ALGORITHME

SOMMAIRE

OBJECTIFS

- Apprendre l'algorithmie
- Comprendre le langage de l'informaticien
- Avoir les bases de la programmation informatique

- Pour qui :
  - Débutant en informatique
  - Toute personne souhaitant comprendre la logique des programmes informatiques
  - aucun pré requis, aide pour tous les langages de programmation

INTRODUCTION

- Le but du cours est de comprendre la logique de la programmation
- IDEE COURANTE : Développer = Programmer
- C'EST FAUX !
- Développer = Analyser + Concevoir + Programmer
  - Analyser : Comprendre les besoins du client
  - Concevoir : identifier le "nécessaire" pour répondre aux besoins du client
  - Programmer : réaliser les lignes de code du logiciel, du site web ou de l'application correspondant aux besoins identifiés

INTRODUCTION

- La logique de programmation est plus important que maîtriser juste un langage de programmation.

- Apprendre la logique de programmation permet de travailler sur tous les langage de programmation et permettre un temps d'adaptation relativement court pour acquérir la syntaxe et particularité du langage.

INTRODUCTION

- Un algorithme qu'est ce que c'est ?
  - C'est une suite d'instructions
  - Exemple :
    - Une notice de montage d'un meuble
    - Une recette de cuisine

  - Mises bout à bout, les instructions permettent de réaliser des actions complexes pour arriver à un résultat

  - En informatique, c'est une suite de lignes de codes (appelée programme) permettant de réaliser des actions « logiques »

- Algorithme —(Langage de programmation)→ Programme informatique

INTRODUCTION

- Pourquoi c'est important ?
  - Les algorithmes sont le fondement même des programmes informatiques et permettent aux informations d'avoir un langage commun.
  - Contrairement aux langages de programmation, un algorithme est une suite logique d'opérations qui est indépendante d'un langage et peut-être représentée de plusieurs manières : graphique, texte ou "pseudo-code".
  - Un informaticien doit être capable de réaliser des algorithmes sans obligatoirement maîtriser tous les langages de programmation.

PSEUDO-CODE

- Le "pseudo-code" permet de décrire les étapes avec un langage commun compris par tous les informaticiens.

- Par exemple:
  - ouvrirPorte seras l'action d'ouvrir une porte par un personnage.
  - On utilise ici le standard d'écriture camelCase :
    - Le mot commence en minuscule et toujours par une lettre
    - Aucun espace et chaque mot sera séparé en mettant la première lettre du prochain mot en majuscule.

EXEMPLE

- Un personnage veut accéder au portail de sa maison, pour cela il doit se retourner complétement et marcher tout droit jusqu'au portail:
  - Seul les « pseudo-code » suivants sont utilisable :
    - tournerAGauche
    - tournerADroite
    - avancer

EXEMPLE

- Ici on écrirait l'algorithme de cette manière :
  - TournerAGauche
  - TournerAGauche
  - avancer

ECRIRE UN ALGORITHME

- Le but d’un algorithme est d’avoir une syntaxe que tout le monde pourrait comprendre.

- Cette syntaxe est parfois légèrement différente suivant les formateurs ou la langue, mais l'important à retenir est la logique.

- Ces algorithmes ne peuvent pas, dans tous les cas, être compris par l'ordinateur. Il faudra ensuite les retranscrire en programme grâce à un langage spécifique.

ECRIRE UN ALGORITHME

- Ecrire son premier algorithme en « pseudo-code »

- Les mots clés seront en majuscule
- On fera une indentation(tabulation) pour chaque bloc.

```
ALGORITHME <nomAlgo>
DEBUT
        <instruction>
        <instruction>
        <instruction>
FIN
```

LES VARIABLES

- Dans les mathématiques supérieures et en logique, une variable est un symbole représentant, a priori, un objet indéterminé.

- L'intérêt est qu'il s'agit d'une valeur qu'on peut modifier avec le temps.

LES VARIABLES

- Imaginons un personnage qui avance et récupère des pièces.

Dans un jeu, on aurait un compteur qui va changer au cours du temps.

Exemple : Etape initiale : piece = 0

Etape1 : piece = 1

etc.

```
ALGORITHME algoJeu1
VARIABLE
On va déclarer toutes les variables (DECLARATION)
    piece : ENTIER
DEBUT
Donner une valeur à la variable (INITIALISATION)
    piece <- 0
FIN
```

LES VARIABLES

- Imaginons un personnage qui avance et récupère des pièces.

Dans un jeu, on aurait un compteur qui va changer au cours du temps.

Exemple : Etape initiale : piece = 0

Etape1 : piece = 1

etc.

ALGORITHME algoJeu1
VARIABLE
On va déclarer toutes les variables (DECLARATION)
    piece : ENTIER
DEBUT
Donner une valeur à la variable (INITIALISATION)
    piece <- 0
    ramassePiece
En algorithmie on assigne une valeur avec une flèche(ASSIGNATION)
    piece <- piece + 1 (En maths : 0 + 1 = 1)
FIN

QUIZZ

Question 1 :

- Quelle phrase définit le mieux une variable ?
  - C'est un conteneur qui peut disposer de plusieurs valeurs
  - C'est une valeur unique
  - C'est un symbole qui n'a pas de valeur

Question 2 :

- Quelle étape permet d'indiquer à un algorithme qu'une variable existe ?
  - L'assignation
  - La concaténation
  - La déclaration

Question 3 :

- En quoi consiste l'initialisation ?
  - A attribuer une valeur à une variable
  - A attribuer une première valeur à une variable
  - A attribuer un type de variable

Question 4 :

- Quel est le symbole de l'assignation ?
  - Une flèche
  - Le signe +
  - Les deux points

LES TYPES

- Exemple :

  Jean, 25 ans est un garçon

  Lana, 20 ans est une fille

- On peut conserver les 3 informations dans des variables :
  - nom
  - age
  - sexe

Nous avons donc les variables:
Nom avec la valeur "Jean" et "Lana"
Age avec la valeur 25 et 20
Sexe avec la valeur "garçon" et "fille"

Nous avons utilisé :
Une chaîne de caractère pour la variable nom
Un entier pour la variable âge
Une chaîne de caractère pour la variable sexe

Note:
On pourrais optimiser la variable sexe car celle-ci ne peut prendre que 2 valeurs.
On va préférer utiliser une variable binaire (qui ne prend que 2 valeurs 0 ou 1 (VRAI ou FAUX))
On appelle cela un booléen

LES TYPES

- Retour sur l'exemple :
  - Variables :

nom	« Jean »	« Lana »
age	25	20
sexe	VRAI	FAUX

Les chaînes de caractères seront toujours délimités par des guillemets

QUIZZ

Question 1 :

- Quel est le type permettant de réaliser des calculs ?
  - Entier
  - Nombre
  - Booleen

Question 2 :

- Que peut contenir le type "Booléen" ?
  - Une variable pouvant contenir des valeurs entières ou des textes
  - Une variable pouvant avoir que deux valeurs
  - Une variable contenant plusieurs valeurs

Question 3 :

- Quelle est la particularité d'une variable de type "chaîne de caractères" ?
  - On peut lui assigner un booléen
  - On peut faire des calculs dessus
  - Pour lui assigner une valeur il faut mettre le texte entre guillemets

Question 4 :

- Qu'est qu'une concaténation ?
  - c'est assigner un texte à une variable
  - c'est réaliser un calcul
  - c'est mettre des textes bout à bout

LES TESTS

- Les tests sont utilisé lorsqu'il y a plus d'une solution possible en fonction de la situation.

- Par exemple :

Si j'ai la clé alors je peux ouvrir la porte sinon elle reste fermée.

- 3 mots clés :
  - SI
  - ALORS
  - SINON

LES TESTS

- L'instruction SI en pseudo code donne :

SI <condition> ALORS

    <bloc d'instructions (toujours indenté)>

SINON

    <bloc d'instructions>

FIN SI

LES TESTS

- Une condition est une expression logique donnant le résultat "FAUX" ou "VRAI". Ce sera toujours un booléen qui devras être renvoyé par la condition.

- Dans le cas où le résultat du test sur la condition est VRAI alors on réalise les actions du bloc ALORS

- Dans le cas où le résultat du test sur la condition est FAUX alors on réalise les actions du bloc SINON

LES TESTS

- Lorsque vous avez des tests qui peuvent s'imbriquer, il existe le SINON SI qui peut optimiser votre code. Vous pouvez mettre autant de SINON SI que vous le souhaitez.

- Syntaxe :

SI <condition> ALORS
        <instructions>
SINON SI <condition> ALORS
        <instructions>
SINON
        <instructions>
FIN SI

LES OPERATEURS LOGIQUES (ET et OU)

- Imaginons le cas suivant :

Seul les hommes de plus de 20 ans peuvent accéder à la fête !

- SOLUTION :

SI sexe = TRUE ET age > 20 ALORS
    AFFICHER "Vous pouvez rentrer"
SINON
    AFFICHER "Vous ne pouvez pas rentrer"
FIN SI

- ET est un opérateur logique permettant de combiner des tests. Pour que le résultat du test soit VRAI, il faut que les deux conditions soient VRAI

LES OPERATEURS LOGIQUES (ET et OU)

- Imaginons le cas suivant :

Il faut soit être un homme ou avoir plus de 20 ans pour accéder à la fête !

- SOLUTION :

SI sexe = TRUE OU age > 20 ALORS
    AFFICHER "Vous pouvez rentrer"
SINON
    AFFICHER "Vous ne pouvez pas rentrer"
FIN SI

- OU est un opérateur logique permettant de combiner des tests. Pour que le résultat du test soit VRAI, il faut que l'une des deux conditions soient VRAI

QUIZZ

Question 1 :

- Quelle est la syntaxe complète de l'instruction SI ?
  - SI ..ALORS .. SINON
  - SI ..SINON .. FIN SI
  - SI ..SINON .. ALORS ... FIN SI
  - SI ..ALORS .. SINON .. FIN SI

Question 2 :

- Quelle phrase est fausse ?
  - Il est possible de tester un BOOLEEN
  - Il est possible de tester un ENTIER
  - Il est possible de tester une CHAINE DE CARACTERES
  - aucune, toutes les phrases sont exactes
  - toutes, les phrases sont toutes fausses

Question 3 :

- Quel mot clef permet d'avoir obligatoirement deux expressions booléennes à VRAI, dans un test, pour que le bloc ALORS se réalise ?
  - ET
  - OU
  - SI

Question 4 :

- Quelle phrase représente le mieux le OU ?
  - une des deux expressions booléennes minimum doit être "VRAI"
  - seulement une des deux expressions booléénnes doit être "VRAI"
  - les deux expressions booléénnes doivent être "VRAI"

L'INSTRUCTION SELON

- L'instruction SELON
- L'instruction "SELON" permet de réaliser certaines actions en fonction d'une variable (comme le permet un "SI").
- L'avantage du "SELON" est qu'il permet de "tester" plusieurs valeurs de manière plus efficace qu'un "SI", et de réduire le nombre de "SI" imbriqués.
- L'inconvénient du "SELON" est qu'il ne permet pas de "tout" tester. Il n'est pas très adapté pour tester des plages de valeur (signe < ou >), exemple entre 5 et 15.

L'INSTRUCTION SELON

- Voici la structure :

SELON <variable>

CAS <resultat1> : <instructions>

CAS <resultat2> : <instructions>

CAS <resultat3> : <instructions>

AUTREMENT : <instructions>

FIN SELON

QUIZZ

Question 1 :

- Que permet de faire l'instruction "selon" ?
  - Réaliser des tests sur plusieurs variables pour effectuer des traitements spécifiques en fonction de leur valeur
  - Réaliser un test sur une variable pour effectuer des traitements spécifiques en fonction de sa valeur
  - Réaliser un traitement en fonction d'une expression Booléenne

Question 2 :

- Quelle condition l'instruction "selon" ne peut pas tester "facilement" ?
  - x=10
  - x="bonjour"
  - x>20

Question 3 :

- A quoi sert le "autrement" ?
  - A réaliser des actions pour les cas qui ne sont pas prévus dans le "SELON"
  - A réaliser des tests supplémentaires en plus du "SELON"
  - Ce mot clé n'est pas directement lié à l'instruction "SELON"

Question 4 :

- Quelle ligne ci-dessous ne permet pas d'initialiser une variable ?
  - saisir x
  - x <- 0
  - x <- x+1

LES BOUCLES

- Lorsque l'on réalise plusieurs fois la même action, il est "dommage" de devoir réécrire les mêmes instructions.
- De plus, dupliquer du code est très limitant et impose d'avoir des algorithmes complexes "pour rien"

LES BOUCLES

- Prenons un exemple de répétition simple :
  - Un personnage doit avancer 5 fois grâce au pseudo-code « avancer » pour arriver à sa destination.

```
ALGORITHME algoAvancerPersonnage
DEBUT
        avancer
        avancer
        avancer
        avancer
        avancer
FIN
```

LES BOUCLES

Il existe 2 façons d'améliorer l'algorithme:

- Si nous connaissons le nombre de fois où le personnage doit avancer,
  - On dirait littéralement : avancer 5 fois
  - On pourra utiliser la boucle POUR qui va permettre d'effectuer la même action X fois.

- Si nous ne connaissons pas le nombre de fois où le personnage doit avancer
  - On dirait littéralement : avancer tant que le personnage n'est pas arrivé à sa destination.
  - On utilisera la boucle TANT QUE (ou FAIRE/TANT QUE) qui va permettre de réaliser une action tant qu'une condition n'est pas atteinte

POUR

- Cette boucle n'est utilisable que si nous connaissons le nombre de fois où l'on doit réaliser une (ou des) instruction(s)

- Syntaxe :

POUR <compteur> ALLANT DE <début> A <fin> PAR PAS DE <x> FAIRE

    <instructions>

FIN POUR

POUR

- Résultat pour l'exemple précédent :

```
ALGORITHME algoAvancerPersonnage
VARIABLE
        compteur<- ENTIER
DEBUT
        POUR compteur ALLANT DE 1 A 5 PAR PAS DE 1 FAIRE
                avancer
        FIN POUR
FIN
```

POUR

- Regardons en détail ce que fait la boucle POUR :
- Le compteur commence à la valeur 1 et s'arrête à la valeur 5
- Le pas à été mit à 1, ce qui veut dire que le compteur augmente de 1 en 1

ETAPE	COMPTEUR	INSTRUCTION	ALGORITHME
0	Pas encore utilisé	-	Personnage n'a pas bougé
1	1	avancer	Personnage a « avancer » 1 fois
2	2	avancer	Personnage a « avancer » 2 fois
3	3	avancer	Personnage a « avancer » 3 fois
4	4	avancer	Personnage a « avancer » 4 fois
5	5	avancer	Personnage a « avancer » 5 fois. La limite du POUR est atteinte on sort de la boucle

TANT QUE

- Cette boucle permet de répéter des instructions TANT QUE la condition n'est pas remplie
- Voici la syntaxe :

TANT QUE <condition> FAIRE

    <instructions>

FIN TANT QUE

TANT QUE

- Reprenons notre exemple:
  - Un personnage doit avancer 5 fois grâce au pseudo-code « avancer » pour arriver à sa destination.

Résultat :

```
ALGORITHME avancerPersonnage
DEBUT
        TANT QUE pasArrive FAIRE
                avancer
        FIN TANT QUE
FIN
```

FAIRE TANT QUE

- On peut parfois améliorer des algorithmes en utilisant une boucle particulière le « FAIRE TANT QUE ».

- Les boucles FAIRE TANT QUE et TANT QUE sont identiques, à une seule différence près : Le test de la condition est réalisée en premier pour le TANT QUE, et après les instructions pour le FAIRE TANT QUE. Cela peut être bien pratique lorsqu'on a besoin d'initialiser une variable avant de boucler par exemple.

Syntaxe :
FAIRE
<instructions>
TANT QUE <condition>

QUIZZ

Question 1 :

- Quel est l'intérêt des boucles ?
  - éviter de déclarer des variables
  - éviter de répéter du code
  - éviter d'initialiser des variables

Question 2 :

- Quand devez-vous choisir la boucle POUR ?
  - dès que nous connaissons pas le nombre de fois où nous réalisons une action
  - dès que l'on connait le nombre de fois qu'une action doit être réalisée
  - dans tous les cas la boucle POUR peut être utilisée

Question 3 :

- Qu'est ce qu'une incrémentation ?
  - C'est l'équivalent d'une assignation.
  - C'est mettre un texte dans une variable ayant le type ENTIER
  - c'est augmenter la valeur d'une variable.

Question 4 :

- Quelle est la différence entre la boucle TANT QUE et FAIRE TANT QUE ?
  - Le FAIRE TANT QUE réalise l'action avant le test
  - Le TANT QUE réalise l'action avant le test

LES FONCTIONS

- Les fonctions permettent de regrouper des instructions dans un même bloc réutilisable.
- Sans le savoir nous avons déjà utiliser des fonctions qu'on a imaginer déjà défini par la machine.
- Exemple : saisir ou afficher.
- Les fonctions sont au final comme des petits algorithmes.

LES FONCTIONS

- Syntaxe : La syntaxe ressemble beaucoup à celle d'un algorithme

FONCTION <nomFonction (toujours en camelCase)>

DEBUT

    <instructions>

FIN

LES FONCTIONS

- On peut aller plus loin avec les fonctions. Il est possible de leur donner des informations (appelés paramètres) pour qu'elles puissent réaliser des traitements spécifiques.

Syntaxe :

```
FONCTION
<nomFonction>(<parametre(s)>)
VARIABLE
        <variable> <- <type>
DEBUT
        <instructions>
FIN
```

LES FONCTIONS

- Exemple :

Nous voulons réaliser une fonction permet de toujours multiplié par 2 un nombre qui lui a été donné comme information en entrée.

Résultat :

```
FONCTION multipliParDeux(nb)
VARIABLE :
        resultat <- ENTIER
DEBUT
        resultat <- nb * 2
        afficher resultat
FIN
```

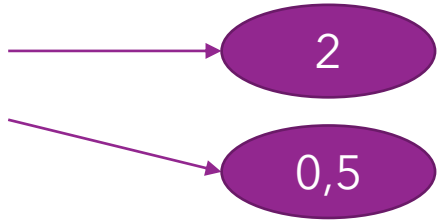
A chaque fois que nous appellerons cette fonction dans un programme, nous devrons lui donner un chiffre en entrée.

LES FONCTIONS

- Comme dit précédemment une fonction peut recevoir plusieurs paramètres.
- Il faudra faire attention à l'ordre des informations passées.

Exemple :
On veut diviser les 2 nombres en paramètres de la fonction:

```
ALGORITHME exempleFonction
VARIABLE
        nb1<- ENTIER
        nb2<-ENTIER
DEBUT
        FONCTION diviser(dividende, diviseur)
           afficher dividende/diviseur
        FIN
        diviser(4, 2)
        diviser(2, 4)
FIN
```



LES FONCTIONS

- Il y a un défaut, sur les fonctions faites jusqu'à présent on pouvait se servir des fonctions que pour afficher des informations.
- Grace au pseudo-code RETOURNER on peut récupérer une (et une seule !) information d'une fonction.
- Cela pourrait nous permettre de récupérer l'information pour faire d'autre calcul ou d'autres affichage lors de l'appel d'une fonction.

QUIZZ

Question 1 :

- A quoi servent les fonctions ?
  - à définir un bloc de code modifiable dynamiquement
  - à définir un bloc de code réutilisable
  - à définir des variables

Question 2 :

- Comment transmettre des informations à une fonction ?
  - via des variables
  - via des types
  - via des instructions
  - via des paramètres

Question 3 :

- Quel point n'est pas important lorsqu'on réalise des fonctions ?
  - l'ordre des paramètres
  - Le nom des paramètres
  - Le type des paramètres

Question 4 :

- Avec les fonctions il n'est pas possible de
  - Modifier la valeur d'un paramètre
  - Utiliser un paramètre

LES TABLEAUX

- Un tableau est un conteneur pouvant disposer de plusieurs valeurs
- C'est un type de variable !
- Un tableau contiendras des valeurs défini ainsi qu'un indice implicite
- Exemple : un tableau contenant des prénoms :

Les indices sont dans les bulles violettes. Le premier élément est à la position 0 donc le dernier élément sera toujours : taille du tableau - 1

LES TABLEAUX

- Définir un tableau en pseudo-code :
- Le symbole des tableaux est : []
- Le nom de la variable précède le symbole.

- Dans le cas d'un tableau contenant des valeurs entière on inscrira [n] et [c] pour le type chaîne de caractères

LES TABLEAUX

Alain	Alice	Pierre	Georges	Inès
0	1	2	3	4

- Voici la syntaxe pour ce tableau :

```
ALGORITHME tableauNom
VARIABLE
        t[c] <- TABLEAU CHAINE DE CARACTERES
DEBUT
        t[0] <- "Alain"
        t[1] <- "Alice"
        t[2] <- "Pierre"
        t[3] <- "Georges"
        t[4] <- "Inès"
FIN
```

LES TABLEAUX

Alain	Alice	Pierre	Georges	Inès
0	1	2	3	4

- Autre syntaxe pour ce tableau :

```
ALGORITHME tableauNom
VARIABLE
        t[c] <- TABLEAU CHAINE DE CARACTERES
DEBUT
        t <- ["Alain", "Alice", "Pierre", "Georges", "Inès"]
FIN
```

LES TABLEAUX

- On ne peut pas afficher directement un tableau complet, il faudra afficher une par une les informations contenu.

- On peut afficher l'élément à une position donnée.
- Exemple :

    afficher t[1] pour afficher le deuxième élément.

LES TABLEAUX

- Pour afficher tous les éléments d'un tableau, c'est possible pour cela on devra utiliser une boucle.
- La boucle POUR est très pratique pour cela.
- Nous aurons pour cela accès au pseudo-code taille(<tableau>) qui est une fonction avec paramètre.
- taille(<tableau>) nous renverra la taille du tableau mis en paramètre
- Exemple :

```
POUR compteur ALLANT DE 0 A taille(t)-1 PAR PAS DE 1 FAIRE
        afficher t[compteur]
FIN POUR
```

QUIZZ

Question 1 :

- A quoi servent les tableaux ?
  - à stocker plusieurs variables
  - à stocker plusieurs types
  - à stocker plusieurs valeurs

Question 2 :

- A quels indices commence et s'arrête un tableau ?
  - Début: 1 fin : taille du tableau
  - Début: 1 fin : taille du tableau -1
  - Début: 0 fin : taille du tableau -1

Question 3 :

- Quel particularité est liée aux tableaux
  - La concaténation des textes ne peux pas se faire avec des valeurs de tableaux
  - Le tableau ne contient que des valeurs d'un seul type
  - Un tableau ne peut pas contenir de valeur de type booléen

Question 4 :

- Quelle syntaxe n'est pas bonne ?
  - x<-5, 6 ,7
  - x<-[5, 6, 7]
  - x[0] <- 5
    x[1] <- 6
    x[2] <- 7