

# L'algorithme



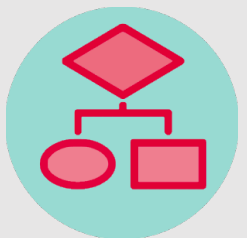
# L'algorithmique

- ❏ Introduction
- ❏ Variables et opérateurs
- ❏ Les structures conditionnelles
- ❏ Les structures répétitives
- ❏ Les structures de données
- ❏ Fonctions et procédures
- ❏ Programmation orientée objet



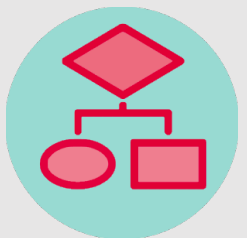
# Concepts importants

- ❏ **Algorithme** : mot dérivé du nom du mathématicien al\_Khwarizmi qui a vécu au 9ème siècle, était membre d'une académie des sciences à Bagdad .
- ❏ Un algorithme prend des **données en entrée**, exprime un **traitement** particulier et fournit des **données en sortie**.
- ❏ **Programme** : série d'instructions pouvant s'exécuter en séquence, ou en parallèle (parallélisme matériel) qui réalise (implémente) un algorithme



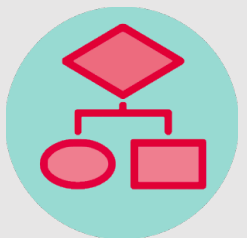
# Pourquoi l'algorithmique ?

- ❖ Permet d'obtenir de la « machine » qu'elle **effectue un travail** à notre place
- ❖ Problème : **expliquer** à la « machine » comment elle doit s'y prendre
- ❖ Besoins :
  - ❖ savoir **explicit**er son raisonnement
  - ❖ savoir **formal**iser son raisonnement
  - ❖ **concevoir** (et écrire) des algorithmes:
    - ❖ séquence d'instructions qui décrit comment résoudre un problème particulier



# Un algorithme

- ❖ Savoir expliquer comment faire un travail sans la moindre ambiguïté
- ❖ Langage simple : des instructions (pas élémentaires)
- ❖ Suite finie d'actions à entreprendre en respectant une chronologie imposée
- ❖ L'écriture algorithmique : un travail de programmation à visée universelle
  - ❖ un algorithme ne dépend pas du langage dans lequel il est implanté,
  - ❖ ni de la machine qui exécutera le programme correspondant.



# Exemples d'algorithmes

## Recette de cuisine

**Recette de la pâte à crêpes**

**Il te faut :**

- 250 g Farine
- 50 cl Lait
- 3 oeufs
- 1 pincée Sel

**1**

Verse la farine et le sel dans le saladier.

**2**

Casse les oeufs, mélange avec la cuillère en bois et ajoute progressivement le lait sans cesser de tourner.

**3**

Laisse reposer la pâte pendant 1 heure.

**4**

Fais chauffer la poêle. Verse-y une louche de pâte. Répartie-la bien bougeant la poêle. Fais cuire une crêpe fine.

*Recette de la pâte à crêpes*

## Notice de montage d'un meuble en kit

**BIRKELAND**

**1**

3x

2x

2x

**2**

1x

**3**

1x

1x

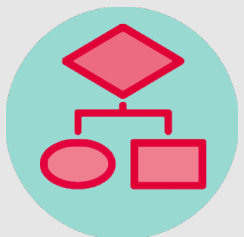
8x

3x

4x

**2**

**3**



# Problèmes fondamentaux en algorithmique

## ❖ Complexité

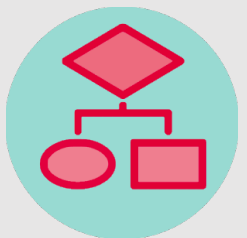
- ❖ En combien de temps un algorithme va -t-il atteindre le résultat escompté ?
- ❖ De quel espace va-t-il avoir besoin ?

## ❖ Calculabilité

- ❖ Existe-t-il des tâches pour lesquelles il n'existe aucun algorithme ?
- ❖ Etant donnée une tâche, peut-on dire s'il existe un algorithme qui la résolve ?

## ❖ Correction

- ❖ Peut-on être sûr qu'un algorithme réponde au problème pour lequel il a été conçu ?



# Exemple de langage algorithmique

## Algorithme ElèveAuCarré

*{Cet algorithme calcule le carré du nombre que lui fournit l'utilisateur}*

**variables** unNombre, sonCarré: entiers

*{déclarations: réservation  
d'espace-mémoire}*

**début**

*{préparation du traitement}*

afficher("Quel nombre voulez-vous élever au carré?")

saisir(unNombre)

*{traitement : calcul du carré}*

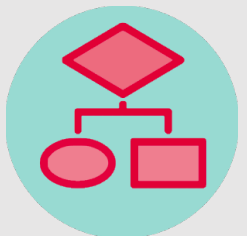
sonCarré  $\leftarrow$  unNombre  $\times$  unNombre

*{présentation du résultat}*

afficher("Le carré de ", unNombre)

afficher("c'est ", sonCarré)

**fin**





# Etapes d'un algorithme

## ❖ Préparation du traitement

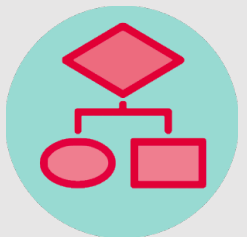
- ❖ données nécessaires à la résolution du problème

## ❖ Traitement

- ❖ résolution pas à pas,
- ❖ après décomposition en sous-problèmes si nécessaire

## ❖ Edition des résultats

- ❖ impression à l'écran,
- ❖ dans un fichier, etc.



# Langage algorithmique

- ❖ Il faut avoir une **écriture rigoureuse**

- ❖ Il faut avoir une écriture soignée : respecter l'**indentation**

- ❖ Il est nécessaire de **commenter** les algorithmes

- ❖ Il existe plusieurs solutions algorithmiques à un problème posé

- ❖ Il faut rechercher l'**efficacité** de ce que l'on écrit

**Algorithme** NomAlgorithme

{ ceci est un commentaire }

**Début**

... Actions

**Fin**

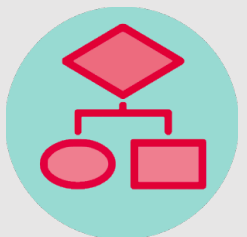
**Algorithme** Bonjour

{ il dit juste bonjour mais ... en anglais ! }

**Début**

afficher('Hello world !!!')

**Fin**



# Déclaration des données

❏ **Variable** <nom de donnée>: **type**

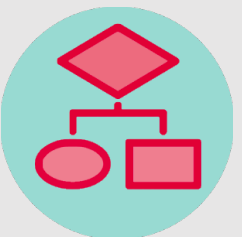
❏ Instruction permettant de réserver de l'espace mémoire pour stocker des données

❏ Dépendant du type des données : entiers, réels, caractères, etc.)

❏ Exemples :

**Variables** val, unNombre: **entiers**

nom, prénom : **chaînes de caractères**



# Déclaration des données

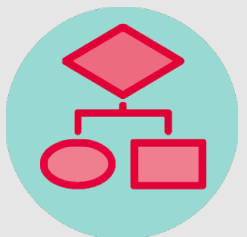
❏ **Constante** <nom de donnée>: **type** ← valeur ou expression

❏ Instruction permettant de réserver de l'espace mémoire pour stocker une constante dont la valeur ne varie pas.

❏ Exemples :

❏ **Constante** MAX : entier ← 10

DEUXFOISMAX : entier ← MAX x 2



# Lecture – écriture de données

❏ **Saisir**<nom de donnée, ...>

❏ **Afficher**<nom de donnée, ...>

❏ **Fonction** : Instructions permettant

- ❏ de **placer en mémoire** les informations fournies par l'utilisateur.

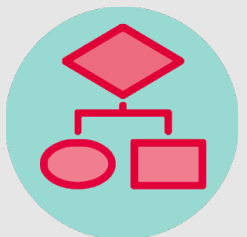
- ❏ De **visualiser** des données placées en mémoire

❏ **Exemples**:

Saisir(unNombre)

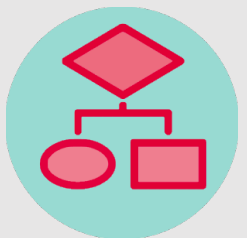
Afficher (« le nom est » , nom, »et le prénom est » ,prénom )

Saisir(val)



# Phase d'analyse

- ❖ Consiste à extraire de l'énoncé du problème des éléments de modélisation
- ❖ Technique : distinguer en soulignant de différentes couleurs :
  - ❖ Quel est le but du programme (**traitement à réaliser**)
  - ❖ **Données en entrée** du problème :
  - ❖ Où vont se situer **les résultats en sortie**

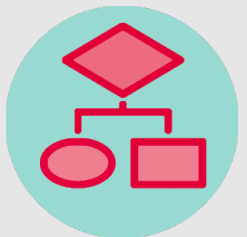


# Exemple d'énoncé d'un problème

❖ On souhaite calculer et afficher, à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

❖ Le montant TTC dépend de :

- ❖ Du prix HT
- ❖ Du taux de TVA de 20 %



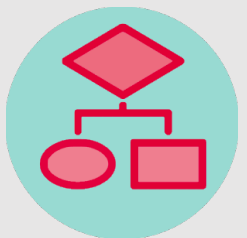
# Exemple d'énoncé d'un problème

❏ On souhaite calculer et afficher, à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

❏ Le montant TTC dépend de :

- ❏ Du prix HT
- ❏ Du taux de TVA de 20 %

TRAITEMENT À RÉLISER





# Exemple d'énoncé d'un problème

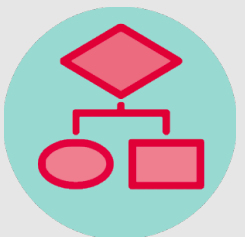
❏ On souhaite calculer et afficher, à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

❏ Le montant TTC dépend de :

❏ Du prix HT

❏ Du taux de TVA de 20 %

DONNÉES D'ENTRÉES



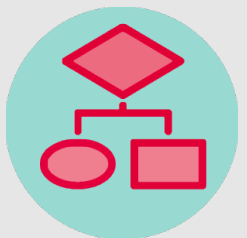
# Exemple d'énoncé d'un problème

❖ On souhaite calculer et afficher, à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

❖ Le montant TTC dépend de :

- ❖ Du prix HT
- ❖ Du taux de TVA de 20 %

DONNÉES DE SORTIES



# Algorithme TVA

## Algorithme CalculTVA

{Saisit un prix HT et affiche le prix TTC correspondant}

**Constantes** TVA : réel. ← 20

TITRE : chaîne ← "Résultat"

**Variables** prixHT : réel

**Variables** prixTTC, montantTVA : réels {déclarations}

**Début** {préparation du traitement}

afficher("Donnez-moi le prix hors taxe :")

saisir(prixHT)

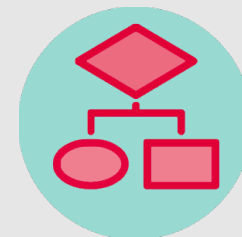
prixTTC ← prixHT \* (1 + TVA / 100) {calcul du prix TTC}

montantTVA ← prixTTC - prixHT

afficher(TITRE) {présentation du résultat}

afficher(prixHT, "€ H.T. + TVA ", TVA, "devient", prixTTC, "€ T.T.C.")

**Fin**



# Simulation d'un algorithme

## Algorithme CaDoitEchanger?

{Cet algorithme .....}

**Variables** valA, valB: réels {déclarations}

**Début** {préparation du traitement}

Afficher ("Donnez-moi deux valeurs :")

Saisir (valA, valB)

Afficher ("Vous m'avez donné ", valA, " et ", valB)

{traitement mystère}

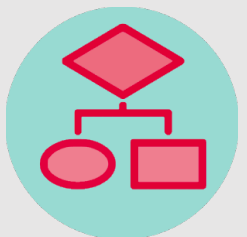
valA ← valB

valB ← valA {présentation du résultat}

Afficher("Maintenant , mes données sont : ", valA, " et ", valB)

**Fin**

Que fait cet algorithme ?



# Simulation d'un algorithme

## Algorithme CaDoitEchanger?

{Cet algorithme .....}

**Variables** valA, valB: réels {déclarations}

**Début** {préparation du traitement}

Afficher ("Donnez-moi deux valeurs :")

Saisir (valA, valB)

Afficher ("Vous m'avez donné ", valA, " et ", valB)

{traitement mystère}

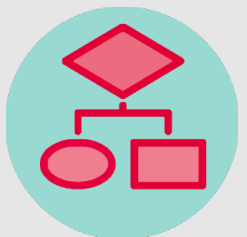
valA ← valB

valB ← valA {présentation du résultat}

Afficher("Maintenant , mes données sont : ", valA, " et ", valB)

**Fin**

Pas ce qui est prévu !!



# Simulation d'un algorithme

Il manque :

- ❖ Déclaration d'une variable supplémentaire

Variables valA, valB, **valTemp**: entiers

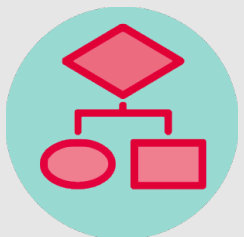
- ❖ Utilisation de cette variable pour stocker provisoirement une des valeurs

Saisir(valA, valB)

**valTemp** ← valA

valA ← valB

valB ← **valTemp**



# Questions ?



# Variables et opérateurs





# Utilité des variables

- ❖ Permet le stockage d'informations

- ❖ Un variable possède :

  - ❖ Un nom

  - ❖ Un type

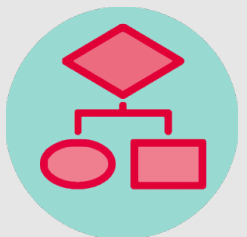
  - ❖ Une valeur

  - ❖ Une adresse

  - ❖ Une portée

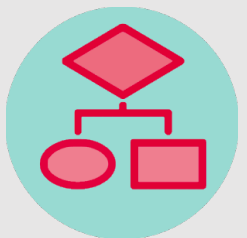
  - ❖ Une visibilité

  - ❖ Une durée de vie



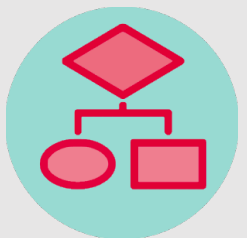
# Utilité des variables

- ❖ **Le nom** : le nom qui est donné à la variable
- ❖ **Le type** : Le type de la variable spécifie l'allocation mémoire nécessaire
- ❖ **La valeur** : ce qui est contenu dans la variable
- ❖ **L'adresse** : c'est l'endroit dans la mémoire où elle est stockée
- ❖ **La portée** : c'est la portion de code source où elle est accessible
- ❖ **La visibilité** : c'est un ensemble de règles qui fixe qui peut utiliser la variable (exemple : mots-clefs *public*, *private*, *protected*...)
- ❖ **La durée de vie** : durée pendant laquelle la variable existe



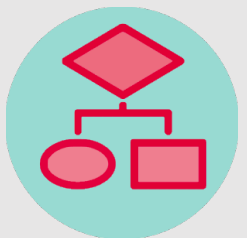
# Utilité des variables

- ❏ **Le nom** : monAge
- ❏ **Le type** : Entier
- ❏ **La valeur** : 25
- ❏ **L'adresse** : f09872cc
- ❏ **La portée** : accessible sur l'ensemble du programme
- ❏ **La visibilité** : public
- ❏ **La durée de vie** : existe tant que le programme est exécuté



# Utilité des variables – La portée

- ❖ La portée d'une variable est l'ensemble des sous-programmes où cette variable est connue (les instructions de ces sous-programmes peuvent utiliser cette variable)
- ❖ Une variable définie au niveau du programme principal (celui qui résout le problème initial, le problème de plus haut niveau) est appelée variable globale
  - ❖ Sa portée est **totale** : tout sous-programme du programme principal peut utiliser cette variable
- ❖ Une variable définie au sein d'un sous programme est appelée variable locale
  - ❖ La portée d'un variable **locale** est uniquement le sous-programme qui la déclare



# Utilité des variables – La portée

Bon exemple :

**Algorithme** ExemplePortee

{Cet algorithme permet de comprendre le principe de portée d'une variable.}

**Variable** valPorteeTotale : **entier**

**début**

Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

Saisir(valPorteeTotale)

**si** valPorteeTotale < 10 { comparaison avec le seuil}

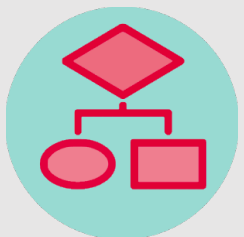
**Variable** valPorteeLocale : **entier** ← 10

**alors** valPorteeTotale ← valPorteeLocale

**Fsi**

Afficher ("Voici la valeur valPorteeTotale : " , valPorteeTotale)

**fin**



# Utilité des variables – La portée

## Mauvais exemple :

### Algorithme ExemplePortee

{Cet algorithme permet de comprendre le principe de portée d'une variable.}

**Variable** valPorteeTotale : **entier**

**début**

Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

Saisir(valPorteeTotale)

**si** valPorteeTotale < 10 { comparaison avec le seuil}

**Variable** valPorteeLocale : **entier** ← 10

**alors** valPorteeTotale ← valPorteeLocale

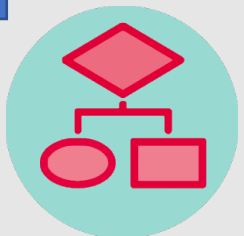
**Fsi**

valPorteeTotale ← valPorteeLocale

Afficher ("Voici la valeur valPorteeTotale :", valPorteeTotale)

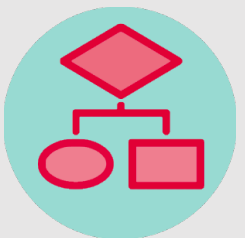
**fin**

La variable **valPorteeLocale** n'existe pas à cet endroit là du programme



# Utilité des variables

- ❖ On distingue généralement **cinq opérations** sur les variables, chacune pouvant revêtir des formes syntaxiques différentes.
  - ❖ la **déclaration** permet de déclarer un nom de variable, éventuellement de lui associer un type,
  - ❖ la **définition** permet d'associer une zone mémoire qui va être utilisée pour stocker la variable, comme lorsqu'on lui donne une valeur initiale,
  - ❖ l'**affectation** consiste à attribuer une valeur à une variable,
  - ❖ la **lecture** consiste à utiliser la valeur liée à la variable,
  - ❖ la **suppression** réalisée soit automatiquement soit par une instruction du langage.



# Convention de nommage

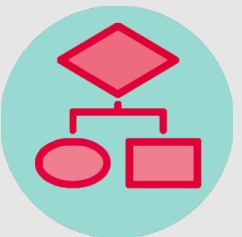
- ❏ Commence toujours par une minuscule
- ❏ Pas d'espace
- ❏ Ne doit pas commencer par un chiffre
- ❏ Pas de caractère spéciaux
- ❏ Majuscule pour séparer les mots (ou des Underscores)



maVariable  
ma\_variable



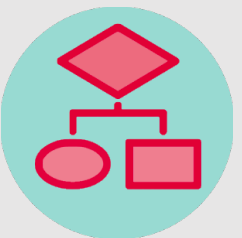
MaVariable  
2variable





# Constantes

- ❏ Une constante est un identificateur associé à une valeur fixe.
- ❏ Cet identificateur a tous les aspects d'une variable.
- ❏ Mais, il n'est possible de lui affecter une valeur qu'une seule fois
  - ❏ généralement au moment du lancement du programme.



# Convention de nommage

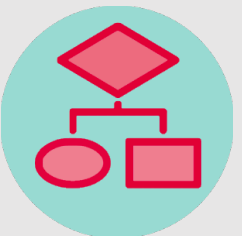
- ✦ Ecrit en MAJUSCULE
- ✦ Pas d'espace
- ✦ Ne doit pas commencer par un chiffre
- ✦ Pas de caractère spéciaux
- ✦ Underscores pour séparer les mots



MACONSTANTE  
MA\_CONSTANTE



maConstante  
2CONSTANTE

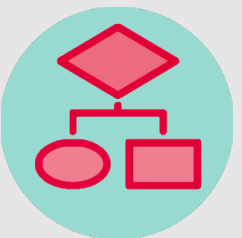


# Questions ?



# Types de données

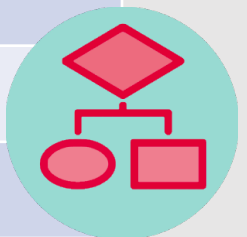
- ❏ **Typage dynamique** : la variable est typé au moment de l'affectation avec la valeur. Le système détecte automatique le type de la variable (Python)
- ❏ **Typage fort** : la variable doit être obligatoirement déclarées dans un type et utilisées dans ce type. (Java)
- ❏ **Type faible** : la variable peut changer de type au cours de son existence (PHP)



# Types de données

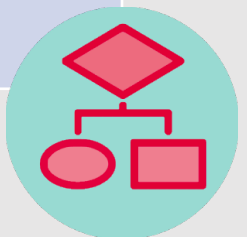
- ❖ Tous les langages, quels qu'ils soient offrent un « bouquet » de types de données, dont le détail est susceptible de varier légèrement d'un langage à l'autre.

TYPES	UTILITÉ	EXEMPLES
BOOLÉEN	Représente un état binaire, vrai ou faux	Vrai
ENTIER	Représente un nombre entier	1234
RÉEL (OU FLOAT)	Représente un nombre à décimal	1234,4321
CARACTÈRE	Représente un caractère unique	'C'
CHAÎNE DE CARACTÈRE	Représente un texte	« Voici un texte »



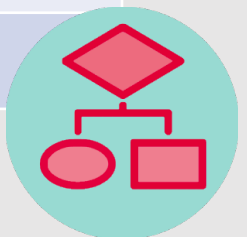
# Les opérateurs mathématiques

OPÉRATEURS	RÔLES	EXEMPLES
+	Addition	3+2 vaut 5
-	Soustraction	3-2 vaut 1
*	Produit	3*2 vaut 6
/	Division	3/2 vaut 1.5
%	Modulo (récupère le reste d'une division entière)	11%3 vaut 2 24%8 vaut 0



# Les opérateurs de comparaisons

OPÉRATEURS	RÔLES	EXEMPLES
>	Supérieur	3 > 2 vaut vrai
≥	Supérieur ou égale	3 ≥ 3 vaut vrai
<	Inférieur	3 < 2 vaut faux
≤	Inférieur ou égale	3 ≤ 3 vaut vrai
==	Parfaitement égale à <b>ATTENTION À NE PAS CONFONDRE AVEC = QUI EST L'AFFECTATION</b>	'C' == 'c' vaut faux
=	Égale à	'C' = 'c' vaut vrai
!=	Différent de	3 != 3 vaut faux



# Questions ?



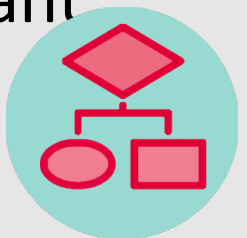


# Structures conditionnelles



# Tests et conditions

- ❖ Souvent les problèmes nécessitent l'étude de **plusieurs situations** qui ne peuvent pas être traitées par les séquences d'actions simples.
- ❖ Puisqu'on a plusieurs situations, et qu'avant l'exécution, on ne sait pas à quel cas de figure on aura à exécuter, dans l'algorithme on doit prévoir **tous les cas possibles**.
- ❖ Ce sont les ***structures conditionnelles*** qui le permettent, en se basant sur ce qu'on appelle ***prédicat*** ou ***condition***.



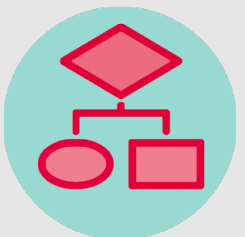
# Tests et conditions – Prédicat

❖ Un prédicat est un énoncé ou proposition qui peut être vrai ou faux selon ce qu'on est entrain de parler.

❖ Exemple :

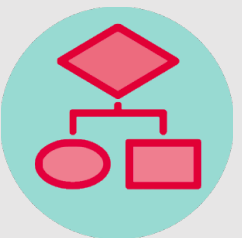
❖  $(10 < 15)$  est un prédicat vrai

❖  $(10 < 3)$  est un prédicat faux



# Tests et conditions – Condition

- ❖ Une condition est une expression de type logique.
- ❖ Ils lui correspondent deux valeurs possibles VRAI et FAUX qu'on note par V ou F.

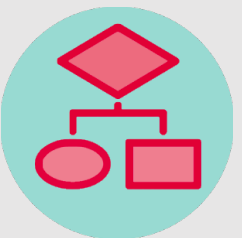


# Questions ?



# Les opérateurs logiques

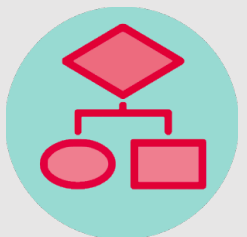
- ❖ La négation : "non"
- ❖ L'intersection : "et"
- ❖ L'union : "ou"



# Les opérateurs logiques

## ❖ La négation d'une condition

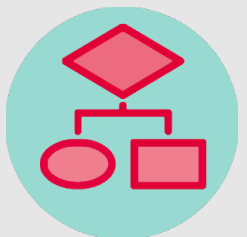
A	NON A
VRAI	FAUX
FAUX	VRAI



# Les opérateurs logiques

✚ L'intersection de deux conditions

A et B	VRAI	FAUX
VRAI	VRAI	FAUX
FAUX	FAUX	FAUX

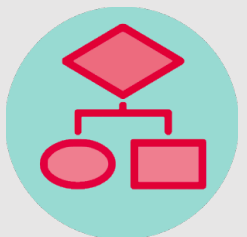




# Les opérateurs logiques

✚ L'union de deux conditions

A ou B	VRAI	FAUX
VRAI	VRAI	VRAI
FAUX	VRAI	FAUX



# Questions ?



# Structure conditionnelle SI

**si** <expression logique>

**alors** instructions

[**sinon** instructions]

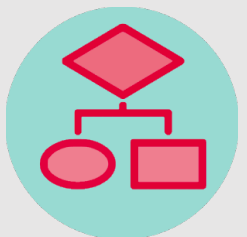
**fsi**

❖ Si l'expression logique (la condition) prend la valeur **vrai**

❖ le premier bloc d'instructions est exécuté;

❖ si elle prend la valeur **faux**

❖ le second bloc est exécuté (s'il est présent, sinon, rien).



# Structure conditionnelle SI

## Algorithme SimpleOuDouble

{Cet algorithme saisit une valeur entière et affiche son double si cette donnée est inférieure à un seuil donné.}

**constante** (SEUIL : **entier**) ← 10

**Variable** val : **entier**

**début**

Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

Saisir(val)

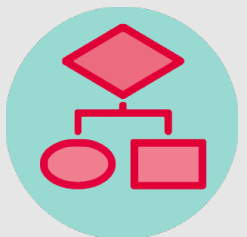
**si** val < SEUIL { comparaison avec le seuil}

**alors** Afficher ("Voici son double :", val x 2)

**sinon** Afficher ("Voici la valeur inchangée :", val)

**Fsi**

**Fin**



# Structure conditionnelle SI

## Autre écriture de l'exemple :

### **Algorithme** SimpleOuDouble

{Cet algorithme saisit une valeur entière et affiche son double si cette donnée est inférieure à un seuil donné.}

**constante** (SEUIL : entier) ← 10

**Variable** val : entier

**début**

Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

Saisir(val)

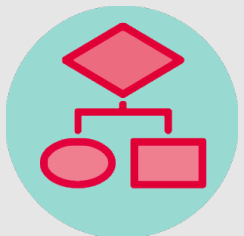
**si** val < SEUIL { comparaison avec le seuil}

**alors** val ← val x2

**Fsi**

Afficher ("Voici la valeur val :", val)

**fin**



# Structure conditionnelle SI

## Structure imbriquée :

Problème: afficher :

"Reçu avec mention Assez Bien " si une note est supérieure ou égale à 12,

" Reçu mention Passable" si elle est supérieure à 10 et inférieure à 12, et

"Insuffisant" dans tous les autres cas.

**si** note  $\geq$  12

**alors** afficher( "Reçu avec mention AB" )

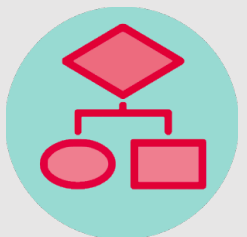
**sinon**      **si** note  $\geq$  10

**alors** afficher( « Reçu mention Passable" )

**sinon** afficher("Insuffisant" )

**fsi**

**fsi**



# Structure conditionnelle SI

## Structure imbriquée :

Problème: afficher :

"Reçu avec mention Assez Bien " si une note est supérieure ou égale à 12,

" Reçu mention Passable" si elle est supérieure à 10 et inférieure à 12, et

"Insuffisant" dans tous les autres cas.

**si** note  $\geq$  12

**alors** afficher( "Reçu avec mention AB" )

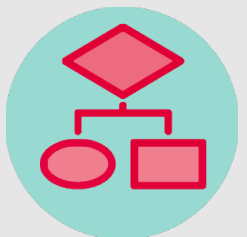
**sinon**      **si** note  $\geq$  10

**alors** afficher( « Reçu mention Passable" )

**sinon** afficher("Insuffisant" )

**fsi**

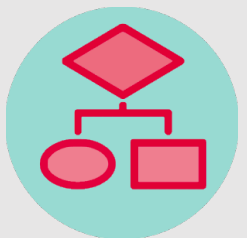
**fsi**



# Structure conditionnelle SI

## Exercice :

Ecrire l'algorithme qui permet de saisir un numéro de couleur de l'arc-en-ciel et d'afficher la couleur correspondante : 1: rouge, 2 : orangé, 3 : jaune, 4 : vert, 5 : bleu, 6 : indigo et 7 : violet.





# Structure conditionnelle SI

## Exercice - correction:

### **Algorithme** CouleurArcEnCiel

{Cet algorithme permet d'afficher les couleurs de l'arc-en-ciel en fonction du numéro saisi.}

**Variable** val : **entier**

**Variable** couleur : **chaîne**

**début**

Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

Saisir(val)

**si** val = 1 { comparaison avec le numéro saisi}

**alors** couleur ← « Rouge »

**Fsi**

**si** val = 2 { comparaison avec le numéro saisi}

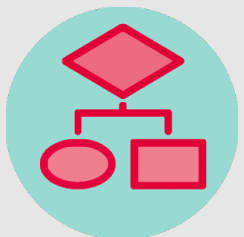
**alors** couleur ← « orangé »

**Fsi**

...

Afficher ("Voici la couleur : ", couleur)

**fin**



# Structure conditionnelle SELON

**selon** <identificateur>

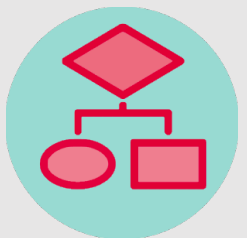
(liste de) valeur(s) : instructions

(liste de) valeur(s) : instructions

...

[**autres**: instructions]

❖ S'il y a **plus** de deux choix possibles, l'instruction selon permet une facilité d'écriture



# Structure conditionnelle SELON

**selon** abréviation

"M" : afficher( " Monsieur " )

"Mme" : afficher( " Madame " )

"Mlle" : afficher( " Mademoiselle " )

**autres** : afficher( " Monsieur, Madame " )

Équivalent avec instruction Conditionnelle

**si** abréviation = "M "

**alors** afficher( "Monsieur" )

**sinon**

**si** abréviation = « Mlle »

**alors** afficher("Mademoiselle")

**sinon**

**si** abréviation = "Mme"

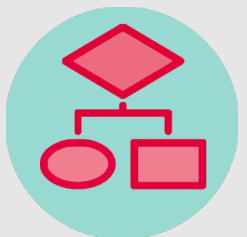
**alors** afficher( "Madame" )

**sinon** afficher( "Monsieur, Madame " )

**fsi**

**fsi**

**fsi**



# Structure conditionnelle SELON

**selon** abréviation

"M" : afficher( " Monsieur " )

"Mme" : afficher( " Madame " )

"Mlle" : afficher( " Mademoiselle " )

**autres** : afficher( " Monsieur, Madame " )

Équivalent avec instruction Conditionnelle (séquentiel)

**si** abréviation = "Mme "

**alors** afficher( « Madame » )

**fsi**

**si** abréviation = « Mlle »

**alors** afficher("Mademoiselle")

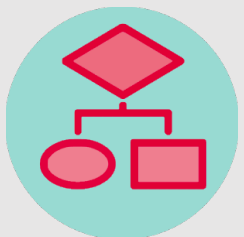
**fsi**

**si** abréviation = "M"

**alors** afficher( "Monsieur" )

**sinon** afficher( "Monsieur,Madame " )

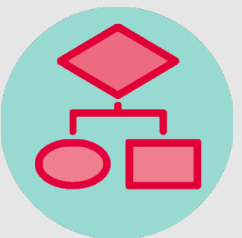
**fsi**



# Structure conditionnelle SELON

## Exercice :

Ecrire l'algorithme qui permet de saisir un numéro de couleur de l'arc-en-ciel et d'afficher la couleur correspondante : 1: rouge, 2 : orangé, 3 : jaune, 4 : vert, 5 : bleu, 6 : indigo et 7 : violet. (en utilisant SELON)



# Structure conditionnelle SELON

## Exercice - correction:

**Algorithme** CouleurArcEnCiel

{Cet algorithme permet d'afficher les couleurs de l'arc-en-ciel en fonction du numéro saisi.}

**Variable** val : entier

**Variable** couleur : chaîne

**début**

Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}

Saisir(val)

**selon** val

1 : couleur ← « rouge »

2 : couleur ← « orangé »

3 : couleur ← « jaune »

4 : couleur ← « vert »

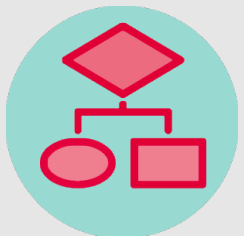
5 : couleur ← « bleu »

6 : couleur ← « indigo »

7 : couleur ← « violet »

**Autres** : couleur ← « Aucune couleurs ne correspond au numéro saisi »

Afficher ("Voici la couleur : ", couleur)



# Questions ?



# Structures répétitives





# Structure répétitive POUR

Valeur initiale

Valeur finale

**pour** <var> ← valinit **à** valfin [**par** <pas>] **faire**

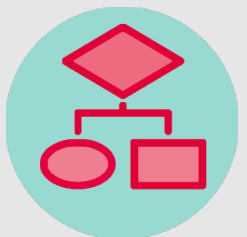
traitement {*suite d'instructions*}

Valeur à ajouter à <var> à  
chaque passage dans la  
boucle

**Fpour**

❖ **Fonction** : répéter une suite d'instructions un certain nombre de fois

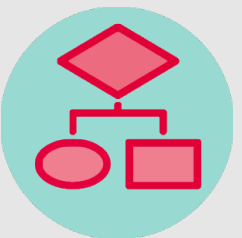
❖ **Pour** est utilisée **quand le nombre d'itération** est connu



# Structure répétitive POUR

❏ L'instruction **pour**:

- ❏ initialise une variable de boucle (le compteur)
- ❏ incrémente cette variable de la valeur de « pas »
- ❏ vérifie que cette variable ne dépasse pas la borne supérieure



# Structure répétitive POUR

Exemple :

**Algorithme** FaitLeTotal

{Cet algorithme fait la somme des nbVal données qu'il saisit}

**variables** nbVal, cpt : **entiers**

          valeur, totalValeurs: **réels**

**début**

    {initialisation du traitement}

**afficher**("Combien de valeurs voulez-vous saisir ?")

**saisir**(nbVal)

    {initialisation du total à 0 avant cumul}

    totalValeurs ← 0

    {traitement qui se répète nbVal fois}

**pour** cpt ← 1 à nbVal **faire**

**afficher**("Donnez une valeur :")

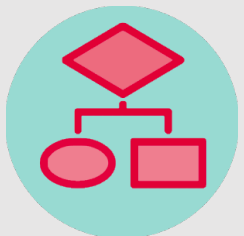
**saisir**(valeur)

        totalValeurs ← totalValeurs+ valeur {cumul}

**fpour**

    {édition des résultats}

**afficher**("Le total des ", nbVal, "valeurs est ", totalValeurs)



# Structure répétitive POUR

## ⚠ Attention :

- ⚠ le traitement ne doit pas modifier la variable de boucle

**Pour** cpt ← 1 à MAX **faire**

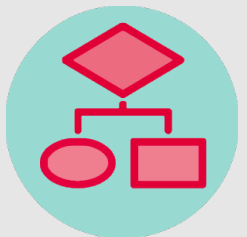
si (...) alors

cpt ← MAX

**Fsi**

**fpour**

INTERDIT !



# Structure répétitive TANT QUE ... FAIRE

amorçage

Initialisation de la (des)  
variable(s) de condition

**Tant que** <expression logique (vraie)> **faire**

traitement {*suite d'instructions*}

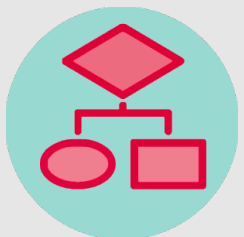
Suite d'instructions à  
exécuter si condition  
vraie

relance

Ré-affectation de la (des)  
variable(s) de condition

**Ftq**

❖ **Fonction** : répéter une suite d'instructions un certain nombre de fois



# Structure répétitive TANT QUE ... FAIRE

## Exemple :

### **Algorithme** FaitLeTotal

{Cet algorithme fait la somme des nbVal données qu'il saisit, arrêt à la lecture de -1}

Constante (STOP : entier) ← -1

**variables** val, totalValeurs : entiers

**début**

totalValeurs ← 0

**afficher**(« Donnez une valeur, »,STOP, « pour finir. ») {*amorçage*}

**saisir**(val)

**Tant que** val ≠ STOP **faire**

totalValeurs ← totalValeurs+ val {*traitement*}

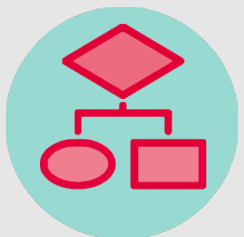
**afficher**("Donnez une autre valeur :")

**saisir**(valeur) {*relance*}

**fpour**

{édition des résultats}

**afficher**(« La somme des valeurs saisies est », totalValeurs)

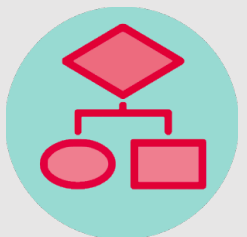


# Structure répétitive TANT QUE ... FAIRE

- ❖ Structure itérative « universelle »

  - ❖ N'importe quel contrôle d'itération peut se traduire par le « tant que »

- ❖ Structure itérative irremplaçable dès que la **conditions d'itération** devient **complexe**



# Structure répétitive TANT QUE ... FAIRE

## Exemple :

{saisir des valeurs, les traiter, et s'arrêter à la saisie de la valeur d'arrêt -1 ou après avoir saisi 5 données.}

**Constantes** (STOP : entier) ← -1

(MAX : entier) ← 5

**Variables.** nbVal, val : entiers

### Début

nbVal ← 0 {compte les saisies **traitées**}

**saisir**(val) {saisie de la 1ère donnée}

**tant que** val ≠ STOP **ou** nbVal < MAX **faire**

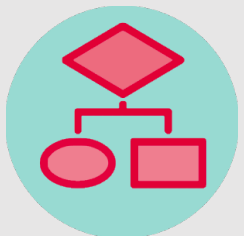
nbVal ← nbVal + 1...{traitement de la valeur saisie}

**saisir**(val) {relance}

**Ftq**

**afficher**(val, nbVal) {valeurs en sortie de boucle}

Remarque : La valeur d'arrêt n'est jamais traitée (et donc, jamais comptabilisée)





# Structure répétitive TANT QUE ... FAIRE

## Interpréter l'arrêt des itérations :

...

nbVal ← 0 {compte les saisies **traitées**}

**saisir**(val) {saisie de la 1ère donnée}

**tant que** val ≠ STOP **ou** nbVal < MAX **faire**

    nbVal ← nbVal + 1...{traitement de la valeur saisie}

**saisir**(val) {relance}

**Ftq**

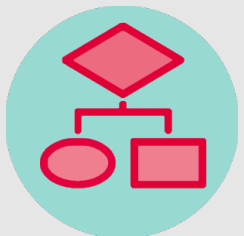
**si** val = STOP

**alors** {la dernière valeur testée était la valeur d'arrêt}

**afficher**(«Sortie de boucle car saisie de la valeur d'arrêt »)

**sinon** {il y avait plus de 5 valeurs à tester}

**afficher**(«Sortie de boucle car nombre maximum de valeurs à traiter atteint»)



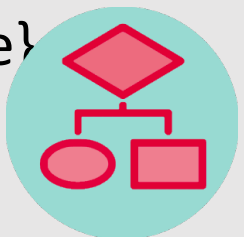
# Structure répétitive TANT QUE ... FAIRE **vs** POUR

❏ Implicitement, l'instruction **pour** :

- ❏ Initialise un compteur
- ❏ Incrémente le compteur à chaque pas
- ❏ Vérifie que le compteur ne dépasse pas la borne supérieur

❏ Explicitement, l'instruction **tant que** doit

- ❏ Initialiser un compteur {amorçage}
- ❏ Incrémenter le compteur à chaque pas {relance}
- ❏ Vérifier que le compteur ne dépasse pas la borne supérieur {test de boucle}



# Structure répétitive TANT QUE ... FAIRE **vs** POUR

**Pour** cpt ← 1 à nbVal **faire**

**afficher**(« Donnez une valeur : »)

**saisir**(valeur)

    totalValeurs ← totalValeurs + valeur {cumul}

**Fpour**

**Est équivalent à**

cpt ← 0

**Tant que** cpt < nbVal **faire**

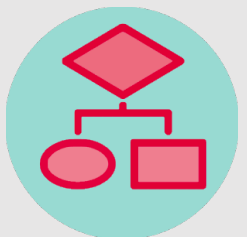
**afficher**(« Donnez une valeur : »)

**saisir**(valeur)

    totalValeurs ← totalValeurs + valeur {cumul}

    cpt ← cpt + 1 {compte le nombre de valeurs traitées}

**ftq**



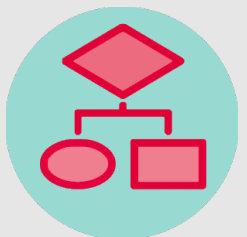
# Structure répétitive RÉPÉTER ... TANT QUE

## Répéter

(ré)affectation de la (des) variable(s) de condition  
traitement

**Tant que** <expression logique (vraie)>

❏ **Fonction** : exécuter une suite d'instruction au ***moins une fois*** et la répéter tant qu'une condition est remplie



# Structure répétitive RÉPÉTER ... TANT QUE

## Algorithme Essai

{cet algorithme a besoin d'une valeur positive paire}

**Variable** valeur : **entier**

**Début**

**Répéter**

**afficher**(« Donnez une valeur positive non nulle : »)

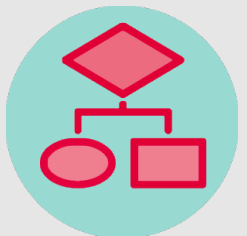
**saisir**(valeur)

**tant que** valeur  $\leq 0$

**afficher**(« la valeur positive non nulle que vous avez saisie est : »)

**afficher**(valeur)    {Traitement de la valeur saisie}

**Fin**



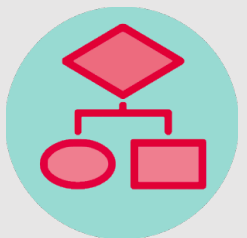
# Structure répétitive RÉPÉTER vs TANT QUE

## ❏ Boucle **tant que**

- ❏ condition vérifiée **avant** chaque exécution du traitement
- ❏ le traitement peut donc ne pas être exécuté
- ❏ la condition porte surtout sur la saisie de nouvelles variables (relance)

## ❏ Boucle **répéter ... tant que**

- ❏ Condition vérifiée **après** chaque exécution du traitement
- => Le traitement est exécuté **au moins une fois**
- ❏ La condition porte surtout sur le résultat du traitement



# Structure répétitive RÉPÉTER vs TANT QUE

## Répéter

**afficher**(« Donnez une valeur positive paire : »)

**saisir**(valeur)

**Tant que** (valeur < 0 **OU** (valeur %2) ≠ 0)

Équivaut à

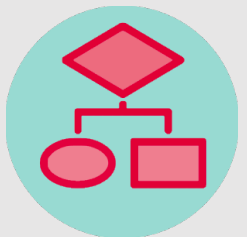
**Afficher**(« Donnez une valeur positive paire : »)

**Saisir**(valeur)

**tant que** (valeur < 0 **OU** (valeur %2) ≠ 0) **faire**

**afficher**(« Donnez une valeur paire : »)

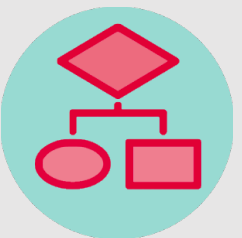
**saisir**(valeur)



# Structure répétitive BOUCLES

## Exercice :

Ecrire l'algorithme qui permet de saisir des données et de s'arrêter dès que leur somme dépasse 500





# Structure répétitive BOUCLES

Somme ← 0

**Répéter**

saisir(val)

somme ← somme + val

**Tant que** (somme ≤ 500)

Équivaut à

saisir(val)

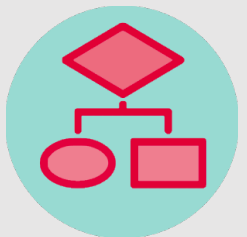
Somme ← val

**tant que** (somme ≤ 500) **faire**

saisir(val)

somme ← Somme + val

**ftp**



# Questions ?



# Les structures de données

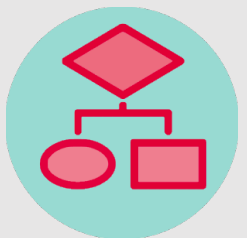


# Utilisation des tableaux

❏ Comment faire pour calculer la moyenne de 12 notes ?

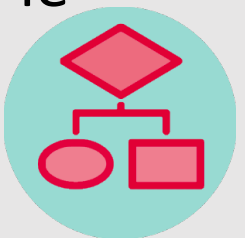
moy ←  $(n1+n2+n3+n4+n5+n6+n7+n8+n9+n10+n11+n12)/12$

Bonne solution ?



# Utilisation des tableaux

- ❖ Un ensemble de valeurs portant le même nom de variable et repérées par un nombre, s'appelle un **tableau**, ou encore une variable indicée.
- ❖ Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle **l'indice**.
- ❖ Chaque fois que l'on doit désigner un élément du tableau, on fait figurer le **nom du tableau**, suivi de **l'indice** de l'élément, entre **crochets**.



# Utilisation des tableaux

## ❏ Création d'un tableau noté note

Nombre d'éléments

Type du tableau

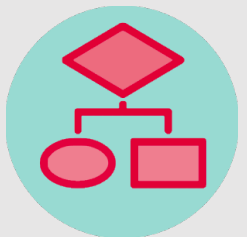
**Tableau note[12] : Entier**

Nom du tableau

❏ Chaque note individuelle (chaque élément du tableau note) sera donc désignée note[0], note[1], etc.

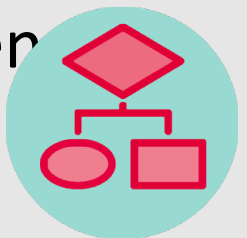
## ❏ **ATTENTION !!**

❏ les indices des tableaux commencent généralement à 0, et non à 1.



# Utilisation des tableaux

- ❖ On peut créer des tableaux contenant des **variables de tous types** : tableaux de **numériques**, tableaux de **caractères**, tableaux de **booléens**, tableaux de **chaîne de caractères**...
- ❖ Par contre, hormis dans quelques rares langages, on **ne peut pas** faire un mixage de **types différents** de valeurs au sein d'un même tableau.
- ❖ L'énorme avantage des tableaux, c'est qu'on va pouvoir les traiter en faisant des **boucles**.



# Utilisation des tableaux

Exemple pour effectuer le calcul de la moyenne des notes :

**Tableau** note [12] : entier

**Variables** moy, som : entier

**Début**

Pour i ← 0 à 11 faire

Afficher(« Entrez la note N° », i)

Saisir(note[i])

**fpour**

som ← 0

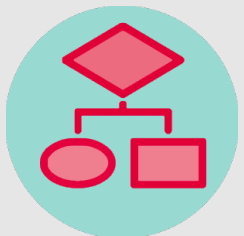
Pour i ← 0 à 11 faire

som ← som + note[i]

**fpour**

moy ← som / 12

**Fin**





# Utilisation des tableaux

✚ Si les valeurs saisies sont :

12, 13, 12, 10, 9, 20, 19, 15, 17, 18, 14, 8

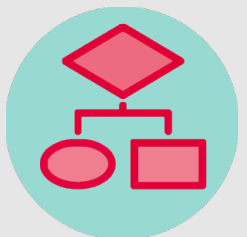
valeur du tableau  
à l'indice 6

Représentation du tableau :

12	13	12	10	9	20	19	15	17	18	14	8
0	1	2	3	4	5	6	7	8	9	10	11

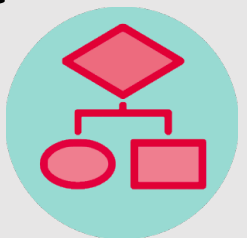
Indice 3 du  
tableau

12 éléments



# Utilisation des tableaux

- ❖ L'indice qui sert à désigner les éléments d'un tableau peut être exprimé directement comme un **nombre en clair**, mais il peut être aussi une **variable**, ou une **expression calculée**.
- ❖ Dans un tableau, la valeur d'un indice doit toujours :
  - ❖ être égale au moins à 0
  - ❖ être un nombre entier Quel que soit le langage, l'élément `truc(3,14)` n'existera jamais.
  - ❖ être inférieure ou égale au nombre d'éléments du tableau (moins 1, si l'on commence la numérotation à 0). Si le tableau `truc` a été déclaré comme ayant 25 éléments, la présence dans une ligne, de `truc(32)` déclenchera automatiquement une **erreur**.



# Utilisation des tableaux

❖ Il arrive fréquemment que l'on ne connaisse pas à l'avance le nombre d'éléments que devra comporter un tableau.

❖ 2 solutions :

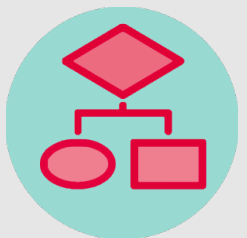
❖ Déclarer un tableau avec un taille gigantesque (10 000 éléments par exemple)

❖ **MAIS :**

❖ On ne sera jamais sûr de la taille du tableau

❖ La place mémoire réservée sera considérable (problème de rapidité)

❖ Déclarer un tableau dynamique sans préciser le nombre d'éléments



# Utilisation des tableaux

Exemple : saisir les notes pour un calcul de moyenne lorsqu'on ne sait pas combien il y aura de notes :

**Tableau** notes[] : **entier**

**Variables** nb : **entier**

**Début**

**Afficher**(« Combien y a-t-il de notes à saisir ? »)

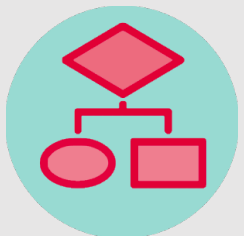
**Saisir**(nb)

**Redim** notes[nb]

...

**Fin**

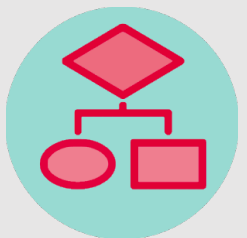
Tant que le nombre d'éléments du tableau n'est pas précisé, le tableau est inutilisable



# Utilisation des tableaux associatifs

- ❏ Un tableau associatif est un tableau dont les indices ne sont pas des nombres, mais des **chaînes de caractères**
- ❏ Un tableau associatif permet donc **d'associer** un nom à une valeur dans un tableau

**Tableau** nomTableau[« clef » ← < valeur >]



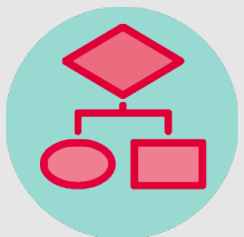
# Utilisation des tableaux associatifs

Exemple : associé le nom de l'élève à sa note :

**Tableau note [12] : entier**

note[« Remi » ← 12, « Florian » ← 13, « Maxime » ← 12, ...]

**Afficher**(note[« Remi »]) {affichera 12}



# Questions ?



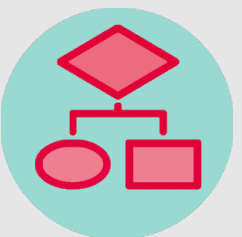
# Fonctions et procédures





# Les fonctions

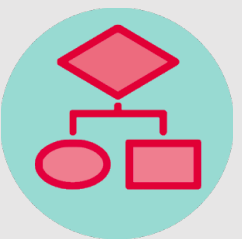
- ❖ Tout langage de programmation propose un certain nombre de **fonctions**.
- ❖ Certaines sont indispensables, car elles permettent d'effectuer des traitements qui seraient, sans elles, impossibles.
- ❖ D'autres servent à soulager le programmeur, en lui épargnant de longs – et pénibles – algorithmes.



# Les fonctions

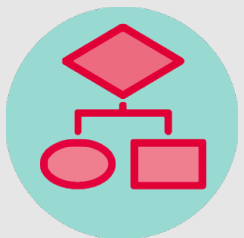
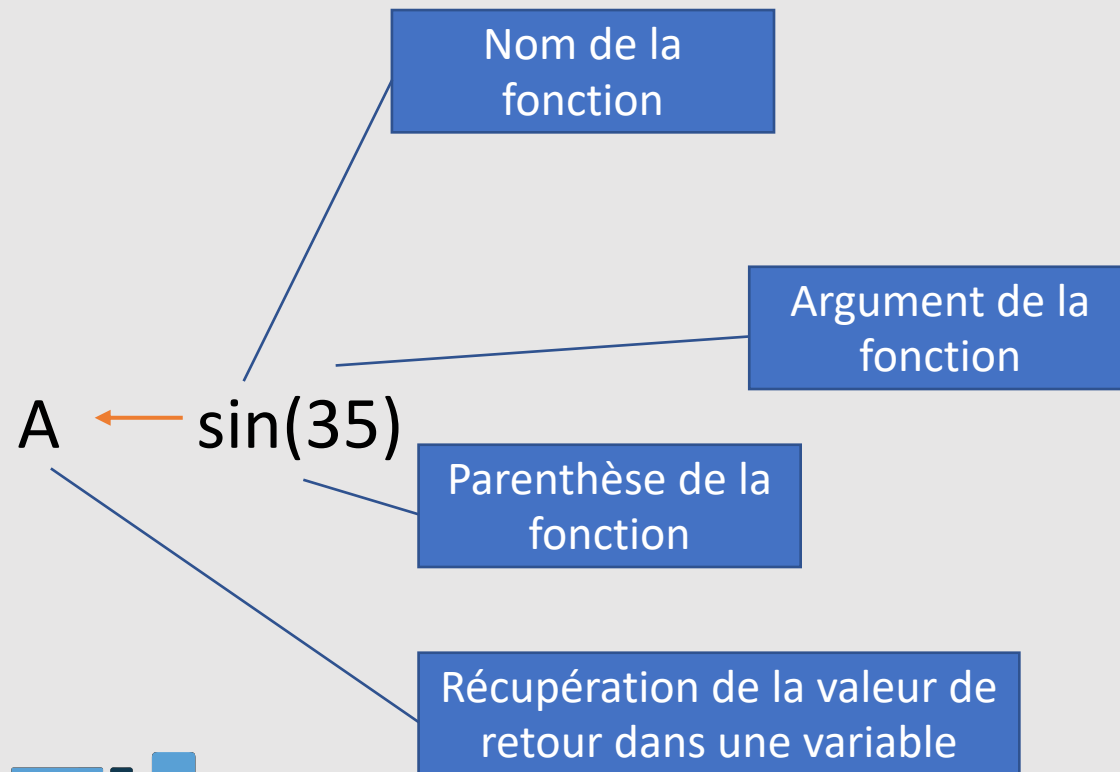
❖ Une fonction est définie par :

- ❖ Un **nom** : ce nom ne s'invente pas. Il doit impérativement correspondre à une fonction proposée par le langage
- ❖ Des **parenthèses** : deux parenthèses, une ouvrante, une fermante. Ces parenthèses sont toujours **obligatoires**, même lorsqu'on n'écrit rien à l'intérieur.
- ❖ Des **paramètres** : une liste de valeurs, indispensables à la bonne exécution de la fonction. Ces valeurs s'appellent des **arguments**, ou des **paramètres**. Certaines fonctions exigent un seul argument, d'autres deux, etc. et d'autres encore aucun.



# Les fonctions

Exemple : calcul du sinus d'un angle :



# Les fonctions

Exemple de fonctions (il en existe une multitude) :

❖ **Len(chaîne)** : renvoie le nombre de caractères d'une chaîne

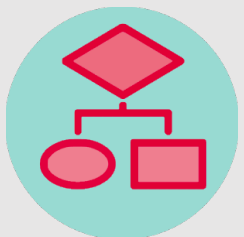
*Len("Bonjour, ça va ?") {vaut 16 (on compte aussi les espaces)}*

❖ **Mid(chaîne,n1,n2)** : renvoie un extrait de la chaîne, commençant au caractère n1 et faisant n2 caractères de long.

*Mid("Zorro is back", 4, 7) {vaut « ro is b » }*

❖ **Trouve(chaîne1,chaîne2)** : renvoie un nombre correspondant à la position de chaîne2 dans chaîne1. Si chaîne2 n'est pas comprise dans chaîne1, la fonction renvoie zéro.

*Trouve("Un pur bonheur", "pur") {vaut 4}*



# Les fonctions

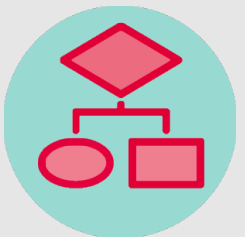
Exemple de fonctions (il en existe une multitude) :

❏ **Left(chaîne,n)** : renvoie les n caractères les plus à gauche dans chaîne

**Left("Et pourtant...", 8) {vaut "Et pourt »}**

❏ **Right(chaîne,n)** : renvoie les n caractères les plus à droite dans chaîne

**Right("Et pourtant...", 4) {vaut "t..."}**



# Les fonctions

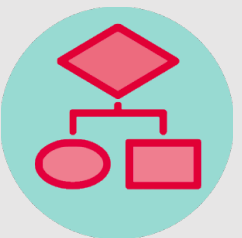
Exemple de fonctions (il en existe une multitude) :

❏ **Ent(nombre)** : récupère la partie entière d'un nombre

**Ent(3,1345) {vaut 3}**

❏ **Mod(chiffre1,chiffre2)** : récupère le reste de la division d'un nombre par un deuxième nombre.

**Mod(10,3) {vaut 1 car  $10 = 3*3+1$ }**



# Questions ?

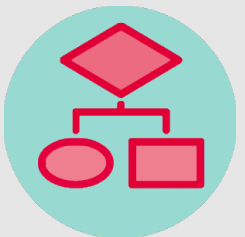


# Les fonctions personnalisées

❖ Une application, surtout si elle est longue, a toutes les chances de devoir procéder aux **mêmes traitements** à plusieurs endroits de son déroulement.

❖ Par exemple :

❖ la saisie d'une réponse par oui ou par non (et le contrôle qu'elle implique), peuvent être **répétés** dix fois à des moments différents de la même application.

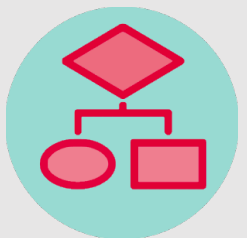




# Les fonctions personnalisées

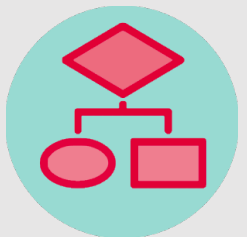
## ❏ Pourquoi utiliser des fonctions ? :

- ❏ Éviter d'avoir un **code lourd**, rempli de répétitions et de réutilisations de lignes de codes
- ❏ Améliorer la **lisibilité** du code
- ❏ Améliorer la **maintenance** et **l'évolutivité** du code
- ❏ **Factoriser** le code



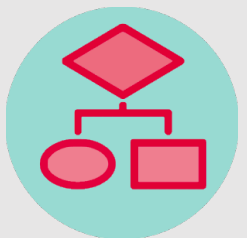
# Les fonctions personnalisées

- ❖ L'utilisation des fonctions consiste à séparer le traitement du corps du programme et à regrouper les instructions en un module séparé.
- ❖ Il ne restera alors plus qu'à appeler ce groupe d'instructions (qui n'existe donc désormais qu'en un exemplaire unique) à chaque fois qu'on en a besoin.
- ❖ Ainsi, la lisibilité est assurée ; le programme devient **modulaire**, et il suffit de faire une seule modification au bon endroit, pour que cette modification prenne effet dans la totalité de l'application.



# Les fonctions personnalisées

- ❖ Le corps du programme s'appelle alors la **procédure principale**, et ces groupes d'instructions auxquels on a recours s'appellent des **fonctions** et des **sous-procédures**
- ❖ Une fonction s'écrit toujours **en-dehors de la procédure principale**



# Les fonctions personnalisées

Exemple d'une mauvaise structure :

...

**Afficher**(« Etes-vous marié ? »)

rep1 ← « »

**Tant que** rep1 != «Oui» ou rep1 != «Non» **faire**

**Afficher**(« Tapez Oui ou Non »)

**Saisir**(rep1)

**ftq**

...

**Afficher**(« Avez-vous des enfants ? »)

rep2 ← « »

**Tant que** rep2 != «Oui» ou rep2 != «Non» **faire**

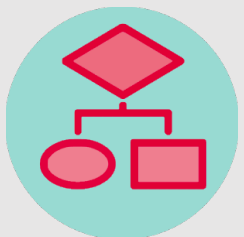
**Afficher**(« Tapez Oui ou Non »)

**Saisir**(rep2)

**Ftq**



Répétition quasi identique du traitement à accomplir. A chaque fois, on demande une réponse par Oui ou pas Non.  
La seule chose qui change, est l'intitulé de la question et le nom de la variable



# Les fonctions personnalisées

Exemple d'une bonne structure :

**Fonction RepOuiNon() : chaine de caractères**

**truc** ← « »

**Tant que** truc != «Oui» ou truc != «Non» **faire**

Afficher(« Tapez Oui ou Non)

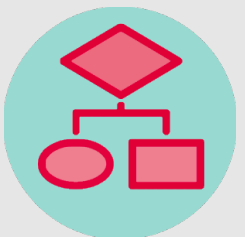
Saisir(truc)

**Ftq**

**Renvoyer** truc

**Fin Fonction**

Le mot-clé **Renvoyer** indique quelle valeur doit prendre la fonction lorsqu'elle est utilisée dans le programme.



# Les fonctions personnalisées

## Exemple d'une bonne structure :

...

**Afficher**(« Etes-vous marié ? »)

rep1 ← RepOuiNon()

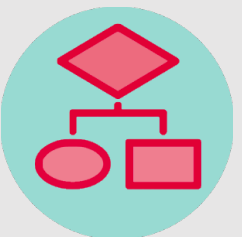
...

**Afficher**(« Avez-vous des enfants ? »)

rep2 ← RepOuiNon()

...

Utilisation de parenthèse obligatoire même si il n'y a pas de paramètres.



# Les fonctions personnalisées

Exemple d'une bonne structure (avec des paramètres) :

**Fonction** RepOuiNon(**msg** : chaine de caractères) : chaine de caractères

Afficher(**msg**)

truc ← « »

**Tant que** truc != «Oui» ou truc != «Non» **faire**

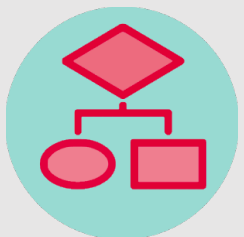
Afficher(« Tapez Oui ou Non)

Saisir(truc)

**Ftq**

Renvoyer truc

**Fin Fonction**



# Les fonctions personnalisées

Exemple d'une bonne structure (avec des paramètres) :

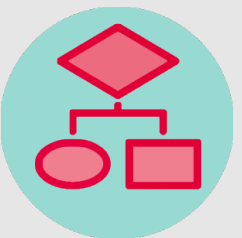
...

rep1 ← RepOuiNon(« Etes-vous marié ? »)

...

rep2 ← RepOuiNon(« Avez-vous des enfants ? »)

...



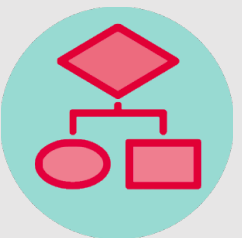


# Questions ?



# Les sous-procédures

- ❖ Une sous-procédure est une fonction qui ne renvoie aucune valeur, ou qui en renvoie plusieurs.
- ❖ L'instruction « **renvoyer** » n'est jamais utilisée dans une sous-procédure



# Les sous-procédures

## Déclaration d'une sous-procédure :

**Procédure** maProcédure(...)

...

**Fin Procédure**

## Appel à la sous-procédure :

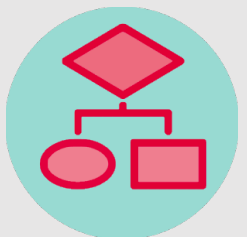
**Appeler** maProcédure(...)

Nom de la sous-procédure

Traitement de la sous-procédure

Arguments ou paramètres d'entrées de la sous-procédure

Appel de la sous-procédure



# Les sous-procédures

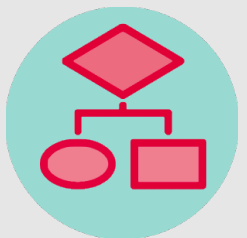
❖ Dans une procédure, les paramètres peuvent être :

❖ Des paramètres **d'entrée**

OU

❖ Des paramètres **de sorties**

Comment faire comprendre à un langage quels sont les paramètres qui doivent fonctionner en entrée et quels sont ceux qui doivent fonctionner en sortie ?

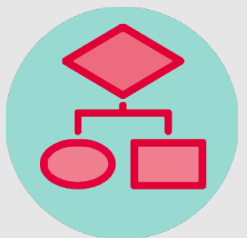


# Les sous-procédures

Comment faire comprendre à un langage quels sont les paramètres qui doivent fonctionner en entrée et quels sont ceux qui doivent fonctionner en sortie ?

❏ Il faut pour cela indiquer le **mode de passage** :

- ❏ Le passage **par valeur**
- ❏ Le passage **par référence**



# Les sous-procédures

Exemple d'une sous-procédure à qui on fournirait une chaîne de caractères et qui devrait extraire (séparément) le premier et le dernier caractère.

**Procédure** FirstLast(msg : chaîne de caractère **par valeur**, prems : ..., dern: ...)

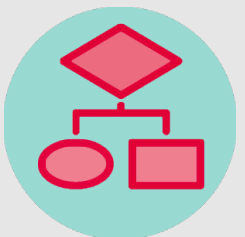
...

??? Left(msg, 1)

??? Right(msg, 1)

...

**Fin procédure**



# Les sous-procédures

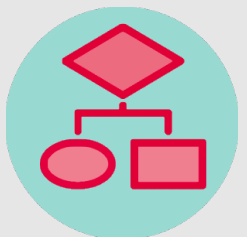
Exemple d'une sous-procédure à qui on fournirait une chaîne de caractères et qui devrait extraire (séparément) le premier et le dernier caractère.

**Variables**      aMouliner : **chaîne de caractères**

alpha, omega : **caractère**

aMouliner ← « Bonjour »

**Appeler** FirstLast(aMouliner, alpha, omega)

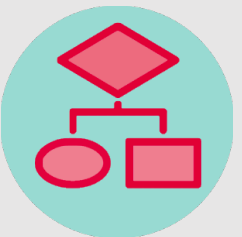


# Les sous-procédures

❏ msg : paramètre passé **par valeur** → msg est une copie de aMouliner

Un paramètre passé par valeur ne peut être qu'un paramètre d'entrée

❏ La valeur de la variable transmise (aMouliner) ne sera jamais modifiée





# Les sous-procédures

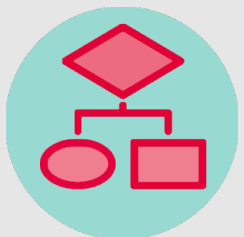
Exemple d'une sous-procédure à qui on fournirait une chaîne de caractères et qui devrait extraire (séparément) le premier et le dernier caractère.

**Procédure** FirstLast(msg : chaîne de caractère **par valeur**, prems : caractère **par référence**, dern : caractère **par référence**)

prems ← Left(msg, 1)

dern ← Right(msg, 1)

**Fin procédure**



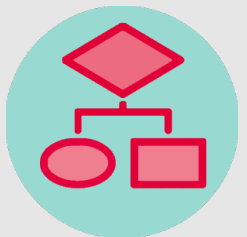
# Les sous-procédures

✚prems et dern : paramètre passé **par référence** → sont des pointeurs vers les adresses des variables alpha et omega.

Toute affectation d'une variable considérée comme un pointeur se traduit automatiquement par la modification de la variable sur laquelle elle pointe

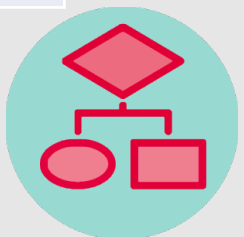
✚La valeur des variables transmises (prems et dern) est donc modifiée.

Passer un paramètre par référence permet d'utiliser ce dernier en lecture (en entrée) et en écriture (en sortie)



# Les sous-procédures

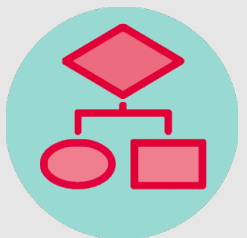
	Passage par valeur	Passage par référence
Utilisation en entrée	Oui	Oui
Utilisation en sortie	Non	Oui



# Pour résumer

❖ Une application bien programmée est une application à **l'architecture claire**, dont les différents modules font ce qu'ils disent, disent ce qu'il font, et **peuvent être testés (ou modifiés)** un par un **sans perturber le reste de la construction**.

1. **Regrouper sous forme de module distincts** tous les morceaux de code qui possèdent une certaine unité fonctionnelle (programmation par « blocs »)
2. Faire de ces modules **des fonctions lorsqu'ils renvoient un résultat unique**, et des **sous-procédures dans tous les autres cas**

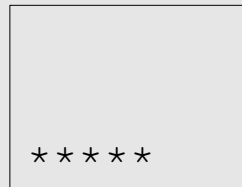


# Questions ?



# TP : boucles

- Demander un entier positif  $n$ .
- Dessiner une ligne d'étoile, un carré d'étoiles et un triangle d'étoiles.
  - Exemple  $n = 5$  :



```
* * * * *
```



```
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```



```
*  
* *  
* * *  
* * * *  
* * * * *
```

- Permettre de choisir, via un menu, quel dessin on souhaite