Langages et programmation

Les quatre concepts de l'informatique

Algorithme
Machine
Langage
Information

Parmi les langages...

Les langages de programmation

Au confluent de ces quatre notions (la clé de voûte de l'informatique)

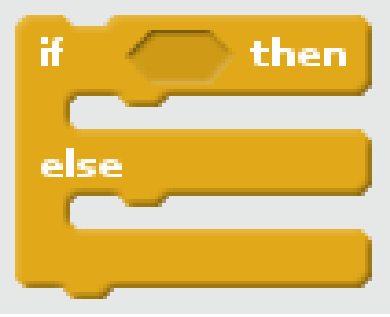
Apprendre à programmer et comprendre (un peu) la notion de langage de programmation

Ce n'est pas une langue

Il y a d'autres langages (la notation musicale, le langage des nombres indo-arabes, romains...)

Comprendre qu'il y a d'autres langages de programmation (dès la seconde : calculatrice, python...)

Mais que beaucoup de concepts sont communs




```
if <expression> <instruction> else <instruction>
if <expression> then <intrudction> else <instruction>
```

Comprendre la notion de test est aussi important que comprendre la manière dont on écrit un test en Scratch

Mais... un savoir réputé difficile à enseigner

Une grande diversité : des querelles de chapelles...

Une grande volatilité : à quoi bon apprendre à conduire une 208 ?

Des détails insignifiants : écrire `:=` et non `=`, terminer par `;`

Difficulté à conceptualiser : la machine, elle fait ceci, elle fait cela

Jusqu'à quel degré de détail descendre ?

Cependant...

Les différents langages sont organisés autour d'un petit nombre de fonctionnalités

Présentes dans de nombreux langages

Qui sont relativement stables

Et que l'on peut décrire simplement avec les outils adéquats

affectation, séquence, test, boucle, fonction, récursivité, enregistrement, cellule, module, objet, événement...

Dans de nombreux langages : quatre instructions

L'affectation, la séquence, le test, la boucle
Le noyau impératif de ce langage

Pour chacune d'entre elles, décrire

- sa syntaxe : la manière dont cette instruction s'écrit
- sa sémantique : ce qu'il se passe quand on l'exécute

Décrire la syntaxe

La question importante : de quoi est constituée cette instruction

La question accidentelle : comment cela s'écrit-il dans le langage particulier que l'on utilise

L'affectation : constituée d'une variable `x` et d'une expression `t`

(`x = t`, `x := t`, `x = t;`...)

La séquence : constituée de deux instructions `p` et `q`

(`p q`, `p;q`...)

Le test : constitué d'une expression `t` et de deux instructions `p` et `q`

(`if (t) p else q`, `if t then p else q`...)

La boucle : constituée d'une expression `t` et d'une instruction `p`

`repeat p until (t)...`

Le besoin de parenthèses

Comme `(a + b)` $\times$ `c` et `a + (b` $\times$ `c)`

Besoin de parenthèses (`{}`) pour disginguer

- la séquence de `if (t) p else q` et `r`
- et le test construit avec `t`, `p` et `q r`

`{if (t) p else q} r` et `if (t) p else {q r}`

Sauf pour une opération associative `a + b + c` ou `p q r`

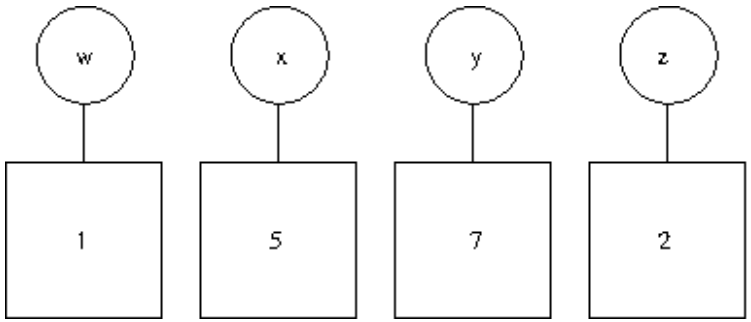
Les instructions et les expressions

`x + 3` : une expression

`y = x + 3` : une instruction (en l'occurrence une affectation)

Décrire la sémantique

Ce que fait un programme : transformer un état
Un état indique la valeur de chaque variable à un instant donné



valeur $\neq$ expression (3 valeur, mais pas `x + 3`)

Une expression a une valeur, mais seulement dans un état

La valeur de `x + 3` dans `[x = 5]` est 8

Une instruction transforme un état en un autre état

Exécuter l'instruction `p` dans l'état `s`, produit l'état `s'`

L'affectation

Une opération sur les états : `s + (x = v)`
même état que `s` sauf pour la variable `x` qui prend la valeur `v`
Exemple :
`s = [x = 4, y = 5, z = 6]`
`s + (y = 7) = [x = 4, y = 7, z = 6]`

Exécuter `x = t;` dans l'état `s`, produit l'état `s + (x = v)`
Qu'est-ce que `v` ?

Un célèbre (non) problème

```
x = x + 1;
```

Les variables qui ne sont pas là

Dans l'état `[x = 12]`

Quelle est la valeur de l'expression `y + 1` ?

Quelle est la sémantique de l'instruction `y = 4;` ?

Quelle est la sémantique de l'instruction `y = y + 2;` ?

La séquence

Exécuter `p1 p2` dans l'état `s`, produit l'état `s''`, produit en exécutant `p2` dans l'état `s'`, produit en exécutant `p1` dans l'état `s`

Le test

Si la valeur de `t` est `true`
alors exécuter `if (t) p1 else p2` dans l'état `s` produit l'état `s'`,
produit par l'exécution de `p1` dans l'état `s`

Si la valeur de `t` est `false`
alors exécuter `if (t) p1 else p2` dans l'état `s` produit l'état `s''`,
produit par l'exécution de `p2` dans l'état `s`

Complet ?

Avant de voir la boucle...

Les points importants :

1. `x` rien à voir avec une variable mathématique

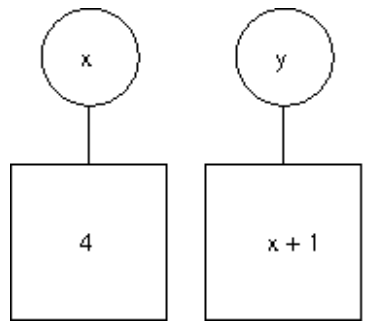
2. La valeur de `x + 3` ne change pas miraculeusement au cours du temps
Une expression n'a une valeur que dans un certain état
Le temps n'est pas un concept donné *a priori* (par exemple : la séquence)

3. Dans une affectation, on évalue l'expression dans l'état courant puis on oublie l'expression

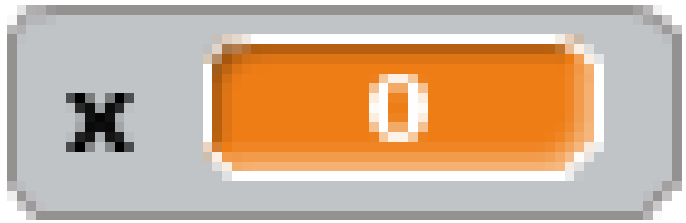
`x = 4; y = x + 1; x = 10;`

`y` prend pas la valeur `5` et non `11`

Pas absurde mais une autre sémantique



4. Il est important que les élèves visualisent l'état (le dessinent)



Scratch vous aide
Qu'est-ce qu'un lutin ?

5. Une question difficile

Est-ce que c'est réellement comme cela à l'intérieur de la machine ?
Est-ce qu'il y a réellement une boîte `x` dans la machine

Plus ou moins, mais de moins en moins

On ne parle pas de la machine mais du langage
On a décrit la sémantique du langage (ce qu'il se passe quand on exécute un programme) sans faire référence à la machine qui l'exécute

Au contraire : parce qu'on a décrit la sémantique du langage indépendamment de la machine, tous les coups sont permis

La boucle

```
repeat q until (t)
repeat x = x * 2; until (x > 1000)
```

Quand on exécute cette instruction : on exécute `q`, puis on calcule la valeur de `t`, si c'est `true` on a terminé, si c'est `false`, on exécute `q`, puis

La boucle

```
repeat q until (t)
repeat x = x * 2; until (x > 1000)
```

Quand on exécute cette instruction : on exécute `q`, puis on calcule la valeur de `t`, si c'est `true` on a terminé, si c'est `false`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `true` on a terminé, si c'est `false`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `true` on a terminé, si c'est `false`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `true` on a terminé, si c'est `true`, on exécute `q`, puis

`repeat q until (t)` : notation finie pour une instruction infinie

```
q if (t) skip else {q if (t) skip else {q if (t) skip
else...}}
```

avec une instruction fictive `skip` qui ne fait rien

Que signifie que `skip` ne fait rien ?

`repeat x = 2; until (x > 1000)` ne termine pas

Instruction infinie : potentialité de non terminaison
La sémantique d'un programme `p` qui ne termine pas dans l'état `s` : quand on exécute le programme `p` dans l'état `s` cela ne produit aucun état (ou produit $\perp$)

Les messages essentiels

Un programme fait quelque chose à quelque chose : un état

On peut expliquer pourquoi les programmes font ce qu'ils font

Ils pourraient faire autre chose

(exemple : `x = 4; y = x + 1; x = 10;`)