LES TYPES SIMPLES 5

PLAN

OBJECTIFS

➤ Maîtriser les données simples et les traitements associés.

5.1 INTRODUCTION

Les programmes informatiques traitent des données qui peuvent être *simples* ou *structurées*.

Les données structurées, par exemple les tableaux, sont généralement constituées d'un ensemble de données simples.

Afin de maîtriser la gestion des différents types de données, il est primordial de bien connaître les types des données simples, leurs limites et les traitements qui leur sont associés.

5.2 LE TYPE ENTIER

Définition

Un entier est un nombre positif ou négatif tel que – 18, 5 ou + 56. Il ne possède aucune partie après le point décimal (notation anglo-saxonne). L'exemple suivant affecte la valeur décimale 138 à la variable `$i` :

```
$i = 138 ;
```

La constante entière 138 peut se noter en :

- *décimal* (base 10). Par exemple : – 138, 138 ou + 138 ;
- *octal* (base 8). Par exemple 0212 (préfixé par 0) équivalent à 138 en décimal ;
- *hexadécimal* (base 16) : par exemple 0x8a ou 0X8A (préfixé par 0x ou 0X) équivalent à 138 en décimal ;
- *binaire* (base 2) par exemple 0b10001010 (préfixé par 0b) équivalent à 138 en décimal.

Le programme `entiers_valeurs_shell.php` présente ces syntaxes et les affichages associés *via* les instructions `echo` ou `printf()` :

Listing 5.1 – Programme entiers_valeurs_shell.php

```
<?php
$a = 1234; // nombre décimal positif
$b = -123; // nombre décimal négatif
$c = 0123; // nombre octal (équivaut à 83 en décimal)
$d = 0x1a; // nombre hexadécimal (équivaut à 26 en décimal)
$e = 0X1A; // nombre hexadécimal (équivaut à 26 en décimal)
$f = 0b11111111; // nombre binaire (équivaut à 255 en décimal)
echo   'Décimal 1234 (affichage par echo) : $a='.$a."\n";
echo   'Décimal -123 (affichage par echo) : $b='.$b."\n";
echo   'Octal   0123 (affichage par echo) : $c='.$c."\n";
printf("Octal   0123 (printf en octal)    : \$c=%o\n",$c);
echo   'Hexadécimal 0x1a (affichage par echo)  : $d='.$d."\n";
printf("Hexadécimal 0x1a (printf au format %%x):\$d=%x\n",$d);
echo   'Hexadécimal 0X1A (affichage par echo)  : $e='.$e."\n";
printf("Hexadécimal 0X1A (printf au format %%X):\$e=%X\n",$e);
echo   'Binaire 0b11111111 (affichage par echo): $f='.$f."\n";
printf("Binaire 0b11111111 (printf format %%b):\$f=%b\n",$f);
?>
```

Voici son exécution :

Listing 5.2 – Exécution de entiers_valeurs_shell.php

```
$ php entiers_valeurs_shell.php
Décimal 1234 (affichage par echo) : $a=1234
Décimal -123 (affichage par echo) : $b=-123
Octal   0123 (affichage par echo) : $c=83
Octal   0123 (printf en octal)    : $c=123
Hexadécimal 0x1a (affichage par echo)  : $d=26
Hexadécimal 0x1a (printf au format %x) : $d=1a
Hexadécimal 0X1A (affichage par echo)  : $e=26
Hexadécimal 0X1A (printf au format %X) : $e=1A
Binaire 0b11111111 (affichage par echo): $f=255
Binaire 0b11111111 (printf format %b):$f=11111111
```

Erreurs de notation

Le type de la variable est défini selon la valeur affectée. Si la valeur est erronée, comme `01093`, qui indique une valeur octale (commençant par 0) contenant une valeur 9 (impossible en octal), **seule la première partie 010 est affectée** à la variable.

Voici des exemples de syntaxes erronées :

```
$i = 01093; // Erreur syntaxe. Vaut 10 (base 8), 8 (base 10)
$i = "18azerty"; // chaîne de caractères
$i = $i+0;       // vaut 18
```

Codage binaire d'un entier

Un entier est codé sur plusieurs octets en une *suite de valeurs binaires avec un bit de signe*. La valeur numérique sera plus ou moins grande selon le nombre d'octets.

La figure 5.1 présente le codage de + 21 331 sur 32 bits. Le bit de gauche est assimilé au signe : 0 pour +, 1 pour –. Les 31 bits restants expriment la « quantité ».

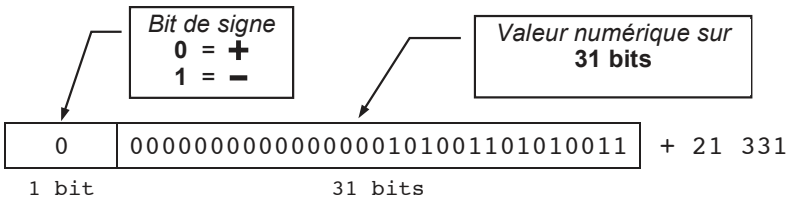

Figure 5.1 – Codage binaire d'un entier.

La plus grande valeur positive correspond à `0` pour le bit de signe suivi de 31 bits à `1` pour la valeur soit :

`01111111111111111111111111111111`

Ce qui correspond à la valeur 2^{31}–1 soit +2 147 483 647.

Cette méthode « simpliste » qui sépare le signe et la valeur numérique est inexacte. Si elle correspond au bon codage pour les entiers positifs, elle est erronée pour les entiers négatifs. Le « vrai » codage ne considère pas le bit de signe séparément du reste. En partant de la valeur centrale 0, on obtient les valeurs négatives successives en retirant la valeur binaire 1 à la valeur précédente. Pour retirer 1 à la valeur 0 on suppose que 0 (sur 32 bits) est en réalité un nombre binaire de 33 bits, dont le 33e bit à 1 a disparu. La figure 5.2 présente ce codage.

Figure 5.2 – Codage binaire d'un entier négatif.

La plus petite valeur d'un entier sur 32 bits est -2^{31} soit –2 147 483 648.

Capacité

Selon que le serveur sur lequel s'exécute PHP est 32 ou 64 bits, la plus grande valeur entière change. PHP prévoit deux constantes pour expliciter ces valeurs : `PHP_INT_MAX` (plus grand nombre entier), `PHP_INT_SIZE` (nombre d'octets d'un entier : 4 ou 8 octets). Le tableau 5.1 récapitule la capacité d'un entier.

Tableau 5.1 – Capacités d'un entier

Nb d'octets	Portée
4 octets (32 bits)	± 2 milliards = ± 2 x 10^9
	– 2 147 483 648 ... + 2 147 483 647
8 octets (64 bits)	± 9 milliards de milliards = ± 9 x 10^{18}
	– 9 223 372 036 854 775 808 ... + 9 223 372 036 854 775 807

Remarques

Le langage PHP ne supporte pas les entiers non signés. Le langage PHP sous Windows propose encore des entiers sur 32 bits.

Dépassement de capacité

Si lors d'un calcul, la valeur dépasse la capacité d'un entier, alors la variable change de type et devient un réel. Ainsi les syntaxes suivantes :

```
$grand_nombre = PHP_INT_MAX;
var_dump($grand_nombre);      // int(9223372036854775807)
$grand_nombre = PHP_INT_MAX+1;
var_dump($grand_nombre);      // float(9.2233720368548E+18)
$petit_nombre = -PHP_INT_MAX-1;
var_dump($petit_nombre);      // int(-9223372036854775808)
```

```
$petit_nombre = -PHP_INT_MAX-2;
var_dump($petit_nombre);      // float(-9.2233720368548E+18)
```

provoquent le changement de type `int` en `float` lorsqu'on ajoute 1 à la plus grande valeur entière sur 64 bits ou que l'on retire 1 à la plus petite valeur entière.

Saisie et affichage

Dans un environnement shell

La saisie et l'affichage en shell sont présentés à la section 3.2 du chapitre 3. La saisie utilise l'instruction `fscanf()` et l'affichage `echo` ou `printf()`.

```
echo "Entrez un entier : "        ;
fscanf(STDIN,"%d",$i)             ;
echo "Entier saisi : ".$i.PHP_EOL ;
printf("Entier saisi : %d\n",$i)  ;
```

Remarque

Les syntaxes précédentes utilisent la constante `PHP_EOL` qui correspond au saut de ligne, aussi bien sous Windows et que sous Unix, à la place de `'\n'` spécifique à Unix.

Dans un environnement web

La saisie par formulaire est détaillée à la section 3.3 du chapitre 3. Le fichier HTML `entiers_saisie_affich_web.html` saisit un entier. Sa valeur est transmise au programme PHP par la méthode POST :

Listing 5.3 – Fichier entiers_saisie_affich_web.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Saisie d'un entier</title>
 </head>
 <body>
  <form action="entiers_saisie_affich_web.php" method="post">
   Entrez un entier : <input type="text" name="i" size="10" /><br/>
   <input type="submit" value="Valider" />
   <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le programme `entiers_saisie_affich_web.php` récupère la valeur « texte » transmise par le formulaire après validation et la convertit au format entier *via* `intval()`. L'instruction `var_dump()` affiche la valeur et le type de `$i` :

Listing 5.4 – Programme entiers_saisie_affich_web.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Traitement d'un entier</title>
 </head>
 <body>
 <?php
 define("WEB_EOL","<br/>");
 $i=$_POST['i'] ; // -- On récupère la donnée --
 $i=intval($i)  ; // -- On traite la donnée --
 echo "Entier saisi : ".$i.WEB_EOL ;
 var_dump($i);
 ?>
 </body>
</html>
```

La figure 5.3 présente la saisie (1) et le traitement par le programme PHP (2).

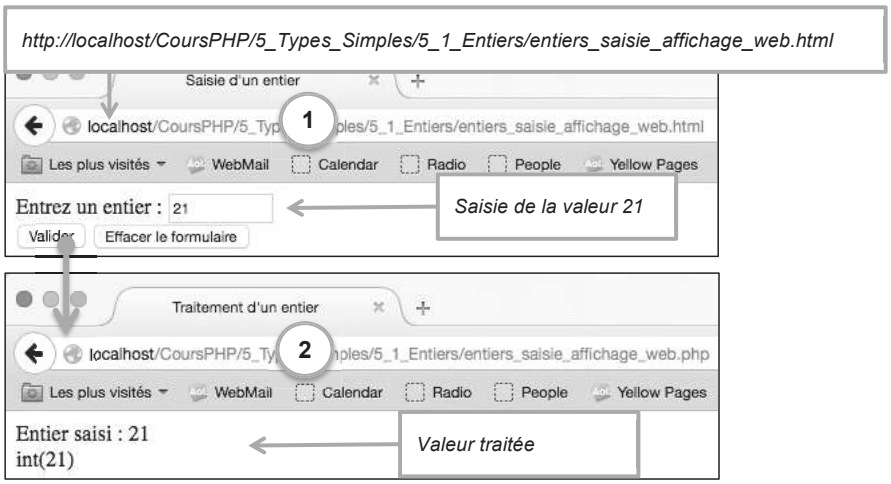

Figure 5.3 – Formulaire de saisie et traitement d'un entier.

Transtypage

L'exemple précédent montre qu'il est préférable de forcer le type entier de la valeur « texte » reçue d'un formulaire. Cela se fait *via* la fonction `intval()`, le préfixe `(int)` ou `(integer)`, ou la procédure `settype()`, comme cela est détaillé dans la section 4.3 du chapitre 4. Alors que `settype()` change le type de la variable, `intval()` ou `(int)` ne modifie que son interprétation dans le calcul. Le programme `entiers_transtypage_shell.php` présente ces conversions :

Listing 5.5 – Programme entiers_transtypage_shell.php

```
<?php
  $i="18";
  var_dump($i); // string(2) "18"
  $i=(int)$i;   // transtypage explicite
  var_dump($i); // int(18)
  $j="texte";
  var_dump($j); // string(5) "texte"
  $j=(int)$j;   // transtypage explicite
  var_dump($j); // int(0)
  $k=18   ;
  $l=10.5 ;
  $m=$k+$l;
  var_dump($m);     // float(28.5)
  $m=(int) ($k+$l); // transtypage explicite
  var_dump($m);     // int(28)
?>
```

La syntaxe avec l'instruction `settype()` est de la forme :

```
settype($i,"int");
settype($j,"integer");
```

Les opérateurs

Arithmétiques

Le tableau 5.2 récapitule les opérateurs arithmétiques sur les entiers.

Tableau 5.2 – Opérateurs arithmétiques sur les entiers

Opérateur	Signification	Exemple
+	Addition	`$k = $i + $j ;`
-	Soustraction	`$k = $i - $j ;`
*	Multiplication	`$k = $i * $j ;`
/	Division entière ou réelle	`$resultat = $i / $j ;`
%	Modulo	`$reste = $i % $j ;`
**	Puissance	`$resultat = $i ** $j ; /* $i à la puissance $j à partir de PHP 5.6 équivalent à exp($j*log($i))*/`
++	Incrémentation (+1)	`$j=$i++;/* affecte PUIS incrémente */` `$k=++$i;/* incrémente PUIS affecte */`
--	Décrémentation (–1)	`$j=$i--;/* affecte PUIS décrémente */` `$k=--$i;/* décrémente PUIS affecte */`
+=	Somme et affectation	`$j += $i ; // $j = $j+$i`
-=	Soustraction et affectation	`$j -= $i ; // $j = $j-$i`

Tableau 5.2 – **Opérateurs arithmétiques sur les entiers (suite)**

Opérateur	Signification	Exemple
*=	Multiplication et affectation	`$j *= $i ; // $j = $j*$i`
/=	Division entière ou réelle et affectation	`$j /= $i ; // $j = $j/$i`
%=	Modulo et affectation	`$j %= $i ; // $j = $j%$i`

Remarque

L'opérateur de division « / » appliqué à deux entiers retourne un entier si le reste est égal à 0. Sinon la valeur retournée est de type réel.

Les deux syntaxes suivantes montrent comment obtenir le quotient d'une division entière quand le reste est différent de 0.

```
$k = ($i-($i%$j))/$j   ; // méthode 1
$k = (int) floor($i/$j) ; // méthode 2
```

De comparaison

Le tableau 5.3 récapitule les opérateurs de comparaison sur les entiers.

Tableau 5.3 – **Opérateurs de comparaison sur les entiers**

Opérateur	Signification	Exemple
<	Inférieur à	`if ($i < $j) …`
>	Supérieur à	`if ($i > $j) …`
<=	Inférieur ou égal à	`if ($i <= $j) …`
>=	Supérieur ou égal à	`if ($i >= $j) …`
==	Est égal (après transtypage)	`if ($i == $j) …`
===	Est égal **et** de même type	`if ($i === $j) …`
!=	Différent (non égal) après transtypage	`if ($i != $j) …`
<>	Différent (non égal) après transtypage	`if ($i <> $j) …`
!==	Différent **ou** pas du même type	`if ($i !== $j) …`

Remarque

L'opérateur « == » compare la valeur entre deux variables. Si l'on compare l'entier 18 avec la chaîne « 18abcd », la chaîne sera convertie en entier, soit 18, et la comparaison sera effectuée numériquement, ce qui donnera VRAI. L'opérateur « === » compare les valeurs et les types sans transtypage. Il en va de même pour « != » et « !== ».

Exemples de programmes

Le programme `somme_entiers_shell.php` affiche la somme de deux entiers.

Listing 5.6 – Programme somme_entiers_shell.php

```
<?php
 echo 'Entrez deux entiers :';
 fscanf(STDIN,"%d %d",$i,$j);
 $somme=$i+$j;
 echo "La somme de $i et de $j = $somme".PHP_EOL;
?>
```

Voici son exécution :

Listing 5.7 – Exécution somme_entiers_shell.php

```
$ php somme_entiers_shell.php
Entrez deux entiers : 12 9
La somme de 12 et de 9 = 21
```

Les fichiers `somme_entiers_web.html` et `somme_entiers_web.php` sont la version web du programme précédent. Le premier saisit deux entiers *via* un formulaire HTML en méthode GET.

Listing 5.8 – Fichier somme_entiers_web.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Somme de deux entiers</title>
 </head>
 <body>
  <form action="somme_entiers_web.php" method="get">
  Entrez deux entiers :<br/>
  Premi&egrave;re valeur : <input type="text" name="i" size="5" /><br/>
  Deuxi&egrave;me valeur : <input type="text" name="j" size="5" /><br/>
  <input type="submit" value="Valider" />
  <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le programme PHP récupère les données *via* `$_GET[]` et les convertit en entier avec `intval()`. La constante `WEB_EOL` est définie pour la balise HTML `<br/>`.

Listing 5.9 – Programme somme_entiers_web.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Somme de deux entiers</title>
 </head>
 <body>
<?php
```

```
define("WEB_EOL","<br/>");
$i=$_GET['i']; // -- On récupère les données --
$j=$_GET['j'];
$i=intval($i); // -- On traite les données --
$j=intval($j);
$somme=$i+$j;
echo "La somme de $i et de $j = $somme".WEB_EOL;
?>
</body>
</html>
```

La figure 5.4 présente le formulaire de saisie (1) et le résultat de la somme (2).

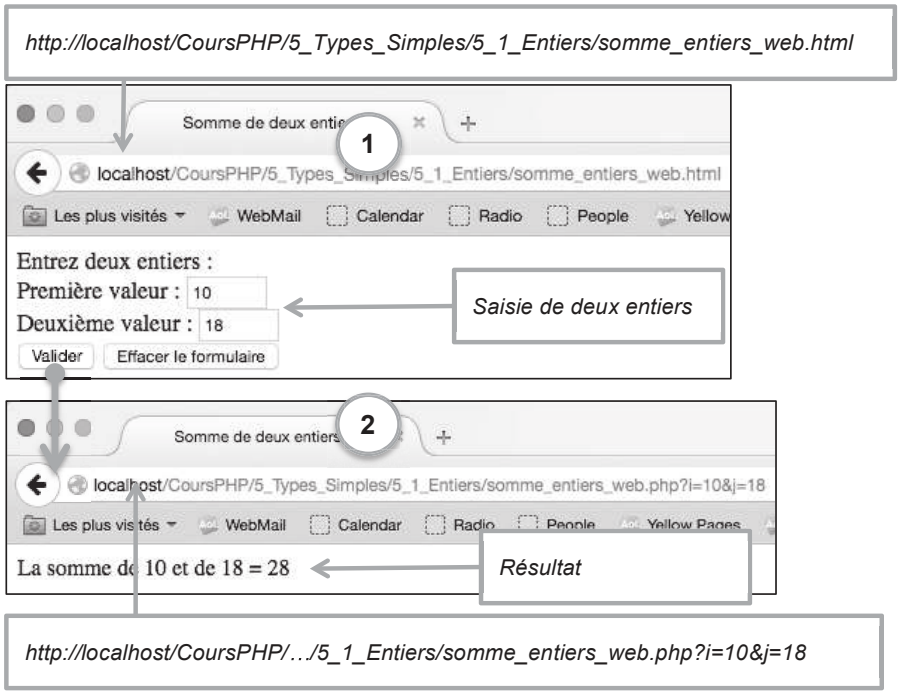

Figure 5.4 – Formulaire de saisie de deux entiers et affichage de leur somme.

Les fonctions

Le tableau 5.4 en présente quelques fonctions sur les entiers.

Tableau 5.4 – Fonctions sur les entiers

Fonction	Signification	Exemple
`abs()`	Valeur absolue	`$i=abs($j);`
`rand()`	Retourne une valeur pseudo aléatoire entre 0 et la valeur indiquée par `getrandmax()`. `srand()`initialise `rand()`	`$i=rand();` `$i=rand(1,49); // valeur entre 1 et 49`

Fonction	Signification	Exemple

`is_int()`
`is_integer()`
`is_long()`
	Retourne VRAI si l'argument est un entier FAUX sinon	`if (is_int($i)) …`

`intval()`	Retourne la valeur entière base 10 de l'argument. Si la base est précisée, on convertit dans la base si le premier argument est une chaîne. Pour un tableau retourne 0 s'il est vide, 1 sinon	`$i=intval(2.4); // 2`
		`$i=intval("2"); // 2`
		`$i=intval(042); // 34`
		`$i=intval("42",8);// 34`
		`$i=intval(42,8); // 42`
		`$i=intval($tableau);`

5.3 Le type réel

Définition

Un réel est un nombre positif ou négatif avec une « virgule » ou plutôt un point décimal qui est la notation anglo-saxonne d'usage. Il se note par exemple : –18.543 ou 1.2e3 ($1.2\times10^{+3}$) ou +1.2E-10 ($+1.2\times10^{-10}$). Les deux syntaxes suivantes affectent la valeur 13.8 à la variable `$x` :

```
$x = 1.38E+1 ;
$x = 13.8    ;
```

Le programme `reels_valeurs_shell.php` présente ces différentes syntaxes :

Listing 5.10 – Programme reels_valeurs_shell.php

```
<?php
$a = -1.56    ;
$b = 7E-10    ;
echo '$a = '.$a."   ";
var_dump($a)  ;
echo '$b = '.$b."   ";
var_dump($b)  ;
?>
```

Voici son exécution :

Listing 5.11 – Exécution de reels_valeurs_shell.php

```
$ php reels_valeur_shell.php
$a = -1.56    float(-1.56)
$b = 7.0E-10  float(7.0E-10)
```

Erreurs de notation

Les syntaxes suivantes présentent le comportement du langage PHP dans le cas où la donnée réelle n'est pas correctement écrite.

```
$x = "18.23azerty"; // $x est chaîne de caractères
$x = $x+0;          // $x est le réel = 18.23
```

Codage binaire d'un réel

Le codage binaire d'un réel est complexe et impacte directement l'exactitude des calculs. Nous présentons son principe afin d'en comprendre les conséquences.

Principe

Le nombre réel est réécrit sous une forme $\pm$ 0.m $\times$ $10^{\pm e}$ où *m* est la **mantisse** et *e* l'**exposant**. Le nombre –22.625 serait exprimé comme $-0.22625 \times 10^{+2}$ en base 10.

Cette règle générale est appliquée en base 2, et non en base 10. Le codage binaire est de la forme $\pm$ 1.m $\times$ $2^{\pm e}$. Ainsi –22.625 s'écrit -1.0110101×2^4.

Pour obtenir ce résultat, le nombre décimal –22.625 est d'abord traduit sous la forme binaire –10110.101, de la manière suivante :

- 22 en base 10 est une somme de puissance de 2 soit : $16 + 4 + 2 = 2^4 + 2^2 + 2^1$ = 10110 en binaire ;
- 0.625 en base 10 est une somme de puissance négative de 2 soit : $0.5 + 0.125 = 2^{-1} + 2^{-3}$ = 0.101 en binaire.

Le nombre –10110.101 est ensuite traduit en -1.0110101×2^4 et finalement écrit sur 32 ou 64 bits.

Représentation avec mantisse et exposant

La valeur -1.0110101×2^4 est présentée en une suite d'octets selon la norme IEEE 754. Les deux formats de cette norme sont :

- sur 32 bits : 1 bit de signe de la mantisse, 8 bits d'exposant (valeurs –126 à 127 avec un décalage), 23 bits de mantisse. C'est le type *float* sur 32 bits ;
- sur 64 bits : 1 bit de signe de la mantisse, 11 bits d'exposant (valeurs –1022 à 1023 avec un décalage), 52 bits de mantisse. C'est le type *float* sur 64 bits ou *double*.

  La formule utilisée par la norme est : valeur = (signe) $\times$ 1.m $\times$ $2^{e - \text{décalage}}$
- *décalage* = $2^{n-1} - 1$ (n = nombre de bits pour l'exposant, soit 8 sur 32 bits ou 11 sur 64 bits), soit 127 ($2^7 - 1$) sur 32 bits, ou 1023 ($2^{10} - 1$) sur 64 bits ;
- *p* = e – décalage. Le calcul de l'exposant binaire est le suivant : e = p + 127 sur 32 bits, ou e = p + 1023 sur 64 bits.

La figure 5.5 présente le codage sur **64 bits** de –22.625 soit $-1.\mathbf{0110101} \times 2^4$ en binaire.

La formule, (signe) $\times$ 1.m $\times$ $2^{e - \text{décalage}}$, se traduit par :

m = ***0110101*** et *e* = *4 + 1023 = 1027* = ***10000000011***

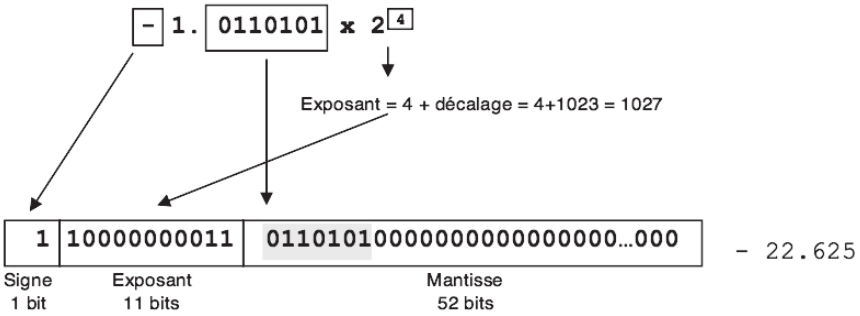

Figure 5.5 – **Représentation d'un réel sur 64 bits.**

Conséquence de cette représentation sur l'exactitude des calculs

Cette représentation binaire d'un réel ne code pas parfaitement tous les nombres. Certains réels *exacts* en base 10 deviennent *infinis* en binaire. C'est le cas du nombre 2.2, dont la mantisse binaire est infinie :

0.2 × 2	=	0.4	→	0	→	0.0
0.4 × 2	=	**0.8**	→	0	→	0.00
0.8 × 2	=	1.6	→	1	→	0.001
0.6 × 2	=	1.2	→	1	→	0.0011
0.2 × 2	=	0.4	→	1	→	0.00110
0.4 × 2	=	**0.8**	→	0	→	0.001100

…

Le calcul continue indéfiniment ! La représentation de **2.2** sur vingt-trois décimales est : 10.00110011001100110011001, ce qui correspond à **2.199999928**.

Le calcul 2 × 2.2 dans un programme donnera le résultat approché 4.399999856 à la place du résultat exact 4.4. Cette erreur grandira au fur et à mesure des calculs.

Limites du codage

- *Valeurs minimales*

Sur 32 bits, le plus petit nombre positif différent de 0 et le plus grand nombre négatif différent de 0 correspondent à : **± 1.17549435×10^{-38}**.

Sur 64 bits, le plus petit nombre positif différent de 0 et le plus grand nombre négatif différent de 0 correspondent à : **± 2.2250738585072014×10^{-308}**.

- *Valeurs maximales*

Sur 32 bits, le plus grand nombre positif fini et le plus petit nombre négatif fini sont : **± 3.40282326×10^{38}**.

Sur 64 bits, le plus grand nombre positif fini et le plus petit nombre négatif fini sont : **± 1.7976931348623157×10^{308}**.

Les erreurs de précision

Pour les nombres décimaux dont la valeur après la virgule est infinie comme PI = 3.1415926535898, et les nombres dont le codage binaire après la virgule est une suite infinie de 0 et de 1, comme 2.2, le fait d'écrire leur valeur dans une variable dont la taille mémoire est par définition finie implique que la valeur est fausse : la partie décimale qui ne « rentre » pas dans la mémoire est perdue.

Les valeurs réelles infinies sont toujours fausses !

Prenons l'exemple des valeurs 0.1 et 0.7 qui n'ont pas de représentation exacte en binaire ! La somme de 0.1 et de 0.7 doit donner 0.8, or le résultat informatique est *0.79999999999999991118, soit presque 0.8 à la 16e décimale près.* Si l'on prend ce résultat et qu'on le multiplie par 10, on pense obtenir 8.0 soit l'entier 8. Or on obtient *7.99999999999999991118,* la conversion en entier donnera … 7 !

Les syntaxes suivantes montrent cette erreur de calcul :

```
$x = 0.1+0.7 ; // Devrait contenir 0.8, mais contient 0.799...
$y = $x*10;    // Devrait contenir 8, mais contient 7.99...
$correction = 1e-15;  // Pour corriger l'erreur de calcul
$z_sans_correct=floor($y); // Retourne 7 et non 8 : FAUX
$z_avec_correct=floor($y+$correction); // Retourne 8 : VRAI
```

Il est préférable de considérer que deux nombres réels sont égaux, quand leur différence est inférieure à une certaine précision. Il faut également contrôler les traitements récupérant un entier à partir d'un réel ayant une valeur approchée de la valeur attendue.

Capacité

En PHP, un réel est défini sur 64 bits selon le format IEEE 754 avec une précision de 1.11e-16. Le tableau 5.5 récapitule sa capacité.

Tableau 5.5 – Capacité d'un réel

Nb d'octets	Type de valeur	Capacité
8 octets (64 bits)	Plus grand nombre positif fini et plus petit nombre négatif fini	$\pm 1.7976931348623157 \times 10^{+308}$
	Plus petit nombre positif différent de zéro et plus grand nombre négatif différent de zéro	$\pm 2{,}2250738585072014 \times 10^{-308}$

Les valeurs *NAN* et *INF*

Si un calcul *dépasse la capacité d'un réel*, la variable résultat contient une valeur infinie, caractérisée par la constante `INF` ou `-INF`. Si un calcul *est impossible*, comme le fait d'appliquer une fonction à des valeurs qui ne sont pas dans son espace de définition, alors la constante `NAN` est retournée. Cette constante est une valeur *indéfinie*

et *non représentable*, elle n'est pas comparable à d'autres valeurs, y compris à elle-même. Il faut utiliser la fonction `is_nan()`pour la détecter.

Les nombres de grande taille

Le langage PHP propose des bibliothèques nommées BCMath ou GMP pour le calcul sur des grands nombres. Les réels sont représentés en chaînes de caractères. Une documentation est disponible à l'URL : *http://php.net/manual/fr/book.bc.php*.

Saisie et affichage

Dans un environnement shell

La saisie et l'affichage en shell sont présentés à la section 3.2 du chapitre 3. La saisie utilise l'instruction `fscanf()`, et l'affichage les instructions `echo` ou `printf()`. Voici le programme `reels_saisie_affichage_shell.php` :

Listing 5.12 – Programme reels_saisie_affichage_shell.php

```
<?php
 echo "Entrez un réel : "        ;
 fscanf(STDIN,"%f",$x)           ;
 echo "Réel saisi : ".$x.PHP_EOL ;
?>
```

Voici son exécution :

Listing 5.13 – Exécution de reels_saisie_affichage_shell.php

```
$ php reels_saisie_affichage_shell.php
Entrez un réel : 2.85
Réel saisi : 2.85
```

Dans un environnement web

Le fichier `reels_saisie_affichage_web.html` propose la saisie d'un réel :

Listing 5.14 – Fichier reels_saisie_affichage_web.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Saisie d'un réel</title>
 </head>
 <body>
  <form action="reels_saisie_affichage_web.php" method="post">
   Entrez un r&eacute;el  <input type="text" name="x" size="10" /><br/>
   <input type="submit" value="Valider" />
   <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le programme `reels_saisie_affichage_web.php` récupère la valeur « texte » transmise et la convertit au format réel *via* `floatval()` :

Listing 5.15 – Programme reels_saisie_affichage_web.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Traitement d'un réel</title>
 </head>
 <body>
 <?php
 define("WEB_EOL","<br/>");
 $x=$_POST['x'] ; // -- on récupère les données --
 $x=floatval($x); // -- on traite les données --
 echo "R&eacute;el saisi : ".$x.WEB_EOL ;
 var_dump($x);
 ?>
 </body>
</html>
```

La figure 5.6 présente la saisie (1) et le traitement par le programme PHP (2).

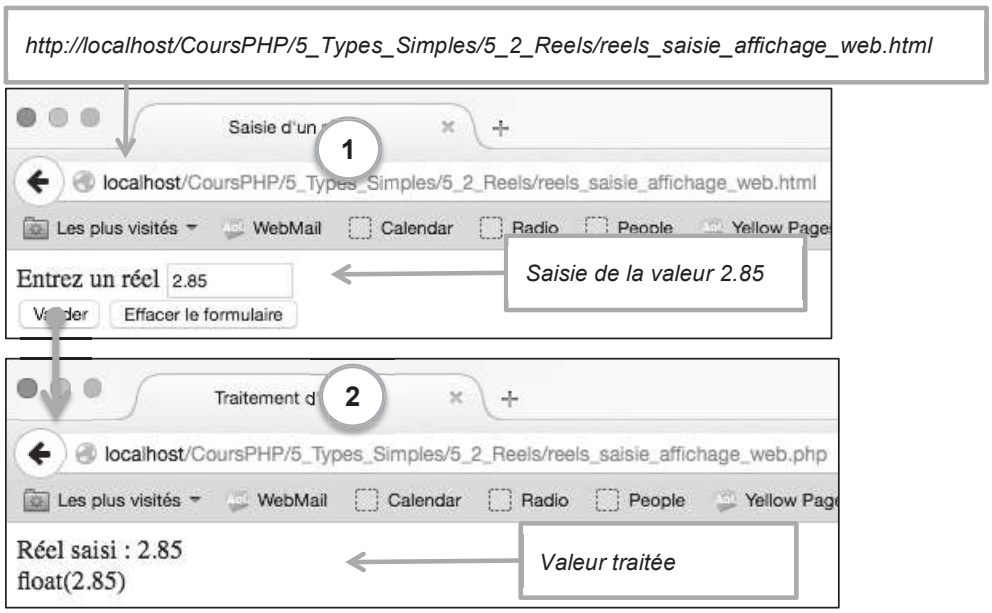

Figure 5.6 – Formulaire de saisie et traitement d'un réel.

Transtypage

Il est préférable de forcer le type réel de la valeur reçue d'un formulaire, *via* la fonction `floatval()`, le préfixe `(float)` ou la procédure `settype()`.

Alors que `settype()` change le type de la variable, `floatval()` ou `(float)` ne modifie que son interprétation dans le calcul. Le programme `reels_transtypage_shell.php` présente la conversion de différentes valeurs en réel :

Listing 5.16 – Programme reels_transtypage_shell.php

```
<?php
  $x="18.2";
  var_dump($x); // string(4) "18.2"
  $x=(float)$x; // transtypage explicite
  var_dump($x); // float(18.2)
  $y="8.2texte";
  var_dump($y); // string(8) "8.2texte"
  $y=(float)$y; // transtypage explicite
  var_dump($y); // float(8.2)
?>
```

La syntaxe avec l'instruction `settype()` est de la forme :

```
settype($x,"float");
```

Les opérateurs

Arithmétiques

Le tableau 5.6 récapitule les opérateurs arithmétiques sur les réels.

Tableau 5.6 – Opérateurs arithmétiques sur les réels

Opérateur	Signification	Exemple
+	Addition	`$z = $x + $y ;`
-	Soustraction	`$z = $x - $y ;`
*	Multiplication	`$z = $x * $y ;`
/	Division réelle	`$resultat = $x / $y ;`
**	Puissance	`$resultat = $x ** $y ;/* PHP 5.6 équivalent à exp($y*log($x))*/`
+=	Somme et affectation	`$y += $x ; // $y = $y+$x`
-=	Soustraction et affectation	`$y -= $x ; // $y = $y-$x`
*=	Multiplication et affectation	`$y *= $x ; // $y = $y*$x`
/=	Division réelle et affectation	`$y /= $x ; // $y = $y/$x`

De comparaison

Le tableau 5.7 récapitule les opérateurs de comparaison sur les réels.

Tableau 5.7 – Opérateurs de comparaison sur les réels

Opérateur	Signification	Exemple
<	Inférieur à	`if ($x < $y) …`
>	Supérieur à	`if ($x > $y) …`
<=	Inférieur ou égal à	`if ($x <= $y) …`
>=	Supérieur ou égal à	`if ($x >= $y) …`
==	Est égal (après transtypage)	`if ($x == $y) …`
===	Est égal **et** de même type	`if ($x === $y) …`
!=	Différent (non égal) après transtypage	`if ($x != $y) …`
<>	Différent (non égal) après transtypage	`if ($x <> $y) …`
!==	Différent **ou** pas du même type	`if ($x !== $y) …`

Exemple de programme

Le programme `somme_reels_shell.php` affiche la somme de deux réels.

Listing 5.17 – Programme somme_reels_shell.php

```
<?php
 echo 'Entrez deux réels :';
 fscanf(STDIN,"%f %f",$x,$y);
 $somme=$x+$y;
 echo "La somme de $x et de $y = $somme".PHP_EOL;
?>
```

Voici son exécution :

Listing 5.18 – Exécution somme_reels_shell.php

```
$ php somme_reels_shell.php
Entrez deux réels :12.3 3.4
La somme de 12.3 et de 3.4 = 15.7
```

Les fonctions

Le tableau 5.8 présente quelques fonctions sur les réels.

Tableau 5.8 – Fonctions sur les réels

Fonction	Signification	Exemple
`abs()`	Retourne la valeur absolue	`$x= abs(-4.2) ; // 4.2`
`ceil()`	Arrondit à l'entier supérieur	`$i=ceil(4.6); // 5`
`floatval()`	Retourne la valeur réelle de l'argument.	`$x=floatval("2.4"); //2.4` `$x=floatval("2.4abc");//2.4`
`floor()`	Arrondit à l'entier inférieur	`$i=floor(4.6);`

`is_float()`
`is_real()`
`is_double()`
	Retourne vrai si l'argument est un entier faux sinon	`if (is_float($x)) …`

`is_finite()`	Indique si un nombre est fini	`$x=18.34 ;` `if (is_finite($x)) …`
`is_infinite()`	Indique si un nombre est infini	`$x=log(0) ;` `if (is_infinite($x)) …`
`is_nan()`	Indique si une valeur n'est pas un nombre	`$x=acos(8) ;` `if (is_nan($x)) …`

`round()`	Arrondit un nombre à virgule flottante	`$x=round(3.4); // 3`
		`$x=round(3.5); // 4`
		`$x=round(3.6,0);// 4`
		`$x=round(2.95583,2);// 2.96`
		`$x=round(2341757,-3); //2342000`
		`$x=round(5.045,2); // 5.05`
		`$x=round(6.055,2); // 6.06`
		`$x=round(9.5,0,PHP_ROUND_HALF_UP);  // 10`
		`$x=round(9.5,0,PHP_ROUND_HALF_DOWN); // 9`
		`$x=round(9.5,0,PHP_ROUND_HALF_EVEN); // 10`
		`$x=round(9.5,0,PHP_ROUND_HALF_ODD);  // 9`
		`$x=round(8.5,0,PHP_ROUND_HALF_UP);  // 9`
		`$x=round(8.5,0,PHP_ROUND_HALF_DOWN); // 8`
		`$x=round(8.5,0,PHP_ROUND_HALF_EVEN); // 8`
		`$x=round(8.5,0,PHP_ROUND_HALF_ODD);  // 9`

`cos()`	Cosinus	`$x=cos(0.9);//0.62160996827066`
`sin()`	Sinus	`$x=sin(-0.9);//-0.78332690962748`
`exp()`	Exponentielle	`$x=exp(4.23) ;//68.717232173846`
`pi()`	Retourne pi	`$x=pi(); // 3.1415926535898`
`sqrt()`	Racine carrée	`$x=sqrt(3.5) ;// 1.870828693387`

Les fonctions mathématiques

Le tableau 5.9 présente quelques fonctions mathématiques travaillant sur des réels ou des entiers.

Tableau 5.9 – Quelques fonctions mathématiques

Fonction	Signification	Exemple
`base_convert()`	Convertit entre différentes bases	`$chaine=base_convert('A3F4',16,2);// '1010001111110100'`
`bindec()`	Convertit de binaire en décimal	`$i=bindec('11111001');// 249`
`decbin()`	Convertit de décimal en binaire	`$chaine=decbin(-28);// "11111111111111111111111111111111111111111111111111111100100"`
`dechex()`	Convertit de décimal en hexadécimal	`$chaine=dechex(-28);// "ffffffffffffffe4"`
`decoct()`	Convertit de décimal en octal	`$chaine=decoct(-28);// "1777777777777777777744"`
`hexdec()`	Convertit de hexadécimal en décimal	`$i=hexdec('AE');// 174`

`max()`	La plus grande valeur, entière, réelle, ou le tableau de plus grand, ou ayant la plus grande valeur dès la première différence, en cas de nombre d'éléments équivalents	`$i=max(2,4) ; // 4`
		`$x=max(2,4.7) ; // 4.7`
		`$x=max(2.5,4.7) ; // 4.7`
		`$chaine=max(2.5,"4.7") ; // "4.7"`
		`$val=max(array(2,3,4),array(1,0,2,1)); // array(1,0,2,1)`
		`$val=max(array(2,5,2),array(2,5,4)); // array(2,5,4)`

`min()`	La plus petite valeur, entière, réelle, ou le tableau de plus petit, ou le plus ayant la plus petite valeur dès la première différence, en cas de nombre d'éléments équivalents	`$i=min(2,4) ; // 2`
		`$i= min (2,4.7) ; // 2`
		`$x=min(2.5,4.7) ; // 2.5`
		`$chaine=min("2.5",4.7) ; // "2.5"`
		`$val=min(array(2,3,4),array(1,0,2,1)); // array(2,3,4)`
		`$val=min(array(2,5,2),array(2,5,4)); // array(2,5,2)`

`octdec()`	Conversion d'octal en décimal	`$i=octdec('77'); // 63`
`pow()`	Nombre élevé à la puissance	`$i=pow(4,3) ;// 64` `$x=pow(4.5,3) ;// 91.125`

Les constantes mathématiques prédéfinies

Le tableau 5.10 présente quelques constantes mathématiques prédéfinies.

Tableau 5.10 – **Quelques constantes mathématiques**

Fonction	Signification	Exemple
M_E	2.7182818284590452354	Valeur de e
M_PI	3.14159265358979323846	Valeur de PI
M_PI_2	1.57079632679489661923	PI/2
M_1_PI	0.31830988618379067154	1/PI
M_2_PI	0.63661977236758134308	2/PI
M_SQRT2	1.41421356237309504880	sqrt(2)
PHP_ROUND_HALF_UP		Arrondi supérieur
PHP_ROUND_HALF_DOWN		Arrondi inférieur
PHP_ROUND_HALF_EVEN		Arrondi au nombre pair
PHP_ROUND_HALF_ODD		Arrondi au nombre impair
INF		L'infini
NAN		N'est pas un nombre

5.4 Le type « caractère » ou chaîne d'un seul caractère

Particularité de PHP

Le langage PHP ne propose pas de type caractère ! *Un caractère* est vu comme une *chaîne de caractères ne contenant qu'un seul caractère*.

Alors que PHP assimile un caractère à une chaîne, il propose néanmoins des traitements spécifiques à un seul caractère. De plus, les chaînes de caractères ne sont qu'une suite de caractères où chaque caractère est numéroté selon une *table de codage*. Il est donc important de comprendre les traitements et la représentation d'un caractère pour détecter les éventuelles erreurs sur les chaînes de caractères.

C'est pourquoi nous décrivons dans ce chapitre, la notion de *caractère*, et de *tables de codage* avant d'aborder les chaînes de caractères dans le chapitre suivant.

Codage binaire

Chaque caractère d'une chaîne est codé sur *un octet*, sauf pour le codage Unicode **UTF-8** qui peut être sur plusieurs octets. Chaque caractère porte un numéro, son *code*, écrit sous la forme binaire. Ainsi la lettre 'A' porte le numéro 65 dans la table ASCII soit 01000001 en binaire et la lettre 'é' porte le numéro 233 dans la table Iso-Latin1 soit 11101001 en binaire, comme le montre la figure 5.7.

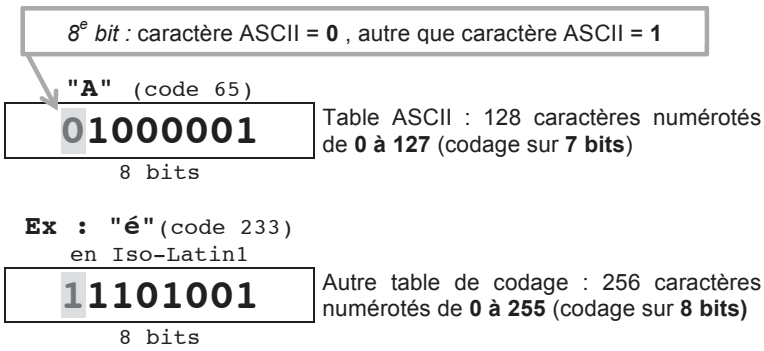

Figure 5.7 – **Codage binaire d'un caractère.**

Les tables de codage

Le code des caractères sans accents est toujours le même. Il est défini dans la table ASCII (*American Standard Code for Information Interchange*). Pour les caractères accentués ou les symboles, le code varie selon la table de codage utilisée. Les principales tables de codage de caractères sont :

- la table **ASCII** de 128 caractères ;
- la table ASCII étendue IBM (page de code 850) de 256 caractères ;
- la table ASCII étendue APPLE de 256 caractères ;
- la table **Iso-Latin1** (page de code **8859-1**) de 256 caractères ;
- la table Unicode **UTF-8** contenant plus de 109 000 caractères.

Nous présentons les tables ASCII et Iso-Latin1, et décrivons le codage UTF-8.

La table ASCII

Cette table présentée à la figure 5.8 correspond au codage des caractères sans accents. Elle est universelle et elle constitue la première partie des autres tables de codage.

Voici quelques remarques sur cette table et les incidences qui en découlent :

- elle contient 128 caractères sans accent numérotés de 0 à 127. Le 8e bit de chaque code sera toujours à 0 ;
- les lettres majuscules sont avant les lettres minuscules. La comparaison de deux caractères, et donc de deux chaînes de caractères, respectera cet ordre. Ainsi, pour deux chaînes de caractères, l'ordre dépendra de la première lettre différente. Monsieur « dupont » sera après monsieur « MARTIN » puisque le 'd' est après le 'M'. Le tri sur deux chaînes de caractères ne sera pas alphabétique si la casse est différente. Pour obtenir un tri alphabétique, il faut que les deux chaînes de caractères soient orthographiées avec la même casse ;
- les codes entre une majuscule et une minuscule équivalente, par exemple 'A' et 'a' diffèrent de 32. Pour convertir 'A' en 'a', et faut ajouter +32 au code ASCII de la majuscule. Il faut retirer 32 pour la conversion inverse ;

- il existe des caractères numériques de '0' à '9' dont les codes vont de 48 à 57 ;
- deux caractères sont particuliers : les caractères LF (n° 10) et CR (n° 13). Ils indiquent un saut de ligne dans les fichiers textes Unix (pour LF) et Windows (pour CR).

Décimal	ASCII	Décimal	ASCII	Décimal	ASCII
0	NUL	43	+	86	V
1	SOH	44	,	87	W
2	STX	45	-	88	X
3	ETX	46	.	89	Y
4	EOT	47	/	90	Z
5	ENQ	48	0	91	[
6	ACK	49	1	92	\
7	BEL	50	2	93	]
8	BS	51	3	94	^
9	HT	52	4	95	_
10	LF	53	5	96	`
11	VT	54	6	97	a
12	FF	55	7	98	b
13	CR	56	8	99	c
14	SO	57	9	100	d
15	SI	58	:	101	e
16	DLE	59	;	102	f
17	DC1	60	<	103	g
18	DC2	61	=	104	h
19	DC3	62	>	105	i
20	DC4	63	?	106	j
21	NAK	64	@	107	k
22	SYN	65	A	108	l
23	ETB	66	B	109	m
24	CAN	67	C	110	n
25	EM	68	D	111	o
26	SUB	69	E	112	p
27	ESC	70	F	113	q
28	FS	71	G	114	r
29	GS	72	H	115	s
30	RS	73	I	116	t
31	US	74	J	117	u
32	Space	75	K	118	v
33	!	76	L	119	w
34	"	77	M	120	x
35	#	78	N	121	y
36	$	79	O	122	z
37	%	80	P	123	{
38	&	81	Q	124	|
39	'	82	R	125	}
40	(	83	S	126	~
41	)	84	T	127	DEL
42	*	85	U		

Figure 5.8 – Table ASCII.

La table Iso-Latin1 (8859-1) ou ANSI 1252

Cette table contient deux parties. Les caractères de 0 à 127 correspondent à la table ASCII. Les caractères de 128 à 255 sont de type « accentués » ou « symboles ». Seule la seconde partie est présentée sur la figure 5.9. Quelques caractères sont particuliers à Windows. Ces caractères différencient la table Iso-Latin1 et la norme américaine ANSI 1252 utilisée par Windows.

Décimal	page code 8859-1		Décimal	page code 8859-1		Décimal	page code 8859-1	
	Caractère	Description		Caractère	Description		Caractère	Description
128			171	«	Guillemet typo ouvrant	214	Ö	O tréma
129			172	¬	négation	215	x	signe de multiplication
130	‚	*windows*	173	_	petit trait d'union	216	Ø	O barré
131	ƒ	*windows*	174	®	marque déposée	217	Ù	U grave
132	„	*windows*	175	¯	accent macron	218	Ú	U aigu
133	…	*windows*	176	°	Degré	219	Û	U circonflexe
134	†	*windows*	177	±	plus ou moins	220	Ü	U tréma
135	‡	*windows*	178	²	exposent 2	221	Y	Y aigu
136	ˆ	*windows*	179	³	exposent 3	222	Φ	Thorn Icelandais Majus
137	‰	*windows*	180	´	accent aigu	223	ß	sz ligature
138	ˇS	*windows*	181	µ	signe micron	224	à	a grave
139	‹	*windows*	182	¶	paragraph	225	á	a aigu
140	Œ	*windows*	183	·	point mi-hauteur	226	â	a circonflexe
141			184	¸	cédille	227	ã	a tilda
142			185	¹	exposent 1	228	ä	a tréma
143			186	º	ordinal masculin	229	å	a ring
144			187	»	Guillemet typo fermant	230	æ	ae ligature
145	‘	*windows*	188	1/4	fraction (un quart)	231	ç	c cédille
146	’	*windows*	189	1/2	fraction (une demi)	232	è	e grave
147	“	*windows*	190	3/4	fraction (trois quarts)	233	é	e aigu
148	”	*windows*	191	¿	point d'interrog inversé	234	ê	e circonflexe
149	•	*windows*	192	À	A grave	235	ë	e tréma
150	–	*windows*	193	Á	A aigu	236	ì	i grave
151	—	*windows*	194	Â	A circonflexe	237	í	i aigu
152	~	*windows*	195	Ã	A tilda	238	î	i circonflexe
153	™	*windows*	196	Ä	A tréma	239	ï	i tréma
154	ˇs	*windows*	197	Å	A ring	240	ð	eth Icelandais
155	>	*windows*	198	Æ	AE (ligature)	241	ñ	n tilda
156	œ	*windows*	199	Ç	C cédille	242	ò	o grave
157			200	È	E grave	243	ó	o aigu
158			201	É	E aigu	244	ô	o circonflexe
159	Ÿ	*windows*	202	Ê	E circonflexe	245	õ	o tilda
160	espace	espace non-sécable	203	Ë	E tréma	246	ö	o tréma
161	¡	point d'exclam. inversé	204	Ì	I grave	247	÷	signe de division
162	¢	cent	205	Í	I aigu	248	ø	o barré
163	£	livre sterling	206	Î	I circonflexe	249	ù	u grave
164	¤	signe monétaire	207	Ï	I tréma	250	ú	u aigu
165	¥	yen	208	Đ	Eth Icelandais	251	û	u circonflexe
166	_	barre verticale brisée	209	Ñ	N tilda	252	ü	u tréma
167	§	section	210	Ò	O grave	253	y	y aigu
168	¨	tréma	211	Ó	O aigu	254	ρ	thorn Icelandais minus
169	©	copyright	212	Ô	O circonflexe	255	ÿ	y tréma
170	ª	ordinal féminin	213	Õ	O tilda			

Figure 5.9 – Table Iso-Latin1 ou ANSI 1252.

La table Unicode UTF-8

Les tables Unicode acceptent plusieurs formats de transformation universelle (UTF = *Universel Transformation Format*) :

- UTF-8 ;
- UTF-16 ;
- UTF-32.

Elles sont développées par le consortium Unicode.

La table UTF-8 correspond à la norme ISO 10646. Elle répertorie plus de 109 000 caractères couvrant 93 écritures (langues). Cette table de codage utilise de 1 à 4 octets pour coder un caractère ou un symbole. Le codage sur 1 octet correspond à la table ASCII.

Un préfixe binaire 0, 110, 1110 ou 11110 pour le premier octet détermine si le caractère est codé respectivement sur 1, 2, 3, ou 4 octets, avec le préfixe 10 pour les octets 2, 3 ou 4, comme l'indique le tableau 5.11. Les valeurs représentées par « b » correspondent aux bits (0 ou 1) significatifs.

Tableau 5.11 – Codage des caractères UTF-8

Nombre d'octets	Nombre de bits significatifs	Numéro Unicode	Numéro en hexadécimal	Représentation binaire
1	7 bits (ASCII)	U+0000 à U+007F	00 à 7F	**0**bbbbbbb
2	11 bits	U+0080 à U+07FF	C280 à DFBF	**110**bbbbb **10**bbbbbb
3	16 bits	U+0800 à U+FFFF	E0A080 à EFBFBF	**1110**bbbb **10**bbbbbb **10**bbbbbb
4	21 bits	U+010000 à U+10FFFF	F0908080 à F7BFBFBF	**11110**bbb **10**bbbbbb **10**bbbbbb **10**bbbbbb

Les caractères de contrôle

Certains caractères non imprimables sont indispensables en programmation. C'est le cas du caractère LineFeed (LF) qui termine chaque ligne dans un fichier texte Unix. Le tableau 5.12 présente quelques caractères de contrôle.

Tableau 5.12 – Caractères de contrôle

Code Caractère	Signification	Exemple
`\e`	Caractère escale (ESC)	`echo "\e" ;`
`\f`	Page suivante (FF)	`echo "\f" ;`
`\n`	Ligne suivante (LF)	`if ( $c == "\n") ...`
`\r`	Retour Chariot (CR)	`if ( $c == "\r") ...`

Tableau 5.12 – **Caractères de contrôle (suite)**

Code Caractère	Signification	Exemple
\t	Tabulation horizontale (TAB)	`echo "\t";`
\v	Tabulation verticale	`if ( $c == "\v" ) ...`
\\	Backslash	`if ( $c == "\\" ) ...`
\'	Apostrophe	`if ( $c == "\'" ) ...`
\"	Guillemet	`if ( $c == "\"" ) ...`
\101	Caractère dont le code ascii est 101 en octal (A)	`echo "\101"  ;`
\x41	Caractère dont le code ascii est 41 en hexadécimal (A)	`echo "\x41"  ;`

Constante caractère

Les guillemets (") ou les apostrophes (') définissent une constante chaîne de caractères, de un ou plusieurs caractères. La différence entre ces deux notations est détaillée à la section 3.2 du chapitre 3. Les syntaxes sont par exemple :

```
$Lettre1 = "A" ; // chaine d'un seul caractère
$Lettre2 = 'A' ; // chaine d'un seul caractère
```

Saisie et affichage

Dans un environnement shell

La saisie et l'affichage dans un environnement shell sont détaillés à la section 3.2 du chapitre 3. En résumé, le format utilisé est `%s` pour le `printf()` ou le `fscanf()`. La saisie lit une chaîne complète. Il faut ensuite extraire le caractère dans le « tableau de caractères » que constitue la chaîne (cf. chapitre sur les données structurées). Le programme `caractere_saisie_affichage_shell.php` présente la saisie et l'affichage *via* les instructions `echo` et `printf()` :

Listing 5.19 – Programme caractere_saisie_affichage_shell.php

```
<?php
 echo "Entrez un caractère : "     ;
 fscanf(STDIN,"%s",$Saisie)        ;
 $Lettre=$Saisie[0]                ;
 echo "Caractère saisi : ".$Lettre.PHP_EOL ;
 printf("Caractère saisi : %s\n",$Lettre)  ;
?>
```

Voici deux exécutions de ce programme.

Listing 5.20 – Exécution de caractere_saisie_affichage_shell.php

```
$ php caractere_saisie_affichage_shell.php
Entrez un caractère : A
Caractère saisi : A
Caractère saisi : A
$ php caractere_saisie_affichage_shell.php
Entrez un caractère : azerty
Caractère saisi : a
Caractère saisi : a
```

La saisie d'un seul caractère peut également se faire par la fonction `fgetc()` et l'affichage par `fputs()` comme indiqué à la section 3.2 du chapitre 3. Le programme `caractere_saisie_affichage2_shell.php` en présente un exemple.

Listing 5.21 – Programme caractere_saisie_affichage2_shell.php

```
<?php
 echo "Entrez un caractère : ";
 $Lettre=fgetc(STDIN);
 echo "Caractère saisi : ".$Lettre.PHP_EOL ;
 fputs(STDOUT,"Caractère saisi : ".$Lettre.PHP_EOL)  ;
?>
```

Remarques

La fonction `fgetc()` lit un seul caractère. Après la lecture, le buffer du clavier contient toujours le caractère `Line Feed` (LF=validation) qui n'est pas lu. Ceci peut perturber la saisie suivante qui lira le caractère restant (LF) sans permettre à l'utilisateur de faire une nouvelle saisie.

Dans un environnement web

La saisie par formulaire est abordée à la section 3.2 du chapitre 3. Le formulaire de saisie retourne une chaîne de caractères. Il faut ensuite extraire le caractère dans le « tableau de caractères » que constitue la chaîne, dans le programme PHP. Le fichier `caractere_saisie_affichage_web.html` présente le formulaire.

Listing 5.22 – Fichier caractere_saisie_affichage_web.html

```
<!DOCTYPE html>
<html>
 <body>
  <form action="caractere_saisie_affichage_web.php" method="post">
   Entrez un caract&egrave;re  <input type="text" name="Saisie"
size="1" /><br/>
   <input type="submit" value="Valider" />
   <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le programme PHP caractere_saisie_affichage_web.php est activé par le formulaire après validation. Il affiche le caractère saisi :

Listing 5.23 – Programme de caractere_saisie_affichage_web.php

```
<!DOCTYPE html>
<html>
 <body>
 <?php
 define("WEB_EOL","<br/>");
 $Saisie=$_POST['Saisie'] ; // -- On récupère les données --
 $Lettre=$Saisie[0]       ; // -- On traite les données --
 echo "Caract&egrave;re saisi : ".$Lettre.WEB_EOL ;
 ?>
 </body>
</html>
```

Les fonctions

Le tableau 5.13 présente les fonctions ord() et chr() qui traitent un seul caractère.

Tableau 5.13 – Fonctions sur un caractère

Fonction	Signification	Exemple
ord()	Retourne le code ASCII du caractère	$code = ord($lettre);
chr()	Retourne le caractère dont le code ASCII est donné en argument	$lettre = chr($code);

5.5 Le type booléen

Définition

Un booléen est une valeur logique qui prend les valeurs *VRAI* ou *FAUX*. Le langage PHP note la valeur VRAI par *true*, *TRUE* ou *1*, et la valeur FAUX par *false*, *FALSE* ou *0* comme le montrent ces différentes syntaxes :

```
$b1=TRUE;  // ou true
$b2=false; // ou FALSE
$b3=1;
$b3=boolval($b3); // Conversion de l'entier en booléen
$b4=0;
$b4=boolval($b4); // Conversion de l'entier en booléen
```

Les opérateurs booléens

Les booléens possèdent les opérateurs *NON*, *ET*, *OU*, et *OU exclusif* (appelé *XOR*). Leur fonctionnement est présenté en *tables de vérité* ou *tables de décision*.

Le ET

Pour comprendre cet opérateur, prenons l'énoncé suivant :

« *Je désire une voiture qui soit **rapide ET** qui soit **rouge*** ».

Selon les propositions suivantes, le résultat sera VRAI ou FAUX :

- Une 2CV GRISE ne convient pas : *rapide* est FAUX, *rouge* est FAUX.
- Une 2CV ROUGE ne convient pas : *rapide* est FAUX, *rouge* est VRAI.
- Une FERRARI GRISE ne convient pas : *rapide* est VRAI, *rouge* est FAUX.
- Une FERRARI ROUGE convient : *rapide* est VRAI, *rouge* est VRAI.

En conclusion, le ET est VRAI quand les deux booléens sont VRAI.

Le OU

Avec cet opérateur, la proposition précédente devient :

« *Je désire une voiture qui soit **rapide OU bien** qui soit **rouge***. »

Voici les différents cas à envisager :

- Une 2CV GRISE ne convient pas : *rapide* est FAUX, *rouge* est FAUX.
- Une 2CV ROUGE convient : *rapide* est FAUX, *rouge* est VRAI.
- Une FERRARI GRISE convient : *rapide* est VRAI, *rouge* est FAUX.
- Une FERRARI ROUGE convient : *rapide* est VRAI, *rouge* est VRAI.

En conclusion, le OU est VRAI si l'un des deux est VRAI (ou les deux).

Le OU exclusif

Le *OU exclusif* ou *XOR* est un OU mais à l'exclusion des deux cas VRAI. La proposition précédente devient :

« *Je désire une voiture qui soit **rapide OU bien** qui soit **rouge, mais pas les deux.*** »

Sa table de vérité est similaire à celle du OU, mais le cas où les deux sont vrais est à exclure.

Le NON

Le NON ne possède qu'un seul opérande. Il est utile toutes les fois où il est plus facile d'exprimer ce qu'on ne veut pas, plutôt que d'énumérer tous les cas désirés. C'est par exemple la proposition suivante :

« *Je désire une voiture qui ne soit **PAS rouge***. »

NON FAUX est VRAI et NON VRAI est FAUX.

La figure 5.10 présente ces différentes tables de vérités.

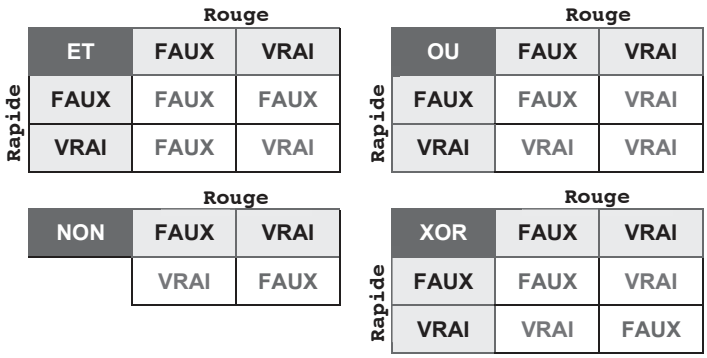

Figure 5.10 – Tables de vérité du ET, OU, XOR et NON.

Les traitements logiques

Ces opérateurs booléens se traduisent en PHP sous la forme de symboles différents selon que le traitement est *logique*, sur la valeur globale de la variable, ou *binaire*, sur chaque élément 0 ou 1 contenu dans la variable.

Les opérateurs logiques

Ces opérateurs s'appliquent à la valeur *logique globale* de la variable, soit les valeurs `true`, `TRUE`, `false`, `FALSE`, `0` ou `1`. Le tableau 5.14 présente ces opérateurs.

Tableau 5.14 – Opérateurs logiques

Opérateur	Signification	Exemple
`&& , and`	ET	`if (($age==10) && ($sexe=='M'))` `if (($age==10) and ($sexe=='M'))`
`|| , or`	OU	`if (($age==10) \|\| ($sexe== 'M'))` `if (($age==10) or ($sexe== 'M'))`
`xor`	OU Exclusif	`if (($age<=65) xor ($impots<=10000))`
`!`	NON	`if (!($age==10))`

Remarque

Les opérateurs de comparaison retournent une valeur booléenne, en comparant des entiers, des réels ou des chaînes de caractères.

Exemple de programme

Le programme `en_activite_shell.php` présente un calcul booléen. Le test `if` porte sur la valeur booléenne de la variable `$actif`.

Listing 5.24 – Programme en_activite_shell.php

```
<?php
    echo "Entrez votre âge : " ;
    fscanf(STDIN,"%d",$age);
    $actif = ( ($age >= 16) && ($age < 65) )  ;
    if ($actif) echo "vous êtes en activité".PHP_EOL;
    else echo "Vous n'êtes pas ou plus en activité".PHP_EOL;
?>
```

Voici un exemple d'exécution :

Listing 5.25 – Exécution de en_activite_shell.php

```
$ php en_activite_shell.php
Entrez votre âge : 15
Vous n'êtes pas ou plus en activité
```

Transtypage

Il est possible de forcer l'interprétation d'une variable ou d'un calcul en booléen, *via* les préfixes `(bool)` ou `(boolean)` positionnés avant la variable ou le calcul, ou *via* `settype()`. Selon la nature et la valeur de la variable, on obtient :

- **0** ou **false** (faux) dans les cas :
  - d'une valeur numérique égale à 0 pour : un entier, un réel ou une chaîne ;
  - d'une chaîne vide ;
  - d'un tableau à 0 éléments ;
  - du type et valeur NULL.
- **1** ou **true** (vrai) pour toutes les autres valeurs.

Voici quelques exemples de syntaxes :

```
// -- valeurs FAUX --
$i=0               ; // cas d'un entier
$bi=(boolean) $i   ; // FAUX
$x=0.0             ; // cas d'un réel
$bx=(boolean) $x   ; // FAUX
$ch="0"            ; // cas d'une chaîne
$bch=(boolean) $ch ; // FAUX
$i=-18             ; // cas d'un entier
$bi=(boolean) $i   ; // VRAI
$x=-18.0           ; // cas d'un réel
$bx=(boolean) $x   ; // VRAI
```

La syntaxe avec l'instruction `settype()` est de la forme :

```
settype($x,"boolean");
```

Les fonctions

Le tableau 5.15 présente quelques fonctions sur les booléens, ou qui retournent une valeur booléenne.

Tableau 5.15 – Quelques fonctions booléennes

Fonction	Signification	Exemple

`boolval()`	Retourne la valeur booléenne de l'argument	`$b=boolval(0); //false`
		`$b=boolval(1); //true`
		`$b=boolval("0");//false`
		`$b=boolval("1");//true`

`is_int()`	Retourne true si l'argument est un entier false sinon	`if (is_int($i)) …`
`is_float()`	Retourne true si l'argument est un réel false sinon	`if (is_float($x)) …`
`is_string()`	Retourne true si l'argument est une chaîne false sinon	`if (is_string($ch)) …`
`is_numeric()`	Retourne true si l'argument est un numérique false sinon	`if (is_numeric($i)) …`

Le traitement binaire

Principe

Ce traitement s'applique uniquement aux entiers. Chaque valeur binaire contenue dans l'entier est assimilée à FAUX (0) ou VRAI (1), comme sur la figure 5.11.

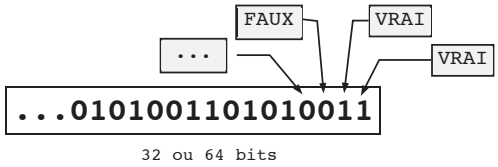

Figure 5.11 – Interprétation des bits en VRAI et FAUX.

Le traitement d'un **ET** ou d'un **OU** ne se fait plus globalement, mais valeur binaire par valeur binaire.

La figure 5.12 présente la variable `$NbAFilter` contenant la valeur `27553`, et la variable `$filtre` contenant la valeur `255`. La représentation du traitement binaire sur les 16 premiers bits montre que cela correspond à un filtrage de l'octet de droite. Le **ET** appliqué bit à bit implique que les bits à 1 de `$filtre` laissent passer les valeurs binaires correspondantes de `$NbAFilter`, et que les valeurs à 0 bloquent le passage. Dans cet exemple, la variable `$resultat` contient la valeur entière `161`, soit la valeur de l'octet de droite de `$NbAFiltrer`.

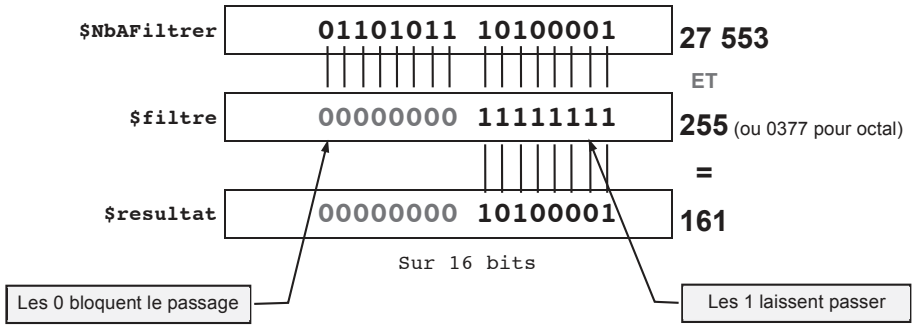

Figure 5.12 – **Traitement binaire.**

Les opérateurs binaires

Le tableau 5.16 présente ces opérateurs binaires.

Tableau 5.16 – **Opérateurs binaires**

Opérateur	Signification	Exemple
&	ET	`$i=$i & 0177;` identique à : `$i&=0177;`
|	OU	`$i=$i | $k ;` identique à : `$i|=$k ;`
^	OU Exclusif	`$i = $j ^ $k ;`
<<	Décalage à gauche	`$j=$i << 2; /* décale 2 bits à gauche */`
>>	Décalage à droite	`$j=$i >> 2; /* décale 2 bits à droite */`
~	Complément à 1 (inversion des bits)	`$k = ~0 ; /* tous les bits à 1 */`

Exemple de programme

Le programme `operateurs_binaires_shell.php` présente le filtrage des 7 bits de droite de la valeur entière saisie :

Listing 5.26 – Programme operateurs_binaires_shell.php

```
<?php
 echo "Entrez le nombre à filtrer : " ;
 fscanf(STDIN,"%d",$NbAFilter);
 $filtre=0b01111111; // laisse passer les 7 bits de droite
 $resultat=$NbAFilter & $filtre ;
 echo "résultat : ".$resultat.PHP_EOL;
?>
```

Voici son exécution :

Listing 5.27 – Exécution de operateurs_binaires_shell.php

```
$ php operateurs_binaires_shell.php
Entrez le nombre à filtrer : 255
resultat : 127
```

5.6 Aucun type NULL

Pour indiquer qu'une variable n'a pas de type et qu'elle est vide, il faut utiliser le mot-clef `NULL` ou `null`. Par exemple :

```
$i=NULL ;
```

Exercices

5.1 Écrire le fichier HTML de saisie de deux entiers *via* un formulaire en méthode POST, et le programme PHP qui effectue leur somme, leur soustraction, leur multiplication, le quotient et le reste de leur division entière (figure 5.13).

Figure 5.13 – Affichage des opérations arithmétiques sur deux entiers.

5.2 Faire un programme qui calcule la *Mensualité* (hors assurance) d'un crédit en fonction du *Capital emprunté*, du Nombre d'années et du *Taux d'intérêt annuel*. La formule de calcul est :

$$MensHA = \text{Cap} \times \text{TauxM} \times \frac{(1 + \text{TauxM})^{\text{NbMois}}}{(1 + \text{TauxM})^{\text{NbMois}} - 1}$$

où :

- *MensHA* est la mensualité hors assurance ;
- *Cap* est le capital emprunté ;
- *TauxM* est le taux mensuel = taux annuel/12 ;
- *NbMois* est le nombre de mois = nombre d'années × 12.

Le programme affichera :

- la mensualité hors assurance (MensHA) ;
- le montant de l'assurance mensuelle (AssM = Cap × TauxM) ;
- la mensualité assurance comprise (MensAC = MensHA + AssM).

Arrondir les calculs à la deuxième décimale. Voici un exemple de l'exécution attendue (les saisies sont en gras) :

```
$ php emprunt_shell.php
Capital : 300000
Nombre d'années : 20
Taux Annuel Hors Assurance (ex : 2.6) : 2.8
Taux de l'Assurance (0.29) : 0.33
Mensualité Hors Assurance     : 1633.92
Coût de l'Assurance par mois  : 82.5
Mensualité Assurance Comprise : 1716.42
```

5.3 Adapter le programme précédent pour effectuer la saisie dans un formulaire HTML en méthode POST, et afficher le résultat dans une page web (figure 5.14).

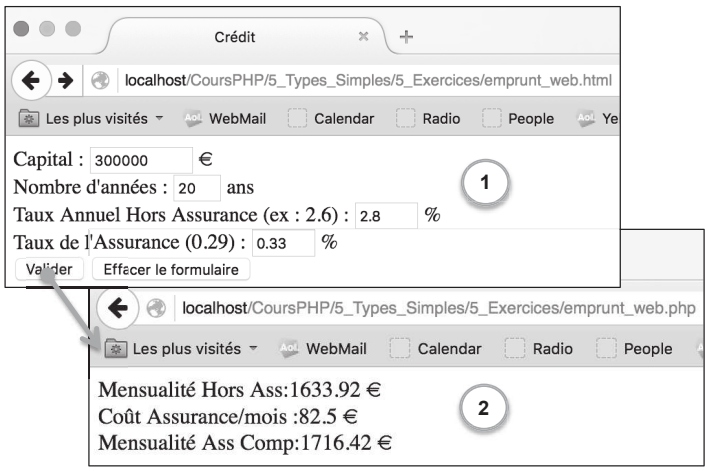

Figure 5.14 – Calcul de la mensualité d'un crédit.

Solutions

5.1 Le fichier `entiers_POST_web.html` présente le formulaire de saisie :

Listing 5.28 – Fichier `entiers_POST_web`***.html***

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Saisie de deux entiers</title>
 </head>
 <body>
  <form action="entiers_POST_web.php" method="post">
  Entrez deux entiers :<br/>
  Premi&egrave;re valeur : <input type="text" name="i"
size="5" /><br/>
  Deuxi&egrave;me valeur : <input type="text" name="j"
size="5" /><br/>
  <input type="submit" value="Valider" />
  <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le programme `entiers_POST_web.php` récupère et traite les données, effectue les opérations et affiche les résultats :

Listing 5.29 – Programme entiers_POST_web.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Opérations de deux entiers</title>
 </head>
 <body>
 <?php
 define("WEB_EOL","<br/>");
 $i=$_POST['i']; // -- On récupère les données --
 $j=$_POST['j'];
 $i=intval($i) ; // -- On traite les données --
 $j=intval($j) ;
 $somme         = $i+$j; // -- On effectue les calculs --
 $soustraction  = $i-$j;
 $mutiplication = $i*$j;
 $reste         = $i%$j;
```

```
$quotient      = ($i-$reste)/$j;
 // -- On affiche les résultats --
echo "$i + $j   = ".$somme.WEB_EOL;
echo "$i - $j   = ".$soustraction.WEB_EOL;
echo "$i * $j   = ".$mutiplication.WEB_EOL;
echo "$i % $j   = ".$reste.WEB_EOL;
echo "$i DIV $j = ".$quotient.WEB_EOL;
?>
</body>
</html>
```

Le programme précédent peut être modifié pour afficher directement les calculs.

```
echo "$i + $j   = ".($i+$j).WEB_EOL;
echo "$i - $j   = ".($i-$j).WEB_EOL;
echo "$i * $j   = ".($i*$j).WEB_EOL;
echo "$i % $j   = ".($i%$j).WEB_EOL;
echo "$i DIV $j = ".(($i-($i%$j))/$j).WEB_EOL;
```

5.2 Le programme `emprunt_shell.php` calcule la mensualité hors assurance, assurance comprise, et le coût mensuel de l'assurance d'un crédit.

Listing 5.30 – Programme emprunt_shell.php

```
<?php
 echo "Capital : "                  ; // Saisie des données
 fscanf(STDIN,"%f",$Cap)           ;
 echo "Nombre d'années : "          ;
 fscanf(STDIN,"%d",$NbAn)          ;
 echo "Taux Annuel Hors Assurance (ex : 2.6) : " ;
 fscanf(STDIN,"%f",$TauxAnnuel)    ;
 printf("Taux Assurance (0.29) : ");
 fscanf(STDIN,"%f",$TauxAssAnnuel) ;
 $NbMois   = $NbAn*12                ; // Calculs
 $TauxM    = ($TauxAnnuel/100)/12    ;
 $calcul1  = $Cap*$TauxM             ;
 $calcul2  = pow((1+$TauxM),$NbMois) ;
 $calcul3  = $calcul2-1              ;
 $TauxAssM = ($TauxAssAnnuel/100)/12 ;
 $MensHA   = round(($calcul1*($calcul2/$calcul3)),2);
 $AssM     = round(($Cap*$TauxAssM),2);
 $MensAC   = $MensHA+$AssM           ;
 // Affichage de la mensualité
 echo "Mensualité Hors Assurance     : ".$MensHA.PHP_EOL;
 echo "Coût de l'Assurance par mois  : ".$AssM.PHP_EOL  ;
 echo "Mensualité Assurance Comprise : ".$MensAC.PHP_EOL;
?>
```

5.3 La solution présente deux fichiers : le formulaire HTML de saisie et le programme PHP de calcul. Voici le fichier de saisie, `emprunt_web.html`.

Listing 5.31 – Fichier emprunt_web.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Crédit</title>
 </head>
 <body>
  <form action="emprunt_web.php" method="post">
  Capital : <input type="text" name="Cap" size="10" /> &euro;<br/>
  Nombre d'ann&eacute;es : <input type="text" name="NbAn" size="3" />
ans<br/>
  Taux Annuel Hors Assurance (ex : 2.6) : <input type="text"
name="TauxAnnuel" size="5" /> %<br/>
  Taux de l'Assurance (0.29) : <input type="text" name="TauxAssAnnuel"
size="5" /> %<br/>
  <input type="submit" value="Valider" />
  <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le programme `emprunt_web.php` calcule la mensualité, hors assurance, assurance comprise et le coût mensuel de l'assurance, à partir des données du formulaire.

Listing 5.32 – Programme emprunt_web.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Crédit</title>
 </head>
 <body>
 <?php
 define("WEB_EOL","<br/>");
 $Cap          = $_POST['Cap'] ; // On récupère les données
 $NbAn         = $_POST['NbAn'];
 $TauxAnnuel   = $_POST['TauxAnnuel']   ;
 $TauxAssAnnuel= $_POST['TauxAssAnnuel'];
 $Cap           = floatval($Cap); // On traite les données
 $NbAn          = intval($NbAn) ;
 $TauxAnnuel    = floatval($TauxAnnuel)   ;
 $TauxAssAnnuel = floatval($TauxAssAnnuel);
 $NbMois   = $NbAn*12 ; // On effectue les calculs
 $TauxM    = ($TauxAnnuel/100)/12;
 $calcul1  = $Cap*$TauxM         ;
 $calcul2  = pow((1+$TauxM),$NbMois);
```

```
$calcul3  = $calcul2-1             ;
$TauxAssM = ($TauxAssAnnuel/100)/12;
$MensHA   = round(($calcul1*($calcul2/$calcul3)),2);
$AssM     = round(($Cap*$TauxAssM),2);
$MensAC   = $MensHA+$AssM           ;
// -- Affichage de la mensualité --
echo "Mensualit&eacute; Hors Ass:".$MensHA." &euro;".WEB_EOL;
echo "Co&ucirc;t Assurance/mois :".$AssM." &euro;".WEB_EOL;
echo "Mensualit&eacute; Ass Comp:".$MensAC." &euro;".WEB_EOL;
?>
</body>
</html>
```