, $date2);

echo ($diff->days);
/* affiche 123 */
```



**Ou aussi**

```php
$date1 = date_create("2017-03-29");
$date2 = new DateTime('2017-07-30');
$diff = $date2->diff($date1);

echo ($diff->days);
/* affiche 123 */
```

# PHP

**Étant données les trois variables suivantes**

```
$mois =  12;
$jour = 31;
$annee = 2019;
```

# PHP

**Étant données les trois variables suivantes**

```
$mois =  12;
$jour = 31;
$annee = 2019;
```

## Exercice

Écrivez un script **PHP** qui permet de vérifier si les trois variables précédentes forment une date valide. Si oui, le script affiche la différence en nombre de jours entre cette date et la date courante.

# PHP

## Correction

```php
$mois = $_GET['mois'] ?? 12;
$jour = $_GET['jour'] ?? 31;
$annee = $_GET['annee'] ?? 2019;

if (checkdate($mois, $jour, $annee)) {
    $date1 = date_create("$annee-$mois-$jour");
    $date2 = new DateTime();
    $diff = $date2->diff($date1);
    echo $diff->days;
} else {
    echo "date incorrecte";
}
```

# PHP

**La fonction `date_create` accepte un seul format de date. On peut utiliser la fonction `date_create_from_format` qui accepte un format personnalisé**

```
$motif = "d/m/Y H:i:s";
$date = date_create_from_format($motif ,'10/04/2020 08:15:00');
```

# PHP

**La fonction `date_create` accepte un seul format de date. On peut utiliser la fonction `date_create_from_format` qui accepte un format personnalisé**

```
$motif = "d/m/Y H:i:s";
$date = date_create_from_format($motif ,'10/04/2020 08:15:00');
```

**Ou la version objet**

```
$motif = "d/m/Y H:i:s";
$date = DateTime::createFromFormat($motif ,'10/04/2020 08:15:00');
```

# PHP

**La méthode `format()` est utilisée pour formater un objet `DateTime` en une chaîne de caractères selon un format spécifié**

```php
$date1 = date_create("2017-03-29");

echo "\n" . $date1->format("y");
// affiche 17

echo "\n" . $date1->format("Y");
// affiche 2017

echo "\n" . $date1->format("m");
// affiche 03

echo "\n" . $date1->format("d");
// affiche 29

echo "\n" . $date1->format("H");
// affiche 00

echo "\n" . $date1->format("L");
// affiche 0 car l'année n'est pas bissextile
```

# PHP

**Attention**

La méthode `format()` de la classe `DateTime` utilise les mêmes formats que la fonction `date()` en **PHP**.

# PHP

**Pour modifier une date, on peut aussi utiliser les méthodes `add` ou `sub` qui prennent comme paramètre `DateInterval`**

```php
$date = new DateTime('2025-02-05');

$date->sub(new DateInterval('P1M'));
// Soustrait 1 mois

echo $date->format('Y-m-d');
// affiche : 2025-01-05
```

# PHP

## L'objet `DateInterval` utilise un format spécifique :

- `P` = "Period" (Période)
- `Y` = Années
- `M` = Mois
- `D` = Jours
- `T` = "Time" (Temps)
- `H` = Heures
- `M` = Minutes
- `S` = Secondes

# PHP

## Exemples de formats valides

- `P1Y` : 1 an
- `P2M` : 2 mois
- `P10D` : 10 jours
- `P1Y2M10D` :1 an, 2 mois et 10 jours
- `PT5H30M` : 5 heures et 30 minutes
- `P1Y2M10DT5H30M` : 1 an, 2 mois, 10 jours, 5 heures et 30 minutes

# PHP

## Exemples

```php
$date = new DateTime('2025-02-05');

// pour jouter 2 mois et 10 jours
$date->add(new DateInterval('P2M10D'));

echo $date->format('Y-m-d');
// affiche 2025-04-15

// pour soustraire 1 an et 5 heures
$date->sub(new DateInterval('P1YT5H'));

echo $date->format('Y-m-d H:i');
//affiche  2024-04-15 19:00
```