L’algorithme

Date de création	@3 juin 2024 08:55
Cours	Algo
Type	O'clock
Relue ?	☐
Prise de note ajoutée	☐

Les variables

En programmation, quelque-soit le langage utilisé, il existe des concepts, des "briques" élémentaires qui sont universels. Les variables en font partie !

Une variable va nous permettre de **stocker une donnée dans la mémoire temporaire de notre ordinateur** (mémoire vive, on parle souvent de "RAM").

Cette donnée va pouvoir être du texte, un nombre, ou d'autres types de donnée que nous découvrirons par la suite.

On peut par exemple **stocker notre prénom dans une variable** !

Les conditions

Pour l'instant, notre programme fait **toujours la même chose** : afficher le prénom de l'utilisateur. Pas génial !

Quand on programme un **algorithme**, on veut en général effectuer certains actions **en fonction de** différents événements ou actions de l'utilisateur. Pour faire cela, on va découvrir une autre brique de base de la programmation : les **conditions**.

Algorithme

Un algorithme, c'est une ***suite d'instructions et d'opérations permettant de résoudre un problème spécifique.***

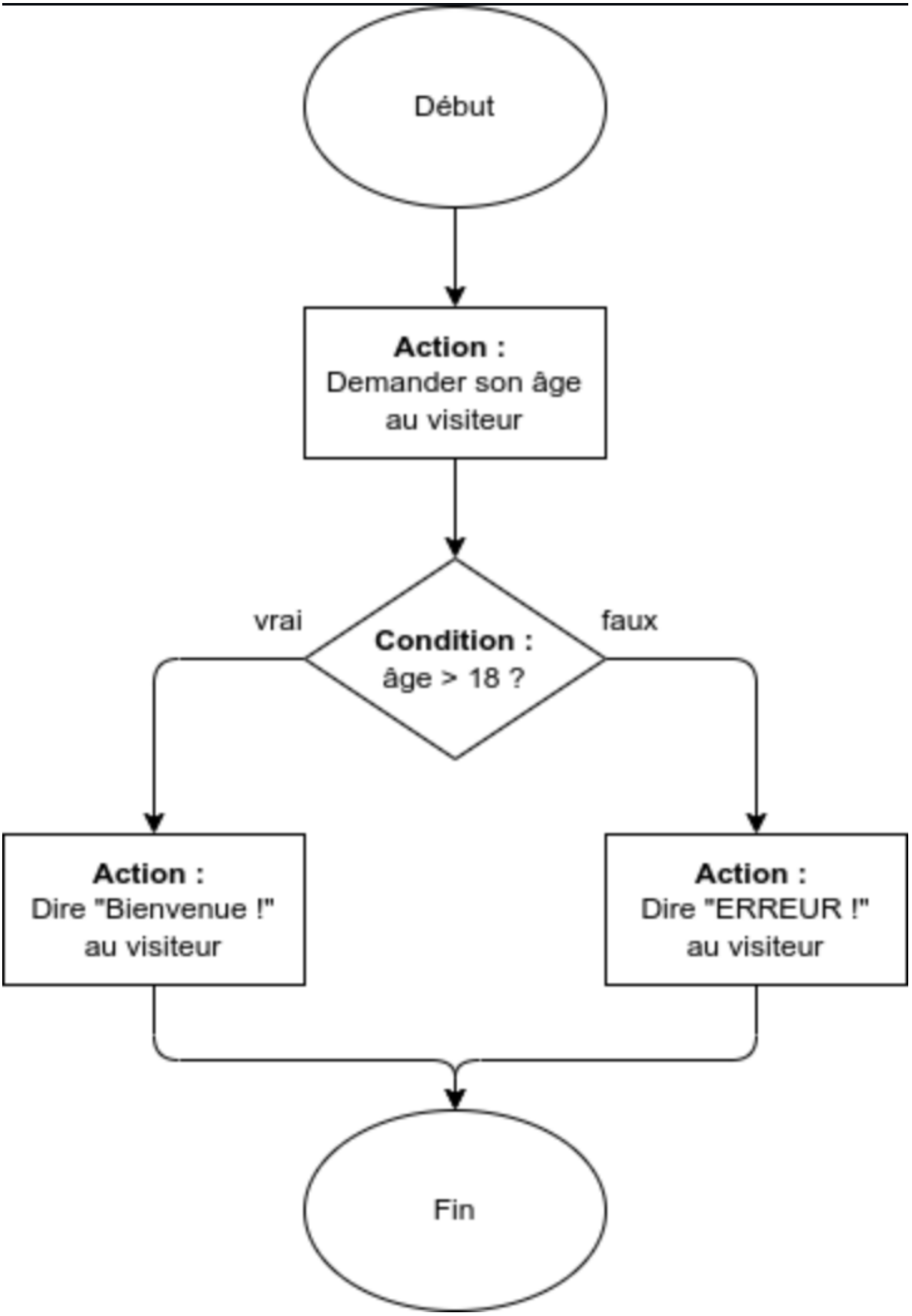
Quelques exemples :

- le thermostat de votre chaudière ou d'un radiateur électrique : `SI` la température `EST INFÉRIEURE OU ÉGAL À` 18°C, `ALORS` on allume le chauffage. `SINON`, on éteint le chauffage.
- le système d'authentification d'un utilisateur sur un site web : `SI` l'adresse email et le mot de passe sont corrects, `ALORS` on dit "Bienvenue" à l'utilisateur, `SINON` on lui met un message d'erreur.
- le site de votre banque en ligne : `SI` le solde du compte `EST SUPÉRIEUR STRICTEMENT À` 0€, `ALORS` le solde est en vert, `SINON` il est en rouge.

Ces algorithmes sont implémentés dans des programmes informatiques, avec du code, mais en réalité, les algorithmes sont aussi présents dans notre vie quotidienne, partout autour de nous :

- le videur à l'entrée de la boîte de nuit : `SI` tu as des baskets, `ALORS` tu rentres pas. `SINON`, tu rentres.
- l'examinateur au passage du permis de conduire : `SI` tu écrases un piéton, `ALORS` tu n'auras pas ton permis. `SINON`, `SI` tu respectes le reste du code de la route, `ALORS` tu auras ton permis, `SINON` non.
- le Rubik's Cube peut être résolu en suivant des étapes précises, basées sur une position initiale.
- les procédures de dépannage, le diagnostic médical, les textes de loi, les recettes de cuisine, tout ça sont également assimilables à des algorithmes.

Représentation graphique d’un algorithme


Début
Action :
Demander son âge
au visiteur
vrai
Condition :
âge > 18 ?
faux
Action :
Dire "Bienvenue !"
au visiteur
Action :
Dire "ERREUR !"
au visiteur
Fin

Le début et la fin de l'algorithme sont représentés par **ronds**, les **actions** par des rectangles et les **conditions** par des losanges. Cette représentation est en général appelée **logigramme**, ou **algorigramme**.

> Quand vous devrez coder des algorithmes plus complexes dans les semaines à venir, n'hésitez pas à prendre un papier et un crayon et à d'abord réfléchir à votre algorithme en dessinant un logigramme

Structure de controle & conditions

En programmation, quelque-soit le langage, on va pouvoir utiliser des **structures de contrôles** pour modifier le comportement de notre programme dans certaines **conditions**.

Les structures de contrôle seront toujours de cette forme :

```
SI condition EST vraie
ALORS
    actions à effectuer quand la condition est vraie
```

On peut également rajouter des actions à effectuer quand la condition est fausse :

```
SI condition EST vraie
ALORS
    actions à effectuer quand la condition est vraie
SINON
    actions à effectuer quand la condition est fausse
```

Et enfin, on peut même imbriquer les structures de contrôle entre elles afin de créer des algorithmes plus complexes :

```
SI condition1 EST vraie
ALORS
    SI condition2 EST vraie
```

```
        actions à effectuer quand la condition1 est vraie E
T que la condition2 est vraie
    SINON
        actions à effectuer quand la condition1 est vraie E
T que la condition2 est fausse
SINON
    actions à effectuer quand la condition1 est fausse
```

> Et condition, condition1 et condition2 dans tout ça, c'est quoi ?

Opérateurs de comparaison

On va utiliser des opérateurs de comparaison pour écrire nos conditions :

- `A = B` pour tester si A et B sont identiques
- `A != B` pour tester si A et B sont différentes
- `A < B` pour tester si A est strictement inférieur à B
- `A > B` pour tester si A est strictement supérieur à B
- `A <= B` pour tester si A est inférieur ou égal à B
- `A >= B` pour tester si A est supérieur ou égal à B

Reprenons l'algorithme du logigramme précédent et traduisons-le en pseudo-code :

```
SI age > 18
ALORS
    afficher "Bienvenue !" au visiteur
SINON
    afficher "ERREUR !" au visiteur
```

`age > 18` est notre **condition**, et notre code va permettre de **déterminer si cette condition est vraie ou fausse**, et d'effectuer des actions différents en fonction de cela. On dit que la condition sera **évaluée** à vrai ou faux.

> Par abus de langage, on parle souvent de "condition" pour décrire la structure de contrôle dans son ensemble.

Conditions complexes : opérateurs booléens

Dans certains cas, on va avoir besoin de conditions plus complexes. Prenons un exemple : si l'âge du visiteur est entre 12 et 18 ans, on voudrait lui dire "Bienvenue, jeune visiteur !".

> Mais on a juste vu comment comparer l'âge avec une seule valeur ! 🤔

Pour résoudre ce problème, on pourrait imbriquer deux structures de contrôles, comme mentionné précédemment :

```
SI age < 18
ALORS
    SI age > 12
        actions à effectuer quand le visiteur a entre 12 et 18 ans.
SINON
    actions à effectuer quand le visiteur n'a pas entre 12 et 18 ans.
```

Mais pour éviter de se retrouver à devoir imbriquer plein de conditions de la sorte, on peut utiliser des **opérateurs booléens** !

Il en existe 3 :

- **ET** : permet de tester si une condition A ET une condition B sont vraies
- **OU** : permet de tester si une condition A OU une condition B est vraie
- **NON** : permet d'inverser, vrai devient faux et inversement

Ne pas se répéter

On vient d'en parler : **en programmation, se répéter, c'est mal**.

Note

Cette "bonne pratique" de ne pas se répéter (quand c'est possible) porte même un nom : **DRY**, acronyme de **Don't Repeat Yourself**. C'est même devenu l'un des **grands principes** de la programmation, qu'il faut tâcher de respecter.

> Mais du coup, si on veut dire 3 fois "Bonjour" de suite, on fait comment ?

Pour ça, il faut qu'on découvre un troisième concept important : les **boucles** !

Les boucles

Les **boucles**, en programmation, nous permettent de **répéter une action un certain nombre de fois**.

Si on veut dire "Bonjour !" trois fois de suite (sans devoir copier-coller l'instruction `dire X pendant Y secondes` trois fois), on peut utiliser le bloc `répéter X fois` (catégorie `Contrôle`).

Boucler, tant que

Souvent, on voudra boucler **tant qu'une condition est vraie** ou **tant qu'une condition est fausse**.

Ici, `réponse = Bob` est ce qu'on appelle la **condition de sortie** de notre boucle. **Dès que** cette condition est vraie, on sort de la boucle et on affiche le message "Bienvenue Bob".

Tant que cette condition est fausse, on continue de boucler ! (et on continue de demander le prénom de l'utilisateur, à l'infini !)