Les opérateurs et opérateurs logiques

Table des matières

I. Contexte

Contexte

Lorsque nous développons des programmes informatiques, la comparaison de valeurs et la prise de décisions conditionnelles sont des concepts fondamentaux. Dans le domaine du code, nous sommes souvent confrontés à des situations où nous devons évaluer différentes variables, valeurs ou conditions pour prendre des décisions éclairées.

La comparaison de valeurs dans le code implique l'évaluation de différentes données afin de déterminer leur relation les unes par rapport aux autres. Nous utilisons des opérateurs de comparaison tels qu'égalité, infériorité, supériorité, etc., pour comparer les valeurs et obtenir un résultat binaire (vrai ou faux). Ces comparaisons sont essentielles pour effectuer des actions spécifiques en fonction des résultats obtenus.

Une fois que nous avons comparé les valeurs dans notre code, nous sommes souvent confrontés à des décisions conditionnelles. Les décisions conditionnelles nous permettent de définir des blocs de code qui seront exécutés uniquement si certaines conditions sont remplies. Par exemple, dans une structure "if-else", si une condition est évaluée comme vraie, le code à l'intérieur du bloc "if" sera exécuté, sinon le code à l'intérieur du bloc "else" sera exécuté. Cela nous permet d'adapter le comportement de notre programme en fonction de différents scénarios.

II. Les opérateurs arithmétiques et de comparaison

A. Les opérateurs arithmétiques

Découverte des symboles arithmétiques

Nous avons tous appris les 4 opérations arithmétiques suivantes dans votre enfance :

- L'addition, symbolisée par le caractère « + »
- La soustraction, symbolisée par le caractère « – »
- La multiplication, symbolisée par le caractère « * »
- La division, symbolisée par le caractère « / »

Nous n'allons pas, ici, refaire les enseignements de l'école primaire. Cependant, nous allons vous montrer comment utiliser ces symboles en programmation pour effectuer ces opérations.

En complément de ces 4 opérations de base, on trouve aussi 2 autres opérations, qui sont connues de ceux qui ont suivi un cursus scolaire scientifique : le modulo, c'est-à-dire le reste de la division ; et l'exponentiation, c'est-à-dire la puissance d'un nombre notée en exposant, symbolisée par « ** ».

Exemple

Penchons-nous sur quelques exemples pour bien comprendre les concepts de ces deux dernières opérations.

Lorsqu'on réalise une division, on divise un dividende par un diviseur. On obtient alors un résultat appelé quotient. Par exemple, dans l'opération « *15 / 5 = 3* » : le dividende est 15, le diviseur est 5, le quotient est 3. La difficulté majeure avec la division, c'est qu'elle amène souvent des fractions.

En effet, quand on divise deux nombres entiers, le résultat exact n'est pas un nombre entier, mais un nombre décimal, c'est-à-dire une fraction. Par exemple, dans l'opération « *15 / 2 = 7,5* », le dividende est 15, le diviseur est 2, le résultat exact est 7,5, mais il n'est pas le quotient. Le quotient est toujours un nombre entier. Dans cet exemple, le quotient est donc 7. Il est possible de représenter l'égalité « *15 / 2 = 7,5* » par « *15 = 2 * 7 + 1* ». L'opération de division est la réciproque de l'opération de multiplication. Dans cet exemple, le nombre 1 est appelé le reste.

Prenons un nouvel exemple pour bien comprendre : calculons « *14 / 3* ». Le dividende 14 n'est pas divisible par le diviseur 3, car on obtient des tiers, impossibles à écrire en nombre décimal. Le nombre multiple de 3 inférieur à 14 à considérer est alors 12, car « *12 = 3 * 4* ». Dans ce cas, le quotient est 4, car « *12 / 3 = 4* ». Nous pouvons alors

écrire l'égalité « *14 = 3 * 4 + 2* ». Le reste de la division de 14 par 3 vaut 2. C'est ce que nous appelons le modulo. En programmation, le modulo est souvent symbolisé par le caractère « *%* », à prononcer « *modulo* » et non « *pourcent* ». Dans le cas de notre dernier exemple, on écrit alors « *14 % 3 = 2* ».

La puissance d'un nombre, indiquée avec une notation en exposant, correspond à la multiplication de ce nombre avec lui-même, et autant de fois qu'il est indiqué par l'exposant. Par exemple, 5^3, prononcé « *5 puissance 3* », correspond à la multiplication 5 * 5 * 5. Autre exemple : 10^6 vaut un million. Cette opération s'appelle l'exponentiation, car elle consiste à utiliser la fonction exponentielle. Elle est souvent notée, en programmation, à l'aide de « ** », prononcé « puissance ». Ainsi, l'écriture « *3 ** 5* » correspond au calcul « *3 * 3 * 3 * 3 * 3* ».

Méthode **Manipulation des opérateurs arithmétiques**

Les exercices de ce cours sont réalisés en pseudo-code pour faciliter votre compréhension.

```
1 afficher 2 + 3 // Affiche 5
2 afficher 10 - 3 // Affiche 7
3 afficher 3 * 4 // Affiche 12
4 afficher 15 / 3 // Affiche 5
```

Mettons maintenant en œuvre les deux dernières opérations arithmétiques présentées :

```
1 afficher 2 **  3 // Affiche 8 (correspond à 2 * 2 * 2)
2 afficher 3 ** 4 // Affiche 81 (correspond à 3 * 3 * 3 * 3)
3 afficher 15 % 2 // Affiche 1 (car 15 = 2 * 7 + 1)
4 afficher 23 % 4 // Affiche 3 (car 23 = 4 * 5 + 3)
```

B. La priorité des opérateurs

Les niveaux de priorité des opérateurs

Comme vous le savez probablement, la multiplication est prioritaire sur l'addition et la soustraction. C'est une chose, mais ce n'est pas tout. Voici la liste des opérateurs ordonnés selon leur niveau de priorité :

- « ** », l'exponentiation est l'opération la plus prioritaire.
- « * », « / », « % », multiplication, division et modulo.
- « + », « - », addition et soustraction sont les opérations les moins prioritaires.

Lorsque des opérateurs de mêmes niveaux se succèdent, il convient de réaliser les opérations correspondantes en respectant l'ordre de lecture de gauche à droite. Par exemple, dans le cas de « *5 - 3 + 2* », il faut d'abord effectuer « *5 - 3* » qui correspond à 2, puis effectuer « *2 + 2* » qui amène au résultat 4.

Méthode **Tester les priorités des opérateurs**

Voici un exemple concret en PHP suivi d'un exemple en pseudo-code :

```
1 affiche  1-2**3/4+3
2 ⇒ résultat 2
3 affiche 2**3
4 ⇒ résultat 8
5 affiche 8/4
6 ⇒résultat 2
7 affiche  1-2
8 ⇒résultat -1
9 affiche -1+3
10 ⇒résultat 2
```

Ici :

- 1 - 2 ** 3 / 4 + 3 : c'est l'expression arithmétique à évaluer.
- 2 ** 3 : l'opérateur "**" représente l'exponentiation. Ici, cela signifie 2 élevé à la puissance 3, donc 2 * 2 * 2, ce qui donne 8.
- 1 - 8 / 4 + 3 : on effectue les opérations dans l'ordre de priorité des opérateurs. La division est effectuée en premier, donc 8 divisé par 4 donne 2.
- 1 - 2 + 3 : ensuite, on effectue les opérations de soustraction et d'addition de gauche à droite. 1 moins 2 donne - 1, puis - 1 plus 3 donne 2.

En résumé, ce code évalue l'expression arithmétique 1 - 2 ** 3 / 4 + 3 et affiche le résultat.

Les parenthèses

Vous avez sans doute appris dans votre jeunesse à utiliser les parenthèses dans vos calculs en mathématiques. Ces symboles « *(« et »)* » définissent les opérations à effectuer dans un ordre précis. Le fonctionnement est identique en programmation.

Méthode **Utiliser des parenthèses**

Regardons quelques exemples pratiques :

```
résultat <- 1 + 2 * 3 // Effectue les opérations arithmétiques
afficher résultat, "\n" // Affiche le résultat suivi d'une nouvelle ligne

résultat <- (1 + 2) * 3 // Effectue les opérations arithmétiques en utilisant des parenthèses
afficher résultat, "\n" // Affiche le résultat suivi d'une nouvelle ligne

résultat <- (3 + 2) / (3 + 2) * (3 + 2) // Effectue les opérations arithmétiques en utilisant des parenthèses
afficher résultat, "\n" // Affiche le résultat suivi d'une nouvelle ligne

résultat <- (3 + 2) / ((3 + 2) * (3 + 2)) // Effectue les opérations arithmétiques en utilisant des parenthèses
afficher résultat, "\n" // Affiche le résultat suivi d'une nouvelle ligne
```

En cas de doute dans vos programmes, mieux vaut mettre des parenthèses qui aident à la compréhension, même si elles sont inutiles pour le programme. Vous saurez maintenant être vigilant sur vos opérations en tenant compte des priorités des opérateurs.

C. Les opérateurs de comparaison

Rappel **Les valeurs booléennes**

Les langages de programmation contiennent les valeurs booléennes « *true* » et « *false* » . True signifie Vrai en anglais, tandis que False correspond à Faux. Ces notions ont été travaillées par George Boole au XIX^{e} siècle en Angleterre. Il est considéré comme le père de la logique moderne et son nom est désormais utilisé comme adjectif en informatique en son honneur. Vous serez amené à utiliser des variables booléennes, c'est-à-dire des variables qui ont comme valeurs « *true* » ou « *false* ».

Rappel **Les opérateurs de comparaison**

Les opérateurs de comparaison sont les suivants :

- a == b : compare l'égalité des expressions a et b, et est évalué à True uniquement si a est égale à b, sinon False.
- a != b : compare la non-égalité des expressions a et b, et est évaluée à True uniquement si a et b ne sont pas égales. Si a est égale à b, cette comparaison est évaluée à False. On prononce cette comparaison « *a différent de b* ».
- a < b : compare si a est plus petit que b, et est évalué à True si a est inférieur à b. Si a est plus grand que b, ou égal à b, cette comparaison est évaluée à False.
- a <= b : compare si a est plus petit ou égal à b, et est évalué à True si a est inférieur ou égal à b. Si a est strictement plus grand que b, cette comparaison est évaluée à False.
- a > b : compare si a est plus grand que b, et est évalué à True si a est supérieur à b. Si a est plus petit que b, ou égal à b, cette comparaison est évaluée à False.
- a >= b : compare si a est plus grand ou égale à b, et est évalué à True si a est supérieur ou égale à b. Si a est strictement plus petit que b, cette comparaison est évaluée à False.

Méthode **Tester les opérateurs de comparaison**

```
afficher type_et_valeur(true) // Affiche le type et la valeur du booléen true
afficher type_et_valeur(false) // Affiche le type et la valeur du booléen false
afficher type_et_valeur(2 == 1) // Affiche le résultat du test "2 est-il égal à 1"
afficher type_et_valeur(2 > 1) // Affiche le résultat du test "2 est-il supérieur à 1"
afficher type_et_valeur(2 != 1) // Affiche le résultat du test "2 est-il différent de 1"
afficher type_et_valeur(2 <= 1) // Affiche le résultat du test "2 est-il inférieur ou égal à 1"
```

Vous connaissez maintenant le principe des expressions booléennes et savez manipuler les opérateurs de comparaison.

D. Utiliser les opérateurs de comparaison

Méthode

L'exercice suivant consiste à vérifier la disponibilité d'un solde suffisant sur votre compte bancaire pour réaliser un achat à l'aide de votre carte de paiement. Nous considérons qu'il s'agit d'une carte à contrôle de solde, qui ne permet pas d'être à découvert. Nous ne connaissons bien évidemment pas à l'avance ni le montant de la transaction ni le solde du compte bancaire au moment du contrôle. Nous représentons le montant par la lettre M et le solde par la lettre S. Si le solde est suffisant, alors le débit est autorisé et l'achat est réalisé. Si le solde est insuffisant, alors l'autorisation de débit est refusée et l'achat n'est pas réalisé. Nous pouvons formaliser cette situation par :

IF (S supérieur ou égal à M), Autoriser la transaction, ELSE Refuser la transaction.

```
m <- Saisir montant
s <- Saisir solde
Si (s >= m) Alors
    Afficher "Autoriser la transaction"
Sinon
    Afficher "Refuser la transaction"
```

Méthode

L'exercice suivant consiste à déterminer la mention obtenue lors d'un examen scolaire. Il convient de considérer tous les cas suivants :

- Pour une note strictement inférieure à 10 : l'examen est "non obtenu".
- Pour une note comprise entre 10 et strictement inférieure à 12 : l'examen est obtenu avec "mention passable".
- Pour une note comprise entre 12 et strictement inférieure à 14 : l'examen est obtenu avec "mention assez bien".
- Pour une note comprise entre 14 et strictement inférieure à 16 : l'examen est obtenu avec "mention bien".
- Pour une note au moins égale à 16 : l'examen est obtenu avec "mention très bien".

Nous allons représenter la note d'examen par la lettre N. Nous vous proposons ci-dessous deux manières de concevoir les tests conditionnels pour mieux comprendre les différentes approches possibles :

1. Première manière :

```
n <-Saisir la note

Si n < 10 Alors
    Afficher "non obtenu"
Sinon Si n < 12 Alors
    Afficher "mention passable"
Sinon Si n < 14 Alors
    Afficher "mention assez bien"
Sinon Si n < 16 Alors
    Afficher "mention bien"
Sinon
    Afficher "mention très bien"
Fin

2.  Seconde manière :
Début
n <-Saisir la note

Si n >= 16 Alors
    Afficher "mention très bien"
Sinon Si n >= 14 Alors
    Afficher "mention bien"
Sinon Si n >= 12 Alors
    Afficher "mention assez bien"
Sinon Si n >= 10 Alors
    Afficher "mention passable"
Sinon
    Afficher "non obtenu"
```

Vous savez maintenant utiliser les opérateurs de comparaison dans les structures de contrôle de type "if".

III. Les opérateurs logiques

A. Découvrir les opérateurs logiques

L'algèbre de Boole

Dans la première partie de ce cours, nous avons introduit les notions de la logique de Boole. L'algèbre consiste en un ensemble de règles, qui décrit la manière d'effectuer des calculs mathématiques. On peut faire l'analogie avec la grammaire d'une langue, qui décrit comment les mots s'articulent les uns aux autres. L'algèbre de Boole définit 3 types d'opérations, qui opèrent donc sur des valeurs VRAI et FAUX. Dans les paragraphes suivants, nous notons A et B, deux propriétés booléennes, qui valent donc VRAI ou FAUX.

Opérateur "!"

La première opération que nous vous présentons se nomme la négation. Elle exprime le contraire d'une valeur. Ainsi, la négation de VRAI est FAUX, et la négation de FAUX est VRAI. C'est assez trivial. Pour écrire mathématiquement la négation, on utilise en français le mot "non". On peut donc écrire NON VRAI = FAUX et NON FAUX = VRAI. En anglais, on écrirait plutôt NOT TRUE = FALSE et NOT FALSE = TRUE. Pour exprimer la négation en langage PHP, on utilise le symbole "!". Nous verrons des exemples concrets plus loin.

Opérateur "ET"

La deuxième opération que nous vous présentons se nomme la conjonction et utilise l'opérateur "et". Cette opération est définie de la façon suivante : (A et B) est Vrai si A est Vrai et B est Vrai (les deux doivent être Vrai). Pour illustrer cette opération, jouons aux devinettes, par exemple avec le jeu « *Qui est-ce ?* ». Vous connaissez probablement ce jeu à deux joueurs, où un joueur choisit un personnage, tandis que l'autre doit retrouver ce personnage parmi tout un groupe. Le joueur qui cherche pose des questions sur les caractéristiques physiques des personnages afin de trouver, par étapes, la correspondance avec le personnage à retrouver, en écartant ceux dont les propriétés ne sont pas satisfaites. Par exemple :

- Le personnage est-il un homme ?
- Le personnage est-il blond ?
- Le personnage porte-t-il des lunettes ?
- Le personnage porte-t-il un chapeau ?

Imaginons que la réponse à chacune des questions ci-dessus ait été OUI, on pourrait dire que le personnage est : un homme ET blond ET portant des lunettes ET portant un chapeau. Chaque réponse amène une conjonction des propriétés. Le "et" se traduit en anglais et en PHP par "and". Nous verrons des exemples concrets plus loin.

Opérateur "OU"

La troisième opération que nous vous présentons se nomme la disjonction et utilise l'opérateur "ou". Cette opération est définie de la façon suivante : (A ou B) est Vrai, si A est Vrai ou B est Vrai (il suffit d'un seul sur les deux). Pour illustrer cette opération, nous allons préparer une fête costumée. Vous êtes invité à participer à un événement qui requiert de porter un costume. Vous vous rendez donc dans votre magasin préféré, pour faire votre choix de costume. Votre seule contrainte est, au final, d'avoir un costume, peu importe le genre de costume. Vous pourriez même acheter plusieurs costumes dans la boutique, tant que vous avez au moins un costume en rentrant chez vous. Votre choix pourrait s'exprimer de la façon suivante : un costume = un costume de super héros OU un costume de gendarme OU un costume de pompier OU un costume de médecin OU un costume de religieux. Si vous avez sélectionné au moins un costume parmi tous les choix disponibles, alors vous avez un costume pour votre occasion. D'un point de vue logique, on pourrait écrire la relation "C = H ou G ou P ou M ou R" où :

- "C" correspond à « *possède un Costume* ».
- "H" correspond à « *possède un costume de Héros* ».
- "G" correspond à « *possède un costume de Gendarme* ».

- “P” correspond à « *possède un costume de Pompier* ».
- “M” correspond à « *possède un costume Médecin* ».
- “R” correspond à « *possède un costume de Religieux* ».

Le “ou” se traduit en anglais par “or”. Nous verrons des exemples concrets plus loin.

Complément **Opérateur “ou exclusif”**

A partir de ces 3 opérations - le “non”, le “et”, le “ou” -, on définit une quatrième opération que l’on appelle le « *ou exclusif* ». Cette opération est notée « *xor* » en PHP, et a le comportement suivant : (A xor B) est Vrai si A est Vrai ou B est Vrai, mais pas les deux ensembles. Pour illustrer ce concept, allons au restaurant pour commander le menu du jour : nous voyons sur la carte la formule économique “entrée + plat” ou “plat + dessert”. On comprend donc qu’avec un plat, il faut choisir entre une entrée ou un dessert, mais qu’on ne peut pas avoir les deux dans cette formule, il s’agit d’un “ou exclusif” : le fait de choisir une option exclut automatiquement l’autre option. Si nous nommons l’entrée par la lettre E et le dessert par la lettre D, nous pouvons écrire que cette formule du menu correspond à “E xor D”. Nous vous disions au début de ce paragraphe que cette opération est définie à partir des 3 précédentes. En effet, dans cet exemple, on a : E xor D = (E et non D) ou (non E et D). Ce qui signifie littéralement en français : (entrée et non dessert) ou (non entrée et dessert). Vous voyez donc que le “xor” est bien défini avec “non”, “et”, et “ou”.

Remarque

Pour finir cette présentation théorique, sachez que dans certains langages, il existe aussi 2 autres notations : le “&&” correspond au “and” et le “||” correspond au “or”. Il est possible, à votre niveau, d’utiliser les deux notations, textuelle ou symbolique.

Méthode **Tester les opérateurs logiques**

Voici quelques exemples commentés, vous permettant de comprendre l’utilisation de ces opérateurs logiques :

```
Afficher (Faux ou Vrai)  # affiche Vrai
Afficher ((Vrai ou Faux) et Vrai)  # affiche Vrai
Afficher ((Vrai xor Faux) ou (non Vrai))  # affiche Vrai
```

B. Utiliser les opérateurs logiques

L’utilité des opérateurs logiques

Les opérateurs logiques sont particulièrement utiles dans les structures de contrôles lorsque les tests conditionnels prennent en compte plusieurs critères. Grâce à eux, on combine des expressions booléennes entre elles en fonction de ce qui est recherché. Dans cette grande section, nous allons développer 3 exemples, du plus simple au plus complexe. Chaque fois, nous expliquerons le contexte de l’exercice ainsi qu’une solution. Nous fournirons un pseudo-code qui répond à la problématique rencontrée. Pour finir, une vidéo vous montrera les fonctionnements implémentés pour chacun de ces 3 problèmes.

Méthode

Pour ce premier exercice, plaçons-nous dans la peau d’un étudiant, qui travaille beaucoup pour réussir ses études, mais qui aime bien aussi sortir pour se divertir. Comme cet étudiant est sage, il ne se permet pas de sortir n’importe quand. Pour s’autoriser à sortir, il a défini 4 critères. Ces critères l’aident à déterminer s’il peut se permettre de sortir, ou si les conditions ne sont pas réunies pour cela, pour qu’il ne sorte pas. Le premier critère consiste à savoir s’il a assez d’argent à dépenser pour sa sortie. Nous noterons A (comme Assez d’argent) ce premier critère. Le deuxième critère indique s’il a fini ses devoirs scolaires, car cet étudiant studieux ne s’autorise pas à sortir s’il a du travail à faire. Nous noterons F (comme devoirs Finis) ce critère. Les troisième et quatrième critères concernent sa mobilité, car il doit bien pouvoir se déplacer. Le troisième critère concerne la disponibilité

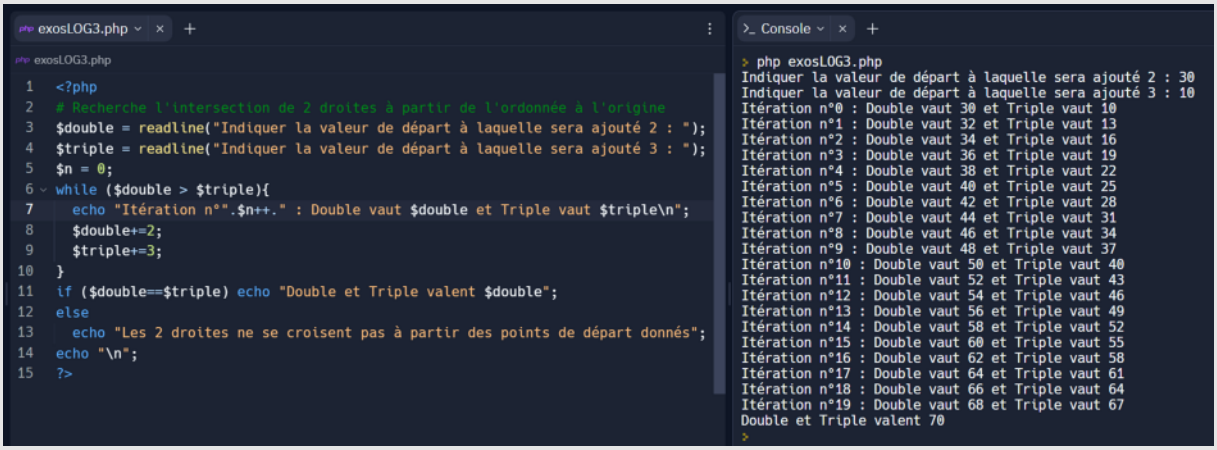
de la voiture familiale. Nous noterons V (comme Voiture) ce critère. Le dernier critère, au cas où la voiture familiale ne serait pas disponible, consiste à s'assurer que les transports en commun fonctionnent bien, en particulier qu'il n'y ait pas de grève. Nous noterons G, le fait que les transports soient en Grève, ou non. Si vous avez bien suivi jusque-là, vous avez compris que cet étudiant va sortir s'il possède assez d'argent ET qu'il a fini ses devoirs et que la voiture est disponible ou bien que les transports ne sont pas en grève. Ainsi on peut écrire : si A et F et (V ou non G). Nous allons donc implémenter cette multi-condition, en PHP puis en pseudo-code. Pour commencer le code, nous allons demander à l'utilisateur de saisir lui-même les valeurs pour chacun de ces 4 critères. L'utilisateur doit donc répondre par "oui" ou "non". Si la réponse est "o" (pour "oui"), alors nous enregistrons True dans la variable correspondante, sinon False.

```
a <- DemanderUtilisateur("Ai-je Assez d'argent pour effectuer cette sortie ? (o/n)") == 'o' ? vrai : faux
f <- DemanderUtilisateur("Ai-je Fini mes devoirs ? (o/n)") == 'o' ? vrai : faux
v <- DemanderUtilisateur("Notre Voiture individuelle est-elle disponible ? (o/n)") == 'o' ? vrai : faux
g <- DemanderUtilisateur("Les transports en commun sont-ils en Grève ? (o/n)") == 'o' ? vrai : faux

Si (a ET f ET (v OU NON g)) Alors
    Afficher "Les conditions sont réunies pour sortir."
Sinon
    Afficher "Les conditions ne permettent pas de sortir."
FinSi

Afficher une nouvelle ligne
```

Méthode

Pour ce deuxième exercice, nous allons utiliser une autre structure de contrôle : "while". Elle permet de créer des boucles, c'est-à-dire que le code sera répété lors de son exécution, tant que le test conditionnel est Vrai. Nous nous plaçons dans le domaine des mathématiques, et allons déterminer par itérations successives, si deux droites se croisent. Nous partons d'une valeur initiale DOUBLE à laquelle on ajoute 2, ainsi que d'une valeur initiale TRIPLE à laquelle on ajoute 3. Donc TRIPLE grandit plus vite que DOUBLE. Ainsi, si au départ DOUBLE est plus grand que TRIPLE, alors TRIPLE va rejoindre DOUBLE. Lorsque DOUBLE et TRIPLE arrivent à la même valeur, cela correspond au point d'intersection des droites 2x+DOUBLE et 3x+TRIPLE.

Nous avons implémenté la situation décrite. Nous avons exécuté le code avec la valeur initiale de 30 pour DOUBLE et la valeur initiale de 10 pour TRIPLE. A chaque itération, on affiche l'accroissement de DOUBLE et TRIPLE, jusqu'à ce qu'ils valent 70. Le test conditionnel du "while" permet d'exécuter la boucle tant que DOUBLE est supérieur à TRIPLE.

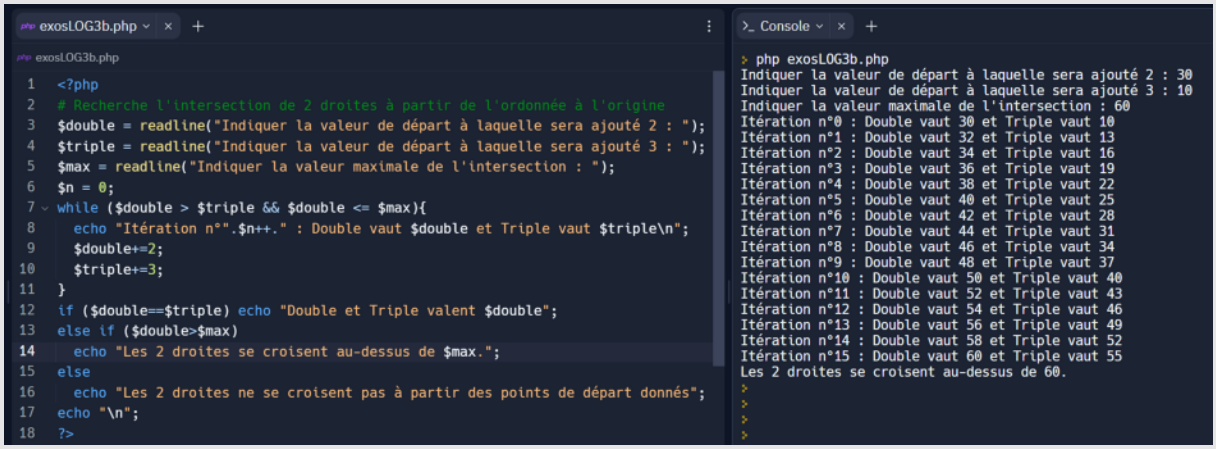


Exécutez ce code plusieurs fois dans la zone ci-dessous pour bien comprendre son fonctionnement. Quand vous l'aurez bien compris, nous le complexifierons davantage juste après.

```php
<?php
# Recherche l'intersection de 2 droites à partir de l'ordonnée à l'origine
$double = readline("Indiquer la valeur de départ à laquelle sera ajouté 2 : ");
$triple = readline("Indiquer la valeur de départ à laquelle sera ajouté 3 : ");
$n = 0;
while ($double > $triple){
  echo "Itération n°".$n++." : Double vaut $double et Triple vaut $triple\n";
  $double+=2;
  $triple+=3;
}
if ($double==$triple) echo "Double et Triple valent $double";
else
  echo "Les 2 droites ne se croisent pas à partir des points de départ donnés";
echo "\n";
?>
```

Maintenant que vous avez bien compris comment croissent DOUBLE et TRIPLE jusqu'à se rejoindre, nous allons comparer leur intersection à une valeur donnée. Nous demandons alors à l'utilisateur une troisième valeur initiale et itérons la boucle "while" jusqu'à cette valeur. Ainsi, cette troisième valeur fait office de borne maximale : soit l'intersection a lieu avant cette borne, soit la borne est atteinte avant l'intersection. C'est-à-dire que l'intersection aurait lieu au-dessus de la borne maximale.

Dans la capture d'écran ci-dessous, nous avons modifié le code précédent afin de prendre en compte cette borne MAX. Nous avons ensuite exécuté ce code avec les mêmes valeurs que dans la capture précédente : 30 pour DOUBLE et 10 pour TRIPLE. Nous avons indiqué la valeur 60 pour MAX. Comme nous avons vu précédemment qu'avec ces valeurs initiales, l'intersection se produit à 70, elle n'a pas lieu d'ici 60. Le test conditionnel du "while" contient donc maintenant un "&&" qui signifie "et".



Voici le code de ce programme, que vous pouvez exécuter vous-même plusieurs fois. Faites plusieurs essais, avec des valeurs initiales différentes, pour bien comparer différents scénarios. La bonne compréhension de ce code vous sera utile pour l'exercice pratique qui vous attend à la fin de ce cours.

```php
<?php
# Recherche l'intersection de 2 droites à partir de l'ordonnée à l'origine
$double = readline("Indiquer la valeur de départ à laquelle sera ajouté 2 : ");
$triple = readline("Indiquer la valeur de départ à laquelle sera ajouté 3 : ");
$max = readline("Indiquer la valeur maximale de l'intersection : ");
$n = 0;
while ($double > $triple && $double <= $max){
  echo "Itération n°".$n++." : Double vaut $double et Triple vaut $triple\n";
  $double+=2;
  $triple+=3;
}
```

```php
if ($double==$triple) echo "Double et Triple valent $double";
else if ($double>$max)
  echo "Les 2 droites se croisent au-dessus de $max.";
else
  echo "Les 2 droites ne se croisent pas à partir des points de départ donnés";
echo "\n";
?>
```

IV. Auto- évaluation

Exercice 1 : Quiz

[solution n°1 p.15]

Question 1

La ligne “echo 5-3**2+2*3 ;” affichera :

- 0
- 2
- - 2

Question 2

Dans l’expression booléenne (A or B and C), nous savons que A vaut True. Que vaut l’expression entière ?

- A
- B
- C

Question 3

Laquelle des expressions booléennes suivantes affichera “bool(true)” ?

- var_dump((True or False) and False or (True xor True)) ;
- var_dump((True xor False) or False and (True or True)) ;
- var_dump((True and False) or False and (True and True)) ;

Question 4

Un étudiant a écrit le programme suivant pour déterminer où se situer dans un mois. Ce programme est-il correct ?

```php
<?php
$jour = intval(readline("Entrez la date du jour (nombre entre 1 et 31 compris) : "));
if ($jour<1 and $jour>31)
  echo "Vous avez fait une erreur de saisie !";
else if ($jour<15)
  echo "Nous sommes dans la première quinzaine.";
else
  echo "Nous sommes dans la deuxième quinzaine.";
?>
```

- Oui
- Non

Question 5

Un étudiant a écrit le programme suivant pour contrôler la saisie d'une date. Le programme suivant est-il correct ?

```php
<?php
$jour = intval(readline("Entrez la date du jour (nombre entre 1 et 31 compris) : "));
$mois = intval(readline("Entrez la date du mois (nombre entre 1 et 12 compris) : "));
if ($jour>30){
  if ($mois==2 or $mois==4 or $mois==6 or $mois==9 or $mois==11)
    # les mois de février, avril, juin, septembre, novembre n'ont pas plus de 30 jours
    echo "Vous avez fait une erreur de saisie !";
  else
    echo "C'est le dernier jour du mois.";
}
else if ($jour>29 and $mois=2){
  # en février il n'y a pas plus de 29 jours
  echo "Vous avez fait une erreur de saisie !";
}
else echo "Vous avez saisi un jour valide.";
?>
```

- ○ Oui
- ○ Non

Solutions des exercices

Exercice p. 12 Solution n°1

Question 1

La ligne “echo 5-3**2+2*3 ;” affichera :

- ○ 0
- ◉ 2
- ○ - 2

Q La première opération calculée est l’exponentiation 3**2=9, puis la multiplication 2*3=6, puis de gauche à droite, car la soustraction et l’addition ont le même niveau de priorité. C’est-à-dire 5-9=-4, puis -4+6=2.

Question 2

Dans l’expression booléenne (A or B and C), nous savons que A vaut True. Que vaut l’expression entière ?

- ○ A
- ○ B
- ◉ C

Q Comme A vaut True, (A or B) vaut également True. Donc l’expression entière a la même valeur que (True and C), c’est-à-dire C.

Question 3

Laquelle des expressions booléennes suivantes affichera “bool(true)” ?

- ○ var_dump((True or False) and False or (True xor True)) ;
- ◉ var_dump((True xor False) or False and (True or True)) ;
- ○ var_dump((True and False) or False and (True and True)) ;

Q Il convient d’évaluer d’abord les sous-expressions entre parenthèses, puis de gauche à droite. La première proposition se simplifie en (True and False or False), puis en (False or False). La deuxième proposition se simplifie en (True or False and True), puis en (True and True). La dernière proposition se simplifie en (False or False and True), puis en (False and True).

Question 4

Un étudiant a écrit le programme suivant pour déterminer où se situer dans un mois. Ce programme est-il correct ?

```php
<?php
$jour = intval(readline("Entrez la date du jour (nombre entre 1 et 31 compris) : "));
if ($jour<1 and $jour>31)
  echo "Vous avez fait une erreur de saisie !";
else if ($jour<15)
  echo "Nous sommes dans la première quinzaine.";
else
  echo "Nous sommes dans la deuxième quinzaine.";
?>
```

- ○ Oui
- ◉ Non

Une erreur s'est glissée : l'opérateur « *and* » devrait être remplacé par l'opérateur « *or* ». En effet, le jour saisi ne peut pas être en même temps inférieur à 1 et supérieur à 31, mais nous voulons que le message d'alerte s'affiche quand l'un ou l'autre est vrai.

Question 5

Un étudiant a écrit le programme suivant pour contrôler la saisie d'une date. Le programme suivant est-il correct ?

```php
<?php
$jour = intval(readline("Entrez la date du jour (nombre entre 1 et 31 compris) : "));
$mois = intval(readline("Entrez la date du mois (nombre entre 1 et 12 compris) : "));
if ($jour>30){
  if ($mois==2 or $mois==4 or $mois==6 or $mois==9 or $mois==11)
    # les mois de février, avril, juin, septembre, novembre n'ont pas plus de 30 jours
    echo "Vous avez fait une erreur de saisie !";
  else
    echo "C'est le dernier jour du mois.";
}
else if ($jour>29 and $mois=2){
  # en février il n'y a pas plus de 29 jours
  echo "Vous avez fait une erreur de saisie !";
}
else echo "Vous avez saisi un jour valide.";
?>
```

- ○ Oui
- ◉ Non

En effet, l'opérateur de comparaison d'égalité est « == » alors que sur la ligne du « *else if* » se trouve l'opérateur « = ». Ainsi, à chaque fois que le jour vaut 30, le programme affiche "Vous avez fait une erreur de saisie !".