

# Les expressions régulières

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Découvrir les expressions régulières</b>	<b>3</b>
A. Principe des expressions régulières .....	3
B. Fonctionnement des expressions régulières .....	4
C. Exercice : Quiz .....	8
<b>III. Les fonctions des expressions régulières</b>	<b>9</b>
A. Les fonctions des expressions régulières .....	9
B. Exercice : Quiz .....	16
<b>IV. Essentiel</b>	<b>17</b>
<b>V. Auto-évaluation</b>	<b>17</b>
A. Exercice .....	17
B. Test .....	18
<b>Solutions des exercices</b>	<b>19</b>

## I. Contexte

**Durée : 1 h**

**Environnement de travail : Replit**

**Prérequis : aucun**

### Contexte

Les expressions régulières sont un outil extrêmement utile pour la manipulation de chaînes de caractères dans les langages de programmation, et PHP ne fait pas exception. En PHP, les expressions régulières vous permettent de rechercher, remplacer, extraire et manipuler des motifs de caractères complexes dans des chaînes de texte. Que vous travailliez sur la validation de données utilisateur, la recherche de chaînes de caractères dans des fichiers ou la transformation de données, les expressions régulières en PHP vous offrent une grande flexibilité et une puissance inégalée.

Au début de ce cours, nous vous présenterons des généralités pour appréhender le concept des expressions régulières. Ensuite, nous entrerons dans le détail du fonctionnement, qui consiste essentiellement à découvrir la syntaxe et les différents outils à disposition. Ensuite, dans la seconde partie du cours, nous vous présenterons les fonctions à utiliser avec les expressions régulières. Elles seront illustrées d'exemples concrets et expliquées en vidéos. Enfin, pour parfaire à votre découverte des expressions régulières, vous vous entraînerez avec des quiz et avec un cas pratique.

## II. Découvrir les expressions régulières

### A. Principe des expressions régulières

#### Introduction historique & références

La notion d'expression régulière a été largement mise en œuvre dans le langage de programmation Perl. Par la suite, un ensemble d'outils a été créé pour utiliser les expressions régulières dans les autres langages de programmation. Il s'agit de PCRE, acronyme de Perl-Compatible Regular Expression. Ce cours présente les bases de l'utilisation de PCRE sous PHP. Il est impossible d'être exhaustif dans ce cours, tant le champ des expressions régulières est vaste. Pour tout complément d'information, vous pourrez vous référer à la documentation officielle : [pcre.org](http://www.pcre.org/)<sup>1</sup>

#### Usages des expressions régulières

Les expressions régulières servent à travailler sur des données textuelles, c'est-à-dire sur des chaînes de caractères. Il peut s'agir d'une simple phrase, ou de millions de phrases. Bien sûr, les exemples de ce cours sont succincts, mais gardez à l'esprit que la mise en œuvre pourrait concerner des milliers de phrases. C'est surtout dans ce contexte de grande quantité de texte que les expressions régulières exploitent toute leur puissance. Elles permettent de rechercher un extrait à l'intérieur d'un texte. À partir de cette capacité de recherche, PCRE fournit des fonctions sophistiquées qui permettent, par exemple, d'extraire différents extraits du texte qui correspondent aux critères de recherche. Encore plus fort, il est possible de remplacer automatiquement une partie de ces extraits. Nous verrons des exemples concrets dans la deuxième partie de ce cours.

### Définition

#### Vocabulaire des expressions régulières

Une expression régulière définit des critères de recherche pour établir des correspondances dans un texte. On utilise le terme « *masque* » pour désigner une expression régulière, ou encore les critères de recherche. Imaginez un masque qui se superpose sur un texte, en cachant des morceaux du texte, et en faisant apparaître d'autres morceaux. Nous pouvons utiliser le terme « *motif* » pour désigner, à l'image d'un vêtement ou d'une tapisserie, un

---

<sup>1</sup> <http://www.pcre.org/>

élément visuel répété. Par exemple, je dirai que la lettre « l » est le motif commun aux mots « *il, elle, lui, le, la, les* ». Le motif révèle une correspondance du masque sur le texte, c'est-à-dire la partie apparente. Un autre exemple, les mots « *pain, entraînement, main, migraine* » contiennent tous le motif « *ain* ».

### Exemple Recherche simple par lettres

Le cas le plus simple pour définir une expression régulière est lorsque le masque est identique au motif. Par exemple, avec le masque « *cou* », une recherche dans un texte peut afficher les mots « *beaucoup, coucou, couper, courte* », c'est-à-dire les mots qui contiennent le motif « *cou* ». Un deuxième exemple, avec le masque « *petit garçon* », on peut révéler les phrases d'un texte qui contiennent ce motif « *petit garçon* », comme :

- Le petit garçon dort.
- Un petit garçon court.
- On voit un petit garçon.
- Où est ton petit garçon ?

### Les options de recherche

Dans notre alphabet, chaque lettre possède deux formes : majuscule et minuscule. Cette distinction se nomme la casse, terme issu de la typographie. Dans l'exemple précédent, les extraits « *Un petit garçon* » et « *un petit garçon* » sont différents, car la première lettre n'est pas de la même casse. On parle alors de sensibilité à la casse. Cependant, il est possible de rechercher un texte sans tenir compte de la casse. On parle alors d'insensibilité à la casse. Ce critère de recherche est précisé avec l'option « *i* », initiale de « *insensible à la casse* ».

Dans les faits, un texte est composé de phrases, de paragraphes, voire de chapitres. Il existe donc des retours à la ligne et des sauts de ligne. Il peut être nécessaire d'exploiter ces retours à la ligne. Imaginez, par exemple, devoir extraire d'un texte chaque premier mot de tous les paragraphes. Au contraire, dans d'autres cas, on préférerait ne pas tenir compte de ces retours à la ligne et s'intéresser qu'aux caractères visibles. En fait, la recherche dans un texte considère par défaut ce texte comme une seule ligne. C'est un peu comme si les retours à la ligne n'étaient pas interprétés : il n'y aurait pas de paragraphes et pas de chapitres, juste une seule ligne. Pour activer la recherche en mode « *multilignes* », il faut utiliser l'option « *m* ».

Nous verrons un peu plus loin que le caractère dollar « *\$* » possède une propriété spéciale dans les expressions régulières : il permet de rechercher à la fin d'une ligne. Par défaut, le caractère *\$* a pour effet de rechercher à la fin de toutes les lignes, comprendre au sens de paragraphe. C'est comme si l'option « *m* » était activée. Si on veut rechercher uniquement à la fin du texte, on doit spécifier l'option « *D* », comme « *Dollar* ».

Nous verrons également un peu plus loin que le caractère point « *.* » possède une propriété spéciale dans les expressions régulières : il permet de remplacer n'importe quel caractère, mais pas les retours à la ligne. Si l'on souhaite que le caractère point considère aussi les retours à la ligne, il convient d'utiliser l'option « *s* », qui signifie « *single line* ».

## B. Fonctionnement des expressions régulières

### Définition

Comme nous l'avons dit précédemment, il est possible de rechercher des mots en début de ligne, et respectivement en fin de ligne. On dit alors que le masque de recherche est ancré, au début ou à la fin. Il existe donc deux types d'ancres : celle placée au début du masque et celle placée à la fin du masque. L'ancre du début se symbolise avec le caractère circonflexe « *^* », tandis que l'ancre de fin se symbolise avec le caractère dollar « *\$* ».

**Exemple Utiliser des ancrs**

Le masque “`^un lapin`” signifie que l’extrait « *un lapin* » doit être placé en début de ligne. Ainsi ce masque correspond avec la phrase « *un lapin saute dans le jardin* », mais ne correspond pas avec la phrase « *je vois un lapin dans le jardin* ».

Le masque “`le jardin$`” signifie que l’extrait « *le jardin* » doit être placé en fin de ligne. Ainsi ce masque correspond avec la phrase « *je cours dans le jardin* », mais ne correspond pas avec la phrase « *le jardin est en fleurs* ».

**Définition Les alternatives**

Les alternatives offrent plusieurs possibilités. Elles peuvent correspondre lors de choix. Elles se notent avec le caractère barre droite “|”, en utilisant les touches “AltGr + 6”. Par exemple, le masque “`garçon|fille`” correspond avec toutes les phrases qui contiennent le mot « *fille* » ou le mot “*garçon*”.

**Définition Les sous masques**

Un sous masque est une partie du masque située entre parenthèses. On peut par exemple utiliser un sous masque pour exprimer une alternative au sein d’un masque plus complexe. Par exemple, le masque “`(père|mère) Noël`” correspond avec les extraits “*père Noël*” et les extraits “*mère Noël*”. Ainsi, ce masque peut afficher les phrases « *la mère Noël emballe les cadeaux* » et « *le père Noël nourrit ses rennes* ».

**Définition Le point et les quantificateurs**

Le caractère point “.” possède la propriété spéciale d’être un joker, c’est-à-dire de correspondre avec n’importe quel caractère. Par exemple, le masque “`l.s`” peut correspondre avec les mots “*las, les, lus, lys*”.

Les quantificateurs permettent d’indiquer aisément la répétition d’un motif dans le masque. Pour indiquer cette répétition, on précise les quantités entre accolades. Il est possible d’indiquer une valeur exacte, ou une borne de valeur, ou deux bornes de valeurs. Enfin, pour les cas les plus courants, il existe une syntaxe courte avec les 3 caractères suivants : “\*”, “+”, “?”. Voyons de suite des exemples concrets.

**Exemple Utiliser le point et les quantificateurs**

Le masque “`(cou){2}`” signifie : le sous masque “*cou*” exactement deux fois à la suite. Cela correspond avec le mot « *coucou* ».

Le masque “`(la){2,4}`” signifie : le sous masque “*la*” présent à la suite entre 2 et 4 fois. Cela correspond par exemple avec la phrase « *On chante : lalalala !* ».

Le masque “`z{1,}`” signifie : la lettre “*z*” présente au moins une fois, mais aussi plusieurs fois de suite, sans préciser de maximum. Cela correspond par exemple avec la phrase « *on ronfle : zzzzz...* ».

Le masque “`{20,}`” signifie : une suite d’au moins 20 lettres. Cela correspond donc avec tous les mots de plus de 19 lettres.

Le quantificateur “?” est l’équivalent de “`{0,1}`”. Par exemple, le masque “`(jolie)? voiture`” correspond aussi bien avec la phrase « *une voiture passe* » qu’avec la phrase « *une jolie voiture passe* », car le sous masque « *jolie* » peut être présent 0 ou 1 fois.

Le quantificateur “\*” est l’équivalent de “`{0,}`”. Par exemple, le masque “`vie*nt`” correspond avec “*vint*”, et “*vient*”, tout comme “*viieeeent*”.

Le quantificateur “+” est l’équivalent de “`{1,}`”. Par exemple, le masque “`bou+h`” correspond avec la phrase “*il pleura : bouuuuuuh...*”, car la lettre “*u*” est présente plusieurs fois, et au moins une fois.

## Définition Les classes

Une classe est un groupe de caractères indiqué entre crochets “[” et “]”. Les classes offrent une syntaxe courte, pour ce qui pourrait être précisé de façon beaucoup plus longue. Leur usage facilite donc l’écriture des masques et leur lecture pour les comprendre plus aisément. Imaginez, par exemple, un masque qui correspond avec n’importe quelle voyelle. Il pourrait s’écrire “(a|e|i|o|u|y)”, mais il peut aussi s’écrire “[aeiouy]”.

Il existe 4 façons d’utiliser les classes :

- En indiquant explicitement les caractères permis, comme dans l’exemple précédent.
- En indiquant une plage de caractères, avec le symbole tiret “-” entre le premier caractère de la plage et le dernier caractère de la plage. Par exemple, le masque “[a-z]” correspond à une lettre parmi toutes les lettres minuscules.
- En indiquant un mot-clé spécifique, appelé “nom de classe”, entouré encore de crochets et de “:”.
- En indiquant les caractères masquants, en commençant par le symbole circonflexe “^”.

Il existe 14 noms de classe. Nous vous présentons ici les plus courants pour vous expliquer leur usage :

- “alpha” : correspond aux lettres
- “digit” : correspond aux chiffres
- “alphanum” : correspond aux lettres et aux chiffres
- “blank” : correspond aux espaces
- “lower” : correspond aux lettres minuscules
- “upper” : correspond aux lettres majuscules
- “word” : correspond aux mots

## Exemple Utiliser les classes

Le masque “gr[aèio]s” correspond avec les mots « gras, grès, gris, gros ».

Le masque “[0-9]” correspond à un seul chiffre, et n’importe quel chiffre. Ce masque est identique au masque “[[:digit:]]”. Le masque “[0-9]” correspond à tous les nombres (avec au moins un chiffre).

Le masque “[^0-9]” est le contraire d’un chiffre, et correspond à ce qui ne contient pas que des chiffres. Par exemple, le motif “123!” correspond grâce au point d’exclamation.

Le masque “[a-zA-Z]” correspond avec tous les mots, qu’ils contiennent des majuscules ou des minuscules. Ce masque correspond donc, par exemple, avec les mots « HELLO, hello, Hello ». Ce masque est identique au masque “[[:alpha:]]”. Notez que dans notre langue française, nous utilisons des lettres accentuées ; ces masques s’appuient donc sur la configuration de PHP, et donneront des résultats différents avec les mêmes données sur un serveur américain par exemple.

## Les séquences d’échappement 1/4 - le cas littéral

Comme vous l’avez vu depuis le début de ce cours, certains caractères du masque activent un comportement spécial : par exemple, les quantificateurs « \* », « + », « ? ». Il en est de même pour les parenthèses (indiquant les alternatives), les accolades (indiquant les répétitions), les crochets (indiquant les classes). Imaginez maintenant que vous souhaitez rechercher dans un texte, l’extrait exact “[a]”, c’est-à-dire de façon littérale. Les crochets ne doivent

donc pas être interprétés comme une classe. Il est alors nécessaire d'échapper à l'interprétation. Pour cela, il convient d'utiliser le caractère d'échappement qui est la barre oblique inverse "\", aussi appelée antislash, devant chaque caractère à échapper. Le masque de cet exemple s'écrit donc "

*a*

". Ainsi, pour rechercher de manière littérale un méta-caractère (un caractère qui définit un comportement spécial), il convient systématiquement d'utiliser ce caractère "\". On utilise donc les masques :

- "\+", pour rechercher le caractère plus
- "\\*", pour rechercher le caractère étoile
- "\?", pour rechercher le caractère point d'interrogation
- "\", pour rechercher le caractère antislash
- "\.", pour rechercher le caractère point
- "\\$", pour rechercher le caractère dollar
- "\|", pour rechercher le caractère barre droite
- "\^", pour rechercher le caractère circonflexe

Imaginez maintenant que vous souhaitiez rechercher exactement, c'est-à-dire littéralement, la suite de caractères "`^+*?\. $[](){}0`". Avec ce que nous venons de dire, il conviendrait d'ajouter un caractère "\" devant chacun des caractères (sauf le 0). C'est possible, mais ce serait compliqué à lire. Nous disposons d'une autre méthode pour rechercher de manière littérale, en utilisant "`\Q...E`" où les points de suspension sont à remplacer par l'expression littérale. Le début "`\Q`" signifie "*Quoting*", qui introduit donc une citation. La fin "`E`" signifie "*End*" qui termine la citation. Dans notre exemple, le masque peut donc s'écrire "`\Q^+*?\. $[](){}0\E`".

### Les séquences d'échappement 2/4 - le cas des caractères invisibles

Il existe plusieurs caractères non imprimables, par exemple les retours à la ligne et les tabulations.

Les caractères de retour à la ligne sont toujours à prendre avec précaution, car ils diffèrent selon l'environnement : pour Windows ils sont `\r\n`, sous Linux ils sont `\n`, et pour Appel ils sont `\r`. Heureusement, PCRE propose `\R` pour considérer les retours à la ligne, quel que soit le système d'exploitation.

Le caractère de tabulation est toujours noté `\t`.

### Les séquences d'échappement 3/4 - le cas des types génériques

Il existe une syntaxe raccourcie pour les classes les plus courantes. C'est un gain d'écriture et de lecture. Voici les équivalences :

- "`\d`" indique un chiffre (digit en anglais), c'est équivalent à "`[0-9]`"
- "`\D`" indique le contraire d'un chiffre, c'est équivalent à "`[^0-9]`"
- "`\w`" indique un mot (word en anglais), c'est équivalent à "`[a-zA-Z0-9_]`"
- "`\W`" indique le contraire d'un mot, c'est équivalent à "`[^a-zA-Z0-9_]`"
- "`\s`" indique un espace blanc (space en anglais), c'est équivalent à "`[\r\n\t\f\v]`"
- "`\S`" indique le contraire d'un espace blanc, c'est équivalent à "`[^\r\n\t\f\v]`"

## Les séquences d'échappement 4/4 - le cas des assertions hors classe

PCRE fournit 6 assertions simples. Nous en présentons ici 4 dont le fonctionnement est similaire aux ancres décrites plus haut :

- “\b” indique une limite de mot, peut être vu un peu comme “\w\W” et “\W\w”
- “\B” indique le contraire d’une limite de mot
- “\A” indique le début du texte, comme l’ancree “^”
- “\Z” indique la fin du texte, comme l’ancree “\$”

### C. Exercice : Quiz

[solution n°1 p.21]

#### Question 1

On considère le texte « *Lundi 13 mai* ». Deux masques sont proposés pour correspondre avec ce texte : “^Lun” et “mai\$”. Ces masques sont corrects.

- ☐ Vrai
- ☐ Faux

#### Question 2

On considère le masque “(Crêpe|Glace) (vanille|chocolat)”. Quel texte ne correspond pas avec ce masque ?

- ☐ Glace vanille
- ☐ Glace chocolat
- ☐ Crêpe au chocolat

#### Question 3

On considère le masque “[A-Z]{0,3}[mbp][0-9]+”. Quel texte ne correspond pas avec ce masque ?

- ☐ CEsP253
- ☐ m1
- ☐ BOb

#### Question 4

On considère le texte « *Quel prix ? 25 \$ ?* ». Quel masque correspond avec ce texte ?

- ☐ “?\$”, car le texte termine par un point d’interrogation
- ☐ “25 \$ ?”, car le texte contient l’extrait “25 \$ ?”
- ☐ “\? 25”, car le texte contient l’extrait “? 25”

#### Question 5

On considère le texte « *14 juillet 1789* ». Quel masque ne correspond pas avec ce texte ?

- ☐ “\d\s\w+\s\d+”
- ☐ “\w\D+\w”
- ☐ “[1-4] \w [1-9]+”



### III. Les fonctions des expressions régulières

#### A. Les fonctions des expressions régulières

##### Les délimiteurs

Pour être utilisés dans les fonctions, les masques doivent être indiqués entre des délimiteurs. Ces délimiteurs peuvent être n'importe quel symbole (ni lettre ni chiffre) : soit le même symbole au début et à la fin, soit un symbole ouvrant au début et son homologue fermant à la fin. Par exemple :

- ~123~ : on utilise le symbole tilde comme délimiteur
- =123= : on utilise le symbole égal comme délimiteur
- /123/ : on utilise le symbole barre oblique (slash), qui est le délimiteur le plus couramment utilisé
- {123} : on utilise les accolades comme délimiteurs
- <123> : on utilise les chevrons comme délimiteurs

##### La fonction preg\_grep

La fonction preg\_grep a besoin de 2 arguments et en accepte 3 :

- Un masque,
- Un tableau, dont chaque élément est comparé au masque,
- Une valeur facultative, par défaut 0, permettant d'inverser les résultats en utilisant la constante PREG\_GREP\_INVERT.

Cette fonction retourne un tableau avec les éléments du tableau passé en argument qui correspondent au masque. Les clés du tableau d'entrée sont conservées dans le tableau de sortie.

##### Méthode Utiliser la fonction preg\_grep

Dans l'exemple ci-dessous, vous verrez un tableau \$entree, contenant 5 chaînes de caractères. Ensuite, 4 masques sont définis et utilisés avec la fonction preg\_grep. Des commentaires ont été ajoutés pour vous aider à comprendre le fonctionnement. Vous êtes invité à exécuter vous-même ces lignes de code. Vous pouvez même modifier les masques avec vos propres expressions régulières, afin de tester ce que vous avez appris dans la première partie de ce cours. Enfin, une vidéo de démonstration vous présentera ces exemples avec Replit.

```
1 <?php
2 $entree = [
3     "bonjour !",
4     "Voici",
5     "un",
6     "tableau",
7     "de 5 éléments."
8 ];
9 $masque1 = "/u/"; # on va rechercher la lettre u
10 $sortie1 = preg_grep($masque1,$entree);
11 # le tableau de sortie contient les éléments
12 # du tableau d'entrée qui contiennent la lettre u
13 print_r($sortie1);
14 /* Affiche :
15 Array
16 (
17     [0] => bonjour !
18     [2] => un
19     [3] => tableau
20 )
```

```

21 Remarquez l'absence de l'élément d'indice 1 (Voici) qui ne contient pas de u
22 */
23 $masque2 = "/^[A-Z]/"; # on va rechercher les éléments commençant par une majuscule
24 $sortie2 = preg_grep($masque2,$entree);
25 # le tableau de sortie contient les éléments
26 # du tableau d'entrée qui commencent par une majuscule
27 print_r($sortie2);
28 /* Affiche :
29 Array
30 (
31     [1] => Voici
32 )
33 */
34 $masque3 = "/\W$/"; # on va rechercher les éléments terminant par un non-mot
35 $sortie3 = preg_grep($masque3,$entree);
36 # le tableau de sortie contient les éléments
37 # du tableau d'entrée qui terminent par un symbole
38 print_r($sortie3);
39 /* Affiche :
40 Array
41 (
42     [0] => bonjour !
43     [4] => de 5 éléments.
44 )
45 Remarquez le dernier caractère sur chaque ligne, qui est un symbole de ponctuation
46 */
47 $masque4 = "/n/"; # on va rechercher les éléments contenant la lettre n
48 $sortie4 = preg_grep($masque4,$entree,PREG_GREP_INVERT);
49 # le tableau de sortie contient les éléments
50 # du tableau d'entrée qui ne contiennent pas la lettre n
51 # Remarquez l'utilisation de l'option d'inversion (le 3ème paramètre de la fonction)
52 print_r($sortie4);
53 /* Affiche :
54 Array
55 (
56     [1] => Voici
57     [3] => tableau
58 )
59 */
60 ?>

```

## La fonction preg\_split

La fonction preg\_split a besoin de 2 arguments et en accepte 4 :

- Un masque,
- Un texte, sur lequel va s'appliquer le masque,
- Un nombre entier facultatif, par défaut à - 1,
- Des options facultatives, par défaut à 0.

Le mot anglais “*split*” se traduit en français par « *diviser* ». La fonction preg\_split va donc découper le texte en plusieurs extraits en fonction du masque. Le résultat de cette « *division* » est un tableau dont chaque élément est un extrait du texte. Le fonctionnement est similaire à celui de la fonction “*explode()*”, mais avec la puissance des expressions régulières.

Le nombre entier facultatif, en 3<sup>ème</sup> argument, permet d'indiquer le nombre maximum de coupes. Par exemple, si 3 est indiqué, la fonction preg\_split va effectuer les 3 premières divisions, et va retourner un tableau avec 4 extraits.

Les options facultatives, qui peuvent se combiner avec l'opérateur "|", sont au nombre de 3 :

- La constante PREG\_SPLIT\_NO\_EMPTY indique que le résultat ne contient pas de chaîne vide.
- La constante PREG\_SPLIT\_DELIM\_CAPTURE permet d'inclure une partie du masque dans le résultat.
- La constante PREG\_SPLIT\_OFFSET\_CAPTURE inclut au résultat la position de chaque extrait dans le texte. Cela modifie la structure du tableau retourné.

#### Méthode Utiliser la fonction preg\_split

Dans l'exemple ci-dessous, nous définissons un texte, puis 4 masques. Nous utilisons ces masques pour découper le texte. Prenez le temps de tester ces exemples et éventuellement de modifier ces masques pour constater le changement des résultats.

```

1 <?php
2 $texte = "Voici un texte ordinaire
3 contenant des numéros de téléphone marqués par des tirets :
4 01-23-45-56-78, 02-34-56-87-91, 03-21-34-65-87";
5 $masque1 = "/ /"; # contient uniquement le caractère d'espace
6 $sortie1 = preg_split($masque1,$texte); # coupe le texte à chaque espace
7 print_r($sortie1);
8 /* Affiche :
9 Array
10 (
11     [0] => Voici
12     [1] => un
13     [2] => texte
14     [3] => ordinaire
15     [4] =>
16 contenant
17     [5] => des
18     [6] => numéros
19     [7] => de
20     [8] => téléphone
21     [9] => marqués
22     [10] => par
23     [11] => des
24     [12] => tirets
25     [13] => :
26     [14] =>
27 01-23-45-56-78,
28     [15] => 02-34-56-87-91,
29     [16] => 03-21-34-65-87
30 )
31 */
32 $masque2 = "/\R| : |, /"; # 3 coupes possibles : retour à la ligne, deux points, et virgule
33 $sortie2 = preg_split($masque2,$texte); # coupe le texte
34 print_r($sortie2);
35 /* Affiche :
36 Array
37 (
38     [0] => Voici un texte ordinaire
39     [1] => contenant des numéros de téléphone marqués par des tirets
40     [2] =>
41     [3] => 01-23-45-56-78
42     [4] => 02-34-56-87-91
43     [5] => 03-21-34-65-87
44 )

```

```

45 */
46 $masque3 = "/[^0-9]/"; # coupe tout ce qui n'est pas chiffre
47 $sortie3 = preg_split($masque3,$texte,-1,PREG_SPLIT_NO_EMPTY); # coupe le texte sans garder
    les chaînes vides
48 print_r($sortie3);
49 /* Affiche :
50 Array
51 (
52     [0] => 01
53     [1] => 23
54     [2] => 45
55     [3] => 56
56     [4] => 78
57     [5] => 02
58     [6] => 34
59     [7] => 56
60     [8] => 87
61     [9] => 91
62     [10] => 03
63     [11] => 21
64     [12] => 34
65     [13] => 65
66     [14] => 87
67 )
68 */
69 $masque4 = "/[^-0-9]/"; # coupe tout ce qui n'est pas tiret ou chiffre
70 $sortie4 = preg_split($masque4,$texte,-1,PREG_SPLIT_NO_EMPTY); # coupe le texte sans garder
    les chaînes vides
71 print_r($sortie4);
72 /* Affiche :
73 Array
74 (
75     [0] => 01-23-45-56-78
76     [1] => 02-34-56-87-91
77     [2] => 03-21-34-65-87
78 )
79 */
80 ?>

```

## Les fonctions preg\_match et preg\_match\_all

Les fonctions preg\_match et preg\_match\_all ont besoin de 2 arguments et en acceptent 5 :

- Un masque,
- Un texte, sur lequel va s'appliquer le masque,
- Une référence facultative d'un tableau qui contiendra les résultats
- Des options facultatives, par défaut à 0,
- Un nombre entier facultatif, par défaut à 0, indiquant la position dans le texte où commencer la recherche.

Ces fonctions effectuent la recherche de correspondances du masque dans le texte. Si le masque correspond dans le texte, les fonctions retournent la valeur 1. Ces fonctions ont un comportement similaire à celui de la fonction strpos(), mais avec la puissance des expressions régulières.

La différence fondamentale entre preg\_match et preg\_match\_all est que la première ne renseigne que la première correspondance, alors que la seconde renseigne toutes les correspondances.

Les options de `preg_match` et `preg_match_all`, qui peuvent se combiner, sont :

- `PREG_OFFSET_CAPTURE`, ajoutant dans les résultats la position de l'extrait dans le texte
- `PREG_UNMATCHED_AS_NULL`, remplaçant les chaînes vides dans le résultat par la valeur null

Les options de `preg_match_all`, qui s'excluent mutuellement, sont `PREG_PATTERN_ORDER` et `PREG_SET_ORDER`. Elles permettent de modifier l'ordre des résultats.

#### Méthode Utiliser les fonctions `preg_match` et `preg_match_all`

Dans l'exemple ci-dessous, on a défini un texte en HTML, donc avec des balises entre chevrons. Nous avons défini un masque pour correspondre avec ces balises. Après avoir testé ces exemples, n'hésitez pas à modifier les options des fonctions afin de voir la différence dans les résultats. Une vidéo de présentation vous montrera le fonctionnement de ces deux fonctions.

```

1 <?php
2 # on définit un texte en HTML :
3 $texte = "<h1>Titre</h1>
4         <p>Un premier paragraphe</p>
5         <p>Un deuxième paragraphe</p>"
6         ;
7 # on définit un masque en utilisant le délimiteur %
8 # ce masque va rechercher les balises HTML
9 $masque = "%<.*?>.*?</.*?>%" ;
10 # par défaut, * et + correspondent avec le maximum de caractères :
11 # on dit qu'ils sont gourmands. Dans ce cas, on ne veut pas * gourmand,
12 # donc on rajoute le caractère ?
13 # on recherche le premier extrait :
14 preg_match($masque,$texte,$resultat);
15 print_r($resultat);
16 /* Affiche :
17 Array
18 (
19     [0] => <h1>Titre</h1>
20 )
21 */
22 # on recherche tous les extraits :
23 preg_match_all($masque,$texte,$resultat);
24 print_r($resultat);
25 /* Affiche :
26 Array
27 (
28     [0] => Array
29         (
30             [0] => <h1>Titre</h1>
31             [1] => <p>Un premier paragraphe</p>
32             [2] => <p>Un deuxième paragraphe</p>
33         )
34 )
35 */
36 # on recherche tous les extraits avec leur position
37 preg_match_all($masque,$texte,$resultat,PREG_OFFSET_CAPTURE);
38 print_r($resultat);
39 /* Affiche :
40 Array
41 (
42     [0] => Array
43         (

```

```

44         [0] => Array
45             (
46                 [0] => <h1>Titre</h1>
47                 [1] => 0
48             )
49         [1] => Array
50             (
51                 [0] => <p>Un premier paragraphe</p>
52                 [1] => 24
53             )
54         [2] => Array
55             (
56                 [0] => <p>Un deuxième paragraphe</p>
57                 [1] => 62
58             )
59     )
60 )
61 */
62 ?>

```

### Les fonctions preg\_replace et preg\_filter

Comme son nom l'indique, la fonction preg\_replace recherche et remplace des extraits dans un texte selon un masque. Cette fonction retourne tout le texte, avec les extraits modifiés. La fonction preg\_filter est quasiment identique à preg\_replace : la seule différence est que preg\_filter ne retourne que les extraits modifiés. C'est un peu comme si preg\_grep était appliqué en même temps que preg\_replace.

Les fonctions preg\_replace et preg\_filter ont besoin de 3 arguments, plus 2 optionnels :

- Un masque de recherche
- Une chaîne de remplacement
- Un texte
- Un nombre entier facultatif, par défaut à - 1, permettant d'indiquer le nombre maximum de remplacements à effectuer
- Une variable facultative, qui contiendra le nombre de remplacements effectués

Ces fonctions fonctionnent aussi bien avec de simples chaînes de caractères qu'avec des tableaux de chaînes. Dans le cas de tableaux, les recherches et remplacements s'effectuent sur l'ensemble des tableaux.

La chaîne de remplacement peut faire référence à des motifs trouvés par les sous masques du masque de recherche. Les sous masques, rappelez-vous, sont indiqués entre parenthèses. Pour y faire référence, nous utilisons le \$ pour signifier utiliser une variable suivie du numéro d'ordre du sous masque. En cas d'ambiguïté, il est possible d'utiliser les accolades comme dans le cas des variables dynamiques.

#### Méthode Utiliser les fonctions preg\_replace et preg\_filter

Plusieurs exemples vous sont présentés ci-dessous. Ils vous sont également expliqués en vidéo. Comme pour les exemples précédents, vous êtes invités à les exécuter, ainsi que tester ces fonctions avec vos propres textes et masques.

```

1 <?php
2 $texte = "J'ai beaucoup de travail.";
3 $recherche = "%(beau)(coup)%"; # on définit 2 sous-masques
4 $remplace = "peu";
5 $resultat = preg_replace($recherche,$remplace,$texte); # on effectue le remplacement
6 print_r($resultat); # on affiche le résultat
7 # Affiche : J'ai peu de travail.

```

```

8 echo "\n";
9 $remplace = "$2$1"; # on fait référence aux sous-masques
10 $resultat = preg_replace($recherche,$remplace,$texte); # on effectue le remplacement
11 print_r($resultat); # on affiche le résultat
12 # Affiche : J'ai coupbeau de travail.
13 echo "\n";
14 # on définit un tableau de textes
15 $texte = [
16     "J'ai beaucoup de travail.",
17     "J'ai beaucoup de vacances.",
18     "J'ai beaucoup d'amis.",
19     "J'ai peu de travail.",
20     "J'ai peu de vacances.",
21     "J'ai peu d'amis.",
22 ];
23 # on définit un tableau de masques de recherche
24 $recherche = [
25     "/vacances/",
26     "/(beaucoup)/"
27 ];
28 # on définit un tableau de chaînes de remplacement
29 $remplace = [
30     "jours de congés",
31     "vraiment $1"
32 ];
33 # donc "vacances" sera remplacé par "jours de congés"
34 # et "beaucoup" sera remplacé par "vraiment beaucoup"
35 $resultat = preg_replace($recherche,$remplace,$texte,-1,$compteur); # on utilise un compteur
36 echo $compteur," remplacements effectués\n";
37 print_r($resultat); # on affiche le résultat
38 /* Affiche :
39 5 remplacements effectués
40 Array
41 (
42     [0] => J'ai vraiment beaucoup de travail.
43     [1] => J'ai vraiment beaucoup de jours de congés.
44     [2] => J'ai vraiment beaucoup d'amis.
45     [3] => J'ai peu de travail.
46     [4] => J'ai peu de jours de congés.
47     [5] => J'ai peu d'amis.
48 )
49 */
50 # "filter" ne retourne que les extraits remplacés :
51 $resultat = preg_filter($recherche,$remplace,$texte); # on effectue le remplacement
52 print_r($resultat); # on affiche le résultat
53 /* Affiche :
54 Array
55 (
56     [0] => J'ai vraiment beaucoup de travail.
57     [1] => J'ai vraiment beaucoup de jours de congés.
58     [2] => J'ai vraiment beaucoup d'amis.
59     [4] => J'ai peu de jours de congés.
60 )
61 */
62 ?>

```

## B. Exercice : Quiz

[solution n°2 p.22]

### Question 1

On considère le code suivant. Combien d'éléments possède le tableau \$sortie ?

```
1 <?php
2 $jours = ["Lun", "Mar", "Mer", "Jeu", "Ven", "Sam", "Dim"];
3 $masque = '/e/';
4 $sortie = preg_grep($masque, $jours);
5 ?>
```

- ☐ 0
- ☐ 3
- ☐ 7

### Question 2

On considère le code suivant. Combien d'éléments possède le tableau \$sortie ?

```
1 <?php
2 $texte = "Atchoum ! Je suis... Atchoum ! enrhumé. C'est... Atchoum ! embêtant.";
3 $masque = "#Atchoum ! #";
4 $sortie = preg_split($masque, $texte, -1, PREG_SPLIT_NO_EMPTY);
5 ?>
```

- ☐ 3
- ☐ 5
- ☐ 7

### Question 3

On considère le code suivant. Combien d'éléments possède le tableau \$resultat ?

```
1 <?php
2 $texte = "Au printemps, les fleurs s'épanouissent.";
3 $masque = "/^[A-Z]([a-z é,']*)\.$/";
4 preg_match($masque, $texte, $resultat);
5 ?>
```

- ☐ 0
- ☐ 1
- ☐ 2

### Question 4

On considère le code suivant. Combien d'éléments possède le tableau \$resultat ?

```
1 <?php
2 $texte = "Au printemps, les fleurs s'épanouissent.";
3 $masque = "/\b(\w{3,6})\b/";
4 preg_match_all($masque, $texte, $resultat);
5 ?>
```

- ☐ 0
- ☐ 2
- ☐ 5



## Question 5

On considère le code suivant. Combien vaut la variable \$compteur ?

```
1 <?php
2 $jours = ["Lun", "Mar", "Mer", "Jeu", "Ven", "Sam", "Dim"];
3 $recherche = ["/^(M.+)/", "/(m|n)$/"];
4 $remplace = ['$1:Journée', '$1:Nuit'];
5 $resultat = preg_filter($recherche, $remplace, $jours, 10, $compteur);
6 ?>
```

- ☐ 2
- ☐ 4
- ☐ 6

## IV. Essentiel

Les expressions régulières permettent de définir des masques. Un masque établit des correspondances sur un texte, afin d'en sélectionner des extraits. Les masques utilisent des caractères de façon littérale, mais aussi des méta-caractères, véritables outils pour critériser les recherches : les ancres, les alternatives, les sous masques, le point et les quantificateurs, les classes, les séquences d'échappement. Les expressions régulières sont utilisées dans des fonctions spécifiques : pour des extractions (grep), du découpage (split), des correspondances (match), et du remplacement (replace). Ces fonctions opèrent sur des tableaux regroupant des textes et des masques, permettant d'effectuer de multiples traitements avec très peu de lignes de code. En apprenant les bases des expressions régulières en PHP, vous serez en mesure de créer des motifs de recherche et de manipulation de texte personnalisés pour répondre à une variété de besoins de traitement de chaînes de caractères.

## V. Auto-évaluation

### A. Exercice

#### Formatage de fichiers téléphonique

Vous travaillez pour une entreprise de démarchage téléphonique. Votre entreprise acquiert régulièrement de nouveaux fichiers de prospection. Chaque nouveau fichier possède son propre format. Vous devez convertir ce fichier, à l'aide des expressions régulières, dans le format utilisé par votre entreprise. Dans ce cas pratique, notre fichier de prospection ne contient que 7 lignes, mais imaginez qu'en situation réelle vous devriez traiter des milliers de lignes toutes les semaines. Le fichier est donné par le code suivant :

```
1 <?php
2 $fichier = ["01.23.45.56.78, MARTIN, PIERRE",
3             "02.56.32.41.87, DURAND, MATHILDE",
4             "03.23.98.87.54, LOTARD, HENRI",
5             "04.32.45.65.10, GEAI, JACQUES",
6             "+333.21.65.98.01, JOYEUX, JEAN-PIERRE",
7             "01.65.87.41.20, TOULIN, MARIE",
8             "03.58.56.21.02, DE BIEN, NOEMIE"];
9 ?>
```

Le format utilisé dans votre entreprise est :

NOM\tPRENOM\t+33X XX XX XX XX

où \t est le caractère de tabulation et les X séparés par des espaces le numéro de téléphone. Remarquez également l'indicatif international + 33.

Pour la prochaine campagne de prospection, vous devez intégrer que les coordonnées des numéros de la zone Nord de la France, c'est-à-dire les numéros commençant par 03 (ou +333).

### Question 1

[solution n°3 p.24]

Expliquez en quelques lignes, quelle fonction vous allez utiliser pour effectuer ce traitement, et comment vous envisagez de définir un masque qui correspond à ce traitement. Pour cela, vous devez bien étudier le format du fichier pour bien le comprendre : par exemple, constatez les séparateurs point et virgule.

### Question 2

[solution n°4 p.24]

Implémentez le traitement que vous avez expliqué ci-dessus, pour répondre au besoin de votre entreprise.

## B. Test

### Exercice 1 : Quiz

[solution n°5 p.24]

#### Question 1

Dans un masque, le caractère circonflexe “^” peut avoir 3 interprétations différentes.

- ☐ Vrai
- ☐ Faux

#### Question 2

On considère le code suivant. Combien d’éléments contient le tableau \$resultat ?

```
1 <?php
2 $texte = ["Avec un nombre de 12",
3           "A l'école ils sont 23",
4           "A cette heure de 10h20",
5           "Après manger",
6           "A4"];
7 $masque = "/A[a-z ]*[0-9]+/";
8 $resultat = preg_grep($masque,$texte);
9 ?>
```

- ☐ 2
- ☐ 3
- ☐ 4

#### Question 3

On considère le code suivant. Quatre éléments sont affichés, lesquels sont-ils ?

```
1 <?php
2 $texte = "A la bataille navale, il a joué : A7, B5, C2, D1.";
3 $masque = "/ ([A-G][1-7])(,|\.)/";
4 preg_match_all($masque,$texte,$resultat);
5 print_r($resultat[1]);
6 ?>
```

- ☐ “A7,” puis “B5,” puis “C2,” puis “D1.”
- ☐ “A7” puis “B5” puis “C2” puis “D1”
- ☐ “,” puis “,” puis “,” puis “.”

#### Question 4

On considère le code suivant. Qu’est-il affiché ?

```
1 <?php
2 $texte = "Ces six saucisses-ci, ces six saucisses aussi.";
3 $recherche = "%(c|s)i%";
4 $remplace = "6";
5 $resultat = preg_replace($recherche,$remplace,$texte);
6 print_r($resultat);
7 ?>
```

- ☐ 6s 6x sau6s6s-6, 6s 6x sau6s6s aus6
- ☐ Ces 6x sau6sses-6, ces 6x sau6sses aus6
- ☐ Ces 6x saucisses-ci, ces 6x saucisses aus6

#### Question 5

On considère le code suivant. Combien d'éléments possède le tableau \$resultat ?

```
1 <?php
2 $texte = ["vingt-trois",
3           "quatre-vingt-cinq",
4           "deux-tiers",
5           "trois-quarts",
6           "soixante-neuf",
7           "quatre-vingt-deux"
8 ];
9 $recherche = "%((deux)|(trois)|(quatre))$%";
10 $remplace = "six";
11 $resultat = preg_filter($recherche,$remplace,$texte);
12 ?>
```

- ☐ 2
- ☐ 5
- ☐ 6

## Solutions des exercices




**Exercice p. 8 Solution n°1****Question 1**

On considère le texte « *Lundi 13 mai* ». Deux masques sont proposés pour correspondre avec ce texte : “*^Lun*” et “*mai\$*”. Ces masques sont corrects.

☒ Vrai

☐ Faux

 Le caractère circonflexe indique le début du texte or, le texte commence bien par « *Lun* ». Le caractère dollar indique la fin du texte or, le texte termine bien par « *mai* ».


**Question 2**

On considère le masque “(Crêpe|Glace) (vanille|chocolat)”. Quel texte ne correspond pas avec ce masque ?

☐ Glace vanille

☐ Glace chocolat

☒ Crêpe au chocolat

 La barre droite “|” indique une alternative. Ainsi, le masque indique le mot « *Crêpe* » ou le mot « *Glace* », suivi d’un caractère espace, puis le mot « *vanille* » ou le mot « *chocolat* ». Le mot « *au* » présent dans la 3<sup>ème</sup> proposition ne correspond pas avec le masque.


**Question 3**

On considère le masque “[A-Z]{0,3}[mbp][0-9]+”. Quel texte ne correspond pas avec ce masque ?

☐ CESp253

☐ m1

☒ BOB

 Le masque contient 3 classes : la première correspond avec les lettres majuscules, la seconde aux 3 lettres « *mbp* », la troisième aux chiffres. Les nombres entre accolades indiquent la quantité de caractères : ainsi, les majuscules initiales peuvent être au maximum au nombre de 3, mais il peut aussi n’y en avoir aucune. Le dernier caractère “+” indique la présence d’au moins un caractère précisé avant : ainsi, tout texte qui correspond doit finir par un chiffre.


**Question 4**

On considère le texte « *Quel prix ? 25 \$ ?* ». Quel masque correspond avec ce texte ?

☐ “?*\$*”, car le texte termine par un point d’interrogation


☐ “25 \$ ?”, car le texte contient l’extrait “25 \$ ?”

☒ “\? 25”, car le texte contient l’extrait “? 25”

 La première proposition est un masque incorrect. Le caractère “?” est un quantificateur qui signifie {0,1}. Pour rechercher de façon littérale le “?”, tout comme le “\$”, il convient d’ajouter le caractère d’échappement “\” avant. La seconde proposition correspondrait avec un texte se terminant par “25”.

**Question 5**

On considère le texte « *14 juillet 1789* ». Quel masque ne correspond pas avec ce texte ?

- ☐ “\d\s\w+\s\d+”
- ☐ “\w\D+\w”
- ☒ “[1-4] \w [1-9]+”
-  Pour la dernière proposition de masque, l'extrait commencerait par un seul chiffre entre 1, 2, 3 ou 4, suivi d'un espace. Or le texte commence par le nombre 14.

## Exercice p. 16 Solution n°2

### Question 1

On considère le code suivant. Combien d'éléments possède le tableau \$sortie ?

```
1 <?php
2 $jours = ["Lun", "Mar", "Mer", "Jeu", "Ven", "Sam", "Dim"];
3 $masque = '/e/';
4 $sortie = preg_grep($masque, $jours);
5 ?>
```

- ☐ 0
- ☒ 3
- ☐ 7


 Le masque correspond avec les 3 jours “Mer”, “Jeu”, et “Ven”.

### Question 2

On considère le code suivant. Combien d'éléments possède le tableau \$sortie ?

```
1 <?php
2 $texte = "Atchoum ! Je suis... Atchoum ! enrhumé. C'est... Atchoum ! embêtant.";
3 $masque = "#Atchoum ! #";
4 $sortie = preg_split($masque, $texte, -1, PREG_SPLIT_NO_EMPTY);
5 ?>
```

- ☒ 3
- ☐ 5
- ☐ 7


 La fonction split découpe le texte en retirant les « Atchoum ! ». Il reste 3 extraits : « Je suis... », « enrhumé. C'est... », et « embêtant. ».

### Question 3

On considère le code suivant. Combien d'éléments possède le tableau \$resultat ?

```
1 <?php
2 $texte = "Au printemps, les fleurs s'épanouissent.";
3 $masque = "/^[A-Z]([a-z é,']*)\.$/";
4 preg_match($masque, $texte, $resultat);
5 ?>
```

☐ 0☐ 1☒ 2

 array(2) {  
 [0]=>  
 string(41) "Au printemps, les fleurs s'épanouissent."  
 [1]=>  
 string(39) "u printemps, les fleurs s'épanouissent"  
 }

L'élément [0] est la chaîne complète qui a correspondu au modèle de l'expression régulière. Dans ce cas, c'est la phrase entière "Au printemps, les fleurs s'épanouissent".


L'élément [1] contient la sous-chaîne qui correspond au premier groupe capturant dans l'expression régulière. Dans ce cas, il s'agit de la partie de la phrase sans la première lettre majuscule, donc "u printemps, les fleurs s'épanouissent".

#### Question 4

On considère le code suivant. Combien d'éléments possède le tableau \$resultat ?

```
1 <?php
2 $texte = "Au printemps, les fleurs s'épanouissent.";
3 $masque = "/\b(\w{3,6})\b/";
4 preg_match_all($masque,$texte,$resultat);
5 ?>
```

☐ 0☒ 2☐ 5


 Le métacaractère “\b” signifie « *non-mot* ». Dans ce cas, il correspond avec les espaces entre les mots. Le sous masque, entre parenthèses, correspond avec les mots qui possèdent entre 3 et 6 lettres. Il s'agit donc des 2 mots « *les* » et « *fleurs* ».

#### Question 5

On considère le code suivant. Combien vaut la variable \$compteur ?

```
1 <?php
2 $jours = ["Lun", "Mar", "Mer", "Jeu", "Ven", "Sam", "Dim"];
3 $recherche = ["/^(M.+)/", "/(m|n)$/"];
4 $remplace = ['$1:Journée', '$1:Nuit'];
5 $resultat = preg_filter($recherche,$remplace,$jours,10,$compteur);
6 ?>
```

☐ 2☐ 4☒ 6

 Il y a 2 masques de recherche. Le premier correspond avec “Mar” et “Mer”. Le deuxième correspond avec “Sam”, “Dim”, “Lun”, “Ven”. Il y a donc 6 remplacements.

**p. 18 Solution n°3**

Tout d'abord, il convient de percevoir qu'il s'agit d'une conversion de format. Donc nous allons faire des remplacements. Pour cela, nous pourrions utiliser la fonction `preg_replace`. Cependant, nous ne devons pas traiter l'intégralité du fichier, mais uniquement les numéros de la zone Nord. C'est pourquoi nous utiliserons la fonction `preg_filter`. Ensuite, les données du fichier d'origine sont utilisées pour créer le nouveau fichier (qui correspond au format de votre entreprise). Par conséquent, il sera nécessaire d'utiliser des sous masques, à l'aide des parenthèses, et d'utiliser les variables correspondantes dans la chaîne de remplacement (\$1, \$2, \$3, etc.). Après quoi, les numéros commençant parfois par « 0 » et parfois par « + 33 », il convient d'indiquer une alternative au début du masque avec la barre droite "|". Enfin, les méta-caractères, comme le "." ou le « + », doivent être échappés avec l'antislash "\" pour correspondre littéralement.

**p. 18 Solution n°4**

Les numéros commençant par 0 ou + 33 s'écrivent dans le masque "(0|+33)". Ensuite vient le chiffre 3. Les paires de chiffres sont notées dans le masque "(\\d{2})", et séparées par "." Vous pouviez aussi utiliser "[0-9][0-9]" par exemple. Les caractères virgule correspondent littéralement. Les prénoms, étant en fin de chaîne, correspondent avec (.?). Les noms doivent correspondre en mode « *non gourmand* » pour que l'étoile ne « *mange* » pas la virgule suivante et le prénom. Cela s'écrit "(.?)". Constatez que le nom et le prénom sont respectivement les 6ème et 7ème groupe de parenthèses, ce qui permet de commencer la chaîne de remplacement par "\$6\\t\$7\\t". Ensuite, le numéro commence par + 333, puis on indique entre espaces les variables correspondant aux autres sous masques "\$2 \$3 \$4 \$5".

```
1 <?php
2 $recherche = "/(0|+33)3\\. (\\d{2})\\. (\\d{2})\\. (\\d{2})\\. (\\d{2}), (.*?), (.*?)"/;
3 $remplace = "$6\\t$7\\t+333 $2 $3 $4 $5";
4 $resultat = preg_filter($recherche,$remplace,$fichier);
5 ?>
```


**Exercice p. 18 Solution n°5**

**Question 1**

Dans un masque, le caractère circonflexe "^" peut avoir 3 interprétations différentes.

☒ Vrai

☐ Faux

 Placé en début du masque, il s'agit de l'ancre « *commençant par* ». Placé en début d'une classe, il indique le contraire de cette classe. Enfin, il peut être recherché de façon littérale, échappé si besoin avec "\\".

**Question 2**

On considère le code suivant. Combien d'éléments contient le tableau \$resultat ?

```
1 <?php
2 $texte = ["Avec un nombre de 12",
3          "A l'école ils sont 23",
4          "A cette heure de 10h20",
5          "Après manger",
6          "A4"];
7 $masque = "/A[a-z ]*[0-9]+/";
8 $resultat = preg_grep($masque,$texte);
9 ?>
```



- ☐ 2
- ☒ 3
- ☐ 4

**Q** Le masque se termine par “[0-9]+”, ce qui indique que le texte qui correspond se termine par un ou des chiffres. Donc le texte “Après manger” ne correspond pas. Entre le A initial et les chiffres finaux, correspondent uniquement les lettres minuscules et le caractère espace. Ainsi, « A l’école » ne correspond pas à cause de l’apostrophe. Les 3 textes restants correspondent.

### Question 3

On considère le code suivant. Quatre éléments sont affichés, lesquels sont-ils ?

```
1 <?php
2 $texte = "A la bataille navale, il a joué : A7, B5, C2, D1.";
3 $masque = "/ ([A-G][1-7])(,|\.)"/;
4 preg_match_all($masque,$texte,$resultat);
5 print_r($resultat[1]);
6 ?>
```

- ☐ “A7,” puis “B5,” puis “C2,” puis “D1.”
- ☒ “A7” puis “B5” puis “C2” puis “D1”
- ☐ “,” puis “,” puis “,” puis “.”

**Q** Remarquez d’abord que le masque contient 2 sous masques. Par conséquent, le tableau \$resultat contient 3 éléments, qui sont eux-mêmes des tableaux. Le premier élément correspond aux éléments capturés avec le masque entier, ce qui correspond à la proposition n° 1. Le deuxième élément correspond aux éléments capturés avec le premier sous masque, ce qui correspond à la proposition n° 2. Le troisième élément correspond aux éléments capturés avec le deuxième sous masque, ce qui correspond à la proposition n° 3. On affiche \$resultat[1], c’est-à-dire le deuxième élément du tableau \$resultat (le premier élément a l’indice 0).

### Question 4

On considère le code suivant. Qu’est-il affiché ?

```
1 <?php
2 $texte = "Ces six saucisses-ci, ces six saucisses aussi.";
3 $recherche = "%(c|s)i%";
4 $remplace = "6";
5 $resultat = preg_replace($recherche,$remplace,$texte);
6 print_r($resultat);
7 ?>
```

- ☐ 6s 6x sau6s6s-6, 6s 6x sau6s6s aus6
- ☒ Ces 6x sau6sses-6, ces 6x sau6sses aus6
- ☐ Ces 6x saucisses-ci, ces 6x saucisses aus6

**Q** Le masque définit une alternative entre les caractères “c” et “s”, suivi de la lettre “i”. Ainsi, les syllabes “ci” et “si” sont concernées par le remplacement.

### Question 5


On considère le code suivant. Combien d’éléments possède le tableau \$resultat ?

```
1 <?php
2 $texte = ["vingt-trois",
3           "quatre-vingt-cinq",
4           "deux-tiers",
5           "trois-quarts",
6           "soixante-neuf",
7           "quatre-vingt-deux"
8 ];
9 $recherche = "%((deux)|(trois)|(quatre))$%";
10 $remplace = "six";
11 $resultat = preg_filter($recherche,$remplace,$texte);
12 ?>
```

☒ 2

☐ 5

☐ 6

 Le masque contient le caractère \$ à la fin, ce qui indique que le texte correspondant termine par « deux », « trois », ou « quatre ». Seuls « vingt-trois » et « quatre-vingt-deux » correspondent.