Premiers pas en Algo - Jour 1

Objectifs du module

Introduction à la programmation, c'est à dire découvertes des concepts communs à tous les langages informatiques.

Compétences

- programme, exécution, instruction, nom
- variables (déclaration, initialisation, affectation, expression)
- fonctions
- Java et Processing
- debugger

Démarche pédagogique

Tous les exercices doivent être réalisés seul(e), en autonomie. Les livrables doivent être validés par le formateur avant de passer à l'exercice suivant.

Exercice A.1 - C’est l’histoire de Toto...

Modalités

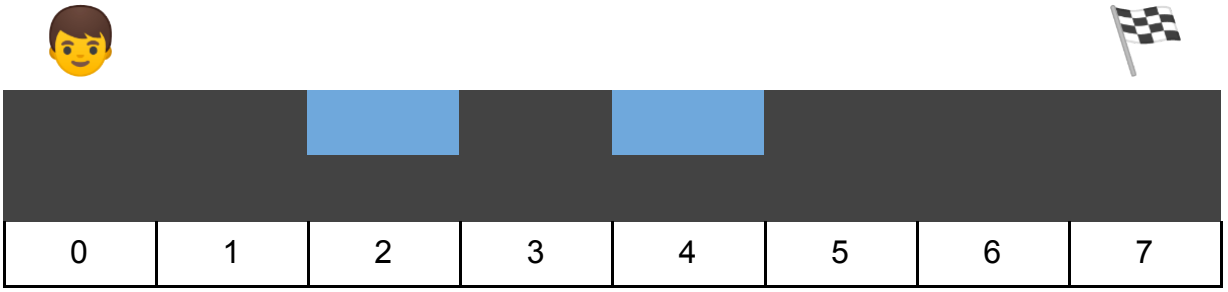
- Travail individuel en autonomie
- Sur papier

Livrable

- Les **instructions** qui permettent à Toto d’arriver au bout du chemin.
- La liste des cases par lesquels Toto est passé

Énoncé

Toto doit aller au bout d’un chemin qui est parsemé de grosses flaques d’eau (en bleu). Le chiffre indiqué représente le nombre de mètres depuis le début du chemin.



Toto ne comprend que deux **instructions** : `walk` et `jump`, et oui Toto parle anglais :-).

Quand on lui dit “`walk`”, il fait un pas d’un mètre. Quand on lui dit “`jump`”, il bondit et se retrouve directement deux mètres plus loin.

Quelle sont les **instructions** qui permettent à Toto d’aller au bout du chemin sans marcher dans l’eau ?

Est-il possible d’arriver au même résultat avec des **instructions** différentes

Exercice A.2 - Où es-tu Toto ?

Modalités

- Travail individuel en autonomie
- Sur papier

Énoncé

Pour suivre la progression de Toto sur le chemin, nous allons noter sa position sur un post-it. Sur le post-it ne peut-être écrit qu'une seule **valeur** (un nombre dans notre cas). Et pour être bien clair sur ce que représente la **valeur** écrite, nous allons écrire un **nom** dans un coin du post-it: `totoPosition`. Et dans un autre coin nous allons aussi écrire le **type** de **valeur** que l'on a le droit d'écrire (un nombre).

Sur un autre post-it nous allons écrire une **instruction** qui permet de changer sa position. L'**instruction** pour changer cette valeur est une **affectation** que l'on va formaliser sous la forme

```
totoPosition = expression
```

Il faut lire cela comme: "remplace la **valeur** du post-it appelé `totoPosition` par la **valeur** de **l'expression** à droite du signe égal".
Une **expression** peut-être soit un nombre, par exemple `12`, soit une opération mathématique, par exemple `5*4`, soit le nom d'un post-it `totoPosition`, soit un mix de tout ça `totoPosition*2 + 1`

Livrables

- Un post-it `totoPosition`
- Ecrire l'**instruction** permettant l'**initialisation** de Toto à sa position de départ
- La suite d'**instructions** pour modifier `totoPosition` suivant la même logique que dans l'exercice précédent.
- Simuler l'**exécution** des **instructions** en modifiant la **valeur** sur le post-it

Exercice A.3 - Processing

Modalités

- Travail individuel en autonomie
- Sur ordinateur dans l'IDE Processing (https://processing.org/download/)

Livrables

- Le **programme** permettant d'afficher la suite des positions de Toto
- Lancer le **debugger** et **exécuter** le **programme** instruction par instruction

Énoncé

Tapez dans l'environnement Processing le **programme** minimal ci dessous.

```
void setup() {

}
```

Puis entre les accolades `{` et `}` ajouter les **instructions**, qui permettent d'afficher les positions successives de Toto en transposant le travail réalisé précédemment sur papier.

Vous aurez besoin de :

- de stocker votre valeur (comme sur le post-it). Il faut pour cela **déclarer** une **variable,** de **type** `int` (c'est à dire integer ou nombre entier en français, voir la doc), appelée `totoPosition`
- **initialiser** cette **variable** avec la position de départ de Toto
- modifier sa **valeur** à l'aide d'une **affectation** (validée dans l'exercice précédent)
- …recommencer les deux dernières étapes autant que nécessaire

Notes:

En Java (et dans de nombreux langages) les instructions doivent se terminer par des point-virgules `;`

Exercice A.4 - Don’t Repeat Yourself (DRY)

Ca y est vous avez fait votre premier programme en Java ! Nous allons maintenant l’améliorer.

Livrables

- Une évolution du **programme** précédent, ayant exactement le même résultat mais utilisant des **fonctions**

Énoncé

Le programme précédent est tout à fait valable mais il n’est pas très facile à lire. Et si on lui disait `walk` pour qu’il marche et `jump` pour qu’il saute, ça serait plus clair, non ?
Vous allez pour cela **déclarer** deux **fonctions** `walk` et `jump` en vous inspirant de la fonction suivante:

```
void walk() {
 // inclure ici les instructions qui font marcher Toto d’un mètre
}

// … ne pas oublier la fonction jump

void setup() {
 // utilisations des fonctions walk et jump
}
```

vous pourrez alors **appeler** vos **fonctions** par son **nom** suivi de parenthèses: `walk();`

Notes

Les fonctions doivent être **déclarées** les unes à côté des autres au même niveau que la fonction `setup`. C’est à dire PAS à l’intérieur des accolades. Dans Processing, la fonction `setup` est appelée automatiquement au démarrage du programme.

Bonus

Utiliser la fonction `println` (voir la doc) qui permet d'afficher la valeur d'une **expression**, et donc d'une variable pour suivre la position de Toto.

Exercice A.5 - Calculs

Livrables

- Afficher le nombre de déplacements effectués par Toto (marche ou saut)
- Afficher la distance moyenne d'un déplacement de Toto
- Répondre aux questions de l'énoncé

Énoncé

Nous allons ajouter du code qui permet de savoir le nombre de déplacements effectués par Toto. A votre avis, de combien de nouvelles variables avons nous besoin ?
Quel est le meilleur endroit pour mettre ces instructions supplémentaires et ne pas se répéter ?

Enfin nous afficherons la distance moyenne des déplacements de Toto, c'est à dire la distance totale divisée par le nombre de déplacements. A-t-on besoin d'une variable pour calculer/afficher cette moyenne ? Justifiez votre réponse au formateur.

Notes:

Attention, en Java si on divise un `int` (un nombre entier et donc sans virgule) par un autre `int`, on obtient un .... `int`. Le résultat de la moyenne sera donc différent de la valeur réelle. Il faut donc utiliser un nouveau **type** de variable, `float`, qui permet de stocker les nombres à virgule (voir la doc).

Si vous avez besoin, d'afficher plusieurs choses sur la même ligne, vous pouvez passer plusieurs **paramètres** à `println` en les séparant par des virgules (voir la doc).

Exercice A.6 - Dessine moi un mouton !

Livrables

- Une fonction qui dessine un mouton
- Un mouton affiché à l'écran

Énoncé

Nous allons laisser Toto sur son chemin pour le moment, et faire un peu de graphisme. Pas de panique, on est en algo, pas besoin de Photoshop, on va dessiner avec du code.

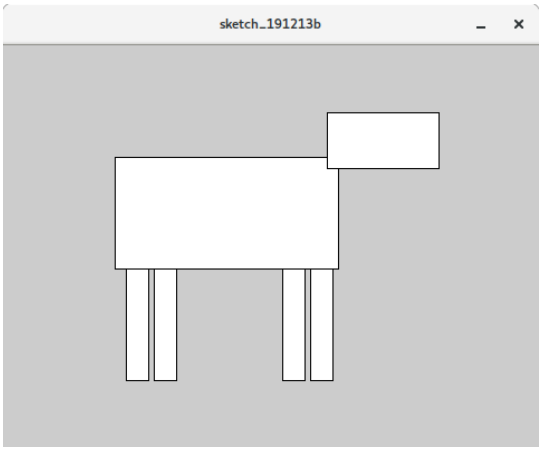
Vous avez créé des fonctions mais il en existe des milliers qui ont déjà été écrites pour vous. Par exemple il existe une fonction pour dessiner un rectangle à l'écran: https://processing.org/reference/rect_.html

Ecrivez une fonction `mouton` qui, à partir de rectangles, va dessiner un mouton, ou autre animal à 4 pattes, à l'écran.

Notes

Si vous avez besoin que la fenêtre soit plus grande, vous pouvez utiliser la fonction `size`: https://processing.org/reference/size_.html

Ne passez pas de temps sur le design, voici à quoi ressemble le mien 😁 :



Exercice A.7 - Dessine moi deux moutons !

Livrables

- Une fonction qui dessine un mouton à une position donnée
- Deux moutons affichés à l'écran

Énoncé

Notre mouton s'ennuie, nous allons lui présenter un copain.
Que se passe-t-il si on appelle deux fois la fonction mouton ?

Pour que l'on voit nos deux moutons, il faut que ceux-ci ne soit pas dessinés au même endroit.
Nous allons donc ajouter deux **paramètres** à notre fonction `mouton` pour indiquer à quelle position `x` (position horizontale) et `y` (position verticale) on souhaite le dessiner, et donc modifier la position de chacun des rectangles en conséquence.

Il faut pour cela renseigner les **paramètres** au moment où l'on **déclare** la fonction:

```
void mouton(int x, int y) {
 // on peut utiliser x et y comme des variables mais uniquement
 // dans la fonction
}
```

et quand on **appelle** la fonction il faut lui dire quelles **valeurs** on souhaite donner aux **paramètres**, on les mettant entre parenthèses:

```
mouton(100, 100);
mouton(300, 200);
```

Bonus

Ajouter en paramètre une couleur de remplissage pour le mouton.
Ajouter un paramètre pour la taille du mouton

Livrables

Prenez le temps d'utiliser le **debugger** et d'exécuter le programme instruction par instruction pour bien comprendre ce qui se passe lorsque les fonctions sont appelées.