Plaquette 4 – Tests et conditions

Algorithmique et Scratch — Quatrième

Corrections détaillées

Boîte à outils

Les deux blocs de test.

Bloc	Effet
si condition alors A	exécute A seulement si la condition est vraie
si condition alors A sinon B	exécute A si vraie, B sinon

Les conditions usuelles.

$$v > a \qquad v < a \qquad v = a$$

Scratch propose aussi « ou », « et » et « non » pour combiner des conditions.

Le cas limite. La condition $v > a$ est **fausse** quand $v = a$: il faut décider si l'égalité relève du « alors » ou du « sinon ». C'est presque toujours là que les scripts se trompent.

Le test dans une boucle. Le test est réévalué **à chaque passage** : la branche suivie peut changer d'un tour à l'autre.

« Répéter jusqu'à ». Cette boucle continue tant que la condition est fausse, et s'arrête dès qu'elle devient vraie. Elle ne garantit aucun nombre fixe de passages.

La méthode de déroulement. Un tableau avec une colonne par variable **et une colonne pour la condition** : à chaque ligne, on note si elle est vraie ou fausse, puis la branche suivie.

Le contrôle. Tester le script sur une valeur de chaque côté de la frontière, **et sur la frontière elle-même**.

Correction de l'exercice 1 – Un test simple

a) La condition $n > 10$ s'évalue avec la valeur actuelle : $12 > 10$ est **vraie**.

b) La branche « alors » est exécutée :

$$n = 12 + 5 = 17$$

La branche « sinon » n'est **pas** exécutée : n ne vaut pas 12.

c) Avec $n = 7$ au départ, la condition $7 > 10$ est **fausse** : c'est la branche « sinon » qui s'applique.

$$n = 7 - 5 = 2$$

Le principe de l'alternative. Une seule des deux branches s'exécute, jamais les deux. C'est ce qui distingue « si … sinon » de deux instructions successives.

Le cas limite à examiner. Avec $n = 10$, la condition $10 > 10$ est **fausse** : on part dans le « sinon » et n devient 5. Pour inclure 10 dans la première branche, il faudrait écrire la condition « $n > 9$ » ou « $n \geq 10$ ».

Réponse. a) vraie — b) $n = 17$ — c) $n = 2$.

Correction de l’exercice 2 – Le test sans « sinon »

a) $8 < 5$ est **fausse** : le contenu du « si » est ignoré.

b) On passe directement à l’instruction suivante, qui n’est **pas** dans le test — elle est alignée sur le « si » :

$$a = 8 + 3 = 11$$

c) Avec $a = 2$, la condition $2 < 5$ est **vraie** : a devient 0, puis l’instruction suivante s’applique de toute façon :

$$a = 0 + 3 = 3$$

Le rôle de l’indentation. « Mettre a à 0 » est décalé : il appartient au test. « Ajouter 3 à a » est aligné sur le « si » : il s’exécute **dans tous les cas**. Le décaler d’un cran le placerait dans le test, et avec $a = 8$ le script ne ferait plus rien du tout.

Le test sans « sinon ». Quand la condition est fausse, il ne se passe simplement rien : le programme continue à la ligne suivante.

Réponse. a) fausse — b) $a = 11$ — c) $a = 3$.

Correction de l’exercice 3 – Le test dans une boucle

La table d’exécution.

Passage	n **au test**	$n > 3$ **?**	s **après**	n **après**
1	1	faux	0	2
2	2	faux	0	3
3	3	faux	0	4
4	4	vrai	4	5
5	5	vrai	9	6
6	6	vrai	15	7

a) La condition est vraie aux passages **4, 5 et 6**.

b) $s = 4 + 5 + 6 = 15$.

c) $n = 7$: la variable a été incrémentée six fois depuis 1.

Le cas limite. Au troisième passage, $n = 3$ et la condition $3 > 3$ est **fausse** : la valeur 3 n’est pas ajoutée. Avec la condition $n \geq 3$, on aurait $s = 3 + 4 + 5 + 6 = 18$.

Ce que le script calcule. La somme des entiers de 4 à 6. Le test filtre les valeurs prises par le compteur : c’est la structure de base de tout comptage conditionnel.

Réponse. a) passages 4, 5 et 6 — b) $s = 15$ — c) $n = 7$.

Correction de l’exercice 4 – L’erreur de Yanis

a) La condition $50 > 50$ est **fausse** : le script dit « perdu ».

b) **Non.** Yanis veut gagner en **atteignant** 50, donc 50 doit être gagnant. Sa condition exclut précisément la valeur frontière.

c) Il faut une inégalité **large** :

```
si score > 49 alors
    dire "gagné"
sinon
    dire "perdu"
```

ou, si Scratch propose le bloc, la condition score ≥ 50.

Le test des trois valeurs. Pour vérifier une condition, on l'essaie toujours sur trois valeurs : une en dessous (49 → perdu ✓), une au-dessus (51 → gagné ✓), et **la frontière** (50 → gagné ✓).

Pourquoi > 49 **fonctionne ici.** Le score est un entier : il n'existe rien entre 49 et 50. Avec des scores décimaux, 49,5 serait déclaré gagnant à tort — il faudrait alors vraiment ≥ 50.

Réponse. a) « perdu » — b) non — c) condition ≥ 50, ou > 49 pour des entiers.

Correction de l'exercice 5 – Deux conditions

a) **Pour** $x = 14$**.** Premier test : $14 > 10$ est vrai, on entre dans la branche « alors ». Second test : $14 < 20$ est vrai. Le script affiche **« entre 10 et 20 »**.

b) **Pour** $x = 25$**.** Premier test vrai ($25 > 10$), second test faux ($25 < 20$ est faux) : le script affiche **« au moins 20 »**.

c) **Pour** $x = 6$**.** Premier test faux ($6 > 10$ est faux) : on part directement dans le « sinon » extérieur, sans évaluer le second test. Le script affiche **« au plus 10 »**.

Les trois zones couvertes. Le script partage tous les nombres en trois catégories, sans recouvrement ni oubli :

$$x \leq 10 \qquad 10 < x < 20 \qquad x \geq 20$$

Les valeurs frontières. Pour $x = 10$, le script affiche « au plus 10 » ; pour $x = 20$, « au moins 20 ». Les deux messages sont cohérents avec les bornes annoncées ✓

Réponse. a) « entre 10 et 20 » — b) « au moins 20 » — c) « au plus 10 ».

Correction de l'exercice 6 – Compter les multiples

a) « i modulo 3 » est le **reste** de la division de i par 3. Il vaut 0 exactement quand i est un multiple de 3 : la condition teste donc la divisibilité par 3.

b) Le compteur i prend les valeurs de 1 à 20. Les multiples de 3 dans cet intervalle sont

$$3; 6; 9; 12; 15; 18$$

soit 6 **valeurs**, donc $c = 6$.

c) Ce sont précisément les six valeurs listées ci-dessus.

Le calcul direct. Le nombre de multiples de 3 jusqu'à 20 est le quotient entier de 20 par 3 :

$$20 = 3 \times 6 + 2$$

soit 6 multiples ✓

Le rôle de « modulo ». C'est l'opérateur qui donne le reste d'une division euclidienne. Tester « reste nul » est la façon standard de reconnaître un multiple — et « i modulo 2 = 0 » testerait la parité.

Réponse. a) si i est multiple de 3 — b) $c = 6$ — c) $3, 6, 9, 12, 15, 18$.

Correction de l'exercice 7 – Le lutin rebondit

a) Tant que le pas vaut 40, x progresse : $40, 80, 120, 160, 200, 240$. La condition $x > 200$ devient vraie au **sixième passage**, quand $x = 240$.

b) Le pas devient -40 : le lutin repart vers la gauche. À partir du septième passage, x décroît de 40 à chaque tour.

c) Après le sixième passage, $x = 240$. Il reste 6 passages, chacun retranchant 40 :

$$x = 240 - 6 \times 40 = 240 - 240 = 0$$

Le lutin termine en $(0; 0)$, à son point de départ.

Pourquoi le lutin ne rebondit qu'une fois. Après le retournement, x diminue et la condition $x > 200$ redevient fausse dès le septième passage : elle ne se redéclenche plus. Pour un vrai rebond aller-retour, il faudrait un second test sur le bord gauche, par exemple « si $x < -200$ alors mettre pas à 40 ».

Le contrôle. Six passages à +40 puis six à −40 se compensent exactement ✓

Réponse. a) au 6^{e} passage — b) le lutin repart vers la gauche — c) en $(0; 0)$.

Correction de l'exercice 8 – Le tarif dégressif

a) $15 > 20$ est faux : on applique le tarif de 4 €.

$$\text{prix} = 15 \times 4 = 60 \text{ €}$$

b) $25 > 20$ est vrai : tarif de 3 €.

$$\text{prix} = 25 \times 3 = 75 \text{ €}$$

c) $20 > 20$ est faux : on applique le tarif de 4 €, soit $20 \times 4 = 80$ €.

Le problème de cohérence. À 21 articles, le prix tomberait à $21 \times 3 = 63$ € : **acheter un article de plus coûterait** 17 **€ de moins**. C'est une anomalie commerciale classique, appelée effet de seuil.

Comment y remédier. On peut appliquer le tarif réduit **à partir** de 20 ($20 \times 3 = 60$ €), ou n'appliquer la réduction qu'aux articles au-delà du seuil :

$$\text{prix} = 20 \times 4 + (q - 20) \times 3$$

ce qui donne 83 € pour 21 articles — croissant, donc cohérent.

La leçon. Un test crée toujours une frontière : il faut vérifier que le comportement reste sensé **de part et d'autre**.

Réponse. a) 60 € — b) 75 € — c) 80 €, incohérent avec les 63 € de 21 articles.

Correction de l'exercice 9 – Pair ou impair

a) 12 modulo 2 = 0 : le nombre est pair, on applique la première branche.

$$r = \frac{12}{2} = 6$$

b) 7 modulo 2 = 1 : le nombre est impair, on applique la seconde branche.

$$r = 3 \times 7 + 1 = 22$$

c) 1 est impair :

$$r = 3 \times 1 + 1 = 4$$

Le test de parité. « n modulo 2 = 0 » est la façon standard de reconnaître un nombre pair : le reste de la division par 2 vaut 0 pour les pairs et 1 pour les impairs.

La règle derrière ce script. C'est celle de la **suite de Syracuse** : on divise par 2 les nombres pairs et on applique $3n + 1$ aux impairs. En répétant, on finit toujours par retomber sur 1 — du moins pour tous les nombres testés à ce jour, car personne n'a su le démontrer.

L'exemple à partir de 7 **:** 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 — seize étapes.

Réponse. a) $r = 6$ — b) $r = 22$ — c) $r = 4$.

Correction de l'exercice 10 – Répéter jusqu'à

a) La boucle continue **tant que** la condition est fausse, c'est-à-dire tant que $n \leq 100$:

$$10 ; 20 ; 40 ; 80 ; 160$$

b) La condition est testée **avant** chaque passage. À $n = 80$, elle est encore fausse ($80 > 100$ est faux), donc un cinquième passage a lieu et n devient 160. La condition devient alors vraie et la boucle s'arrête : 5 **passages**.

c) $n = 160$.

Le contrôle par les puissances. $n = 5 \times 2^5 = 5 \times 32 = 160$ ✓

La différence avec « répéter *n* fois ». Ici le nombre de passages n'est pas écrit dans le script : il dépend des valeurs. C'est ce qui rend cette boucle utile quand on ne sait pas d'avance combien d'étapes seront nécessaires.

Le danger. Si la condition ne devient jamais vraie — par exemple avec « mettre n à $n / 2$ » et la condition $n > 100$ — la boucle tourne indéfiniment. C'est une boucle infinie, et il faut toujours vérifier qu'elle peut se terminer.

Réponse. a) 10, 20, 40, 80, 160 — b) 5 passages — c) $n = 160$.

Correction de l'exercice 11 – Le score du jeu

a) Le compteur i prend les valeurs de 1 à 10. Les valeurs paires sont $2, 4, 6, 8, 10$: la branche « alors » est exécutée 5 **fois**, et la branche « sinon » également 5 **fois** (pour $1, 3, 5, 7, 9$).

b) Le score reçoit cinq fois +10 et cinq fois −3 :

$$\text{score} = 5 \times 10 + 5 \times (-3) = 50 - 15 = 35$$

c) Avec 20 répétitions, i irait de 1 à 20 : dix valeurs paires et dix impaires.

$$\text{score} = 10 \times 10 + 10 \times (-3) = 100 - 30 = 70$$

Le score double, ce qui était prévisible : deux fois plus de passages, répartis de la même façon.

Le gain moyen par passage. $\dfrac{35}{10} = 3{,}5$ points, soit la moyenne de +10 et −3 ✓ Cette lecture permet de prévoir le score pour n'importe quel nombre pair de répétitions.

Réponse. a) 5 fois chacune — b) score = 35 — c) score = 70.

Correction de l'exercice 12 – Le lutin et le bord

a) Tant que le test ne se déclenche pas, x progresse : $-100, -50, 0, 50, 100, 150, 200, 250$. Au **huitième passage**, $x = 250 > 240$: la condition devient vraie.

b) Le test **bloque** le lutin au bord : x est ramené à 240. C'est ce que fait Scratch automatiquement, mais ici le script le programme explicitement.

c) Aux neuvième et dixième passages, x passe à 290 puis est ramené à 240, et de même au dixième. La position finale est donc

$$(240 ; 0)$$

Ce que le test empêche. Sans lui, le lutin finirait à $x = -150 + 10 \times 50 = 350$, hors de la scène et invisible.

La structure appelée « saturation ». Le test impose un plafond à une variable : elle croît normalement jusqu'à la limite, puis y reste. On la retrouve partout — jauge de batterie, barre de vie dans un jeu, volume sonore.

Réponse. a) au 8^{e} passage — b) il bloque x à 240 — c) $(240; 0)$.

Correction de l'exercice 13 – Chercher le plus grand

a) **La table d'exécution.**

v	$v >$ max **?**	**max après**
12	vrai ($12 > 0$)	12
7	faux	12
25	vrai	25
19	faux	25
3	faux	25

b) La valeur finale est max = 25, qui est bien le plus grand élément de la liste ✓

c) Avec des valeurs toutes négatives — par exemple $-4, -9, -2$ — aucune ne dépasserait 0 : la variable resterait à 0, qui n'appartient pas à la liste. Le script donnerait un **résultat faux**.

La correction. Il faut initialiser « max » avec la **première valeur** de la liste, et non avec 0 :

```
mettre max à première valeur de liste
```

L'algorithme du maximum. Il parcourt la liste en gardant en mémoire le plus grand élément vu jusque-là. C'est l'un des algorithmes les plus utilisés en informatique, et son initialisation est le point où il échoue le plus souvent.

Réponse. b) max = 25 — c) le script renverrait 0, valeur absente de la liste.

Correction de l'exercice 14 – Deux tests successifs

a) **Premier test :** $18 > 10$ est vrai, donc n devient 23.

Second test : il s'évalue avec la **nouvelle** valeur. $23 > 20$ est vrai, donc n devient 123.

b) La valeur finale est $n = 123$.

c) Avec un « sinon », les deux tests deviendraient exclusifs :

```
si n > 10 alors
    ajouter 5 à n
sinon
    si n > 20 alors
        ajouter 100 à n
```

Le premier test étant vrai, la seconde branche ne serait **jamais** atteinte : n vaudrait 23.

La différence essentielle. Deux tests successifs indépendants peuvent se déclencher **tous les deux** ; deux tests liés par « sinon » s'excluent mutuellement.

Le piège de cet exercice. Le second test porte sur la valeur **modifiée** par le premier. Un script qui modifie une variable puis la teste à nouveau doit être déroulé ligne à ligne, sans anticiper.

Réponse. a) $n = 23$ puis 123 — b) $n = 123$ — c) avec un « sinon », $n = 23$.

Correction de l'exercice 15 – Synthèse : le jeu du nombre

a) n prend les valeurs de 1 à 15, puis vaut 16 à la sortie.

b) Les valeurs paires entre 1 et 15 sont $2, 4, 6, 8, 10, 12, 14$:

$$\text{pairs} = 2 + 4 + 6 + 8 + 10 + 12 + 14 = 56$$

c) Les valeurs impaires sont $1, 3, 5, 7, 9, 11, 13, 15$:

$$\text{impairs} = 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 = 64$$

d) La somme des entiers de 1 à 15 vaut

$$\frac{15 \times 16}{2} = 120$$

et $56 + 64 = 120$ ✓ Les deux accumulateurs se partagent exactement tous les entiers.

La remarque sur les impairs. $64 = 8^2$: la somme des k premiers nombres impairs vaut toujours k^2. Ici $k = 8$, d'où 64 ✓

Le bilan de la plaquette. Un test dans une boucle trie les valeurs en deux catégories, et la somme des deux accumulateurs redonne le total : c'est le contrôle qui valide tout l'algorithme.

Réponse. a) de 1 à 15 — b) pairs = 56 — c) impairs = 64 — d) total 120 ✓