Licence STS Université Claude Bernard Lyon I

# LIFAPI : ALGORITHMIQUE ET PROGRAMMATION IMPÉRATIVE, INITIATION

## COURS 1 : INTRODUCTION À L'ALGORITHMIQUE

# COORDONNÉES ET SITE WEB

Responsables de L'UE

- Elodie DESSEREE elodie.desseree@univ-lyon1.fr
- Marie LEFEVRE marie.lefevre@univ-lyon1.fr

Responsables d'amphis

- Elodie DESSEREE (séquence 1)
- Nicolas PRONOST (séquence 3)
- Marie LEFEVRE (séquence 5)

Site WEB de l'UE (pour infos pratiques, supports, corrections …)

➔ http://perso.univ-lyon1.fr/elodie.desseree/LIFAPI/

# DÉTAIL DES ENSEIGNEMENTS DE L'UE

CM : 8 séances de 1h30

- Présentation des concepts fondamentaux
- Illustration par des exemples

TD : 16 séances de TD de 1h30

- Exercices d'application des notions vues en CM
- Ecriture d'algorithmes sur papier
- Indispensable d'avoir appris le cours avant la séance

TP : 16 séances de 1h30

- Exercices de difficulté similaires à ceux des TDs mais traduits en langage de programmation → sur machine

# PLANNING DE L'ANNÉE 2025-2026

| Semaine | Creneaux | | | | |
|---|---|---|---|---|---|
| | matin de la séquence | | | après-midi de la séquence | |
| | 8h00 - 9h30 | 9h45 - 11h15 | 11h30 - 13h00 | 14h00 - 15h30 | 15h45 - 17h15 |
| 08/09/2025 | CM1 : Algo | TD1 : Algo 1 | TD2 : Algo 2 | CM2 : C/C++ | |
| 15/09/2025 | TD3 : Algo 3 | TP1 : Prise en main | TP2 : Etoiles | CM3 : Fonctions / procédures | |
| 22/09/2025 | TD4 : Fonctions / procédures | TP3 : Fonctions / procédures | Soutien 1 | CM4 : Passage de parametres | |
| 29/09/2025 | TD5 :Passage de paramètres | CC1 TD 25% | TP4 :Fonctions / procédures | CM5 : Tableaux | |
| 06/10/2025 | TD6 : Passage de paramètres | TP5 : Passage de paramètres | | CM6 : Chaines de caractères | |
| 13/10/2025 | TD7 : Tableaux 1D | TP6 : Tableaux 1D | Soutien 2 | CM7 : Structures | |
| 20/10/2025 | TD8 : Tableaux 1D / 2D | TP7 : Tableaux 2D | | CM8 :CC Blanc | |
| 27/10/2025 | Vacances de Toussaint | | | | |

# PLANNING DE L'ANNÉE 2025-2026

| | | | | | |
|---|---|---|---|---|---|
| 03/11/2025 | TD9 : Chaines de caractères | TP8 : Chaines de caractères | | TD10 : Chaines de caractères | |
| 10/11/2025 | CC mi-parcours 8h - 9h **(seq 1 et 5)** | | | CC mi-parcours 14h-15h **(seq 3)** | |
| 17/11/2025 | TD11 : Mémory | TP9 : Chaines de caractères | | | |
| 24/11/2025 | TD12 : Structures | TP10 : Structures | | TP noté 25 % | |
| 01/12/2025 | TD13 : Démineur | TP11 : Memory | | TP12 : Démineur | |
| 08/12/2025 | TD14 : Révisions | TP13 : démineur | Soutien 3 | TP noté 25 % | |
| 15/12/2025 | TD15 : Grapic ou corrections | TP14 : Grapic ou corrections | | | |
| 22/12/2025 | Vacances de Noël | | | | |
| 29/12/2025 | | | | | |
| 05/01/2026 | Seconde Chance | | | | |

# Modalités de Contrôle des Connaissances et des Compétences (MCCC)

4 épreuves avec un coefficient de 25% chacune

- semaine du 29 septembre ➔ 1h en TD
- semaine du 10 novembre ➔ 1h en TD
- semaine du 24 novembre ➔ 1 en TP sur machine
- semaine du 8 décembre ➔ 1h30 en TP sur machine

1 épreuve de seconde chance (mais pas de rattrapage en juin !!)

- pendant la semaine d'examens en janvier 2026
- pour rattraper **1 absence** ou améliorer une note (pas obligatoire si aucune absence)

# MCCC ET GESTION DES ABSENCES / CALCUL DE LA MOYENNE

Gestion des absences

- À justifier dans les 48h qui suivent l'absence auprès de la scolarité
- **1 absence max** (justifiée ou non) aux épreuves de contrôle continu ➔ au-delà **DEF**

Calcul de la moyenne de l'UE

- Si pas d'absence et pas de seconde chance alors moyenne des 4 notes de CC
- Si pas d'absence mais seconde chance alors moyenne des 4 meilleures notes parmi les 5
- Si 1 absence alors seconde chance obligatoire et moyenne 3 notes CC + seconde chance
- Si 2 absences ou plus alors défaillant pour l'année

# INFORMATIONS PRATIQUES

Début des TDs
- Juste après ce CM

Début des TPs
- Semaine du 15 septembre

Environnement de travail
- Windows
- Répertoire utilisateur W:

Outils complets pour le développement
- C5 (https://c5.univ-lyon1.fr/)
- CodeBlocks (gratuit)
- Dev-Cpp (gratuit)
- Microsoft Visual C++ (logiciel payant)
- Xcode sous Mac-OS

# PLAN

LIFAPI : programme de l'UE, objectifs

LIFAPI / Culture Numérique / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Quelques exemples complets

# PROGRAMME DE L'UE

Algorithmique :

- syntaxe algorithmique, écriture d'algorithmes
- structures de contrôle : itérations, conditions
- sous-programmes (fonctions / procédures)
- mode de passage des paramètres dans des sous-programmes
- tableaux / chaînes de caractères
- structures

Programmation impérative :

- Traduction dans un langage de programmation adapté des notions algorithmiques étudiées (fonction / procédure, alternative, séquence, structures, tableaux, chaînes de caractères, …)

(Utilisation d'une bibliothèque graphique)

# OBJECTIFS DE L'UE

Analyser un problème

Le formaliser

Concevoir une solution (algorithme)

Programmer l'algorithme

Exécuter le programme sur un ordinateur

# PLAN

LIFAPI : programme de l'UE

LIFAPI / PIX / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Quelques exemples complets

# AUTRES UE INFORMATIQUES DE LA L1

- PIX dans TR1 : concepts informatiques généraux (bureautique, outil internet, réseaux, messagerie, création de pages WEB …) ➔ certification PIX (pour TOUS)
- Pour licence Info ou Math-Info
  - LIFUNIX : Unix
  - LIFBAP : Bases de l'architecture pour la programmation

  - LIFAMI : Applications aux maths et à l'info
  - LIFAPR : Algorithmique programmation fonctionnelle et récursive
  - LIFIRW : Introduction aux réseaux et au Web

# PLAN

LIFAPI : programme de l'UE

LIFAPI / Culture Numérique / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Quelques exemples complets

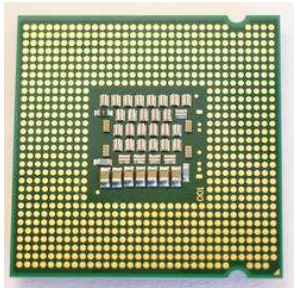
# COMPOSITION D'UN ORDINATEUR

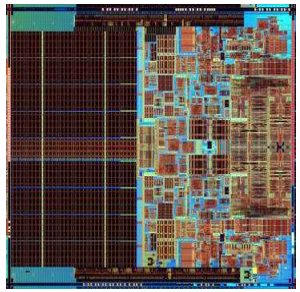
Vision **simpliste** du contenu d'un ordinateur

- Processeur : effectue les opérations
- Mémoire(s), disques : stockage données, instructions
- …

Effectue des opérations à partir de données

Vues d'un processeur





# LE PROCESSEUR COMPREND

Programme (séquence d'instructions du processeur)

| Code machine | Assembleur |
|---|---|
| cc2: 55 | push %ebp |
| cc3: 89 e5 | mov %esp,%ebp |
| cc5: 53 | push %ebx |
| cc6: 83 ec 14 | sub $0x14,%esp |
| cc9: e8 fc ff ff ff | call cca |
| cce: 81 c3 02 00 00 00 | add $0x2,%ebx |
| cd4: 8b 45 08 | mov 0x8(%ebp),%eax |
| cd7: 89 44 24 04 | mov %eax,0x4(%esp) |
| cdb: 8b 45 08 | mov 0x8(%ebp),%eax |
| cde: 89 04 24 | mov %eax,(%esp) |

Seul langage compris par le processeur

Codage hexadécimal des instructions

→ Quasi inutilisable pour programmeur

# PLAN

LIFAPI : programme de l'UE

LIFAPI / Culture Numérique / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Quelques exemples complets

# POURQUOI PROGRAMMER ?

Programmation existe partout

- Réveil
- Digicode
- Téléphone
- Tablette …

Besoin d'effectuer des nouvelles tâches

➔ besoin d'écrire des programmes nouveaux

- Par non informaticien : formalisation en français
- Par informaticien : langage compréhensible par lui et la machine

# UN PROGRAMME C'EST QUOI ?

Un programme, c'est tout ce qui fonctionne sur votre ordinateur, par exemple :

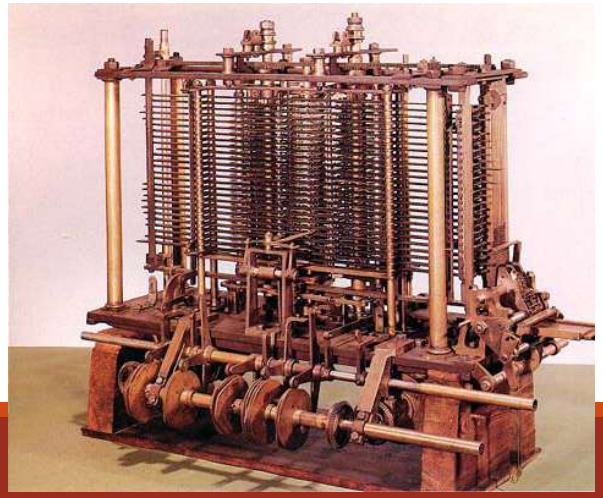
- Un jeu vidéo (Fortnite, Call of Duty, …)
- Un lecteur vidéo (comme VLC par exemple ou Youtube),
- Ou même un truc tout simple comme OpenOffice, Mozilla Firefox.
- Et le plus important **le système d'exploitation** (Windows, Android …)

**Sans programme pas d'application sur votre ordinateur** !

# LA NAISSANCE DE LA PROGRAMMATION

- Première machine programmable : métier à tisser de Jacquard en **1801** (suite de cartons perforés avec le motif à reproduire lors du tissage). Repris par IBM bien plus tard !

- En **1936**, création de l'ordinateur programmable : la machine de Turing
  - premier calculateur universel programmable
  - invention des concepts et des termes de programmation et de programme.



# LE LANGAGE DE PROGRAMMATION

Langage commun entre

- Le programmeur
- Le processeur : traduit en assembleur puis en code machine

Grande diversité

- Langage C/C++ (LIFAPI, ce semestre)
- Python (NSI au lycée pour certains)
- Scheme / Racket (LIFAPR, prochain semestre)
- Java, Matlab, Mathematica, macros word / excel (écrites en Visual Basic for Applications VBA)…
- …
- Plus de 700 langages de programmation !!

# DU PROBLÈME AU PROGRAMME

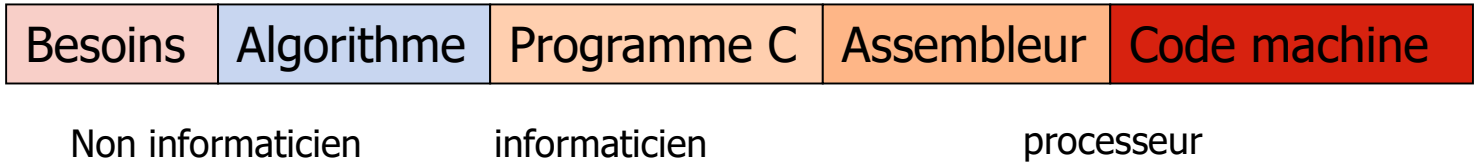
Besoins exprimés en français (cahier des charges)

Traduction dans un langage "universel" ➔ algorithmique

Traduction de l'algorithme en programme C

Puis en code assembleur

Puis en code machine compréhensible par le processeur



# PLAN

LIFAPI : programme de l'UE

LIFAPI / Culture Numérique / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Quelques exemples complets

# L'ALGORITHME AU QUOTIDIEN

L'algorithmique intervient dans la vie de tous les jours

- Une **recette** de cuisine :
  - des entrées (les ingrédients, le matériel utilisé) ;
  - des instructions élémentaires simples, dont l'exécution amène au résultat voulu ;
  - un résultat : le plat préparé.
- Le **tissage**, surtout tel qu'il a été automatisé par le métier Jacquard est une activité que l'on peut dire algorithmique.
- Un **casse-tête**, tel le Rubik's Cube, peut être résolu de façon systématique par un algorithme qui mécanise sa résolution.
- En **sport**, l'exécution de séquences répondant à des finalités d'attaque, de défense, de progression.

# ALGORITHME : DÉFINITION

Un algorithme est une méthode

- Suffisamment générale pour permettre de traiter toute une classe de problèmes
- Combinant des opérations suffisamment simples pour être effectuées par une machine

Pour un problème donné, il peut y avoir plusieurs algorithmes ou aucun

# ALGORITHME : PROPRIÉTÉS

**Lisible** : l'algorithme doit être compréhensible même par un non-informaticien

**Haut niveau** : doit pouvoir être traduit en n'importe quel langage de programmation➔ ne pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné

**Précis / non ambigu** : chaque élément de l'algorithme ne doit pas porter à confusion

**Concis :** ne doit pas dépasser une page, sinon décomposer le problème en plusieurs sous-problèmes

**Structuré :** un algorithme doit être composé de différentes parties facilement identifiables

# ALGORITHME : MÉTHODOLOGIE

Trois étapes caractérisent la résolution d'un problème

1. **comprendre la nature du problème** posé
   et préciser les **données** fournies
   ("entrées" ou **"input"** en anglais)
2. **préciser les résultats** que l'on désire obtenir
   ("sorties" ou **"output"** en anglais)
3. **déterminer le processus de transformation**
   des données en résultats.

# ALGORITHMIQUE / LANGAGE PROGRAMMATION

Un algorithme est

- Une **suite d'instructions élémentaires**

décrites dans un langage universel exécutées de manière **séquentielle**

- Indépendant du langage de programmation

Un langage de programmation

- Est un langage commun entre machine et programmeur
- Implante ou réalise un algorithme

# PLAN

LIFAPI : programme de l'UE

LIFAPI / Culture Numérique / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Quelques exemples complets

# L'INSTRUCTION, LA SÉQUENCE

Instruction

- Étape dans un programme informatique / brique de base
- Dicte à l'ordinateur l'action nécessaire qu'il doit effectuer avant de passer à l'instruction suivante.
- Opération élémentaire
- Comprise et exécutée par le processeur

Séquence / suite d'instructions

- Suite bloquée d'instructions qui sont exécutées **dans l'ordre où elles sont écrites**, dans toutes les circonstances du traitement.
- Délimitée par **Début** et **Fin (→ bloc)**

```
Début
        instruction1
        instruction2
        …
        instructionN
Fin
```

# LA VARIABLE / LA CONSTANTE

Une **variable**

- nom utilisé dans un programme pour faire référence à une donnée manipulée par programme
- peut contenir un entier, un réel, un caractère…
- associe un nom ou symbole à une valeur
- sa valeur peut éventuellement varier au cours du temps

Une **constante**

- nom utilisé pour faire référence à une valeur permanente (dont la valeur ne change pas au cours du programme).

  ➔ PI = 3.14159…

# TYPE DES DONNÉES

Définit les valeurs que peut prendre une donnée, ainsi que les opérateurs qui peuvent lui être appliqués

Types de base utilisés en algorithmique

- Caractère : ‘c’ , ‘a’, ‘-’, ‘!’ …
- Entier : 3 0 -3 -789
- Réel : 0 3,345 -7,678
- Booléen : VRAI / FAUX
- …

# DÉCLARATION DES VARIABLES

La *déclaration* permet de donner un nom à la variable

- Eventuellement de lui associer un type,
- Ainsi qu'une valeur initiale.

Exemples

- indice : entier

permettra de déclarer une variable "indice" de type entier

- Est_majuscule : booléen

permettra de déclarer une variable booléenne

La variable doit avoir un nom aussi évocateur que possible de son contenu

# AFFECTATION

Attribue une valeur à une **variable**

Symbolisée en algorithmique par le symbole "←"

La valeur peut être

- une valeur numérique

  **a ← 2** (la variable *a* contient la valeur 2)

- le résultat d'une expression

  **variable ← expression**

  **var1 ← a + 2*racine(15)**

# OPÉRATIONS SUR LES VARIABLES

Affectation : variable ← expression

La variable contient la valeur de l'expression

Cette valeur est conservée jusqu'à la prochaine affectation

Une variable peut apparaître dans une expression,
elle sera remplacée par la valeur qu'elle contient au moment du calcul
de l'expression

# CONTENU D'UNE VARIABLE

Pour pouvoir stocker la valeur et vérifier qu'une variable est correctement utilisée, une variable a un **type.**

Un type est

- un **domaine de valeurs** (ensemble des valeurs possibles)
  - Entiers, réels
  - Booléen
  - caractères
- un **ensemble d'opérations** pour manipuler ces valeurs
  - Addition, soustraction, multiplication …
  - Opérations logiques
  - Concaténation, substitution …

# Entrées / Sorties

Assurent la communication programmeur / machine

Données du problème (utilisateur → machine)

**Lire** (valeur) ou **Saisir** (Valeur)

Résultats affichés à l'écran (machine → utilisateur)

**Afficher** (valeur) ou **Ecrire** (Valeur)

# STRUCTURES DE CONTRÔLE

- Contrôlent l'ordre dans lequel seront effectuées les instructions
- 2 catégories
  - Les structures de choix
  - Les structures de boucles

# CONDITIONNELLE

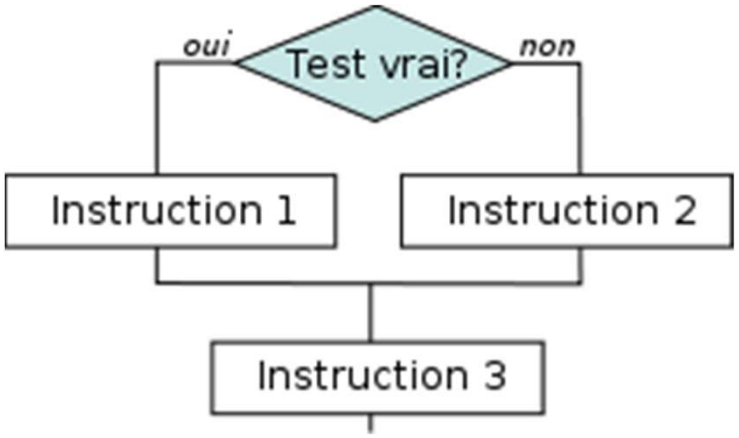
**Si** condition **alors**

instruction(s)

**Sinon**

instruction(s)

**FinSi**




Condition = expression booléenne (vrai / faux)

- Élémentaire
- Complexe (conjonction, négation ou disjonction de conditions élémentaires et/ou complexes)

Partie sinon facultative : il n'y a pas nécessairement de traitement à effectuer.

# CONDITIONNELLE : EXEMPLES

- Exemple 1 sans "sinon"

```
Si (A>2) alors
    B←A*3
FinSi
```

- Exemple 2 avec "sinon"

```
Si ((A<10) et (B>racine(A*5))) alors
        B←A*3
        A← A+B
Sinon
        A←A+2

        B←A*B
FinSi
```

Les opérateurs de comparaison sont

- = ➔ égal à…
- ≠ ➔ différent de…
- < ➔ strictement plus petit que…
- > ➔ strictement plus grand que…
- ≤ ➔ plus petit ou égal à…
- ≥ ➔ plus grand ou égal à…

Expression booléenne

3 opérateurs logiques ET, OU, NON

# CONDITION / TEST

- Apparaît dans les "Si" et les "Tant Que"
- Variable **booléenne** qui renvoie comme valeur **VRAI** ou **FAUX**
- Combinaison de conditions
  - conjonction (**ET**)
  - disjonction (**OU**)
  - négation (**NON**)
- Tables de vérité

| X | Y | X et Y |
|---|---|---|
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

| X | Y | X ou Y |
|---|---|---|
| V | V | V |
| V | F | V |
| F | V | V |
| F | F | F |

| X | Non X |
|---|---|
| V | F |
| F | V |

# CONDITIONNELLE : IMBRICATION

- gérer des décisions complexes ou multiniveaux
- placer un test conditionnel à l'intérieur d'un autre permet de
  - raffiner les choix
  - réagir à plusieurs niveaux

- Pourquoi utiliser l'imbrication ?
  - Pour éviter les répétitions.
  - Pour organiser la logique de manière hiérarchique.
  - Pour gérer des cas complexes sans tout mélanger.

# CONDITIONNELLE : IMBRICATION

```
si temperature > 25 alors
   si budget ≥ 100 alors
      afficher "On part à la plage !"
   sinon
      afficher "On fait un pique-nique au parc."
   fin si
sinon
      afficher "On reste à la maison avec un bon film.«
fin si
```

# STRUCTURE CONDITIONNELLE SELON

- lorsqu'on a plusieurs niveaux d'imbrication
- aussi appelée ***à choix multiple*** ou ***sélective***
- sélectionne entre plusieurs choix à la fois, et non entre deux choix alternatifs (le cas de la structure SI).

**SELON** (sélecteur) **FAIRE**
Cas <valeurs-1> : <suite d'action (s)-1>
[Cas <valeur-2> : <suite d'action (s)-2>
…….. ]
[Autrement : <suite d'action (s)-n> ]
**FIN SELON**

Le ***sélecteur*** est une variable de type entier ou caractère

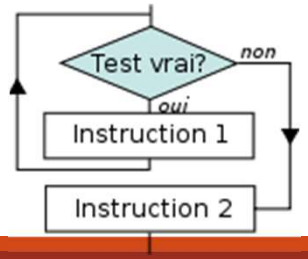
# STRUCTURE CONDITIONNELLE SELON : EXEMPLE

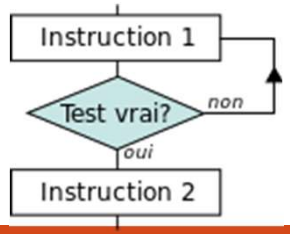
Afficher la couleur en fonction d'un entier = 1: rouge, 2 : orangé, 3 : jaune, 4 : vert, 5 : bleu, 6 : indigo et 7 : violet.

```
Selon couleur Faire
    Cas 1 : afficher(" rouge")
    Cas 2 : afficher(" orangé")
    Cas 3 : afficher(" jaune")
    Cas 4 : afficher(" vert")
    Cas 5 : afficher(" bleu")
    Cas 6 : afficher(" indigo ")
    Cas 7 : afficher(" violet")
   Autrement : afficher ("Couleur inconnue")
Fin selon
```

# STRUCTURE DE CONTRÔLE ITÉRATIVE

- Une **boucle** ou **itération** est une structure de contrôle destinée à exécuter une portion de code plusieurs fois de suite
- Les langages proposent en général plusieurs types de boucles :
  - boucle à pré-condition : la condition est vérifiée avant la première boucle
  - boucle à post-condition : la condition est vérifiée après la première boucle
  - boucle à condition d'arrêt : la condition est vérifiée au milieu de la boucle
  - boucle itérative : un compteur est utilisé pour compter le nombre d'itérations
  - boucle de parcours : la boucle est exécutée sur chacun des éléments d'une liste



# ITÉRATION : BOUCLE CONDITIONNELLE

Permet de réitérer une instruction
ou une suite d'instructions
jusqu'à ce qu'une condition ne soit plus vraie

Condition évaluée **avant** d'effectuer les instructions

**TantQue** condition **faire**

instruction(s)

**FinTantQue**

# BOUCLE CONDITIONNELLE : EXEMPLE

**i← 1**

**TantQue** i<10 **faire**

afficher(i)

i ← i+1

**FinTantQue**

Instruction qui modifie la condition
pour éviter les boucles infinies

# ITÉRATIVE : BOUCLE CONDITIONNELLE

Autre construction

Condition évaluée **après** avoir effectué les instructions

**Faire**

instruction(s)

**TantQue** condition

Les instructions sont effectuées au moins une fois

# BOUCLE CONDITIONNELLE : EXEMPLE

**i← 1**

**Faire**

afficher(i)

i ← i+1

**Tant que** i< 10

instruction qui modifie la condition pour éviter les boucles infinies

# BOUCLE INCONDITIONNELLE : POUR

Cas particulier du TantQue

**Pour** compteur **allant de** … **à** … **par pas de** … **faire**

instruction(s)

**FinPour**

Permet de répéter un nombre connu de fois
une suite d'instructions

# BOUCLE INCONDITIONNELLE : EXEMPLES

Compter de 1 à 10 (incrémentation)

**Pour** i **allant de** 1 **à** 10 par pas de 1 **faire**
    afficher(i)
**FinPour**

Résultat affiché ?

Compter de 10 à 1 (décrémentation)

**Pour** i **allant de** 10 **à** 1 **par pas de** -1 **faire**
    afficher(i)
**FinPour**

Résultat affiché ?

Compter de (deux en deux)

**Pour** i **allant de** 0 **à 10 par pas de** 2 **faire**
    afficher(i)
**FinPour**

Résultat affiché ?

# BOUCLE INCONDITIONNELLE : PARTICULARITÉS

- **Break** ➔ Interrompt **immédiatement** la boucle

```
pour i de 1 à 10 faire
   si i mod 2 = 0 alors
      afficher "Premier nombre pair : " + i
      break
   fin si
fin pour
```

- **Continue** ➔ Passe directement à l'itération suivante

```
pour i de 1 à 10 faire
   si i mod 3 = 0 alors
      continue
   fin si
   afficher i
fin pour
```

- Inconvénients
  - Le flux logique devient plus difficile à suivre
  - Risque de masquer des erreurs
  - Contredit la logique structurée
  - Difficile à tester et à maintenir

# PLAN

LIFAPI : programme de l'UE

LIFAPI / Culture Numérique / Autres UE informatiques

Le fonctionnement interne d'un ordinateur

La programmation

Le langage algorithmique

La syntaxe algorithmique

Un exemple complet

# EXEMPLE : CALCUL DU PRODUIT DE 2 ENTIERS

On veut calculer le produit de a par b deux entiers et stocker le résultat dans une variable C

Version 1 ➔ en utilisant l'opérateur "*"

**Début**

| | |
|---|---|
| a,b,c : entiers | déclaration des variables |
| afficher ("Donnez deux entiers") | entrées / sorties |
| saisir (a) | |
| saisir (b) | |
| c ← a*b | stockage du résultat |
| afficher ("**Le produit de**" , a, "**par**" , b, " **est :** ", **c**) | |

**Fin**

# EXEMPLE : CALCUL DU PRODUIT

Dans cet algorithme, on n'utilisera pas la multiplication !!

Raisonnement : $5 * 4 = \underbrace{5 + 5 + 5 + 5}_{4 \text{ fois}}$

Généralisation : a * b = a+a+…+a (b fois)

Formalisation

tant qu'on n'a pas ajouté **b** fois **a**,
on ajoute **a** à la somme

# EXEMPLE : CALCUL DU PRODUIT

```
Début
        a,b,c : entiers                          déclaration des variables
        afficher ("Donnez deux entiers")         entrées / sorties
        saisir (a)
        saisir (b)
        c ← 0                                    initialisation à 0 du résultat
        TantQue b ≠ 0 faire
                c ← c + a                        incrémentation
                b ← b – 1                        modification de la condition
        FinTantQue
        afficher ("Le produit de" , a, "par" , b, " est : ", c)
Fin
```

# EXEMPLE 2 : CALCUL DU PRODUIT

## Traduction algorithmique avec un "pour"

**Début**

| | |
|---|---|
| a,b,c ,i: entiers | déclaration des variables |
| afficher ("Donnez deux entiers") | entrées / sorties |
| saisir (a) | |
| saisir (b) | |
| c ← 0 | initialisation à 0 du résultat |
| **Pour i allant de 1 à b par pas de 1 faire** | |
| c ← c + a | incrémentation |
| **Fin Pour** | |
| afficher ("**Le produit de**" , a, "**par**" , b, " **est :** ", **c**) | |

**Fin**

# CONCLUSION

Tour d'horizon des notions de bases de l'algorithmique

- Variable : déclaration, type
- Instruction : séquence, bloc
- Structures de contrôle
  - Conditionnelles
    - Conditionnelle ➔ si
    - Sélecteur ➔ selon
  - Boucles
    - Conditionnelles ➔ Tant que
    - Inconditionnelles ➔ Pour