RÉSUMÉ THÉORIQUE – FILIÉRE DÉVELOPPEMENT DIGITAL

M102 - Acquérir les bases de l'algorithmique

120 heures

SOMMAIRE

01 MODÉLISER UN PROBLÈME

Analyser un problème

Identifier les approches d'analyse d'un problème

02 FORMULER UN TRAITEMENT

Reconnaitre la structure d'un algorithme

Connaitre les bases

Structurer un algorithme

Structurer les données

03 PROGRAMMER EN PYTHON

Transformer une suite d'étapes algorithmique en une suite d'instructions Python

Manipuler les données

04 DÉPLOYER LA SOLUTION PYTHON

Débloquer le code Python

Déployer une solution Python

MODALITÉS PEDAGOGIQUES

1

Le guide de soutien

Il s'agit du résumé théorique et du manuel des travaux pratiques.

2

La version PDF

Une version PDF du guide de soutien est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life.

3

Des ressources téléchargeables

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life

4

Du contenu interactif

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life.

5

Des ressources en lignes

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage

Copyright - Tout droit réservé - OFPPT

PARTIE 1
MODELISER UN PROBLEME

Dans ce module, vous allez :

- Comprendre comment analyser un problème
- Différencier les approches d'analyse d'un problème

6 heures

OFPPT
La Voie de l'Avenir

CHAPITRE 1
ANALYSER UN PROBLÈME

Ce que vous allez apprendre dans ce chapitre:

- Acquérir une compréhension de la méthode d'analyser d'un problème
- Identifier les entrées/ sorties d'un problème
- Comprendre les différents types de traitement de données

3 heures

OFPPT
La Voie de l'Avenir

CHAPITRE 1
ANALYSER UN PROBLÈME

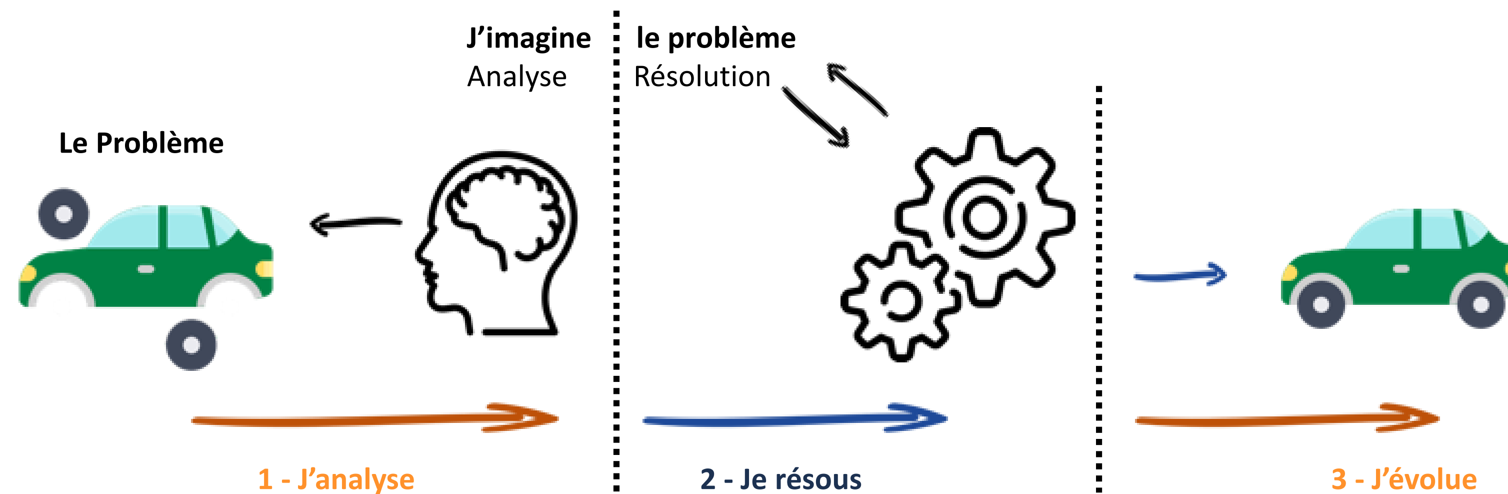
1- Définition du problème (Contexte, Entrées/Sorties, traitements)

2- Types de traitement des données

Définition du problème

Le développement d'un logiciel est la transformation d'une idée ou d'un besoin (problème) en un logiciel fonctionnel.
Le processus de résolution d'un problème peut être décrit en 3 phases :

- Analyse du problème
- Résolution du problème (conception et réalisation de la solution)
- Evaluation de la solution



Copyright - Tout droit réservé - OFPPT

Un problème peut se définir comme une question à résoudre, qui prête à discussion.

Il est soumis à des critères bien définis

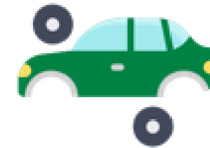
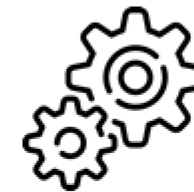
Exemple 1 :

Un magasin d'électroménager contient 380 aspirateurs

Il s'approvisionne de 40 autres appareils
et fait 3 ventes de 5 aspirateurs chacune.

Quel est le stock actuel ?

Un problème peut aussi se définir par un écart entre ce qui est, et ce qui devrait ou pourrait être.

Entrée **Algorithme** **Résultat**

Il est important d'apprendre à identifier le problème parce que :

- c'est la première étape cruciale du processus de la modélisation d'un problème;
- cela aide à clarifier et à préciser les problèmes pour qu'ils puissent être résolus efficacement
- cela peut aider à comprendre que tous ont une logique de résolution.

Copyright - Tout droit réservé - OFPPT

01- ANALYSER UN PROBLÈME

Définition du problème

L'analyse d'un problème est une étape préalable indispensable dans le processus de résolution de problème.

En effet, Un problème bien analysé est un problème à moitié résolu.

L'analyse des problèmes s'intéresse aux éléments suivants :

- Les résultats souhaités (sorties),
- Les traitements (actions réalisées pour atteindre le résultat),
- Les données nécessaires aux traitements (entrées)

L'objectif de cette étape est de :

- Bien comprendre l'énoncé du problème,
- Déterminer les dimensions du problème (entrées et sorties),
- Déterminer la méthode de de sa résolution par décomposition et raffinements successifs,
- Déterminer les formules de calculs, les règles de gestion, … etc

Le problème posé est souvent en langue naturelle et comporte plusieurs ambiguïtés d'où la nécessité de :

- Lecture entière et itérative pour comprendre et délimiter le problème à résoudre.
- Reformulation du problème sous la forme d'une question ou bien en utilisant un modèle mathématique
- il est impératif de relire ensuite le sujet pour bien vérifier qu'il n'y manque rien d'important.

Copyright - Tout droit réservé - OFPPT

Bien comprendre le problème à résoudre :

Exemple 2 :

On se donne un ensemble d'entiers positifs, on souhaite calculer la moyenne ('K') de ces entiers.

Quelle est la moyenne de 'K' entiers positifs ?

***Conseils* :**

- Se consacrer entièrement à la compréhension du sujet
- Eviter de chercher des idées de résolution (le comment faire ?)
- Identifier les données d'entrée et les résultats attendus

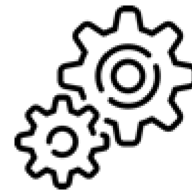
Copyright - Tout droit réservé - OFPPT

Bien comprendre le problème à résoudre

***Données d’entrée* :**

- Concernent le jeu de données sur lequel on souhaite résoudre le problème
- Représente l’ensemble des données en entrée, à partir desquelles on doit calculer puis afficher un résultat

Entrée



Algorithme

Résultat

***Conseils* :**

- Décrire précisément et d'avoir bien en tête les valeurs qu'elles peuvent prendre.

Copyright - Tout droit réservé - OFPPT

Exemple 1 :

Un magasin d'électroménager contient 380 aspirateurs

Il s'approvisionne de 40 autres appareils
et fait 3 ventes de 5 aspirateurs chacune.

Quel est le stock actuel ?

Entrées :
- Le nombre des entiers
- Les entiers positifs

Exemple 2 :

On se donne un ensemble d'entiers positifs, on souhaite calculer la moyenne ('K') de ces entiers.

Entrées :
- Valeur du stock initial
- Quantité d'approvisionnement
- Quantités vendues

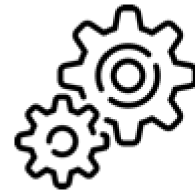
Copyright - Tout droit réservé - OFPPT

Bien comprendre le problème à résoudre :

Les résultats ou sorties :

Ils correspondent à ce que l'on demande de calculer ou de déterminer pour pouvoir obtenir le résultat

Entrée

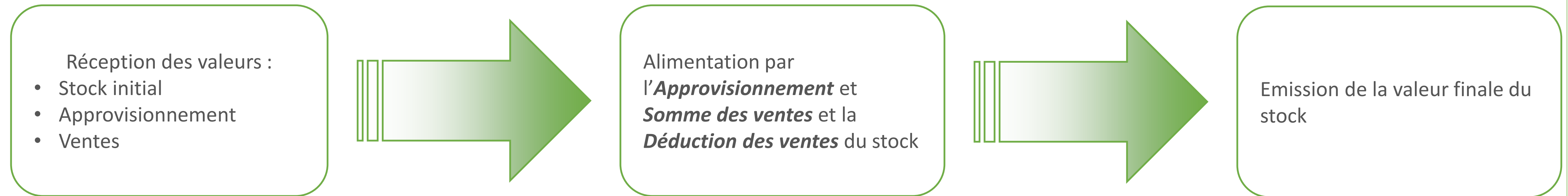


Algorithme

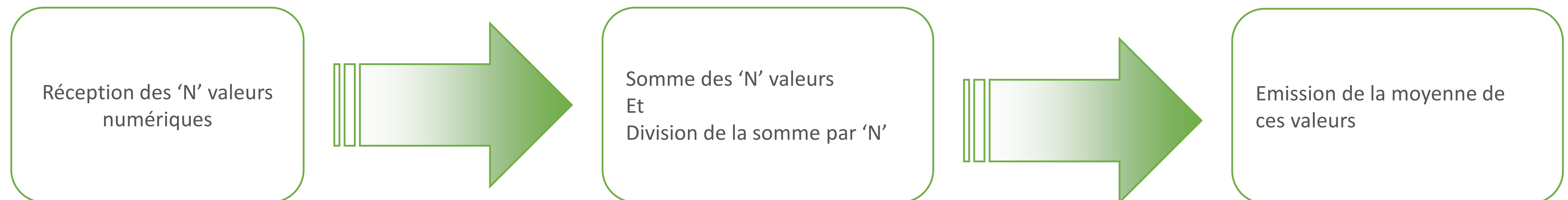
Résultat

Copyright - Tout droit réservé - OFPPT

Exemple 1 :



Exemple 2 :



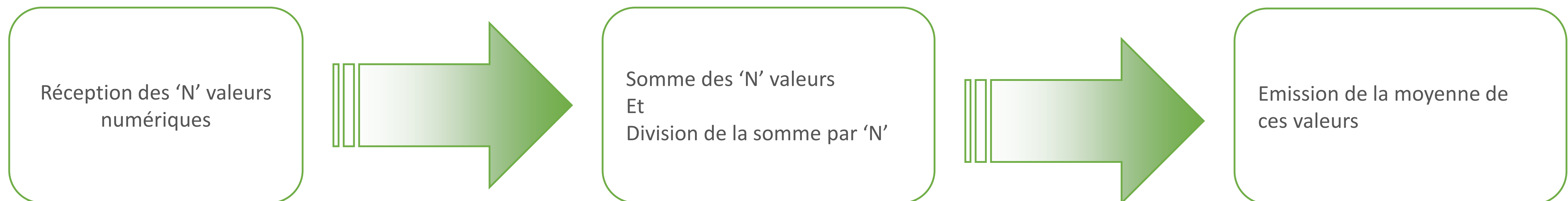
Copyright - Tout droit réservé - OFPPT

Le traitement des données

- L’analyse d’un problème se base aussi sur spécification de toutes les relations liant les résultats aux données et éventuellement les résultats entre eux
- La spécification des relations est la partie liée aux traitements à développer afin de résoudre le problème

- Le traitement est décrit à travers une suite finie et ordonnées de règles opératoires à suivre en vue de résoudre un problème.

Exemple 2 :



Le traitement des données est donc la formulation d’une solution imaginée par :

- Analogie: recherche des ressemblances, des rapprochements à partir d'idées déjà trouvées pour un problème précédent plus ou moins similaire.
- Contraste: recherche des différences, des oppositions, des arguments antagonistes.
- Contigüité: recherche des faits se produisant en même temps, des parallélismes, des simultanéïtés et autres concomitances.

Il est nécessaire d’avoir du bon sens, d’adopter une démarche rigoureuse et d’utiliser des outils adaptés

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
ANALYSER UN PROBLÈME

1- Définition du problème (Contexte, Entrées/Sorties, traitements)

2- Types de traitement des données

Le traitement des données

- Tout traitement est effectué par l'exécution séquencée d'opérations appelées ***instructions***.
- Selon la nature du problème, un traitement est classé en 4 catégories:

Traitement séquentiel	Traitement conditionnel
Traitement itératif	Traitement récursif

Copyright - Tout droit réservé - OFPPT

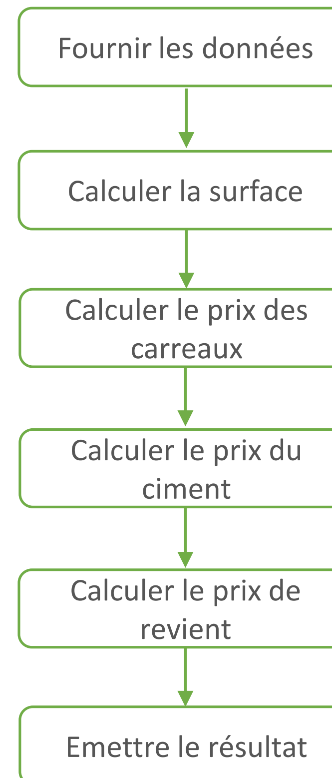
Le traitement séquentiel

Le traitement est décrit à travers l'enchaînement d'une suite d'actions primitives.

La séquence des actions sera exécutée dans l'ordre

Traitement séquentiel | Traitement conditionnel | Traitement itératif | Traitement récursif

Exemple :



Copyright - Tout droit réservé - OFPPT

Le traitement conditionnel

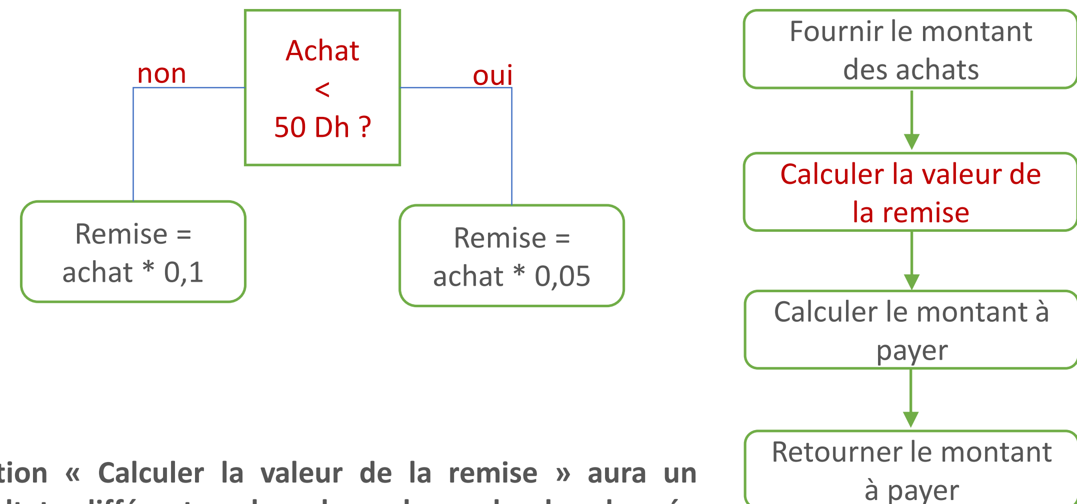
Le traitement est utilisé pour résoudre des problèmes dont la solution ne peut être décrite par une simple séquence d'actions mais **implique un ou plusieurs choix entre différentes possibilités**.

Traitement séquentiel	Traitement conditionnel
Traitement itératif	Traitement récursif

Exemple :

Un magasin accorde une remise sur les achats de ses clients.

Le taux de la remise est de 5% si le montant de l'achat est inférieur à 50Dh
Le taux de la remise est de de 10% si le montant dépasse 50Dh.
Connaissant le montant d'achat d'un client on souhaite déterminer la valeur de la remise et calculer le montant à payer.



L'action « Calculer la valeur de la remise » aura un résultat différent selon la valeur de la donnée « montant de l'achat » = Traitement conditionnel

Copyright - Tout droit réservé - OFPPT

Le traitement itératif

L'analyse d'un problème peut révéler le besoin de répéter un même traitement plus d'une fois.
Recours à des outils permettant d'exécuter ce traitement un certain nombre de fois sans pour autant le réécrire autant de fois.

Traitement séquentiel	Traitement conditionnel
Traitement itératif	Traitement récursif

Exemple :

Considérons le problème qui consiste de calculer la somme de 10 entiers positifs donnés

Entrer un entier
Ajouter l'entier à la somme
Répéter 1 et 2 10 fois
Afficher le résultat

Copyright - Tout droit réservé - OFPPT

Le traitement récursif

Un problème peut être exprimé en fonction d'un ou de plusieurs sous-problèmes tous de même nature que lui mais de complexité moindre

Traitement séquentiel

Traitement conditionnel

Traitement itératif

Traitement récursif

Exemple :

Considérons le problème qui consiste à calculer la factorielle d'un entier N positif ou nul.

On peut formuler le problème de cette manière :

Si N > 0
    Factorielle (N) = N * Factorielle
Si N = 0
    Factorielle (N) = 1

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
IDENTIFIER LES APPROCHES D'ANALYSE D'UN PROBLÈME

Ce que vous allez apprendre dans ce chapitre:

- Différencier les différentes approches d'analyse d'un problème
- Les maitriser

3 heures

CHAPITRE 2
IDENTIFIER LES APPROCHES D'ANALYSE D'UN PROBLÈME

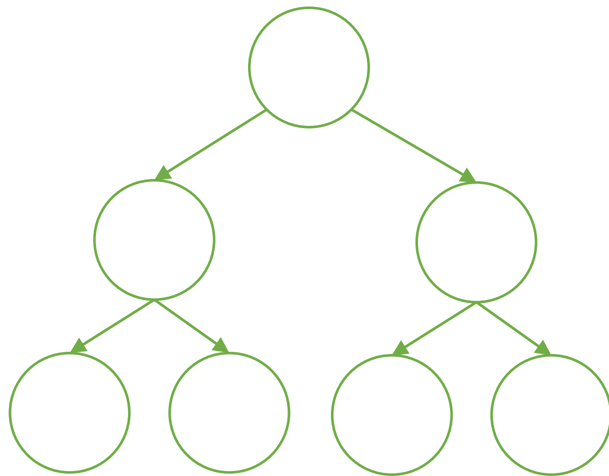
1- Approche descendante

2- Approche ascendante

Approche descendante

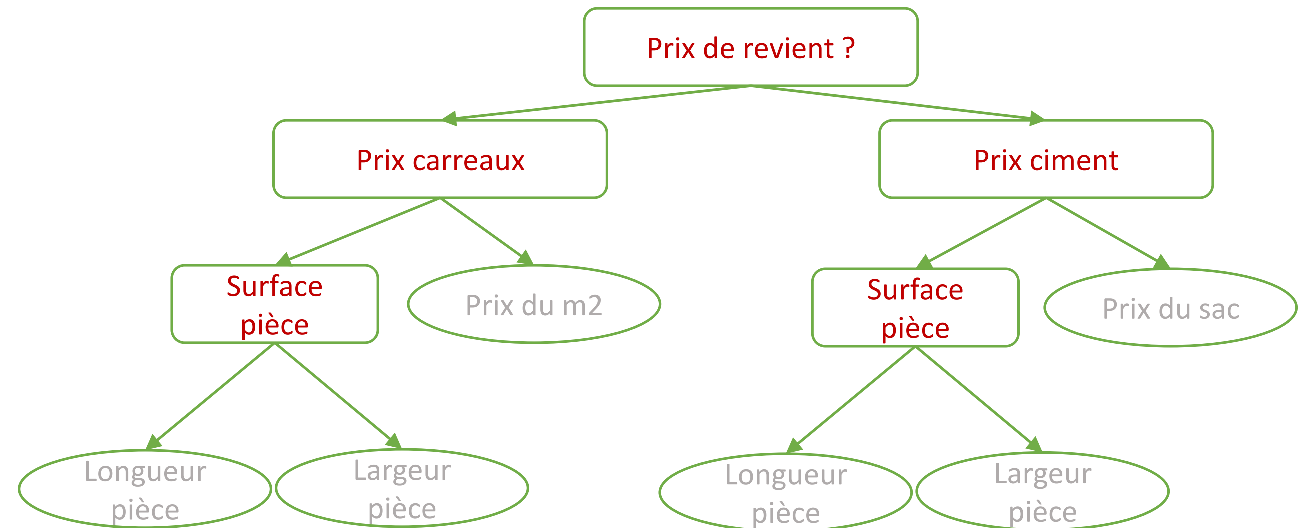
L'approche descendante divise un problème complexe en plusieurs parties plus simples et plus petites (modules) pour organiser le traitement de manière efficace

Ces modules sont ensuite décomposés jusqu'à ce que le module résultant constitue l'action primitive comprise et ne peut plus être décomposée



Exemple :

Une pièce rectangulaire de 4 sur 3 mètres doit être carrelée. Le carrelage d'un m² nécessite 1 sac de ciment. On cherche le prix de revient du carrelage de cette pièce sachant que le prix des carreaux est de 58 Dh / m² et le prix d'un sac de ciment est de 75 Dh.



Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
IDENTIFIER LES APPROCHES D’ANALYSE D’UN PROBLÈME

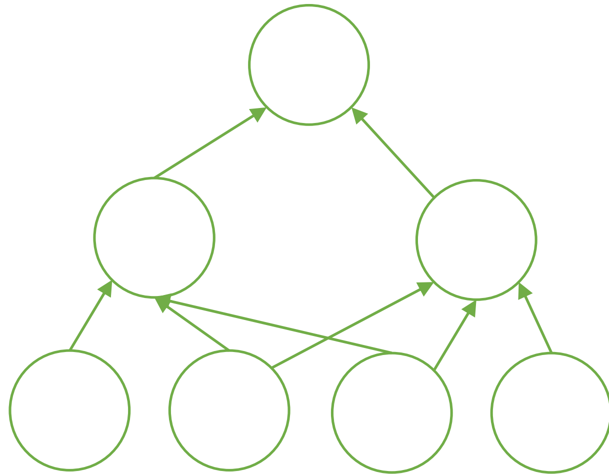
1- Approche descendante

2- Approche ascendante

Approche ascendante

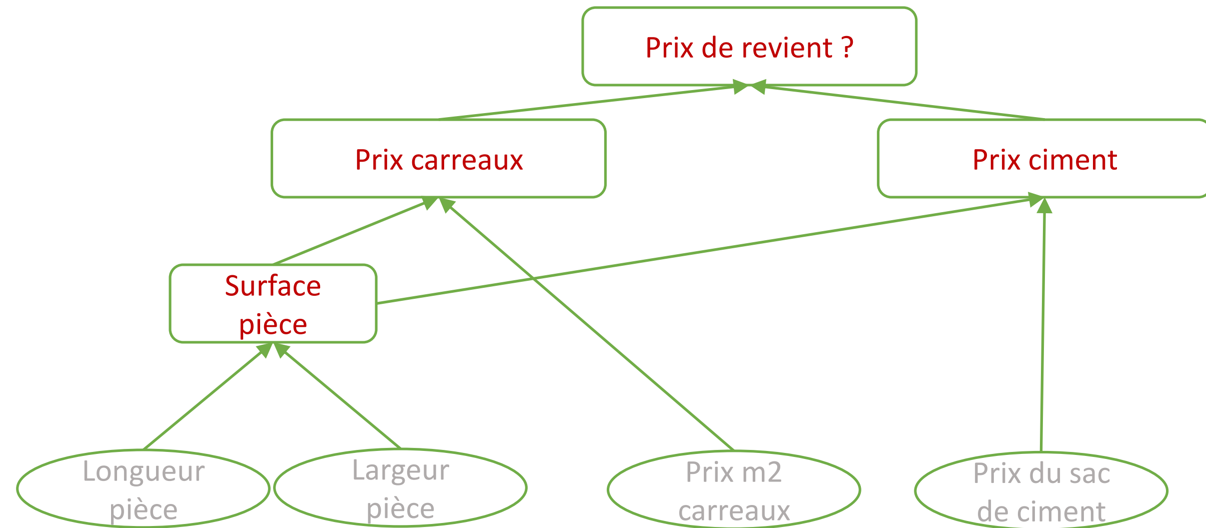
L'approche ascendante fonctionne de manière inverse, les actions primitives étant d'abord conçues puis poursuivis au niveau supérieur.

- Conception des pièces les plus fondamentales qui sont ensuite combinées pour former le module de niveau supérieur.
- Intégration de sous-modules et de modules dans le module de niveau supérieur est répétée jusqu'à l'obtention de la solution complète requise



Exemple :

Une pièce rectangulaire de 4 sur 3 mètres doit être carrelée. Le carrelage d'un m² nécessite 1 sac de ciment. On cherche le prix de revient du carrelage de cette pièce sachant que le prix des carreaux est de 58 Dh / m² et le prix d'un sac de ciment est de 75 Dh.



Copyright - Tout droit réservé - OFPPT

PARTIE 2
FORMULER UN TRAITEMENT

Dans ce module, vous allez :

- Maitriser la structure d'un algorithme
- Connaitre les différents types de traitement
- Maitriser la programmation structurée
- Manipuler les structures de données

48 heures

OFPPT
La Voie de l'Avenir

CHAPITRE 1
RECONNAITRE LA STRUCTURE D’UN ALGORITHME

Ce que vous allez apprendre dans ce chapitre :

- Comprendre la notion d’algorithme
- Différencier la notion de variable et de constante
- Connaitre les différents types d’objets informatiques
- Maitriser la structure d’un algorithme

7 heures

CHAPITRE 1
RECONNAITRE LA STRUCTURE D'UN ALGORITHME

1-Définition d'un algorithme

2- Objets informatiques (variable, constante, type)

3-Structure d'un algorithme

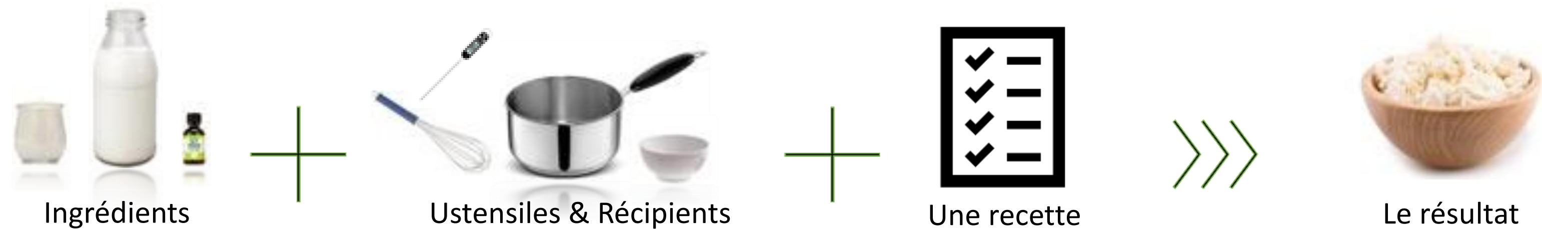
Définition d’un algorithme

OFPPT

Un algorithme est une ***suite d'instructions détaillées*** qui, si elles sont ***correctement exécutées***, conduit à un **résultat donné**.

"détaillées" signifie que les instructions sont suffisamment précises pour pouvoir être mises en œuvre correctement par l'exécutant (homme ou machine)

En algorithmique, nous utiliserons un langage situé à mi-chemin entre le langage courant et un langage de programmation appelé pseudo-code.



Explication : Lorsque nous cuisinons, nous utilisons des ingrédients et ustensiles, dans un ordre précis qui est régis par notre recette (une liste d’instruction dans un ordre donné), afin d’obtenir un résultat précis

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
RECONNAITRE LA STRUCTURE D'UN ALGORITHME

1-Définition d'un algorithme

2- Objets informatiques (variable, constante, type)

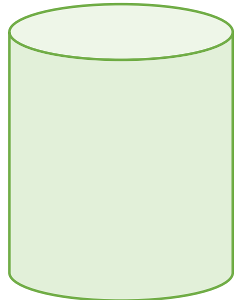
3-Structure d'un algorithme

01- STRUCTURE D’UN ALGORITHME

Objets informatiques (variable, constante, type)

Un algorithme manipule des objets (données) pour obtenir un résultat.

- ***Un objet est composé de*** :
  - Un identificateur (son nom) : pour le désigner. Celui-ci doit être parlant.
  - Un type : pour déterminer la nature de l’objet simple (entier, caractère, ect…) ou composé (tableau,…)
    - Un type détermine en particulier les valeurs possibles de l'objet, la taille mémoire réservée à l’objet et les opérations primitives applicables à l'objet.
  - Une valeur : détermine le contenu unique de l’objet



Un objet

Un objet composé d’un identificateur

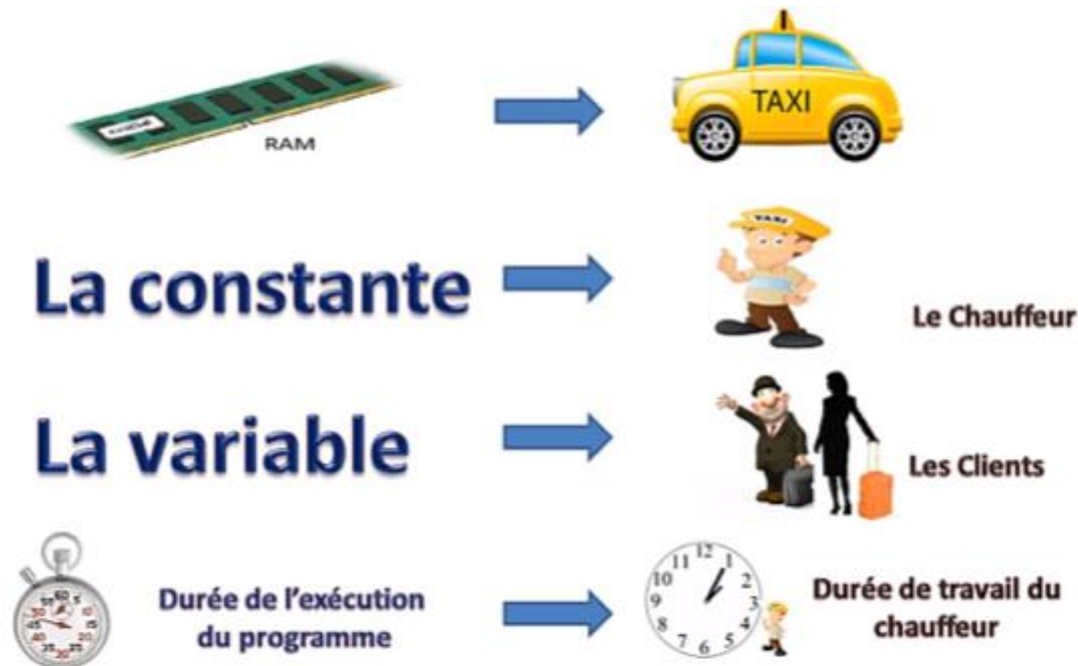
Un objet composé d’un identificateur + type

Un objet composé d’un identificateur + type + valeur

Copyright - Tout droit réservé - OFPPT

Les objets sont de deux types: les constantes et les variables.

- Une **constante** est un objet dont l'état **reste inchangé** durant toute l'exécution d'un programme. On ne peut jamais modifier sa valeur et celle-ci doit donc être précisée lors de la définition de l'objet.
- Une **variable** est un objet dont le contenu (sa valeur) **peut être modifié** par une action
- Exemple:



Copyright - Tout droit réservé - OFPPT

Types des objets

- A chaque variable utilisée dans le programme, il faut associer un type qui permet de définir :
  - l'ensemble des valeurs que peut prendre la variable
  - l'ensemble des opérations qu'on peut appliquer sur la variable
- Les principaux types utilisés en algorithmique sont :
  - le type entier
  - le type réel
  - le type caractère
  - le type chaîne de caractères
  - le type logique ou booléen.

Type entier

- Une variable est dite entière si elle prend ses valeurs dans Z (ensemble des entiers relatifs)
- Elle peut supporter les opérations suivantes :

Opération	Notation
Addition	+
Soustraction	-
Multiplication	*
Division entière	div
Modulo (reste de la division)	mod

Exemples

13 div 5 = 2

13 mod 5 = 3

Copyright - Tout droit réservé - OFPPT

Type réel ou décimal

- Il existe plusieurs types de réels représentant chacun un ensemble particulier de valeurs prises dans R (ensemble des nombres réels).
- Il existe deux formes de représentation des réels :
  - la forme usuelle avec le point comme symbole décimal.
    - **Exemples**

    -3.2467 2 12.7 +36.49
  - la notation scientifique selon le format aEb, où : a est la mantisse, qui s’écrit sous une forme usuelle b est l’exposant représentant un entier relatif.
    - **Exemples** :

    347 = 3.47E2 = 0.347E+3 = 3470E-1

- Les opérations définies sur les réels sont :

Opération	Notation
Addition	+
Soustraction	-
Multiplication	*
Division (réelle)	/

Copyright - Tout droit réservé - OFPPT

Type caractère

- Un caractère peut appartenir au domaine des chiffres de ”0” à ”9”, des lettres (minuscules et majuscules) et des caractères spéciaux (”*”, ”/”, ”{”, ”$”, ”#”, ”%” …).
- Un caractère sera toujours noté entre des guillemets.
- Le caractère espace (blanc) sera noté ” ”.
- Les opérateurs définis sur les données de type caractère sont :

Opération	Notation
Égal	=
Différent	#
Inférieur	<
Inférieur ou égal	<=
Supérieur	>
Supérieur ou égal	>=

- La comparaison entre les caractères se fait selon leur codes ASCII
- **Exemple:**

  ” ” < ”0” < ”1” < ”A” < ”B” < ”a” < ”b” < ”{”

Copyright - Tout droit réservé - OFPPT

Type logique ou booléen

- Une variable logique ne peut prendre que les valeurs "Vrai" ou "Faux".
- Elle intervient dans l'évaluation d'une condition.
- Les principales opérations définies sur les variables de type logique sont : la négation (NON), l'intersection (ET) et l'union (OU).
- L'application de ces opérateurs se fait conformément à la table de vérité suivante :

Table de vérité des opérateurs logiques

A	B	NON (A)	A ET B	A OU B
Vrai	Vrai	Faux	Vrai	Vrai
Vrai	Faux	Faux	Faux	Vrai
Faux	Vrai	Vrai	Faux	Vrai
Faux	Faux	Vrai	Faux	Faux

Copyright - Tout droit réservé - OFPPT

Expressions

- Ce sont des combinaisons entre des variables et des constantes à l’aide d’opérateurs.
- Elles expriment un calcul (expressions arithmétiques) ou une relation (expressions logiques).

Les expressions arithmétiques:

- **Exemple** : x * 53.4 / (2 + pi)
- L’ordre selon lequel se déroule chaque opération de calcul est important.
- Afin d’éviter les ambiguïtés dans l’écriture, on se sert des parenthèses et des relations de priorité entre les opérateurs arithmétiques :

Ordre de priorité des opérateurs arithmétiques

Priorité	Opérateurs	
1	-	signe négatif (oprérateur unaire)
2	( )	parenthèses
3	^	puissance
4	* et /	multiplication et division
5	+ et -	addition et soustraction

- En cas de conflit entre deux opérateurs de même priorité, on commence par celui situé le plus à gauche

Copyright - Tout droit réservé - OFPPT

Expressions

Les expressions logiques

- Ce sont des combinaisons entre des variables et des constantes à l'aide d'opérateurs relationnels (=, <, <=, >, >=, #) et/ou des combinaisons entre des variables et des constantes logiques à l'aide d'opérateurs logiques (NON , ET, OU, etc).
- On utilise les parenthèses et l'ordre de priorité entre les différents opérateurs pour résoudre les problèmes de conflits.

Opérateurs logiques

Priorité	Opérateur
1	NON
2	ET
3	OU

Opérateurs relationnels

Priorité	Opérateur
1	>
2	>=
3	<
4	<=
5	=
6	#

- **Exemple:**

5 + 2 * 6 – 4 + (8 + 2 ^ 3) / (2 – 4 + 5 * 2) = 15

Copyright - Tout droit réservé - OFPPT

Déclaration d’une variable

- Toute variable utilisée dans un programme doit avoir fait l’objet d’une déclaration préalable.
- En pseudo-code, la déclaration de variables est effectuée par la forme suivante :

Var liste d'identificateurs : type

- **Exemple**

```
Var
    i, j, k : Entier x, y : Réel
    OK: Booléen
    C1, C2 : Caractère
```

Déclaration d’une constante

- En pseudo-code, la déclaration des constante est effectuée par la forme suivante :
- Par convention, les noms de constantes sont en majuscules
- Une constante doit toujours recevoir une valeur dès sa déclaration

Const identificateur=valeur : type

- **Exemple** Const PI=3.14 : réel

Pour calculer la surface des cercles, la valeur de pi est une constante mais le rayon est une variable

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
RECONNAITRE LA STRUCTURE D'UN ALGORITHME

1-Définition d'un algorithme

2- Objets informatiques (variable, constante, type)

3-Structure d'un algorithme

02- STRUCTURE D’UN ALGORITHME

Structure d’un algorithme

```
<NOM_ALGORITHME>

Const
      Const1= val1 : type
      Const2=val2 : type
………
Var
     v1 : type
     v2 : type
………
Début
     Instruction 1
     Instruction 2
     …..;
Fin
```

- Const … : Liste des constantes
- Var … : Liste des variables
- Début … : Corps de l’algorithme

```
Cercle

Const
pi = 3.14

Var
r, p, s : Réel

Début
   Ecrire(”Entrer le rayon du cercle : ”)
  Lire(r)
  p := 2 * pi * r
  s :=pi * r ^ 2
  Ecrire (”Périmètre = ”, p)
  Ecrire (”Surface = ”, s)
Fin.
```

Copyright - Tout droit réservé - OFPPT

OFPPT
La Voie de l'Avenir

CHAPITRE 2

RECONNAITRE LES BASES

Ce que vous allez apprendre dans ce chapitre :

- Maitriser les instructions d'affectation et les instructions d'entrée/Sortie
- Reconnaitre les différents types de traitement des instructions dans un algorithme

15 heures

OFPPT
La Voie de l'Avenir

CHAPITRE 2

RECONNAITRE LES BASES

1-Traitement séquentiel (affectation, lecture et écriture)

2-Traitement alternatif(conditions)

3-Traitement itératif (boucles)

Instruction d'affectation

- L'affectation consiste à attribuer une valeur à une variable (c'est-à-dire remplir ou modifier le contenu d'une zone mémoire)
- En pseudo-code, l'affectation est notée par le signe **:=**

Var:= e : attribue la valeur de e à la variable Var

  - **e** peut être une valeur, une autre variable ou une expression
  - **Var et e** doivent être de même type ou de types compatibles

- L'affectation ne modifie que ce qui est à gauche de la flèche

Exemple:

  - l'instruction : A := 6 signifie « mettre la valeur 6 dans la case mémoire identifiée par A ».
  - l'instruction : B := (A + 4) Mod 3 range dans B la valeur 1 (A toujours égale à 6).

- La valeur ou le résultat de l'expression à droite du signe d'affectation doit être de même type ou de type compatible avec celui de la variable à gauche.

Copyright - Tout droit réservé - OFPPT

Instruction de lecture

- Les instructions de lecture et d'écriture (Entrée/Sortie) permettent à la machine de communiquer avec l'utilisateur
- La lecture permet d'entrer des donnés à partir du clavier.
- En pseudo-code, on note :

```
lire (var)
```

- La machine met la valeur entrée au clavier dans la zone mémoire nommée **var.**
- Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la frappe d'une valeur au clavier et de la touche **Entrée.**

Instruction d'écriture

- L'écriture permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)
- En pseudo-code, on note :

```
Ecrire (var)
```

- La machine affiche le contenu de la zone mémoire **var**

Exemple :

Ecrire(a, b+2, "Message")

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2

RECONNAITRE LES BASES

1-Traitement séquentiel (affectation, lecture et écriture)

2-Traitement alternatif(conditions)

3-Traitement itératif (boucles)

OFPPT

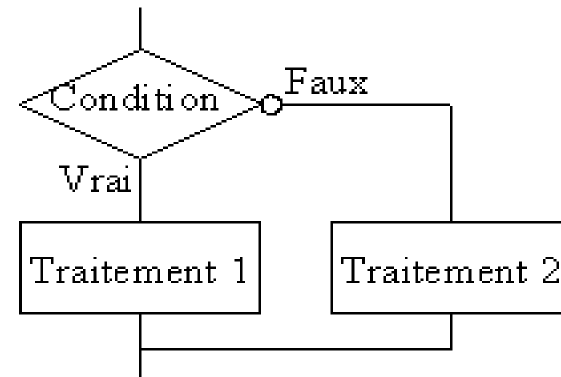
Traitement alternatif

Rappel :

- Les instructions conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.
- **Syntaxe: forme simple**

```
Si <Condition> Alors
        <Séquence d'instructions>
FinSi
```

- Cette primitive a pour effet d'exécuter la séquence d'instructions si et seulement si la condition est vérifiée.
- L'exécution de cette instruction se déroule selon l'organigramme suivant:



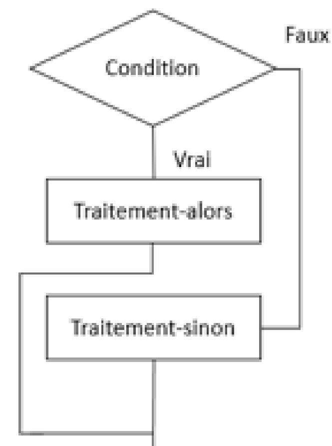
Copyright - Tout droit réservé - OFPPT

Traitement alternatif

- **Syntaxe: Forme alternative**

```
Si <Condition> Alors
        <Séquence d'instructions 1>
Sinon
        <Séquence d'instructions 2>
FinSi
```

- Cette primitive a pour effet d'exécuter la première séquence d'instructions si la condition est vérifiée ou bien la deuxième séquence d'instructions dans le cas contraire.
- L'exécution de cette instruction se déroule selon l'organigramme suivant:



Copyright - Tout droit réservé - OFPPT

Exemple 1

```
Si ( a≠0 )    alors
        a := 0
Sinon
        a := b
        c := d
Finsi
```

- Si la condition est vraie c'est à dire la variable **a** est différente de **0** alors on lui affecte la valeur **0**, sinon on exécute le bloc **sinon**.

Exemple 2

```
Si ( a – b ≠ c ) alors
        a := c
Sinon
        a := d
Finsi
```

- Si la condition est vraie, la seule instruction qui sera exécutée est l'instruction d'affectation **a : = c.**
- Sinon la seule instruction qui sera exécutée est l'instruction d'affectation **a : = d.**

Copyright - Tout droit réservé - OFPPT

Traitement alternatif

- **Syntaxe: Schéma conditionnel à choix multiple**

```
Cas <var> de:
        <valeur 1> : <action 1>
        < valeur 2> : <action 2>
        ...
        < valeur n> : <action n>
        Sinon : <action_sinon>
FinCas
```

la partie action-sinon est facultative

Exemple:

- On dispose d’un ensemble de tâches que l’on souhaite exécuter en fonction de la valeur d’une variable choix de type entier, conformément au tableau suivant :

Valeur de choix	Tâche à exécuter
1	Commande
2	Livraison
3	Facturation
4	Règlement
5	Stock
Autre valeur	ERREUR

Copyright - Tout droit réservé - OFPPT

Traitement alternatif

Forme alternative

```
Si choix = 1  alors
   Commande
sinon
   si    choix = 2 alors
         Livraison
   sinon
        si choix = 3  alors
              Facturation
        sinon
             si      choix = 4   alors
                       Règlement
-             sinon
                     si  choix = 5   alors
                            Stock
                     sinon
                            ecrire (''Erreur'')
                     finsi

               finsi
             finsi
        finsi
finsi
```

Schéma conditionnel à choix multiple

```
Cas choix de :
     1: Commande
     2: Livraison
     3: Facturation
     4: Règlement
     sinon  ecrire (''Erreur'')
finCas
```


Copyright - Tout droit réservé - OFPPT

CHAPITRE 2

RECONNAITRE LES BASES

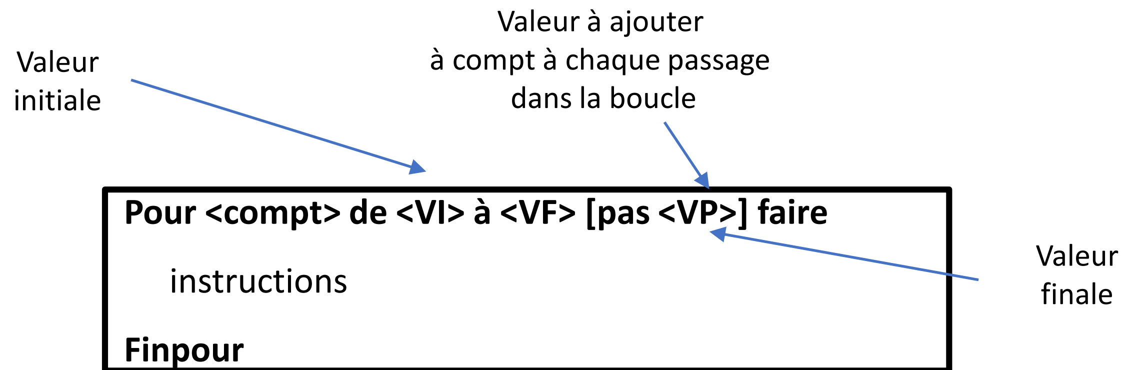
1-Traitement séquentiel (affectation, lecture et écriture)

2-Traitement alternatif(conditions)

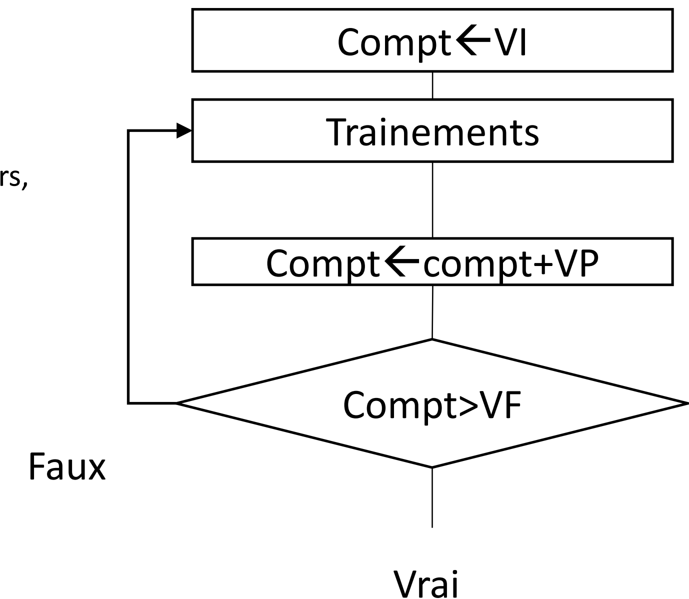
3-Traitement itératif (boucles)

Traitement itératif

- **Structure « Pour ……….Faire »**



- Le compteur (variable de contrôle) prend la valeur initiale au moment d'accès à la boucle puis, à chaque parcours, il passe automatiquement à la valeur suivante dans son domaine jusqu'à atteindre la valeur finale
- L'exécution de cette instruction se déroule selon l'organigramme suivant:



Copyright - Tout droit réservé - OFPPT

Traitement itératif

- **Structure « Pour………….Faire »**

Exemple : un algorithme permettant de lire N réels, de calculer et d'afficher leur moyenne

```
moyenne
var   n, i, x, s : réel
Début
        lire( n )
        s := 0
        Pour  i  de  1  à  n  faire
                lire( x )
                s := s + x
        Finpour
        ecrire( “la moyenne est :”, s / n )
Fin
```

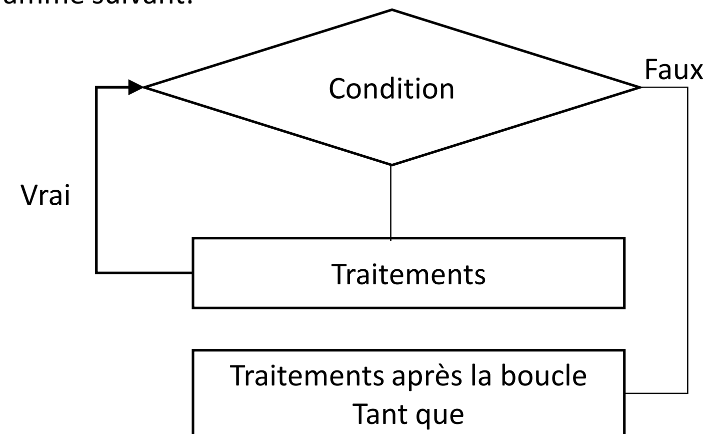
Copyright - Tout droit réservé - OFPPT

Traitement itératif

- **Structure « TantQue Faire »»**
  - Le traitement est exécuté aussi longtemps que la condition est vérifiée. Si dès le début cette condition est fausse, le traitement ne sera exécuté aucune fois.
  - Une boucle « tantQue » peut s'exécuter 0, 1 ou n fois

```
TantQue <condition> Faire
        <Séquence d'instructions>
FinTQ
```

  - L'exécution de cette instruction se déroule selon l'organigramme suivant:



Copyright - Tout droit réservé - OFPPT

Traitement itératif

- **Structure « TantQue Faire »»**

Exemple : un algorithme permettant de lire une suite de réels, de calculer et d'afficher leur moyenne.

```
moyenne
var    i, x, s : réel

Début

        lire( x )
        s := 0
        i := 0
        TantQue  x > 0  faire
                i := i + 1
                s := s + x
                lire( x )
        FinTQ
        si  i ≠ 0
          alors  écrire( “la moyenne est :”, s / i )
        finsi
Fin
```

Condition obligatoire pour éviter de diviser par 0 si le premier entier lu est 0

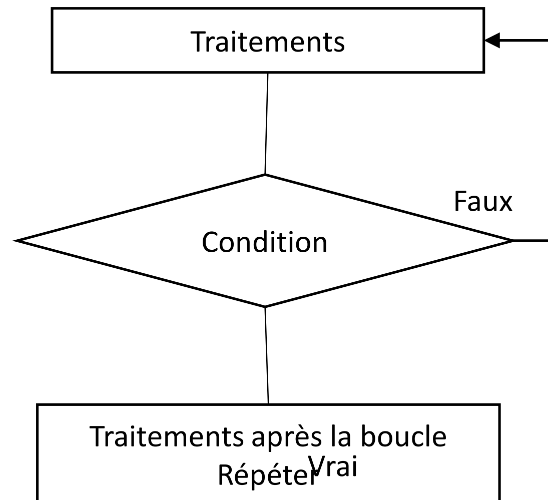
Copyright - Tout droit réservé - OFPPT

Traitement itératif

- **Structure « Répéter…..Jusqu' à »**
  - La séquence d'instructions est exécutée une première fois, puis l'exécution se répète jusqu'à ce que la condition de sortie soit vérifiée.
  - Une boucle « répéter » **s'exécute toujours au moins une fois**

```
Répéter
        <Séquence d'instructions>
Jusqu'à <condition>
```

- L'exécution de cette instruction se déroule selon l'organigramme suivant:



Traitement itératif

- **Structure « Répéter…..Jusqu' à »**

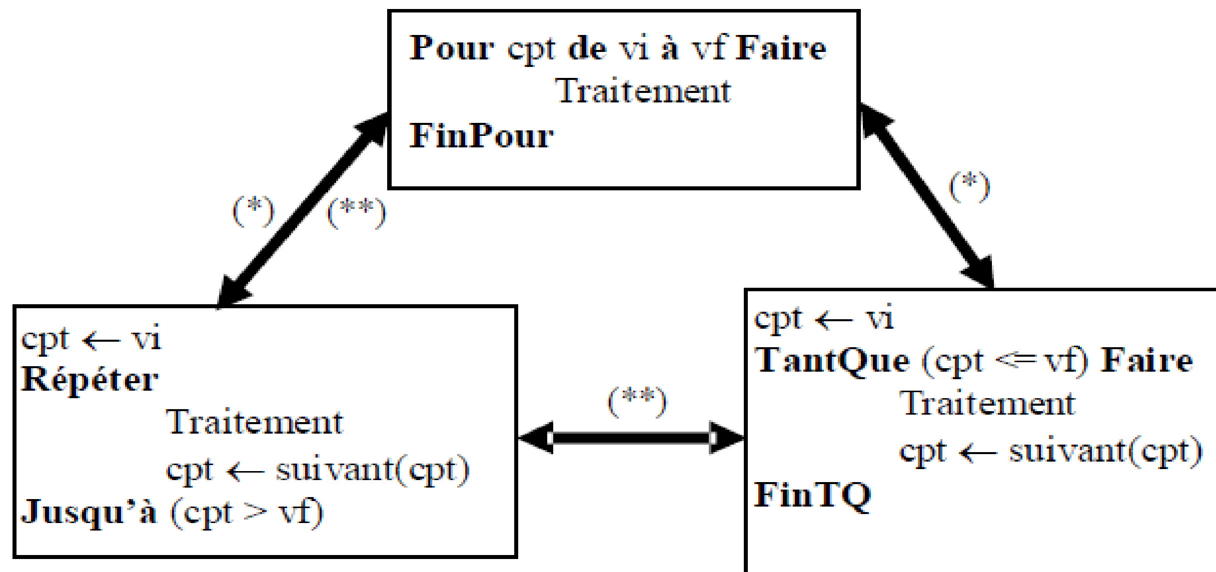
Exemple : un algorithme permettant de lire deux entiers, de calculer et d'afficher le résultat de la division du premier par le second (quotient)

```
quotient
var   x, y : entier

Début

        lire( x )
        répéter
            lire( y )
        jusqu'à  y > 0
        écrire( x / y )
Fin
```

un contrôle obligatoire doit être effectué lors de la lecture de la deuxième valeur

Copyright - Tout droit réservé - OFPPT

Passage d’une structure itérative à une autre



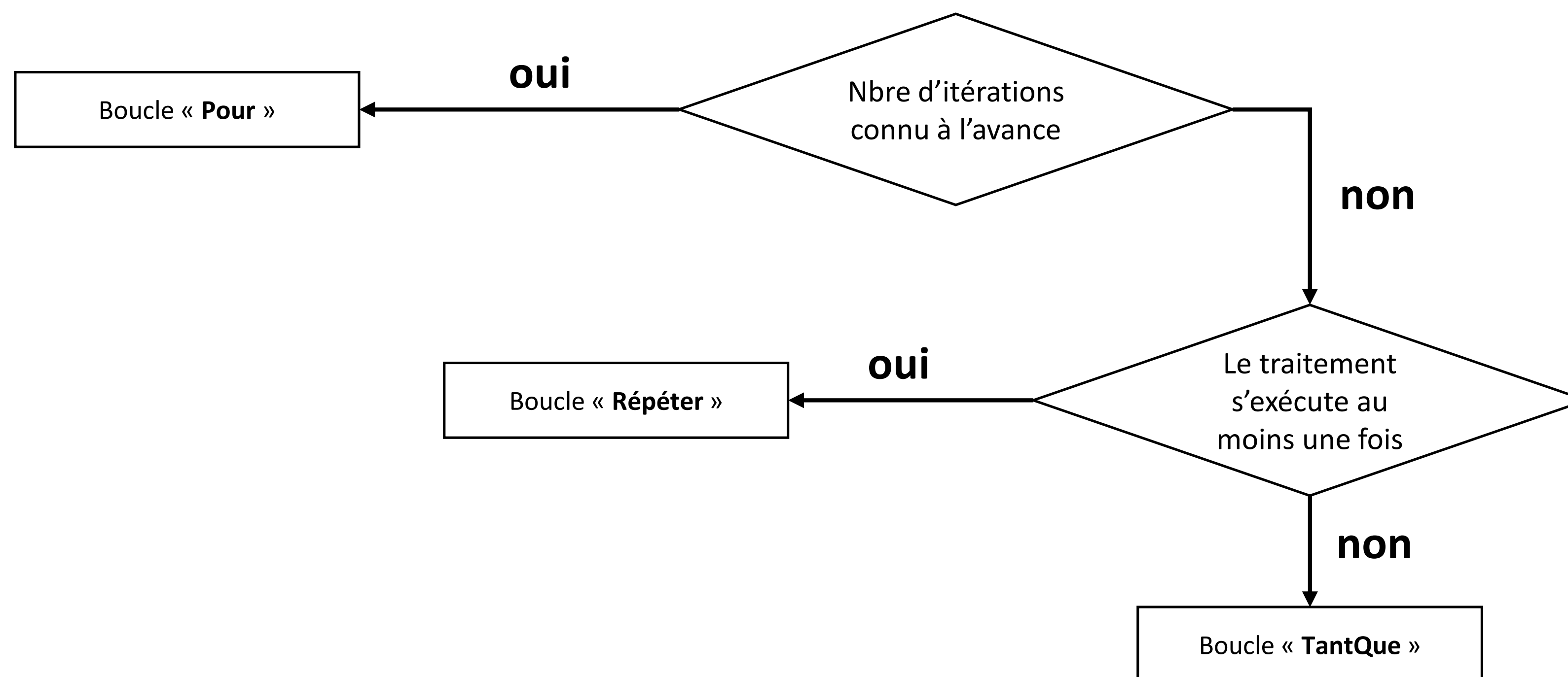
(*) : Le passage d’une boucle « répéter » ou « tantque » à une boucle « pour » n’est possible que si le nombre de parcours est connu à l’avance

(**) : Lors du passage d’une boucle « pour » ou « tantque » à une boucle « répéter », faire attention aux cas particuliers (le traitement sera toujours exécuté au moins une fois)

Copyright - Tout droit réservé - OFPPT

OFPPT

Choix de la structure itérative



Copyright - Tout droit réservé - OFPPT

OFPPT
La Voie de l'Avenir

CHAPITRE 3

STRUCTURER UN ALGORITHME

Ce que vous allez apprendre dans ce chapitre :

- Maîtriser la définition des procédures et des fonctions
- Maîtriser les notions de paramètre formel et paramètre effectif
- Définir les différents types de passage des paramètres
- Connaître la notion de variable locale et de variable globale

10 heures

CHAPITRE 3

STRUCTURER UN ALGORITHME

1-Procédures et Fonctions

2-Portée d'une variable

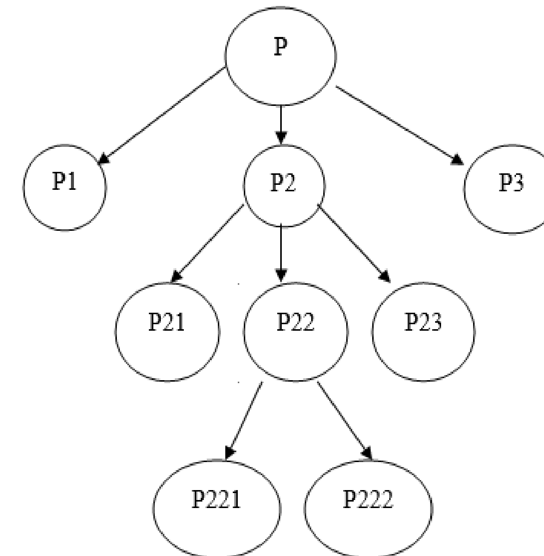
OFPPT

Programmation structurée

- La résolution d'un problème complexe peut engendrer des milliers de lignes de code :
  - Algorithme long
  - Algorithme difficile à écrire
  - Algorithme difficile à interpréter
  - Algorithme difficile à maintenir
- **Solution : utiliser une méthodologie de résolution:**

Programmation Structurée

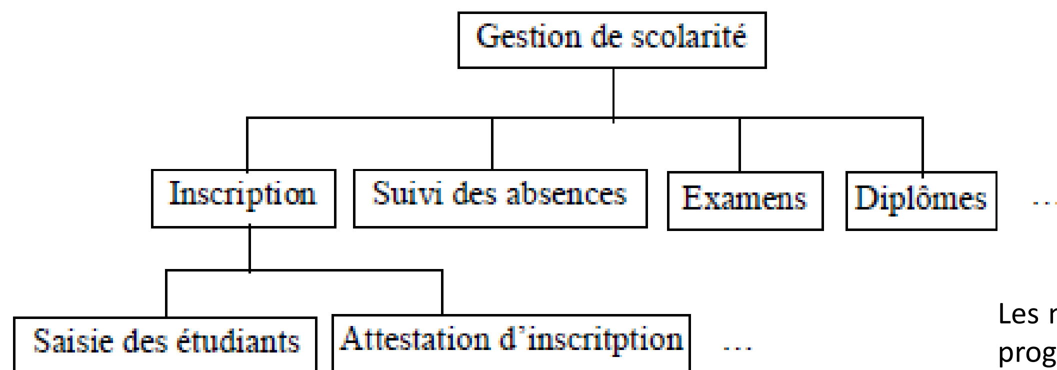
- **Idée :** Découpage d'un problème en des sous_problèmes moins complexes
- **Avantages**
  - clarté de l'algorithme
  - lisibilité de la lecture d'un algorithme
  - facilité de maintenance
  - réutilisation des sous algorithmes



Copyright - Tout droit réservé - OFPPT

Programmation structurée

Exemple : Un programme de gestion de scolarité peut être découpé en plusieurs modules : inscription, suivi des absences, examens, diplômes, etc



Décomposition d'un programme en sous-programmes

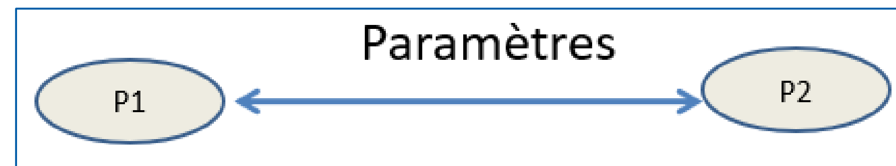
Les modules développés peuvent être réutilisés plusieurs fois dans le même programme ou dans d'autres programmes une fois intégrés à des bibliothèques.

Copyright - Tout droit réservé - OFPPT

OFPPT

Procédures et fonctions

- Deux types de sous_algorithmes/sous_programmes sont possibles :
    - procédure
    - fonction
- La Communication entre sous_ programmes se fait via des paramètres :



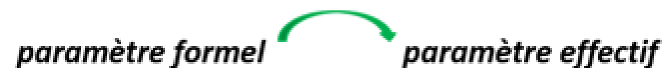
- Il existe 3 types de paramètres:
    - paramètres données : **les entrées**
    - paramètres résultats : **les sorties**
    - paramètres données/résultats : **à l'appel des données transformés par la procédure/fonction en résultats**

Copyright - Tout droit réservé - OFPPT

Paramètre formel/Paramètre effectif

- **Paramètres formels:** objets utilisés pour la description d'un sous_algorithme
- **Paramètres effectifs :** objets utilisés lors de l'appel d'un sous_algorithme
- **Un paramètre formel** est toujours une **variable**.
- **Un paramètre** effectif peut être :
    - une **variable**
    - une **constante**
    - une **expression arithmétique**
    - un **appel de fonction**

- Pour tout paramètre formel on fait correspondre un paramètre effectif.



- Le paramètre formel et le paramètre effectif correspondant doivent avoir le même type ou être de types compatibles.
- La correspondance entre paramètres formels et paramètres effectifs se fait selon l'ordre de leurs apparitions dans la définition et dans l'utilisation de la procédure ou la fonction.

Copyright - Tout droit réservé - OFPPT

Syntaxe Définition d’une procédure

- Pour définir une procédure on adoptera la syntaxe suivante

```
<Nom_proc> (<liste_par_form>)
  Var   <declaration_variables>
  Debut
      <Corps_procédure>
 Fin
```

<Nom_proc> : désigne le nom de la procédure.

<liste_par_form> : la liste des paramètres formels. Un paramètre résultat ou donnée/résultat doit être précédé par le mot clé var.

<declaration_varibales> : la liste des variables

<Corps_procédure> : la suite des instructions décrivant le traitement à effectuer

Copyright - Tout droit réservé - OFPPT

OFPPT

Syntaxe Définition d'une procédure

Exemple:

- La procédure suivante permet de lire N valeurs entières et de calculer la plus petite et la plus grande parmi ces N valeurs.
- Les entiers saisis doivent être supérieurs à 0 et inférieurs à 100.

  - Le nom de cette procédure est **Min_Max**
  - Les paramètres formels sont : l'entier N comme paramètre donné, les entiers min et max comme paramètres résultats (précédés par le mot clé var).
  - 2 variables locales de type entier : **i et x**

```
Min_Max (N : entier ; var min: entier, var max : entier)
Var   i, x : entier
Début
            min := 100
            max := 0
            pour  i  de  1  à  N  faire
                        Répéter
                          Lire ( x )
                        Jusqu'à  (x > 0) et (x < 100)
                        Si  x < min Alors
                            min := x
                        Finsi
                        Si  x > max Alors
                            max := x
                        Finsi
            Finpour
Fin
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Syntaxe Définition d'une fonction

- Pour définir une fonction, on adoptera la syntaxe suivante :

```
Nom_fonction>(<liste_par_form>) : <Type-fonction>

Var   <declarat_var_locales>
Début
  <Corps_fonction>
  retourner <valeur>
Fin
```

Nom_fonction>: désigne le nom de la fonction

<liste_par_form> : désigne la liste des paramètres formels de la fonction

<declarat_var_locales> définissent les mêmes concepts que pour la procédure

<Type-fonction> : est le type de la valeur retourner par la fonction.

<Corps_fonction>: en plus des instructions décrivant le traitement à effectuer, une instruction d'affectation du résultat que devrait porter la fonction au nom de la fonction elle-même.

retourner <valeur>: est la valeur retournée par la fonction

Copyright - Tout droit réservé - OFPPT

Syntaxe Définition d’une fonction

Exemple:

- La fonction suivante permet de lire N valeurs entières et de calculer la plus petite parmi ces N valeurs.
- Les entiers saisis doivent être supérieurs à 0 et inférieurs à 100.

  - Le nom de cette fonction est **Min**
  - La fonction retour un entier
  - Les paramètres formels sont : l’entier **N** comme paramètre donné
  - 3 variables locales de type entier : **i , x et min**

```
Min (N : entier): entier
Var   i, x, min : entier
Début
            min := 100
            pour  i  de  1  à  N  faire
              Répéter
                  Lire ( x )
              Jusqu'à  (x > 0) et (x < 100)
              Si  x < min Alors
                  min := x
              Finsi

            Finpour
 retourner min
Fin
```

Copyright - Tout droit réservé - OFPPT

Appel d’une procédure/une fonction

- Lors de l’appel d’un sous algorithme (procédure ou fonction) à partir d’un algorithme appelant, on utilisera le nom de la procédure ou la fonction suivi par la liste de ses paramètres effectifs

<Nom>(<liste_par_effectif>)

  - **<Nom> :** est le nom de la procédure ou la fonction
  - **<liste_par_effectif> :** une suite d’objets désignant les paramètres effectifs séparés par des virgules (‘,’).

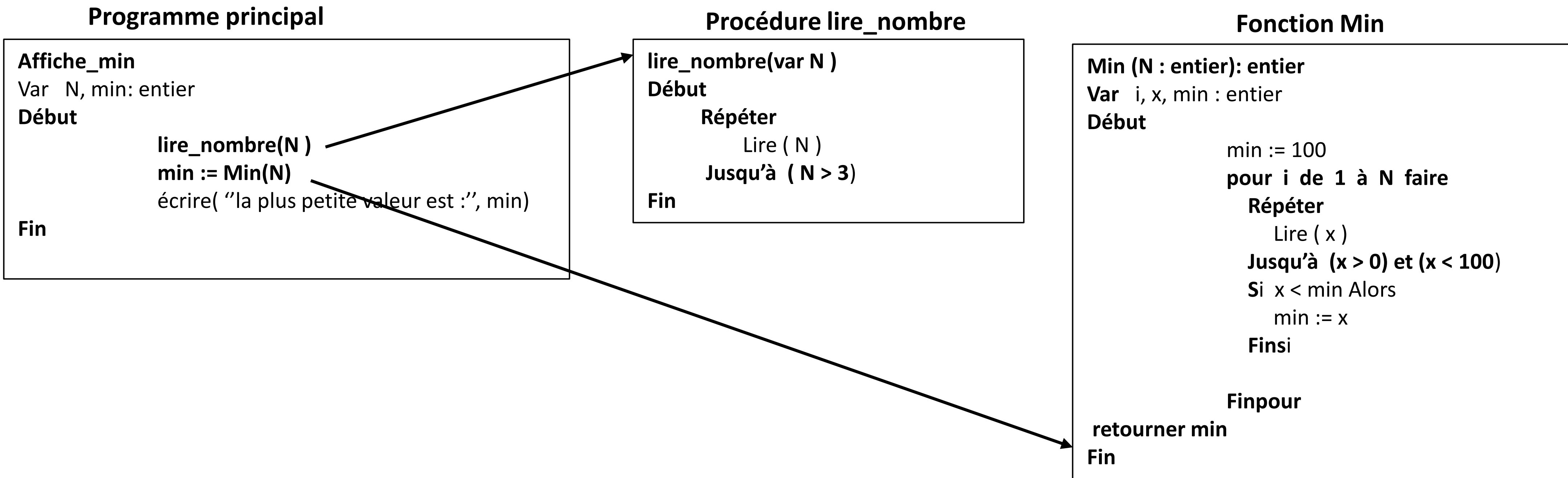
- Les paramètres effectifs et les paramètres formels doivent être compatibles en nombre et en type.
- La correspondance entre les 2 types de paramètres se fait selon l’ordre d’apparition.

Copyright - Tout droit réservé - OFPPT

Appel d'une procédure/une fonction

Exemple:

- On souhaite écrire un algorithme qui lit un entier N supérieurs à 3, puis saisit N valeurs entières et affiche la plus petite parmi ces N valeurs. Les entiers saisis doivent être supérieurs à 0 et inférieurs à 100.

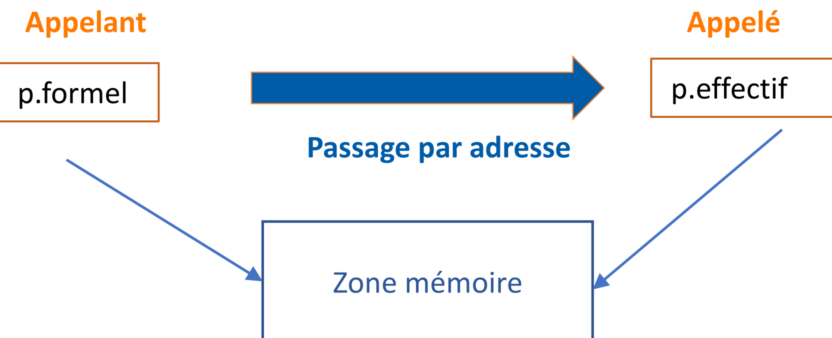


Passage de paramètre

- **Passage par valeur :** valeur du paramètre effectif transmise au paramètre formel.
    - **il s'agit d' paramètre donnée**

- **Passage par adresse** : l'adresse du paramètre effectif transmise au paramètre formel
    - **il s'agit d'un paramètre résultat ou d'un paramètre donnée/résultat**

- Une copie est transmise
- Toute modification sur le paramètre formel n'altère pas le paramètre effectif



- Une adresse est transmise
- Toute modification sur le paramètre formel altère le paramètre effectif

Copyright - Tout droit réservé - OFPPT

Passage de paramètre

Exemple:

Passage de paramètres par valeur

```
ajoute_un (a : entier)
Debut
        a := a+1
Fin

Appel :
Programme Principal
var  x : entier
Debut
        x := 9
        ajoute_un(x)
        ecrire(x)
Fin
```

Valeur affichée   9

Passage de paramètres par adresse

```
inc(var x : entier)
Debut
        x := x+1
Fin

Appel :
Programme Principal
var  y : entier
Debut
        y := 9
        inc(y)
        ecrire(y)
Fin
```

Valeur affichée 10

Copyright - Tout droit réservé - OFPPT

OFPPT
La Voie de l'Avenir

CHAPITRE 3

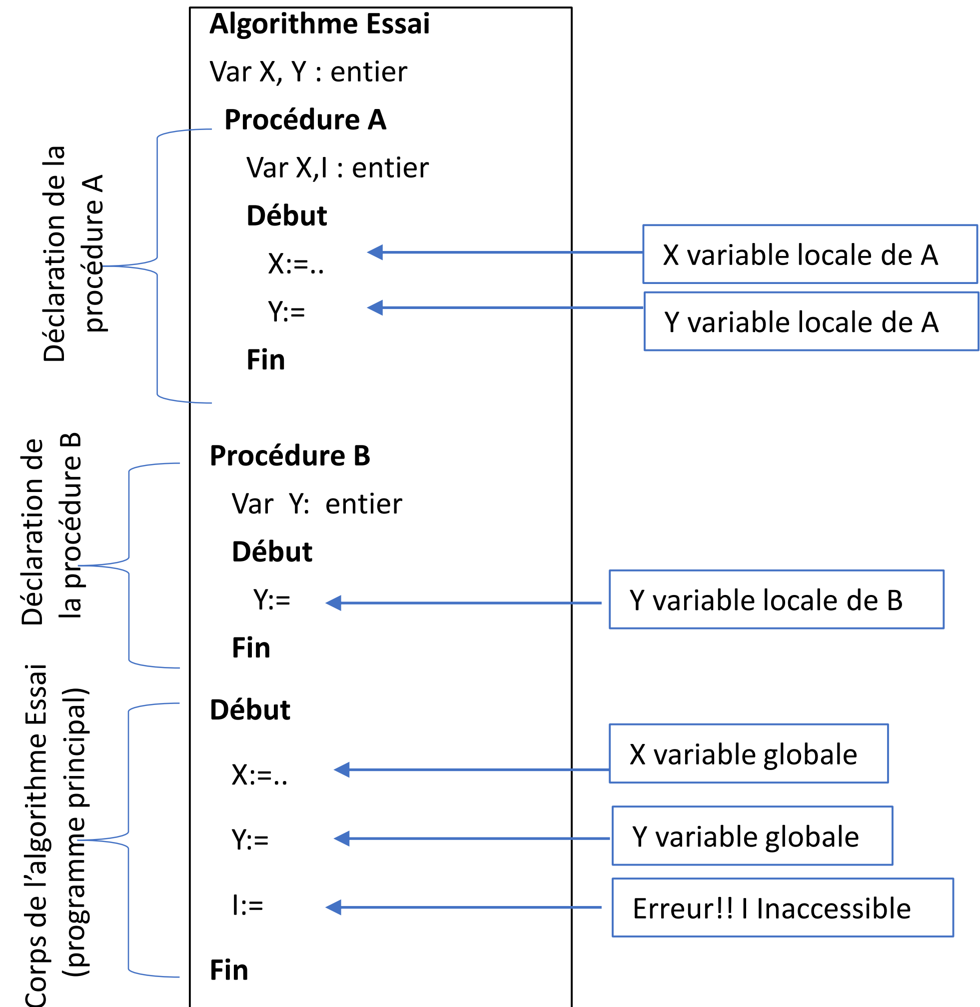
STRUCTURER UN ALGORITHME

1-Procédures et Fonctions

2-Portée d'une variable

Variable globale/variable locale

- La portée d'une variable est l'ensemble des sous-algorithmes (procédures ou fonctions) où cette variable est connue (les instructions de ces sous-algorithmes 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**
- **La portée d'une variable globale est totale** : tout sous-algorithme du programme principal peut utiliser cette variable
- Une variable définie au sein d'un sous-algorithme est appelée **variable locale**
- **La portée d'une variable locale est uniquement le sous-algorithme qui la déclare**
- Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée Dans ce sous-programme la variable globale devient inaccessible.



Copyright - Tout droit réservé - OFPPT

CHAPITRE 4

STRUCTURER LES DONNÉES

Ce que vous allez apprendre dans ce chapitre:

- Maîtriser la manipulation d'un tableau vecteur et d'un tableau multidimensionnel
- Connaître les principaux algorithmes de tri d'un tableau
- Maîtriser la manipulation des chaines de caractères

16 heures

CHAPITRE 4

STRUCTURER LES DONNÉES

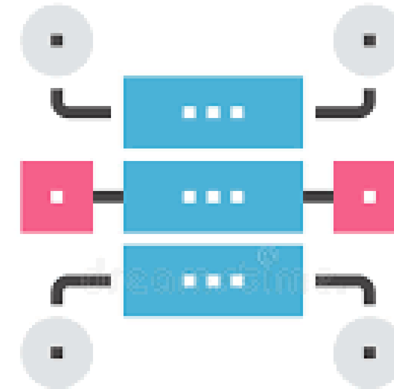
1-Différents types de tableaux

2-Chaines de caractères

Structure de données

- Une structure de données est une manière particulière de stocker et d’organiser des données dans un ordinateur de façon à pouvoir être utilisées efficacement.
- **Une structure de données regroupe :**
  - Un certain nombre de données à gérer,
  - Un ensemble d’opérations pouvant être appliquées à ces données

- **Dans la plupart des cas, il existe :**
  - plusieurs manières de représenter les données,
  - différents algorithmes de manipulation.



Copyright - Tout droit réservé - OFPPT

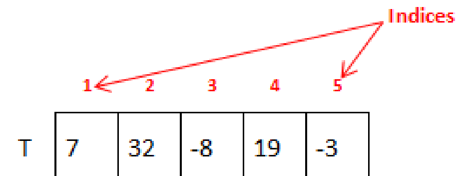
Structure Tableau Vecteur

- Un tableau est une structure de données qui permet de stocker à l'aide d'une seule variable un ensemble de valeurs de même type
- Un tableau unidimensionnel est appelé vecteur
- Syntaxe de déclaration d'un tableau vecteur

Type vecteur = tableau [1..MAX] de type des éléments
avec **MAX** est le nombre maximum d'éléments pour le type vecteur

- **Exemple**: déclaration d'un **tableau de 5 cases**

T : tableau [1..5] d'entier



- L'accès à un élément du tableau se fait via la position de cet élément dans le tableau

nom_tableau **[indice]**
avec **indice** est la position de l'élément dans le tableau

- **Exemple:** T[2] correspond à la case 2 ayant la valeur 32

Copyright - Tout droit réservé - OFPPT

Structure Tableau Vecteur

- **Caractéristiques**
  - Un tableau vecteur possède un nombre maximal d'éléments défini lors de l'écriture de l'algorithme (les bornes sont des constantes explicites, par exemple MAX, ou implicites, par exemple 10)
  - Le nombre d'éléments maximal d'un tableau est différent du nombre d'éléments significatifs dans un tableau

- **Exemple** d'un algorithme permettant de lire un tableau vecteur de 12 entiers

```
LectutreTabVecteur

Var   i : entier
      T : tableau[1..12] de  Réel
Debut
   Pour  i  de  1  à  12  faire
        Lire(T[i])
   Finpour
Fin
```

Copyright - Tout droit réservé - OFPPT

Structure de tableau multi-dimensions

- Par extension, on peut définir et utiliser des tableaux à n dimensions
- Syntaxe de déclaration d'un tableau à n dimensions:

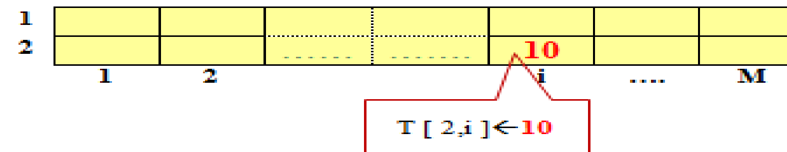
tableau [intervalle1,intervalle2,…,intervallen] de type des éléments

- Les tableaux à deux dimensions permettent de représenter les matrices
- Syntaxe de déclaration d'une matrice:

tableau [intervalle1,intervalle2] de type des éléments

- Chaque élément de la matrice est repéré par deux indices :
  - le premier indique le numéro de la ligne
  - le second indique le numéro de la colonne.
- <u>**Exemple**</u> de déclaration d'un tableau à 2 dimensions: **T : tableau [1..2,1..M] d'entier**

T[2,i] correspond à signe l'élément situé à la 2ème ligne et la ième colonne.



Copyright - Tout droit réservé - OFPPT

Structure de tableau multi-dimensions

<u>Exemple</u> : un algorithme permettant de lire un tableau matrice d'entiers de 12 lignes et 8 colonnes

```
LectutreTabMatrice
Var   i , j : entier
     T: tableau[1..12 , 1..8] de  Réel
Debut
   Pour  i  de  1  à  12  faire
        Pour  j  de  1  à  8  faire
             Lire(T[i , j])
        Finpour
   Finpour
Fin
```

indices

j de 1 à 8

i

de 1 à 12

Copyright - Tout droit réservé - OFPPT

OFPPT

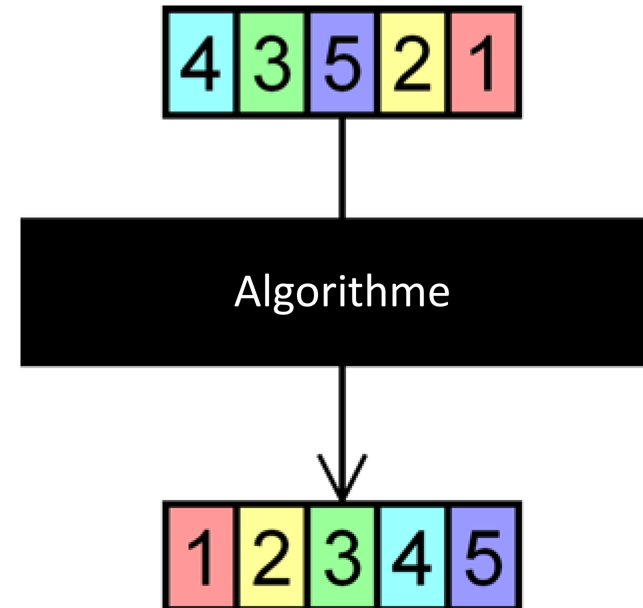
Tri d'un tableau

- Il existe plusieurs méthodes de tri qui se différencient par leur complexité d'exécution et leur complexité de compréhension pour le programmeur.
- Parmi les méthodes de Tri d'un tableau on cite :

Tri par insertion

Tri à bulles

Tri par sélection



Copyright - Tout droit réservé - OFPPT

Tri par sélection

Le tri par sélection est la méthode de tri la plus simple, elle consiste à :

- chercher l'indice du plus petit élément du tableau T[1..n] et permuter l'élément correspondant avec l'élément d'indice 1
- chercher l'indice du plus petit élément du tableau T[2..n] et permuter l'élément correspondant avec l'élément d'indice 2
- ……………………………………………………………………………………
- chercher l'indice du plus petit élément du tableau T[n-1..n] et permuter l'élément correspondant avec l'élément d'indice (n-1).

Tableau initial	6	4	2	3	5
Après la 1ère itération	2	4	6	3	5
Après la 2ème itération	2	3	6	4	5
Après la 3ème itération	2	3	4	6	5
Après la 4ème itération	2	3	4	5	6

```
Tri_Selection(Var T : Tab)
Var
  i, j, x, indmin : Entier
Début
   Pour i de 1 à (n-1) Faire
      indmin := i
      Pour j de (i+1) à n Faire
         Si (T[j] < T[indmin]) Alors
            indmin := j
         FinSi
      FinPour
      x:= T[i]
      T[i]:= T[indmin]
      T[indmin]:= x
   FinPour
Fin
```

Copyright - Tout droit réservé - OFPPT

Tri à bulles

La méthode de tri à bulles nécessite deux étapes :

- Parcourir les éléments du tableau de 1 à (n–1) ; si l'élément i est supérieur à l'élément (i+1), alors on les permute
- Le programme s'arrête lorsqu'aucune permutation n'est réalisable après un parcours complet du tableau.

Tableau initial	6	4	3	5	2
Après la 1ère itération	4	3	5	2	6
Après la 2ème itération	3	4	2	5	6
Après la 3ème itération	3	2	4	5	6
Après la 4ème itération	2	3	4	5	6

```
Tri_Bulle (Var T : Tab)
Var
  i, x : Entier
  échange : Booléen
Début
  Répéter
    échange← Faux
    Pour i de 1 à (n-1) Faire
      Si (T[i] > T[i+1]) Alors
        x:= T[i]
        T[i]:= T[i+1]
        T[i+1]:= x
        échange:= Vrai
      FinSi
    FinPour
  Jusqu'à (échange = Faux)
Fin
```

Copyright - Tout droit réservé - OFPPT

Tri par insertion

Le tri par insertion consiste à prendre les éléments de la liste un par un et insérer chacun dans sa bonne place de façon que les éléments traités forment une sous-liste triée.
Pour ce faire, on procède de la façon suivante :

- comparer et permuter si nécessaire T[1] et T[2] de façon à placer le plus petit dans la case d'indice
- comparer et permuter si nécessaire l'élément T[3] avec ceux qui le précèdent dans l'ordre (T[2] puis T[1]) afin de former une sous-liste triée T[1..3]
- ……………………………………………………
- comparer et permuter si nécessaire l'élément T[n] avec ceux qui le précèdent dans l'ordre (T[n-1], T[n-2], …) afin d'obtenir un tableau trié.

Tableau initial	6	4	3	5	2
Après la 1ère itération	4	6	3	5	2
Après la 2ème itération	3	4	6	5	2
Après la 3ème itération	3	4	5	6	2
Après la 4ème itération	2	3	4	5	6

```
Tri_Insertion(Var T : Tab)
Var
  i, j, x, pos : Entier
Début
  Pour i de 2 à n Faire
     pos:= i - 1
     TantQue (pos>=1) et (T[pos]>T[i]) Faire
        pos := pos – 1
      FinTQ
      pos:= pos + 1
      x← T[i]
      Pour j de (i-1) à pos [pas = -1] Faire
         T[j+1]:= T[j]
      FinPour
      T[pos] := x
  FinPour
Fin
```

[Pas = -1] signifie que le parcours se fait dans le sens décroissant

Copyright - Tout droit réservé - OFPPT

CHAPITRE 4

STRUCTURER LES DONNÉES

1-Différents types de tableaux

2-Chaines de caractères

Chaine de caractères

- Une chaîne est une suite de caractères. La chaîne ne contenant aucun caractère est appelée chaîne vide.
- **Syntaxe** de déclaration d'un chaine

ch : Chaîne
chn : Chaîne[Max]
La variable ch peut contenir jusqu'à 255 caractères alors que chn peut contenir au maximum Max caractère

- **Les opérations sur les chaines de caractères**

  - La **concaténation** C'est l'assemblage de deux chaînes de caractères en utilisant l'opérateur « + ».

Exemple:

chn1 :="Structure"

chn2 := "de données"

chn3 := chn1+" "+chn2

la variable chn3 contiendra "Structure de données"

Copyright - Tout droit réservé - OFPPT

04- STRUCTURER LES DONNÉES

Chaines de caractères

Chaine de caractères

- **Les opérations sur les chaines de caractères**
  - les opérateurs relationnels (>, >=, <, <=, =, #)
    - Il est possible d'effectuer une comparaison entre deux chaînes de caractères, le résultat est de type booléen. La comparaison se fait caractère par caractère de la gauche vers la droite selon le code ASCII.
    - **Exemples:**
      - L'expression («a" > "A") est vraie puisque le code ASCII de "a" (97) est supérieur à celui de "A" (65)
      - L'expression ("programme" < "programmation") est fausse puisque "e" > "a »
      - L'expression ("" = " ") est fausse (le vide est différent du caractère espace).
  - Accès à un caractère dans une chaîne
    - il suffit d'indiquer le nom de la chaîne suivi d'un entier entre crochets qui indique la position du caractère dans la chaîne.
    - **Exemple:** chn:= "Turbo Pascal"

      c:= chn[7] (la variable c contiendra le caractère "P")
    - En général, ch[i] désigne le ième caractère de la chaîne ch.

Copyright - Tout droit réservé - OFPPT

OFPPT

Chaine de caractères

- **Procédures standards sur les chaines de caractères**

Procédure	Rôle	Exemple
Efface(Chaîne, P, N)	Enlève N caractères de Chaîne à partir de la position P donnée.	chn ← "Turbo Pascal" efface(chn,6,7) ⇨ chn contiendra "Turbo"
Insert(Ch1, Ch2, P)	Insère la chaîne Ch1 dans la chaîne Ch2 à partir de la position P.	ch1 ← "D" ch2 ← "AA" insert(ch1,ch2,2) ⇨ ch2 contiendra "ADA"
Convch(Nbr, Ch)	Converti le nombre Nbr en une chaîne de caractères Ch.	n = 1665 convch(n,chn) ⇨ chn contiendra la chaîne "1665"

- **Fonctions standards sur les chaines de caractères**

Fonction	Rôle	Exemple
Long(Chaîne)	Retourne la longueur de la chaîne.	chn ← "Turbo Pascal" n ← Long(chn) ⇨ n contiendra 12
Copie(Chaîne, P, N)	Copie N caractères de Chaîne à partir de la position P donnée.	ch1 ← "Turbo Pascal" ch2 ← Copy(ch1,7,6) ⇨ ch2 contiendra "Pascal"
Position(Ch1, Ch2)	Retourne la position de la première occurrence de la chaîne Ch1 dans la chaîne Ch2.	ch1 ← "as" ch2 ← "Turbo Pascal" n ← Position(ch1,ch2) ⇨ n contiendra 8

Copyright - Tout droit réservé - OFPPT

OFPPT
La Voie de l'Avenir

PARTIE 3
PROGRAMMER EN PYTHON

Dans ce module, vous allez :

- Maitriser les bases de la programmation Python
- Appliquer les bonnes pratiques de la programmation Python
- Manipuler les fonctions en Python
- Maitriser la manipulation des données en Python

OFPPT

La Voie de l'Avenir

CHAPITRE 1
TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

Ce que vous allez apprendre dans ce chapitre :

- Connaitre les critères de choix d'un langage de programmation
- Connaitre les caractéristiques du langage Python
- Maitriser la structure générale d'un programme Python
- Traduire un algorithme en langage Python
- Appliquer les bonnes pratiques du codage en Python

20 heures

OFPPT
La Voie de l'Avenir

CHAPITRE 1
TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

1- Critères de choix d'un langage de programmation

2- Blocs d'instructions

3-Conversion de l'algorithme en Python

4- Optimisation du code (Bonnes pratiques de codage, commentaires,…)

OFPPT

Langage de programmation

- Langage de programmation est un outil à l'aide duquel le programmeur écrit des programmes exécutables sur un ordinateur
- Il y a toute une panoplie de langages disponibles

Exemples : FORTRAN, COBOL, Pascal, Ada, C, Java, Python

- Il est important de pouvoir évaluer ces langages afin de pouvoir les choisir de manière appropriée et de les améliorer
- Trois critères d'évaluation sont généralement utilisés

Critère	Description
la **lisibilité**	• correspond à la facilité avec laquelle un programme peut-être lu et compris
la **facilité d'écriture**	• correspond a la facilité avec laquelle un langage peut être utilisé pour créer un programme
la **fiabilité**	• correspond au degré de confiance avec lequel un programme peut être exécuté sous différentes conditions et aboutir aux mêmes résultats

Copyright - Tout droit réservé - OFPPT

OFPPT

Critères d'évaluation

Critères affectant la lisibilité

Critères affectant la facilité d'écriture

Critères affectant la fiabilité

La simplicité

- S'il y a beaucoup de composantes de bases (tels que les mots clés), il est difficile toutes les connaître
- S'il existe plusieurs façons d'exprimer une commande, il est aussi difficile de toutes les connaître
- Trop de simplicité cause la difficulté de lecture
- Il faut alors trouver un compromis entre la simplicité et la facilité d'écriture

L'orthogonalité

- L'orthogonalité est la propriété qui signifie "Changer A ne change pas B".
- Dans les langages de programmation, cela signifie que lorsque vous exécutez une instruction, rien que cette instruction ne se produit
- De plus, la signification d'un élément du langage doit être indépendante du contexte dans lequel il apparait

Instructions de contrôle

- Pour la lisibilité d'un langage de programmation, il est important d'avoir des structures de contrôle adéquates (structures itératives, structures conditionnelles, etc)
- Par exemple, l'un des plus grands problèmes du premier ,BASIC est que sa seule instruction de contrôle était le « goto »

Types et structures de donnée

- La présence de moyens appropriés pour définir des types et des structures de données dans un langage peut améliorer considérablement la lisibilité

Copyright - Tout droit réservé - OFPPT

OFPPT

Critères d’évaluation

Critères affectant la lisibilité

Critères affectant la facilité d'écriture

Critères affectant la fiabilité

La simplicité et l'orthogonalité

- Si un langage a une grande variation de constructeurs syntaxiques, il est fort possible que certains programmeurs ne les connaissent pas
- De même, il se peut que le programmeur ne connaisse certains constructeurs que superficiellement et les utilises de manière erronée

L'abstraction

- L ’abstraction est la possibilité de définir des structures ou des opérations compliquées tout en cachant leurs détails (abstraction de processus et abstraction des données)
- **Abstraction de processus** : Quand un processus est abstrait dans un sous-programme il n’est pas nécessaire de répéter son code à chaque fois qu’il est utilisé. Un simple appel de la procédure/fonction est suffisant
- **Abstraction des données:** les données peuvent être abstraites par les langages de programmation de haut niveau dans des objets à interface simple. L’utilisateur n’a pas besoin de connaitre les détails d’implémentation pour les utiliser (utilisation des arbres, tables de hachage,….)

L'expressivité

- Un langage est expressif s’il offre des outils simples, commodes et intuitifs pour permettre au programmeur d'exprimer les différents concepts de programmation

Exemple: Utiliser des boucles "for" et "while" au lieu de "goto

Copyright - Tout droit réservé - OFPPT

OFPPT

Critères d'évaluation

Critères affectant la lisibilité

Critères affectant la facilité d'écriture

Critères affectant la fiabilité

Vérification de types

- La vérification de type signifie qu'un langage est capable de détecter les erreurs relatives aux types de données lors de la compilation etbde l'exécution

Exemple:

- Le langage de programmation C e détecte pas ces erreurs, le programme peut-être exécuté, mais les résultats ne seront pas significatifs

Prise en Charge des Exceptions

- La possibilité pour un programme d'intercepter les erreurs faites pendant l'exécution, de les corriger, et de continuer l'exécution augmente de beaucoup la fiabilité du langage de programmation

Exemple:

- Des langages tels que Python, Ada, C++ et Java, Ruby, C# ont des capacités étendues de prise en charge des exceptions, mais de tels capacités sont absentes dans d'autres langages tels que le C ou le FORTRAN

Lisibilité et facilité d'écriture

- La lisibilité et la facilite d'écriture influencent la fiabilité des langages de programmation
- si il n'y a pas de moyens naturels d'exprimer un algorithme, des solutions complexes seront utilisées, et le risque d'erreurs (bugs) augmente

Copyright - Tout droit réservé - OFPPT

Coût d’un langage de programmation

- Si le langage n’est pas simple et orthogonal alors :
  - les coûts de formation de programmeurs seront plus élevés
  - l'écriture de programmes coûtera plus cher
- Autres facteurs
  - Les coûts de la compilation et de l’exécution de programmes
  - Les coûts de la maintenance de programmes
  - le coût de la mise en marche du langage
  - le coût lié au manque de fiabilité
  - le coût de la maintenance du langage (correction, modification et ajout de nouvelles fonctionnalités)

Autres critères

- Il y a aussi d’autres critères tels que:
  - La portabilité
  - La généralité/spécificité
  - La précision et la complétude de la description
  - La vitesse d'exécution
  - etc

Copyright - Tout droit réservé - OFPPT

Langage python

- Python est un langage de programmation développé **depuis 1989 par Guido van Rossum** et de nombreux contributeurs bénévoles
- **En février 1991**, la première version publique, numérotée 0.9.0
- Afin de réparer certains défauts du langage, **la version Python 3.0 a été publié en décembre 2008**.
- Cette version a été suivie par une version 3.1 qui corrige les erreurs de la version 3.0

Caractéristiques de Python

- Python est **portable**, non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires: MacOS, BeOS, NeXTStep, MS-DOS et les différentes variantes de Windows
- Python est **gratuit**, mais on peut l'utiliser sans restriction dans des projets commerciaux
- La **syntaxe de Python est très simple** et, combinée à des **types de données évolués** (listes, dictionnaires,...), conduit à des programmes à la fois très compacts et très lisibles.
- Python **gère ses ressources** (mémoire, descripteurs de fichiers...) sans intervention du programmeur

Copyright - Tout droit réservé - OFPPT

Caractéristiques de Python

- Python est **orienté-objet**.
- Python intègre un système d'**exceptions**, qui permettent de simplifier considérablement la gestion des erreurs.
- Python est **dynamique** (l'interpréteur peut évaluer des chaînes de caractères représentant des expressions ou des instructions Python), **orthogonal** (un petit nombre de concepts suffit à engendrer des constructions très riches) et **introspectif** (un grand nombre d'outils de développement, comme le debugger sont implantés en Python lui-même).
- Python est **dynamiquement typé** c’est à dire tout objet manipulable par le programmeur possède un type bien défini à l'exécution, qui n'a pas besoin d'être déclaré à l'avance.
- Python est **extensible**, on peut facilement l'interfacer avec des bibliothèques C existantes.
- **La bibliothèque standard** de Python, et les paquetages contribués, donnent accès à une grande variété de services: chaînes de caractères et expressions régulières, services UNIX standard (fichiers, pipes, signaux, sockets, threads...), protocoles Internet (Web, News, FTP, CGI, HTML...), persistance et bases de données, interfaces graphiques

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
Transformer une suite d'étapes algorithmique en une suite d'instructions Python

1- Critères de choix d'un langage de programmation

2- Blocs d'instructions

3-Conversion de l'algorithme en Python

4- Optimisation du code (Bonnes pratiques de codage, commentaires,…)

Structuration et notion de bloc

- En Python, chaque instruction s'écrit sur une ligne sans mettre d'espace:

Exemple:

```
a = 10
b = 3
print(a, b)
```

- Ces instructions simples peuvent cependant être mises sur la même ligne en les séparant par des points virgules ; les lignes étant exécutées dans l'ordre de gauche à droite:

Exemple:

```
a = 10; b = 3; print(a, b)
```

- La séparation entre les en-têtes qui sont des lignes de définition de boucles, de fonction, de classe qui se terminent par les deux points :
- Le contenu ou 'bloc' d'instructions correspondant se fait par indentation des lignes
- Une indentation s'obtient par le bouton tab (pour tabulation) ou bien par 4 espaces successifs.
- L’ensemble des lignes indentées constitue un bloc d’instructions.

```
instruction-1
en-tête-1:
    instruction-2
    instruction-2
    en-tête-2:
        instruction-3
        instruction-3
    en-tête-2:
        instruction-3
        instruction-3
    instruction-2
    instruction-2
instruction-1
instruction-1
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

1- Critères de choix d'un langage de programmation

2- Blocs d'instructions

3-Conversion de l'algorithme en Python

4- Optimisation du code (Bonnes pratiques de codage, commentaires,…)

Script et langage python

- Peu de ponctuation
- Pas de point virgule ";"
- Tabulations ou 4 espaces significatifs
- Scripts avec exécution d'un fichier ayant l'extension .py

Exemple: script.py

Invite de commande

```
Affichage d'une invite (prompt)
>>> 5 + 3        read : l'utilisateur tape une expression
8                eval et print : calcul et affichage du résultat
>>>              Réaffichage d'une invite
```

Fichier script.py

```
1 a=int(input('Donner a :'))
2 b=int(input('Donner b :'))
3 def pgcd(a,b):
4     if a<b:
5         a,b=b,a
6     while b!=0:
7         r,a,b=a%b,b,r
8     returne(a)
9 print(pgcd)
```

- Python utilise un **identifiant** pour nommer chaque objet.
- Python n'offre pas la notion de variable, mais plutôt celle de **référence** (adresse) d'objet.

```
>>> b=2
>>> c=b
>>> c
2
>>> b=3
>>> c
2
```

Copyright - Tout droit réservé - OFPPT

Types de données

Les types de données les plus utilisés sont:

- **Type entier (integer)**

```
>>> a=1
>>> b=555
>>> c=-6
```

- **Type réel (float)**

```
>>> b=0.003
>>> b=2.
```

- **Type Boolean**

```
>>> b=True
>>> c=False
```

- **Type caractère**

```
>>> b=True
>>> c=False
>>> phrase='les oeufs durs'
>>> phrase1='les oeufs durs'
>>> phrase2='Oui, répondit-il'
>>> phrase3="j'aime bien"
>>> print(phrase2, phrase3, phrase1)
Oui, répondit-il j'aime bien les oeufs durs
```

Copyright - Tout droit réservé - OFPPT

Variables

- Une variable est créée au moment où vous lui attribuez une valeur pour la première fois.

```
>>> x=5
>>> y='Jhon'
>>> print(x)
5
>>> print(y)
Jhon
```

- Les variables de chaîne peuvent être déclarées à l'aide de guillemets simples ou doubles:

```
>>> x="Jhon" #ceci est identique à faire 'Jhon'
```

- Règles pour les variables Python:
  - Un nom de variable doit commencer par une lettre ou le caractère de soulignement
  - Un nom de variable ne peut pas commencer par un nombre
  - Un nom de variable ne peut contenir que des caractères alphanumériques et des traits de soulignement (A-z, 0-9 et _)
  - Les noms de variable sont sensibles à la casse (age, Age et AGE sont trois variables différentes)

- Python permet d'affecter des valeurs à plusieurs variables sur une seule ligne:

```
>>> x,y,z='orange','Banana','Cherry'
>>> print(x)
orange
>>> print(y)
Banana
>>> print(z)
Cherry
```

Copyright - Tout droit réservé - OFPPT

Variables d'entrée

- La fonction **input()** retourne une valeur correspondant à ce que l'utilisateur a entré. Cette valeur peut alors être assignée à une variable quelconque
- **input()** renvoie une valeur dont le type est une chaine de caractère
- Utiliser **int()** pour convertir la sortie en entier

```
prenom = input("Entrez votre prénom (entre guillemets) : ")
print ("Bonjour,", prenom)
```

```
print ("Veuillez entrer un nombre positif quelconque : ")
nn = input()
print ("Le carré de", int(nn), "vaut", int(nn)**2)
```

Variables de sortie

- La fonction **print** Python est souvent utilisée pour afficher des variables et des chaines de caractères.
- Pour combiner à la fois du texte et une variable, Python utilise le caractère +:

```
>>> x="cool"
>>> print("Python est" + x)
Python estcool
>>> x="Python est"
>>> y="Cool"
>>> z=x+y
>>> print(z)
Python estCool
```

Copyright - Tout droit réservé - OFPPT

Variables de sortie

- Le mot clé **end** évite le retour à la ligne

```
>>> print("Hello"); print ("Joe")
Hello
Joe
>>> print("Hello", end=""); print("Joe")
HelloJoe
>>> print("Hello", end=" "); print("Joe")
Hello Joe
```

- Le mot clé **sep** précise une séparation entre les variables chaines

```
>>> x=32
>>> nom="John"
>>> print(nom," a",x," ans")
John  a 32  ans
>>> x=32
>>> nom="John"
>>> print(nom,"a",x,"ans")
John a 32 ans
>>> print(nom,"a",x,"ans",sep="")
Johna32ans
>>> print(nom,"a",x,"ans",sep=" ")
John a 32 ans
```

- **La méthode .format()** permet une meilleure organisation de l'affichage des variables

```
>>> x=32
>>> nom="John"
>>> print("{} a {} ans." . format(nom,x))
John a 32 ans.
>>> print("{0} a {1} ans." . format(nom,x))
John a 32 ans.
```

Copyright - Tout droit réservé - OFPPT

Manipulation des Types numériques

- Les quatre arithmétiques de base se font de manière simple sur les types numériques (nombres entiers et réels)

```
>>> x=45
>>> x+2
47
>>> x-2
43
>>> x*3
135
>>> y=205
>>> y=2.5
>>> x-y
42.5
```

```
>>> (x*10)+y
452.5
>>> 3/4
0.75
>>> 2**3
8
>>> 5/4
1.25
>>> 5//4
1
>>> 5%4
1
```

```
>>> i=0
>>> i=i+1
>>> i
1
>>> i+=1
>>> i
2
>>> i+=2
>>> i
4
```

Copyright - Tout droit réservé - OFPPT

Manipulation des chaînes de caractères

- En Python une chaîne de caractères est un objet de la classe **str**
- **Les opérateurs de concaténation (+) et de répétition (*)**

```
>>> s1="Welcome"
>>> s2="Python"
>>> s3=s1 +" to " + s2
>>> s3
'Welcome to Python'
>>> s4= 3 * s1
>>> s4
'WelcomeWelcomeWelcome'
```

- **Les opérateurs in et not in**

```
>>> s1="Welcome"
>>> "come" in s1
True
>>> "come" not in s1
False
```

- **Création des chaînes de caractères en utilisant le mot clé str**

```
>>> s1=str() # créer un objet chaine de caractère vide
>>> s2=str("Welcome") # créer l'objet chaine de caractères "Welcome"
>>>
>>> s3="" #identique à s3=str()
>>> s4="Welcome" # identique à s2=str("Welcome")
```

Copyright - Tout droit réservé - OFPPT

Manipulation des chaînes de caractères

- **Fonctions des chaînes de caractères**
  - Plusieurs fonctions intégrées en Python sont utilisées avec les chaînes de caractères.
    - Puisque s a 7 caractères, **len(s)** renvoie 7
    - les lettres minuscules ont une valeur ASCII supérieure à celle des lettres majuscules, donc **max(s)** retourne ‘o’ et **min(s)** retourne ‘W’ (ligne 7).

```
>>> s1="Welcome"
>>> len(s1)
7
>>> max(s1)
'o'
>>> min(s1)
'W'
```

- **L’opérateur indice [ ]**
  - Une chaîne de caractères est une séquence de caractères.
  - Un caractère de la chaîne est accessible par l’opérateur indice [ ]

```
>>> s="Welcome"
>>> for i in range (0, len(s),2):
        print(s[i], end=' ')


W l o e
```

- **Découpage en tranche ([début :fin])**

```
>>> s="Welcome"
>>> s[1:4]
'elc'
```

```
>>> s="Welcome"
>>> s[ :6]
'Welcom'
>>> s[4:]
'ome'
```

Copyright - Tout droit réservé - OFPPT

Manipulation des chaînes de caractères

- **Comparaison de chaînes de caractères**
  - La comparaison des caractères un par un selon leurs code ASCII

```
>>> "green" <= "glow"
False
>>> "ab"<= "abc"
True
```

- **Recherche de sous-chaînes**
  - **endswith**: vérifie si une chaine se termine par une autre
  - **startswith**: vérifie si une chaine se commence par une autre
  - **Find**: recherche de la position d'une chaine dans une autre
  - **Count**: retourne ne nombre d'occurrence d'une chaine dans une autre

```
>>> s="welcome to Python"
>>> s.endswith("thon")
True
>>> s.startswith("good")
False
>>> s.find("come")
3
>>> s.find("become")
-1
>>> s.count("o")
3
```

Copyright - Tout droit réservé - OFPPT

Structure conditionnelle

Exemple1:

```
>>> a=33
>>> b=200
>>> if b>a:
        print("b is greater than a")

b is greater than a
```

Exemple 2:

```
>>> a=33
>>> b=33
>>> if b>a: print("b is greater than a")
elif a==b: print("a and b are equals")

a and b are equals
```

Exemple 3:

```
>>> a=200
>>> b=33
>>> if b>a : print("b is greater than a")
elif a==b: print ("a and b are equals")
else: print("a is greater than b")

a is greater than b
```

- Le mot clé **or** est un opérateur logique et est utilisé pour combiner des instructions conditionnelles:

```
>>> a=200
>>> b=33
>>> c=500
>>> if a>b or a>c: print ("At least one of the conditions is true")

At least one of the conditions is true
```

- Le mot clé **and** est un opérateur logique et est utilisé pour combiner des instructions conditionnelles:

```
>>> a=200
>>> b=33
>>> c=500
>>> if a>b and c>a: print("Both conditions are True")

Both conditions are True
```

Copyright - Tout droit réservé - OFPPT

Boucles d'itérations

- **Boucle While**

  - Avec la boucle **while**, il est possible d'exécuter un ensemble d'instructions tant qu'une condition est vraie:

```
>>> i=1
>>> while i<6:
        print(i)
        i+=1


1
2
3
4
5
```

  - Avec l'instruction **break**, nous pouvons arrêter la boucle même si la condition **while** est vraie:

```
>>> i=1
>>> while i<6:
        print(i)
        if i==3:
                break
        i+=1


1
2
3
```

  - Avec l'instruction continue, nous pouvons arrêter l'itération en cours et continuer avec la suivante:

```
>>> i=0
>>> while i<6:
        i+=1
        if i==3:
                continue
        print(i)


1
2
4
5
6
```

Copyright - Tout droit réservé - OFPPT

Boucles d'itérations

- **Boucle For**
  - Une boucle **for** est utilisée pour itérer sur une séquence (c'est-à-dire une liste, un tuple, un dictionnaire, un ensemble ou une chaîne):

```
>>> fruits =["apple","banana","cherry"]
>>> for x in fruits:
        print(x)


apple
banana
cherry
```

  - Même les chaînes sont des objets itérables, elles contiennent une séquence de caractères

```
>>> for x in "banana":
        print(x)


b
a
n
a
n
a
```

  - Avec l'instruction break, nous pouvons arrêter la boucle avant d'avoir bouclé tous les éléments:

```
>>> fruits =["apple","banana","cherry"]
>>> for x in fruits:
        print(x)
        if x=="banana":
                break


apple
banana
```

Copyright - Tout droit réservé - OFPPT

Boucles d'itérations

- **Boucle For**
    - Pour parcourir un ensemble de codes un nombre spécifié de fois, nous pouvons utiliser la fonction **range (),**
    - La fonction **range ()** renvoie une séquence de nombres, commençant à **0 par défaut**, et incrémentant de **1 (par défaut),** et se termine à un nombre spécifié;

```
>>> for x in range(6):
        print(x)


0
1
2
3
4
5
```

    - La fonction range () par défaut est 0 comme valeur de départ, mais il est possible de spécifier la valeur de départ en ajoutant un paramètre: range (2, 6), ce qui signifie des valeurs de 2 à 6 (mais pas 6):

```
>>> for x in range(2,6):
        print(x)


2
3
4
5
```

    - La fonction range () par défaut incrémente la séquence de 1, mais il est possible de spécifier la valeur d'incrément en ajoutant un troisième paramètre: range (2, 30, 3):

```
>>> for x in range(2,12,3):
        print(x)


2
5
8
11
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
Transformer une suite d'étapes algorithmique en une suite d'instructions Python

1- Critères de choix d'un langage de programmation

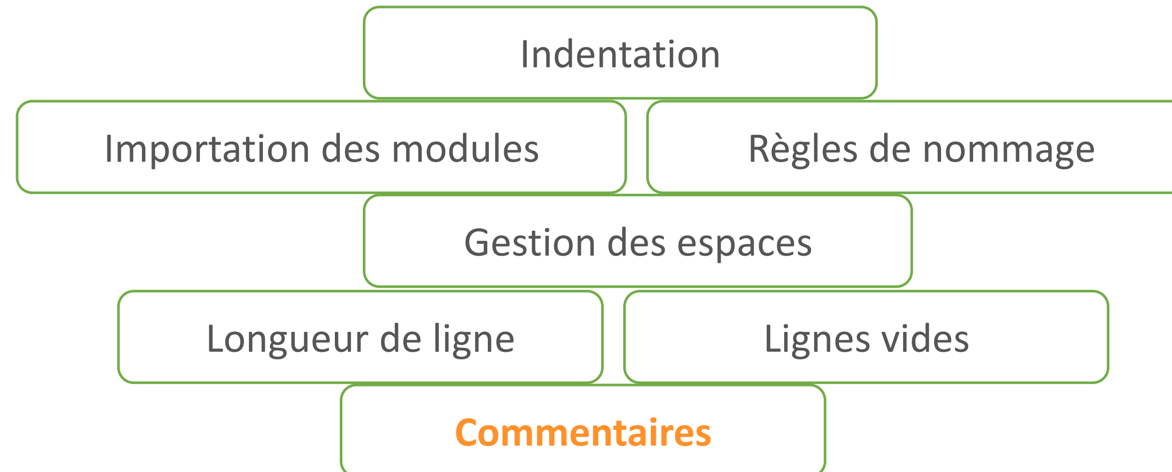
2- Blocs d'instructions

3-Conversion de l'algorithme en Python

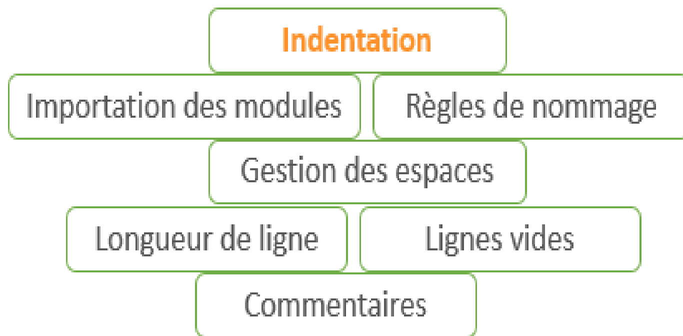
4- Optimisation du code (Bonnes pratiques de codage, commentaires,…)

Python Enhancement Proposal

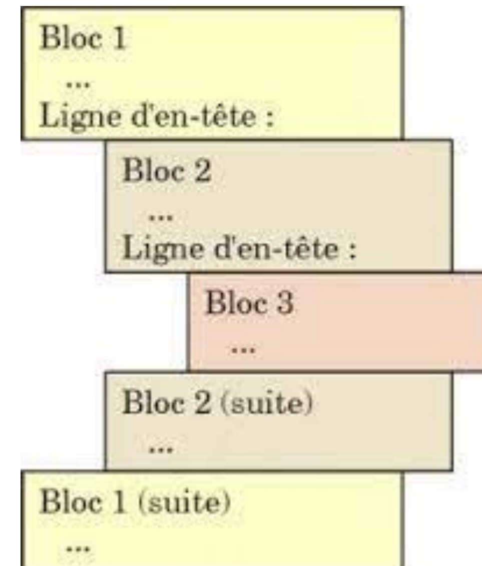
- Afin d'améliorer le langage Python, la communauté qui développe Python publie régulièrement des **Python Enhancement Proposal (PEP),** suivi d'un numéro.
- Il s'agit de propositions concrètes pour améliorer le code, ajouter de nouvelles fonctionnalités, mais aussi des recommandations sur la manière d'utiliser Python, bien écrire du code, etc.
- On parle de code **pythonique** lorsque ce dernier respecte les règles d'écriture définies par la communauté Python mais aussi les règles d'usage du langage.
- La PEP 8 Style Guide for Python Code 2 est une des plus anciennes PEP (les numéros sont croissants avec le temps). Elle consiste en un nombre important de recommandations sur la syntaxe de Python
- Quelques concepts de PEP



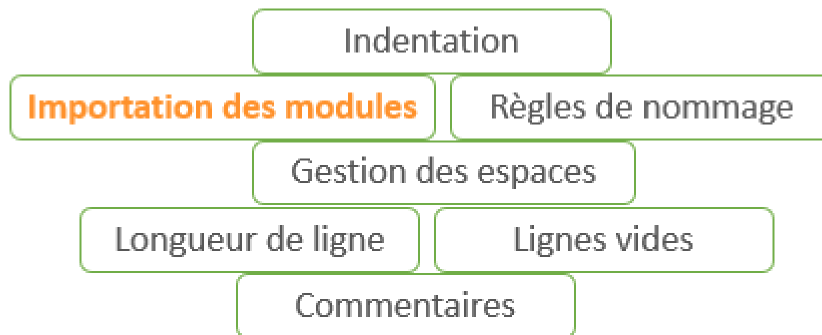
Copyright - Tout droit réservé - OFPPT



- L'indentation est obligatoire en Python pour séparer les blocs d'instructions.
- Cela vient d'un constat simple, l'indentation améliore la lisibilité d'un code
- Dans la PEP 8, la recommandation pour la syntaxe de chaque niveau d'indentation est très simple : 4 espaces



Copyright - Tout droit réservé - OFPPT



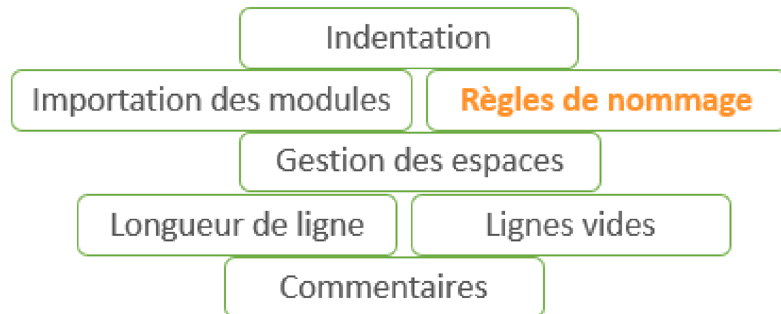
- Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi bibliothèques ou libraries). Ce sont des « boîtes à outils » qui vont vous être très utiles.
- l'utilisation de la syntaxe **import module** permet d'importer tout une série de fonctions organisées par «thèmes ».

Exemple:

les fonctions gérant les nombres aléatoires avec **random** et les fonctions mathématiques avec **math**. Python possède de nombreux autres modules internes (c'est-à-dire présent de base lorsqu'on installe Python)

```
>>> import math
>>> math.cos(math.pi / 2)
6.123233995736766e-17
>>> math.sin(math.pi / 2)
1.0
```

Copyright - Tout droit réservé - OFPPT



- Les noms de variables, de fonctions et de modules doivent êtres en minuscules avec un caractère « souligné » (« tiret du bas » ou underscore en anglais) pour séparer les différents « mots » dans le nom.

```
ma_variable
fonction_test_27()
mon_module
```

- Les constantes sont écrites en majuscules :

```
MA_CONSTANTE
VITESSE_LUMIERE
```

Copyright - Tout droit réservé - OFPPT

- La PEP 8 recommande d'entourer les opérateurs (+, -, /, *, ==, !=, >=, not, in, and, or. . .) d'un espace avant et d'un espace après. Par exemple :

```
# code recommandé :
ma_variable = 3 + 7
mon_texte = "souris"
mon_texte == ma_variable
# code non recommandé :
ma_variable=3+7
mon_texte="souris"
mon_texte== ma_variable
```

- Il n'y a, par contre, pas d'espace à l'intérieur de crochets, d'accolades et de parenthèses :

```
# code recommandé :
ma_liste[1]
mon_dico{"clé"}
ma_fonction(argument)
# code non recommandé :
ma_liste[ 1 ]
mon_dico{"clé" }
ma_fonction( argument )
```

- Ni juste avant la parenthèse ouvrante d'une fonction ou le crochet ouvrant d'une liste ou d'un dictionnaire :

```
# code recommandé :
ma_liste[1]
mon_dico{"clé"}
ma_fonction(argument)
# code non recommandé :
ma_liste [1]
mon_dico {"clé"}
ma_fonction (argument)
```

Copyright - Tout droit réservé - OFPPT

- On met un espace après les caractères : et , (mais pas avant) :

```
# code recommandé :
ma_liste = [1, 2, 3]
mon_dico = {"clé1": "valeur1", "clé2": "valeur2"}
ma_fonction(argument1, argument2)
# code non recommandé :
ma_liste = [1 , 2 ,3]
mon_dico = {"clé1": "valeur1", "clé2": "valeur2"}
ma_fonction(argument1 ,argument2)
```

- Par contre, pour les tranches de listes, on ne met pas d’espace autour du :

```
ma_liste = [1, 3, 5, 7, 9, 1]
# code recommandé :
ma_liste[1:3]
ma_liste[1:4:2]
ma_liste[::2]
# code non recommandé :
ma_liste[1 : 3]
ma_liste[1: 4:2 ]
ma_liste[ : :2]
```

- On n’ajoute pas plusieurs espaces autour du = ou des autres opérateurs

```
# code recommandé :
x1 = 1
x2 = 3
x_old = 5
```

Copyright - Tout droit réservé - OFPPT

- Une ligne de code ne doit pas dépasser 79 caractères
- Le caractère \ permet de couper des lignes trop longues.

```
>>> ma_variable = 3
>>> if ma_variable > 1 and ma_variable < 10 \
... and ma_variable % 2 == 1 and ma_variable % 3 == 0:
...     print(f"ma variable vaut {ma_variable}")
...
ma variable vaut 3
```

- À l'intérieur d'une parenthèse, on peut revenir à la ligne sans utiliser le caractère \. C'est particulièrement utile pour préciser les arguments d'une fonction ou d'une méthode, lors de sa création ou lors de son utilisation :

```
>>> def ma_fonction(argument_1, argument_2,
...                 argument_3, argument_4):
...     return argument_1 + argument_2
...
>>> ma_fonction("texte très long", "tigre",
...             "singe", "souris")
'texte très longtigre'
```

Copyright - Tout droit réservé - OFPPT

- Les parenthèses sont également très pratiques pour répartir sur plusieurs lignes une chaîne de caractères qui sera affichée sur une seule ligne

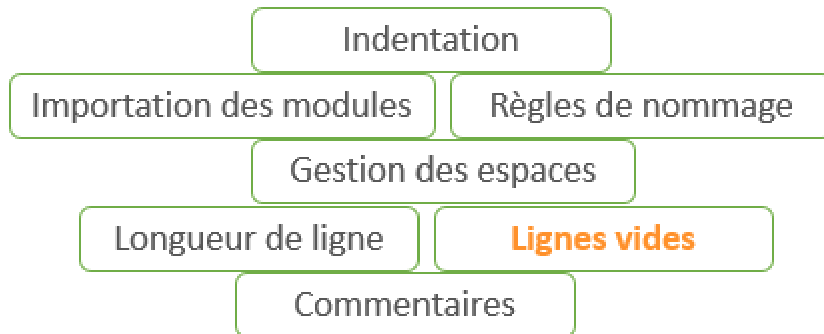
```
>>> print("ATGCGTACAGTATCGATAAC"
...       "ATGACTGCTACGATCGGATA"
...       "CGGGTAACGCCATGTACATT")
ATGCGTACAGTATCGATAACATGACTGCTACGATCGGATACGGGTAACGCCATGTACATT
```

- L' opérateur + est utilisée pour concaténer les trois chaînes de caractères et que celles-ci ne sont pas séparées par des virgules.
- À partir du moment où elles sont entre parenthèses, Python les concatène automatiquement.
- On peut aussi utiliser les parenthèses pour évaluer un expression trop longue

```
>>> ma_variable = 3
>>> if (ma_variable > 1 and ma_variable < 10
... and ma_variable % 2 == 1 and ma_variable % 3 == 0):
...     print(f"ma variable vaut {ma_variable}")
...
ma variable vaut 3
```

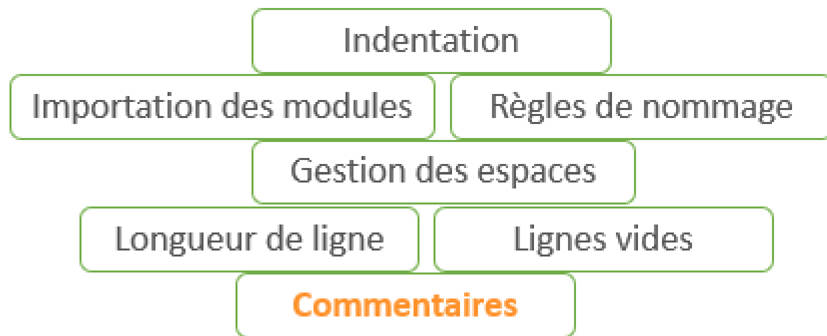
Copyright - Tout droit réservé - OFPPT

- Dans un script, les lignes vides sont utiles pour séparer visuellement les différentes parties du code.
- Il est recommandé de laisser deux lignes vides avant la définition d'une fonction
- . On peut aussi laisser une ligne vide dans le corps d'une fonction pour séparer les sections logiques de la fonction, mais cela est à utiliser avec parcimonie.




```
from math import *

def maximum(a, b):
  if a > b:
    return a
  else:
    return b

def maximum3(a, b, c):
  m = maximum(a, b)
  return maximum(m, c)
```

Copyright - Tout droit réservé - OFPPT



- Les commentaires débutent toujours par le **symbole #** suivi d'un espace.
- Les commentaires donnent des explications claires sur l'utilité du code et doivent être synchronisés avec le code, c'est-à-dire que si le code est modifié, les commentaires doivent l'être aussi (le cas échéant).
- Les commentaires sont sur le même niveau d'indentation que le code qu'ils commentent.
- Les commentaires sont constitués de phrases complètes, avec une majuscule au début (sauf si le premier mot est une variable qui s'écrit sans majuscule) et un point à la fin
- PEP 8 recommande la cohérence entre la langue utilisée pour les commentaires et la langue utilisée pour nommer les variables. Pour un programme scientifique, les commentaires et les noms de variables sont en anglais.

```
# Premier essai de script Python
# petit programme simple affichant une suite de Fibonacci, c.à.d. une suite
# de nombres dont chaque terme est égal à la somme des deux précédents.

print "Suite de Fibonacci :"

a,b,c = 1,1,1                # a & b servent au calcul des termes successifs
                             # c est un simple compteur
print 1                      # affichage du premier terme
while c<15:                  # nous afficherons 15 termes au total
    a,b,c = b,a+b,c+1
    print b
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
MANIPULER LES DONNÉES

Ce que vous allez apprendre dans ce chapitre:

- Manipuler les fonctions et les fonctions Lambda en Python
- Maitriser les structures de données Python et les différencier
- Maitriser le manipulation des différents types de fichiers de données
- Connaitre les principales bibliothèques standards de Python

40 heures

OFPPT
La Voie de l'Avenir

CHAPITRE 2
MANIPULER LES DONNÉES

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

3- Fichiers de données

4- Bibliothèques standards

Manipulation des fonctions

- Une fonction est un bloc de code qui ne s'exécute que lorsqu'elle est appelée.
  - Vous pouvez transmettre des données, appelées paramètres, à une fonction.
  - Une fonction peut renvoyer des données en conséquence.
  - Pour appeler une fonction, utilisez le nom de la fonction suivi de parenthèses

```
>>> def my_function():
        print("Hello from a function")


>>> my_function()
Hello from a function
```

- Les informations peuvent être transmises aux fonctions comme arguments.
  - Les arguments sont spécifiés après le nom de la fonction, entre parenthèses.
  - Vous pouvez ajouter autant d'arguments que vous le souhaitez, séparez-les simplement par une virgule.

```
>>> def my_function(fname):
        print(fname+ ":Hello")


>>> my_function("Emil")
Emil:Hello
>>> my_function("Linus")
Linus:Hello
```

- Il est possible de donner une valeur par défaut à un paramètre d’une fonction

```
>>> def my_function(country="Norway"):
        print("I am from " + country)


>>> my_function("sweden")
I am from sweden
>>> my_function()
I am from Norway
```

- Une fonction peut retourner une valeur

```
>>> def my_function(x):
        return 5*x

>>> print(my_function(3))
15
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Fonction Lambda

- En Python, le mot clé **Lambda** est utilisé pour déclarer une fonction **anonyme** (sans nom), raison pour laquelle ces fonctions sont appelées « fonction Lambda » .
- Une fonction Lambda est comme n'importe quelle fonction Python normale, sauf **qu'elle n'a pas de nom lors de sa définition** et qu'elle est **contenue dans une ligne**
- Tout comme la définition d'une fonction normale par l'utilisateur à l'aide du mot clé '**def**', une fonction Lambda est définie à l'aide du mot clé '**Lambda** '.
- Une fonction Lambda peut avoir 'n' nombre d'arguments mais une seule expression

Syntaxe

Exemple:

- Une fonction lambda qui ajoute 10 au nombre passé en argument et affiche le résultat:

```
>>> x = lambda a : a + 10
>>> print(x(5))
15
```

- Une définition de fonction qui prend un argument, et cet argument sera multiplié par un nombre inconnu

```
>>> def myfunc(n):
        return lambda a : a * n

>>> mydoubler = myfunc(2)
>>> print(mydoubler(11))
22
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
Manipuler les données

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

3- Fichiers de données

4- Bibliothèques standards

Tableaux dynamiques(Liste)

- Une **liste** est une **collection** qui est **commandée et modifiable.**
  - En Python, les liste sont écrites entre crochets.

```
>>> thislist= ["apple","banana","cherry"]
>>> print(thislist)
['apple', 'banana', 'cherry']
>>> print(thislist[1])
banana
```

```
>>> thislist= ["apple","banana","cherry"]
>>> print(thislist)
['apple', 'banana', 'cherry']
>>> print(thislist[1])
banana
>>> thislist= ["apple","banana","cherry"]

>>> print(thislist[-1]) #afficher la valeur de la position -1 (cycle)

cherry
>>> thislist=["apple","banana","cherry","orange","kiwi","melon","mango"]

>>> print(thislist[2:5])#afficher les valeurs de la position 2 jusqu'à 5 (5 non inclus)
['cherry', 'orange', 'kiwi']
```

Copyright - Tout droit réservé - OFPPT

Tableaux dynamiques(Liste)

- Modification de la valeur d''un élément du tableau

```
>>> thislist= ["apple","banana","cherry"]
>>> thislist[1] ="blackcurrant" # modifier la valeur de la 1ere position
>>> print(thislist)
['apple', 'blackcurrant', 'cherry']
```

- Parcours une liste:

```
>>> for x in thislist:
        print(x)


apple
banana
cherry
```

- Recherche d'un élément dans une liste:

```
>>> thislist= ["apple", "banana", "cherry"]
>>> if "apple" in thislist:
        print("Yes, 'apple' is in the fruits list")


Yes, 'apple' is in the fruits list
```

Copyright - Tout droit réservé - OFPPT

Tableaux dynamiques(Liste)

- **Fonction len():** Longueur d'une liste **(fonction len())**

```
>>> thislist= ["apple", "banana", "cherry"]
>>> print(len(thislist))
3
```

- **Fonction append():** Ajout d'un élément à la liste

```
>>> thislist= ["apple", "banana", "cherry"]

>>> thislist.append("orange")

>>> print(thislist)
['apple', 'banana', 'cherry', 'orange']
```

- **Fonction insert():** Ajout d'un élément à une position de la liste:

```
>>> thislist= ["apple", "banana", "cherry"]
>>> thislist.insert(1, "orange")
>>> print(thislist)
['apple', 'orange', 'banana', 'cherry']
```

Copyright - Tout droit réservé - OFPPT

Tableaux dynamiques(Liste)

- **Fonction pop():** Suppression du dernier élément de la liste

```
>>> thislist= ["apple", "banana", "cherry"]
>>> thislist.pop()
'cherry'
>>> print(thislist)
['apple', 'banana']
```

- **Fonction del():** Suppression d' un élément de la liste

```
>>> thislist= ["apple", "banana", "cherry"]
>>> del(thislist[0])
>>> print(thislist)
['banana', 'cherry']
```

- **Fonction extend():** Fusion de deux listes

```
>>> list1 = ["a", "b" , "c"]

>>> list2 = [1,2,3]
>>> list1.extend(list2)

>>> print(list1)
['a', 'b', 'c', 1, 2, 3]
```


Copyright - Tout droit réservé - OFPPT

OFPPT

Tableaux dynamiques(Liste)

- **Autres méthodes**

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Copyright - Tout droit réservé - OFPPT

Tableaux statiques (tuple)

- Un **tuple** est une collection **ordonnée** et non **changeable**.
  - En Python, les tuples ont écrits avec des **crochets ronds.**

```
>>> thistuple= ("apple","banana","cherry")
>>> print(thistuple)
('apple', 'banana', 'cherry')
>>> print(thistuple[1])#accéder à la valeur 1
banana
```

  - Convertissez le tuple en liste pour pouvoir le modifier:

```
>>> x = ("apple","banana","cherry")
>>> y =list(x)#convertir le tupleen une liste

>>> y[1] ="kiwi"#changerla valeur de la 1er position de la liste

>>> x =tuple(y)#convertir la liste en un tuple

>>> print(x)
('apple', 'kiwi', 'cherry')
```

Copyright - Tout droit réservé - OFPPT

Tableaux statiques (tuple)

- **Parcours d'un tuple:**

```
>>> thistuple= ("apple", "banana", "cherry")

>>> for x in thistuple:
        print(x)


apple
banana
cherry
```

- **Vérification si un élément est dans un tuple:**

```
>>> thistuple= ("apple", "banana", "cherry")

>>> if "apple" in thistuple:
        print("Yes, 'apple' is in the fruits tuple")


Yes, 'apple' is in the fruits tuple
```

- Surpression d'un tuple

```
>>> thistuple= ("apple", "banana", "cherry")

>>> del(thistuple)

>>> print(thistuple) #this will raise an error because the tuple no longer exists
Traceback (most recent call last):
  File "<pyshell#521>", line 1, in <module>
    print(thistuple) #this will raise an error because the tuple no longer exists
NameError: name 'thistuple' is not defined
```


Copyright - Tout droit réservé - OFPPT

Tableaux statiques (tuple)

- Fusion de deux tuples

```
>>> tuple1 = ("a","b","c")

>>> tuple2 = (1,2,3)

>>> tuple3 = tuple1 +tuple2

>>> print(tuple3)
('a', 'b', 'c', 1, 2, 3)
```

- Autres méthodes

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Copyright - Tout droit réservé - OFPPT

OFPPT

Tableaux statiques (set)

- Un set est une **collection** non **ordonnée** et non **indexée**.
- En Python, les **sets** sont écrits avec des **accolades**.
- **Création d'un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> print(thisset)
{'apple', 'cherry', 'banana'}
```

- **Parcours d'un set:**

```
>>> for x in thisset:
        print(x)


apple
cherry
banana
```

Copyright - Tout droit réservé - OFPPT

Tableaux statiques (set)

- **Vérification si un élément est dans un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> print("banana" in thisset)
True
```

- **Une fois qu'un ensemble est créé, vous ne pouvez pas modifier ses éléments, mais vous pouvez ajouter de nouveaux éléments.**
- **Ajout d'un élément un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> thisset.add("orange")

>>> print(thisset)
{'orange', 'apple', 'cherry', 'banana'}
```

- **Ajout de plusieurs éléments à un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> thisset.update(["orange", "mango", "grapes"])

>>> print(thisset)
{'banana', 'mango', 'orange', 'grapes', 'apple', 'cherry'}
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Tableaux statiques (set)

- **Suppression d'un élément d'un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> thisset.remove("banana")

>>> print(thisset)

{'apple', 'cherry'}
>>> #ou bien
thisset.discard("apple")

>>> print(thisset)
{'cherry'}
```

- **suppression du dernier élément d'un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> x = thisset.pop()

>>> print(x)

apple
>>> print(thisset)
{'cherry', 'banana'}
```

Copyright - Tout droit réservé - OFPPT

Tableaux statiques (set)

- **Suppression de tous les éléments d'un set**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> thisset.clear()

>>> print(thisset)
set()
>>>
```

- **Suppression d'un set:**

```
>>> thisset= {"apple", "banana", "cherry"}

>>> del(thisset)

>>> print(thisset)
Traceback (most recent call last):
  File "<pyshell#583>", line 1, in <module>
    print(thisset)
NameError: name 'thisset' is not defined
>>>
```

- **Fusion de deux sets**

```
>>> set1 = {"a", "b" , "c"}

>>> set2 = {1, 2, 3}

>>> set3 = set1.union(set2)

>>> print(set3)

{'b', 1, 2, 3, 'a', 'c'}
>>> #Ou bien
set1.update(set2)

>>> print(set1)
{'b', 1, 2, 3, 'a', 'c'}
>>>
```

Copyright - Tout droit réservé - OFPPT

Tableaux statiques (set)

- **Autres méthodes:**

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element

Copyright - Tout droit réservé - OFPPT

Dictionnaires

- Un **dictionnaire** est une collection non **ordonnée**, **modifiable** et **indexée**.
- En Python, les dictionnaires sont écrits avec des **accolades**, et ils ont des clés et des valeurs.
- **Création d'un dictionnaire:**

```
>>> thisdict={"brand":"Ford","model":"Mustang","year":1964}

>>> print(thisdict)

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>> x = thisdict["model"] #pour accéderà unevaleurprécise

>>>
>>> print(x)
Mustang
```

- **Parcours d'un dictionnaire**

```
>>> for x in thisdict:
        print(x)


brand
model
year
```

Dictionnaires

- **Modification d'une valeur d'une clé particulière:**

```
>>> thisdict={"brand":"Ford","model":"Mustang","year":1964}

>>> thisdict["year"] =2018 #modier une valeur

>>> print(thisdict)
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

- **Retour d'une valeur particulière d'une clé:**

```
>>> thisdict={"brand":"Ford","model":"Mustang","year":1964}

>>> x = thisdict.get("model")
>>> print(x)
Mustang
```

- **Parcours les clés et les valeurs d'un dictionnaire à la fois**

```
>>> for x, y in thisdict.items():
        print(x, y)


brand Ford
model Mustang
year 1964
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Dictionnaires

- **Vérification si une clé existe:**

```
>>> if "model" in thisdict:
        print("Yes, 'model' is one of the keys in the thisdictdictionary")


Yes, 'model' is one of the keys in the thisdictdictionary
```

- **Recherche de la longueur:**

```
>>> thisdict= {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

>>> print(len(thisdict))
3
```

- **Ajout à un dictionnaire:**

```
>>> thisdict= {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

>>> thisdict["color"] = "red"

>>> print(thisdict)
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Copyright - Tout droit réservé - OFPPT

Dictionnaires

- **Suppression d'un élément d'un dictionnaire**

```
>>> thisdict= {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

>>> thisdict.pop("model")

'Mustang'
>>> print(thisdict)

{'brand': 'Ford', 'year': 1964}
>>> #oubienpour supprimer le dernier élément: thisdict.popitem()
```

- **Suppression de tous les éléments d'un dictionnaire**

```
>>> thisdict= {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

>>> del(thisdict["model"])

>>> print(thisdict)

{'brand': 'Ford', 'year': 1964}
>>> #oubienpour supprimerde dictionnaireentier: delthisdict
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Dictionnaires

- **Copie d’un dictionnaire**

```
>>> thisdict= {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

>>>
>>> mydict= thisdict.copy()

>>> print(mydict)
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- **Autres méthodes**

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
Manipuler les données

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

3- Fichiers de données

4- Bibliothèques standards

Utilisation des fichiers

- Python a plusieurs fonctions pour **créer**, **lire**, **mettre à jour** et **supprimer** des fichiers texte.
- La fonction clé pour travailler avec des fichiers en Python est **open().**
  - Elle prend deux paramètres; **nom de fichier** et **mode**.
  - Il existe quatre méthodes (modes) différentes pour ouvrir un fichier:
    - **"r" -Lecture -Par défaut. Ouvre un fichier en lecture, erreur si le fichier n'existe pas**
    - **"a" -Ajouter -Ouvre un fichier à ajouter, crée le fichier s'il n'existe pas**
    - **"w" -Écrire -Ouvre un fichier pour l'écriture, crée le fichier s'il n'existe pas**
    - **"x" -Créer -Crée le fichier spécifié, renvoie une erreur si le fichier existe**
- **Ouvrir et lire un fichier**
  - Pour ouvrir le fichier, utilisez la **fonction open()** intégrée.
  - La fonction **open**() renvoie un objet fichier, qui a une méthode **read()** pour lire le contenu du fichier

```
>>>f = open("demofile.txt", "r")
>>>print(f.read())
```

- **Renvoyez les 5 premiers caractères du fichier**:

```
>>>f = open("demofile.txt", "r")
>>>print(f.read(5))
```

Copyright - Tout droit réservé - OFPPT

Utilisation des fichiers

- **Vous pouvez renvoyer une ligne en utilisant la méthode readline ():**

```
>>>f = open("demofile.txt", "r")
>>>print(f.readline())
```

- **Ecrire dans un fichier**
  - Pour écrire dans un fichier existant, vous devez ajouter un paramètre à la fonction **open():**

```
>>>f = open("demofile2.txt", "a")
>>>f.write("Now the file has more content!")
>>>f.close()
>>>f = open("demofile2.txt", "r")#afficher le ficher après modification
>>>print(f.read())

>>>f = open("demofile3.txt", "w")
>>>f.write("Woops! I have deleted the content!")
>>>f.close()
# ouvrez et lisez le fichier après l'ajout:
>>>f = open("demofile3.txt", "r")
>>>print(f.read())
```

La méthode "w" écrasera tout le fichier.

Copyright - Tout droit réservé - OFPPT

Utilisation des fichiers

- **Fermer un fichier**
  - Il est recommandé de toujours fermer le fichier lorsque vous en avez terminé.

```
>>>f = open("demofile.txt", "r")
>>>print(f.readline())
>>>f.close()
```

- **Supprimer d'un fichier**
  - Pour supprimer un fichier, vous devez importer le module OS et exécuter sa fonction **os.remove ():**

```
>>>import os
>>>os.remove("demofile.txt")
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Format CSV

- Il existe différents formats standards de stockage de données. Il est recommandé de favoriser ces formats car il existe déjà des modules Python permettant de simplifier leur utilisation.
- Le fichier ***Comma-separated values*** **(CSV)** est un format permettant de stocker des tableaux dans un fichier texte. Chaque ligne est représentée par une ligne de texte et chaque colonne est séparée par un **séparateur** (virgule, point-virgule …).
- Les champs texte peuvent également être délimités par des guillemets.
- Lorsqu'un champ contient lui-même des guillemets, ils sont doublés afin de ne pas être considérés comme début ou fin du champ.
- Si un champ contient un signe pouvant être utilisé comme séparateur de colonne (virgule, point-virgule …) ou comme séparateur de ligne, les guillemets sont donc obligatoires afin que ce signe ne soit pas confondu avec un séparateur.

- **Données sous la forme d'un tableau**

Nom	Prenom	Age
Dubois	Marie	29
Duval	Julien "Paul"	47
Jacquet	Bernard	51
Martin	Lucie;Clara	14

- **Données sous la forme d'un fichier CSV**

```
Nom;Prénom;Age
"Dubois";"Marie";29
"Duval";"Julien ""Paul""";47
Jacquet;Bernard;51
Martin;"Lucie;Clara";14
```

Copyright - Tout droit réservé - OFPPT

Format CSV

- Le module csv de Python permet de simplifier l'utilisation des fichiers CSV
- **Lecture d'un fichier CSV**
  - Pour lire un fichier CSV, il faut ouvrir un flux de lecture de fichier et ouvrir à partir de ce flux un lecteur CSV.

```
import csv
fichier = open("noms.csv", "rt")
lecteurCSV = csv.reader(fichier,delimiter=";")  # Ouverture du lecteur CSV en lui fournissant le caractère séparateur (ici ";")
for ligne in lecteurCSV:
        print(ligne)
fichier.close()
```

```
['Nom ', 'Prenom', 'Age']
['Dubois', 'Marie', '29']
['Duval', 'Julien "Paul" ', '47']
['Jacquet', 'Bernard', '51']
['Martin', 'Lucie;Clara', '14']
```

  - Il est également possible de lire les données et obtenir un dictionnaire par ligne contenant les données en utilisant **DictReader** au lieu de **reader**

```
import csv
fichier = open("noms.csv", "rt")
lecteurCSV = csv.DictReader(fichier,delimiter=";")
for ligne in lecteurCSV:
        print(ligne
fichier.close()
```

```
{'Nom ': 'Dubois', 'Prenom': 'Marie', 'Age': '29'}
{'Nom ': 'Duval', 'Prenom': 'Julien "Paul" ', 'Age': '47'}
{'Nom ': 'Jacquet', 'Prenom': 'Bernard', 'Age': '51'}
{'Nom ': 'Martin', 'Prenom': 'Lucie;Clara', 'Age': '14'}
```

Copyright - Tout droit réservé - OFPPT

Format CSV

- **Écriture dans un fichier CSV**
  - À l'instar de la lecture, on ouvre un flux d'écriture et on ouvre un écrivain CSV à partir de ce flux :

```
import csv
fichier = open("annuaire.csv", "wt")
ecrivainCSV = csv.writer(fichier,delimiter=";")
ecrivainCSV.writerow(["Nom","Prénom","Téléphone"])      # On écrit la ligne d'en-tête avec le titre des colonnes
ecrivainCSV.writerow(["Dubois","Marie","0198546372"])
ecrivainCSV.writerow(["Duval","Julien \"Paul\"","0399741052"])
ecrivainCSV.writerow(["Jacquet","Bernard","0200749685"])
ecrivainCSV.writerow(["Martin","Julie;Clara","0399731590"])
fichier.close()
```

```
Nom;Prénom;Téléphone
Dubois;Marie;0198546372
Duval;"Julien ""Paul""";0399741052
Jacquet;Bernard;0200749685
Martin;"Julie;Clara";0399731590
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Format CSV

- Il est également possible d'écrire le fichier en fournissant un dictionnaire par ligne à condition que chaque dictionnaire possède les mêmes clés.
- Il faut également fournir la liste des clés des dictionnaires avec l'argument **fieldnames** :

```
bonCommande = [
        {"produit":"cahier", "reference":"F452CP", "quantite":41, "prixUnitaire":1.6},
        {"produit":"stylo bleu", "reference":"D857BL", "quantite":18, "prixUnitaire":0.95},
        {"produit":"stylo noir", "reference":"D857NO", "quantite":18, "prixUnitaire":0.95},
        {"produit":"équerre", "reference":"GF955K", "quantite":4, "prixUnitaire":5.10},
        {"produit":"compas", "reference":"RT42AX", "quantite":13, "prixUnitaire":5.25}
]
fichier = open("bon-commande.csv", "wt")
ecrivainCSV = csv.DictWriter(fichier,delimiter=";",fieldnames=bonCommande[0].keys())
ecrivainCSV.writeheader()       # On écrit la ligne d'en-tête avec le titre des colonnes
for ligne in bonCommande:
        ecrivainCSV.writerow(ligne)
fichier.close()
```

```
reference;quantite;produit;prixUnitaire
F452CP;41;cahier;1.6
D857BL;18;stylo bleu;0.95
D857NO;18;stylo noir;0.95
GF955K;4;équerre;5.1
RT42AX;13;compas;5.25
```

Copyright - Tout droit réservé - OFPPT

OFPPT

Format JSON

- Le format JavaScript Object Notation (JSON) est issu de la notation des objets dans le langage JavaScript.
- Il s'agit aujourd'hui d'un format de données très répandu permettant de stocker des données sous une forme structurée.
- Il ne comporte que des associations **clés → valeurs** (à l'instar des dictionnaires), ainsi que des listes ordonnées de valeurs (comme les listes en Python).
- Une valeur peut être une autre association clés → valeurs, une liste de valeurs, un entier, un nombre réel, une chaîne de caractères, un booléen ou une valeur nulle.
- Sa syntaxe est similaire à celle des dictionnaires Python.
- <u>**Exemple de fichier JSON**</u>

```
{
  "Dijon":{
          "nomDepartement": "Côte d'Or",
          "codePostal": 21000,
          "population": {
                  "2006": 151504,
                  "2011": 151672,
                  "2014": 153668
          }
  },
  "Troyes":{
          "nomDepartement": "Aube",
          "codePostal": 10000,
          "population": {
                  "2006": 61344,
                  "2011": 60013,
                  "2014": 60750
          }
  }
}
```

Copyright - Tout droit réservé - OFPPT

Format JSON

- **Lire un fichier JSON**
  - La fonction loads (texteJSON) permet de décoder le texte JSON passé en argument et de le transformer en dictionnaire ou une liste.

```
import json
fichier = open("villes.json","rt")
villes = json.loads(fichier.read())
print(villes)
fichier.close()
```

```
{'Troyes': {'population': {'2006': 61344, '2011': 60013, '2014': 60750}, 'codePostal': 10000,
'nomDepartement': 'Aube'}, 'Dijon': {'population': {'2006': 151504, '2011': 151672, '2014':
153668}, 'codePostal': 21000, 'nomDepartement': "Côte d'Or"}}
```

- **Écrire un fichier JSON**
  - la fonction **dumps(variable, sort_keys=False)** transforme un dictionnaire ou une liste en texte JSON en fournissant en argument la variable à transformer.
  - La variable **sort_keys** permet de trier les clés dans l'ordre alphabétique.

```
import json
quantiteFournitures = {"cahiers":134, "stylos":{"rouge":41,"bleu":74}, "gommes": 85}
fichier = open("quantiteFournitures.json","wt")
fichier.write(json.dumps(quantiteFournitures))
fichier.close()
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
Manipuler les données

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

3- Fichiers de données

4- Bibliothèques standards

Bibliothèques standards

- Python dispose d'une très riche bibliothèque de modules (classes (types d'objets), fonctions, constantes, …) étendant les capacités du langage dans de nombreux domaines: nouveaux types de données, interactions avec le système, gestion des fichiers et des processus, protocoles de communication (internet, mail, FTP, etc.), multimédia, etc.
- Certains des modules importants sont:
  - **math** pour les utilitaires mathématiques
  - **re** pour les expressions régulières
  - **Json** pour travailler avec JSON
  - **Datetime** pour travailler avec des dates

Le module Math

- Le module math nous fournit un accès à de nombreuses fonctions permettant de réaliser des opérations mathématiques comme le calcul d'un sinus, cosinus, d'une tangente, d'un logarithme ou d'une exponentielle
- Les fonctions les plus couramment utilisées sont les suivantes :
  - **ceil()** et **floor()** renvoient l'arrondi du nombre passé en argument en arrondissant respectivement à l'entier supérieur et inférieur ;
  - **fabs()** renvoie la valeur absolu d'un nombre passé en argument ;
  - **isnan()** renvoie True si le nombre passé en argument est NaN = Not a Number (pas un nombre en français) ou False sinon ;
  - **exp()** permet de calculer des exponentielles ;

Copyright - Tout droit réservé - OFPPT

Le module Math

- **log()** permet de calculer des logarithmes ;
- La fonction **sqrt()** permet de calculer la racine carrée d'un nombre ;
- **cos(), sin() et tan()** permettent de calculer des cosinus, sinus et tangentes et renvoient des valeurs en radians.

- les fonctions de ce module ne peuvent pas être utilisées avec des nombres complexes. Pour cela, il faudra plutôt utiliser les fonctions du module **cmath**.
- Le module math définit également des constantes mathématiques utiles comme pi ou le nombre de Neper, accessibles via math.pi et math.e.

```
import math
print(math.ceil(3.1))
print(math.ceil(3.9))
print(math.floor(3.1))
print(math.floor(-4))
print(math.pi)
```

```
4
4
3
-4
3.141592653589793
```

Copyright - Tout droit réservé - OFPPT

Le module random

- random fournit des outils pour générer des nombres pseudo-aléatoires de différentes façons.
- La fonction random() est le plus utilisée du module. Elle génère un nombre à virgule flottante aléatoire de façon uniforme dans la plage semi-ouverte [0.0, 1.0).
- La fonction uniform() génère un nombre à virgule flottante aléatoire compris dans un intervalle. Cette fonction a deux arguments : le premier nombre représente la borne basse de l'intervalle tandis que le second représente la borne supérieure.

```
import random
print(random.random())
print(random.random())
print(random.uniform(10 , 100))
```

```
0.28349953961279983
0.007815000784710868
44.309364272132456
```

Copyright - Tout droit réservé - OFPPT

PARTIE 4
DEPLOYER LA SOLUTION python

Dans ce module, vous allez :

- Déployer une solution Python
- Déboguer une solution Python

6 heures

CHAPITRE 1
DEBOGUER LE CODE PTHON

Ce que vous allez apprendre dans ce chapitre :

- Différencier entre les erreurs de syntaxe et celles de compilation
- Maitriser la gestion des exceptions en Python
- Maitriser la manipulation de l'outil de débogage de Python

3 heures

CHAPITRE 1
DEBOGUER LE CODE PTHON

1- Gestion des erreurs (compilation-syntaxe)

2-Débogage

3-Outils de suivi et de visualisation de l'exécution d'un code python

Erreurs de syntaxe

- Les erreurs de syntaxe sont des erreurs d'analyse du code

Exemple:

```
>>> while True print('Hello world')
SyntaxError: invalid syntax
```

- L'analyseur indique la ligne incriminée et affiche une petite « flèche » pointant vers le premier endroit de la ligne où l'erreur a été détectée.
- L'erreur est causée par le symbole placé avant la flèche.
- Dans cet exemple, la flèche est sur la fonction print() car il manque deux points (':') juste avant. Le nom du fichier et le numéro de ligne sont affichés pour vous permettre de localiser facilement l'erreur lorsque le code provient d'un script.

Copyright - Tout droit réservé - OFPPT

Exceptions

- Même si une instruction ou une expression est syntaxiquement correcte, elle peut générer une erreur lors de son exécution.
- Les erreurs détectées durant l'exécution sont appelées des **exceptions** et ne sont pas toujours fatales
- La plupart des exceptions toutefois ne sont pas prises en charge par les programmes, ce qui génère des messages d'erreurs comme celui-ci :

```
>>> 10 * (1/0)

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    10 * (1/0)
ZeroDivisionError: division by zero
```

- La dernière ligne du message d'erreur indique ce qui s'est passé. Les exceptions peuvent être de différents types et ce type est indiqué dans le message : le types indiqué dans l'exemple est **ZeroDivisionError**

Copyright - Tout droit réservé - OFPPT

Instruction try-except

- Python présente un **mécanisme d'exception** qui permet de gérer des exécutions exceptionnelles qui ne se produisent qu'en cas d'erreur.
- La construction de base à utiliser est l'instruction **try-except** qui se compose de deux blocs de code.
- On place le code « risqué » dans le bloc try et le code à exécuter en cas d'erreur dans le bloc except.
- **Exemple:**
- On souhaite calculer l' âge saisi par l'utilisateur en soustrayant son année de naissance à 2016. Pour cela, il faut convertir la valeur de la variable birthyear en un int. Cette conversion peut échouer si la chaine de caractères entrée par l'utilisateur n'est pas un nombre.

```
birthyear = input('Année de naissance ? ')

try:
    print('Tu as', 2016 - int(birthyear), 'ans.')
except:
    print('Erreur, veuillez entrer un nombre.')

print('Fin du programme.')
```

Copyright - Tout droit réservé - OFPPT

Instruction try-except

```
birthyear = input('Année de naissance ? ')

try:
    print('Tu as', 2016 - int(birthyear), 'ans.')
except:
    print('Erreur, veuillez entrer un nombre.')

print('Fin du programme.')
```

1er cas

- Si l'utilisateur entre un nombre entier, l'exécution se passe sans erreur et son âge est calculé et affiché

```
Année de naissance ? 1994
Tu as 22 ans.
Fin du programme.
```

2ème cas

- Si l'utilisateur entre une chaine de caractères quelconque, qui ne représente pas un nombre entier, un message d'erreur est affiché

```
Année de naissance ? deux
Erreur, veuillez entrer un nombre.
Fin du programme.
```

- Dans le premier cas, la conversion s'est passée normalement, et le bloc try a donc pu s'exécuter intégralement sans erreur.
- Dans le second cas, une erreur se produit dans le bloc try, lors de la conversion. L'exécution de ce bloc s'arrête donc immédiatement et passe au bloc except, avant de continuer également après l'instruction try-except.

Copyright - Tout droit réservé - OFPPT

Type Exception

- Lorsqu'on utilise l'instruction **try-except**, le bloc **except** capture toutes les erreurs possibles qui peuvent survenir dans le bloc try correspondant.
- Une **exception** est en fait représentée par un objet, instance de la classe Exception.
- On peut récupérer cet objet en précisant un nom de variable après **except**

```
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)
```

- On récupère donc l'objet de type Exception dans la variable e.
- Dans le bloc except, on affiche son type et sa valeur.

Copyright - Tout droit réservé - OFPPT

Type Exception

Voici deux exemples d'exécution qui révèlent deux types d'erreurs différents :

- Si on ne fournit pas un nombre entier, il ne pourra être converti en **int** et une erreur de type **ValueError** se produit :

```
>>> try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)

a ? trois
<class 'ValueError'>
invalid literal for int() with base 10: 'trois'
```

- Si on fournit une valeur de 00 pour b, on aura une division par zéro qui produit une erreur de type ZeroDivisionError :

```
>>> try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)

a ? 5
b ? 0
<class 'ZeroDivisionError'>
division by zero
```

```
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)
except Exception as e:
    print(type(e))
    print(e)
```

Copyright - Tout droit réservé - OFPPT

Capture d’erreur spécifique

- Chaque type d'erreur est donc défini par une classe spécifique.
- Il est possible d’associer plusieurs blocs except à un même bloc try, pour exécuter un code différent en fonction de l'erreur capturée.
- Lorsqu'une erreur se produit, les blocs except sont parcourus l'un après l'autre, du premier au dernier, jusqu'à en trouver un qui corresponde à l'erreur capturée.

Exemple:

```
>>> try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)

except ValueError:
    print('Erreur de conversion.')
except ZeroDivisionError:
    print('Division par zéro.')
except:
    print('Autre erreur.')
```

- Lorsqu'une erreur se produit dans le bloc **try** l'un des blocs **except** seulement qui sera exécuté, selon le type de l'erreur qui s'est produite.
- Le dernier bloc **except** est là pour prendre toutes les autres erreurs.
- Lorsqu'une erreur se produit dans le bloc **try** l'un des blocs **except** seulement qui sera exécuté, selon le type de l'erreur qui s'est produite.
- Le dernier bloc **except** est là pour prendre toutes les autres erreurs.

- **L'ordre des blocs except est très important et il faut les classer du plus spécifique au plus général, celui par défaut devant venir en dernier.**

Copyright - Tout droit réservé - OFPPT

Gestionnaire d'erreur partagé

- Il est possible d'exécuter le même code pour différents types d'erreur, en les listant dans un tuple après le mot réservé **except**.
- Si on souhaite exécuter le même code pour une erreur de conversion et de division par zéro, il faudrait écrire :

```
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print(a, '/', b, '=', a / b)

except (ValueError, ZeroDivisionError) as e:
    print('Erreur de calcul :', e)
except:
    print('Autre erreur.')
```

Copyright - Tout droit réservé - OFPPT

Bloc finally

- le mot réservé **finally** permet d'introduire un bloc qui sera exécuté soit après que le bloc try se soit exécuté complètement sans erreur, soit après avoir exécuté le bloc except correspondant à l'erreur qui s'est produite lors de l'exécution du bloc try.
- On obtient ainsi une instruction **try-except-finally**

```
print('Début du calcul.')
try:
    a = int(input('a ? '))
    b = int(input('b ? '))
    print('Résultat :', a / b)
except:
    print('Erreur.')
finally:
    print('Nettoyage de la mémoire.')
print('Fin du calcul.')
```

- Si l'utilisateur fournit des valeurs correctes pour a et b l'affichage est le suivant:

```
Début du calcul.
a ? 2
b ? 8
Résultat : 0.25
Nettoyage de la mémoire.
Fin du calcul.
```

- si une erreur se produit l'affichage est le suivant:

```
Début du calcul.
a ? 2
b ? 0
Erreur.
Nettoyage de la mémoire.
Fin du calcul.
```

Dans les 2 cas le bloch finally a été exécuté

Copyright - Tout droit réservé - OFPPT

Génération d'erreur

- Il est possible de générer une erreur dans un programme grâce à l'instruction **raise**.
- Il suffit en fait simplement d'utiliser le mot réservé **raise** suivi d'une référence vers un objet représentant une exception.
- **Exemple:**

```
def fact(n):
    if n < 0:
        raise ArithmeticError()
    if n == 0:
        return 1
    return n * fact(n - 1)

print(fact(-12))
```

- si n est strictement négatif, une exception de type **ArithmeticError** est généré

- Le programme suivant permet de capture spécifiquement l'exception de type **ArithmeticError** lors de l’appel de la fonction **fact**

```
try:
    n = int(input('Entrez un nombre : '))
    print(fact(n))
except ArithmeticError:
    print('Veuillez entrer un nombre positif.')
except:
    print('Veuillez entrer un nombre.')
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
DEBOGUER LE CODE PTHON

1- Gestion des erreurs (compilation-syntaxe)

2-Débogage

3-Outils de suivi et de visualisation de l'exécution d'un code python

Débogueur

- Un débogueur est un outil de développement qui s’attache à une application en cours d’exécution et qui permet d’inspecter le code
    - il permet d'exécuter le programme pas-à-pas
    - d'afficher la valeur des variables à tout moment
    - de mettre en place des points d'arrêt sur des conditions ou sur des lignes du programme.
- De nombreux débogueurs permettent, en plus de l'observation de l'état des registres processeurs et de la mémoire, de les modifier avant de rendre la main au programme débogué.
- Ils peuvent alors être utilisés pour localiser certaines protections logicielles et les désactiver, amenant à la conception d'un crack.

Copyright - Tout droit réservé - OFPPT

CHAPITRE 1
DEBOGUER LE CODE PTHON

1- Gestion des erreurs (compilation-syntaxe)

2-Débogage

3-Outils de suivi et de visualisation de l'exécution d'un code python

Débogueur Python

•Python est livré avec un débogueur intégré appelé **Python debugger** ou **pdb**

•**Pdb** est un environnement de débogage interactif pour les programmes Python il permet:

- Une visualisation des valeurs des variables
- Une visualisation de l'exécution du programme étape par étape.
- Une compréhension de ce que fait le programme et de trouver des bogues dans la logique.

- **Exécution du débogueur à partir d'un interpréteur interactif**
  - Utilisez **run ()** ou **runeval ()**
  - L'argument de run () est une expression de chaîne qui peut être évaluée par l'interpréteur Python
  - Le débogueur l'analysera, puis interrompra l'exécution juste avant l'évaluation de la première expression.
  - **set_trace ()** est une fonction Python qui peut être appelé à partir de n'importe quel point d'un programme

Copyright - Tout droit réservé - OFPPT

Débogueur Python

- **Exemple:** Soit le code Python suivant

Python

```python
# debug_code.py
def log(number):
    print(f'Processing {number}')
    print(f'Adding 2 to number: {number + 2}')

def looper(number):
    for i in range(number):
        log(i)


if __name__ == '__main__':
    looper(5)
```

Définition de la procédure log

Définition de la procédure looper

Appel de log(i) pour i de 1 à number

Programme principal Appel de looper pour number =5

- **Lancement de déboguer avec run()**

```
>>> import pdb
>>> import debug_code
>>> pdb.run('debug_code.looper(5)')
> <string>(1)<module>()
(Pdb)
```

Importation du débogueur

Importation du fichier à déboguer

Lancement du déboguer à l'aide la fonction **run()**

La ligne suivante est préfixée par (Pdb). Cela signifie que vous êtes maintenant dans le débogueur. **Succès!**

Copyright - Tout droit réservé - OFPPT

Débogueur Python

- Pour exécuter un code dans le débogueur, tapez **continue** ou **c** . Cela exécutera votre code jusqu'à ce que l'un des événements suivants se produise :
    - Le code lève une exception (une erreur).
    - Vous arrivez à un point d'arrêt (expliqué plus loin).
    - Le code se termine.

```
Processing 0
Adding 2 to number: 2
Processing 1
Adding 2 to number: 3
Processing 2
Adding 2 to number: 4
Processing 3
Adding 2 to number: 5
Processing 4
Adding 2 to number: 6
>>>
```

- **Le mode Trace**

- Il est plus pratique de démarrer le débogueur depuis l'intérieur d'un programme en utilisant la fonction **set_trace ()**
    - Le débogueur Python vous permet d'importer le module pbd et d'ajouter directement un point d'arrêt (**breakpoint**) à votre code à l'aide de la fonction **set_trace ()**

Copyright - Tout droit réservé - OFPPT

Débogueur Python

- **Exemple:**

```
def log(number):
  print(f'Processing {number}')
  print(f'Adding 2 to number: {number + 2}')

def looper(number):
    for i in range(number):
        import pdb; pdb.set_trace()
        log(i)

if __name__ == '__main__':
    looper(5)
```

Un breakpoint
Arrêt avant d'exécuter
l'instruction log(i)

```
(Pdb) c
Processing 0
Adding 2 to number: 2
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(7)looper()
-> import pdb; pdb.set_trace()
(Pdb) c
Processing 1
Adding 2 to number: 3
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(8)looper()
-> log(i)
(Pdb) c
Processing 2
Adding 2 to number: 4
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(7)looper()
-> import pdb; pdb.set_trace()
(Pdb) c
Processing 3
Adding 2 to number: 5
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(8)looper()
-> log(i)
(Pdb) c
Processing 4
Adding 2 to number: 6
>>>
```

Trace d'exécution du programme : **number=2,3,4→5**

Copyright - Tout droit réservé - OFPPT

Débogueur Python

- Au niveau de chaque **breakpoint**, il est possible de vérifier le contenu des variables
- **Exemple:**

```python
def log(number):
  print(f'Processing {number}')
  print(f'Adding 2 to number: {number + 2}')

def looper(number):
    for i in range(number):
        import pdb; pdb.set_trace()
        log(i)

if __name__ == '__main__':
    looper(5)
```

```
-> log(i)
(Pdb) c
Processing 0
Adding 2 to number: 2
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(9)looper()
-> import pdb; pdb.set_trace()
(Pdb) i
1
(Pdb) c
Processing 1
Adding 2 to number: 3
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(10)looper()
-> log(i)
(Pdb) i
2
(Pdb)
```

Demande de continuer l’exécution avec c

Demande de la valeur de **i**

Copyright - Tout droit réservé - OFPPT

Débogueur Python

- Au niveau de chaque **breakpoint**, il est possible de changer le contenu des variables
- **Exemple:**

```python
def log(number):
  print(f'Processing {number}')
  print(f'Adding 2 to number: {number + 2}')

def looper(number):
    for i in range(number):
        import pdb; pdb.set_trace()
        log(i)

if __name__ == '__main__':
    looper(5)
```

```
== RESTART: C:/Users/DELL/AppData/Local/Programs/Python/Python39/debug_code.py =
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(10)looper()
-> log(i)
(Pdb) i=20
(Pdb) c
Processing 20
Adding 2 to number: 22
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(9)looper()
-> import pdb; pdb.set_trace()
(Pdb)
```

Modification de la valeur de **i**

Copyright - Tout droit réservé - OFPPT

Débogueur Python

- Cas de plusieurs **breakpoints**

```
def log(number):
  print(f'Processing {number}')
  import pdb; pdb.set_trace()
  print(f'Adding 2 to number: {number + 2}')



def looper(number):
    for i in range(number):
        import pdb; pdb.set_trace()
        log(i)

if __name__ == '__main__':
    looper(5)
```

```
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(11)looper()
-> log(i)
(Pdb) c
Processing 0
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(4)log()
-> print(f'Adding 2 to number: {number + 2}')
(Pdb) c
Adding 2 to number: 2
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(10)looper()
-> import pdb; pdb.set_trace()
(Pdb) c
Processing 1
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(4)log()
-> print(f'Adding 2 to number: {number + 2}')
(Pdb) c
Adding 2 to number: 3
> c:\users\dell\appdata\local\programs\python\python39\debug_code.py(11)looper()
-> log(i)
(Pdb)
```

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
DÉPLOYER UNE SOLUTION PYTHON

Ce que vous allez apprendre dans ce chapitre :

- Connaitre les outils de déploiement en Python
- Créer un fichier d'installation en Python
- Générer de la documentation en Python

3 heures

CHAPITRE 2
DÉPLOYER UNE SOLUTION PYTHON

1-Outils de déploiement de solution Python

2- Création de fichiers d'installation de solution Python

3- Documentation du programme

Packaging Python

- le packaging est une étape importante lorsque l'on veut partager et réutiliser du code sans avoir à le dupliquer dans chacun de nos projets
- **pip** est un gestionnaire de paquets utilisé pour installer et gérer des paquets écrits en Python.
- De nombreux paquets peuvent être trouvés sur le PyPI
- PyPI est le dépôt tiers officiel du langage de programmation Python
- PyPI est de doter la communauté des développeurs Python d'un catalogue complet recensant tous les paquets Python libres
- Depuis 2003 c'est un index central accessible avec Distutils, créé par Richard Jones

Distutils

- Distutils a été initié par Greg Ward en 1998.
- Organise un projet Python autour d'un fichier setup.py.
- Distutils permit de configurer, compiler, deployer, etc
- Possède des extensions pour l'interface avec C, Fortran.
- Ajouté à la librairie standard de Python 1.6 en 2000.

Copyright - Tout droit réservé - OFPPT

Setuptools

- Initié par Phillip Eby en 2005, c’est une extension des Distutils.
- Permet de créer des packages au format binaires (eggs).
- Un outil d’installation automatique (easy_install).
- Génère automatiquement le fichier MANIFEST.in avec l’aide du gestionnaire de versions,
- Capable de fouiller sur le web pour trouver des packages

Distribute

- En 2008, Setuptools est une partie vitale de Python mais son développement ralentit.
- Cet autre projet voit le jour et supporte Python 3 (Ian Bicking)
- Setuptools et Distribute sont deux packages distincts et concurrents.
- Il est fortement conseillé de choisir entre ces deux outils (pas de cohabitation)

Copyright - Tout droit réservé - OFPPT

Packaging un projet Python

- Certaines commandes nécessitent une version plus récente de pip, alors commencez par vous assurer que la dernière version est installée par la commande:

```
py -m pip install --upgrade pip
```

- Créez localement la structure de fichiers suivante:

```
packaging_tutorial/
└── src/
    └── example_package/
        ├── __init__.py
        └── example.py
```

- **__init__.py** est requis pour importer le répertoire en tant que package et doit être vide.
- **example.py** est un exemple de module dans le package qui pourrait contenir la logique (fonctions, classes, constantes, etc.) de votre package.

- Ouvrez le fichier example.py et saisissez le contenu suivant:

```
def add_one(number):
    return number + 1
```

Copyright - Tout droit réservé - OFPPT

Packaging un projet Python

- Une fois la structure des fichiers est définie , toutes les commandes suivantes seront exécutées dans le répertoire packaging_tutorial
- Ajouter des fichiers qui seront utilisés pour préparer le projet pour la distribution. La structure du projet ressemblera à ceci :

```
packaging_tutorial/
├── LICENSE
├── pyproject.toml
├── README.md
├── setup.cfg
├── src/
│   └── example_package/
│       ├── __init__.py
│       └── example.py
└── tests/
```

- **Création de pyproject.toml**
  - Le fichier pyproject.toml indique aux outils de construction (comme pip et build) ce qui est nécessaire pour construire votre projet.
  - On utilise dans ce qui suit **setuptools**, alors ouvrez pyproject.toml et saisissez son contenu

```
[build-system]
requires = [
    "setuptools>=42",
    "wheel"
]
build-backend = "setuptools.build_meta"
```

- **build-system.requires** donne une liste des packages nécessaires pour construire votre package.
- **build-system.build-backend** est le nom de l'objet Python qui sera utilisé pour effectuer la construction

Copyright - Tout droit réservé - OFPPT

Packaging un projet Python

- **Création du fichier setup.py**
  - setup.py est le script de construction pour setuptools.
  - Il indique à setuptools votre package (comme le nom et la version) ainsi que les fichiers de code à inclure.
  - setup() prend plusieurs
  - Exemple:
    - **Name**: est le nom de distribution de votre package.
    - **Version:** est la version du paquet
    - **author** et **author_email** sont utilisés pour identifier l'auteur du package
    - **Description**: est une brève description du package
    - **url** est l'URL de la page d'accueil du projet.
    - **project_urls**: permet de répertorier un nombre quelconque de liens supplémentaires à afficher sur PyPI

```
import setuptools

with open("README.md", "r", encoding="utf-8") as fh:
    long_description = fh.read()

setuptools.setup(
    name="example-pkg-YOUR-USERNAME-HERE",
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/pypa/sampleproject",
    project_urls={
        "Bug Tracker":
"https://github.com/pypa/sampleproject/issues",
    },
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    package_dir={"": "src"},
    packages=setuptools.find_packages(where="src"),
    python_requires=">=3.6",
)
```

Copyright - Tout droit réservé - OFPPT

Packaging un projet Python

- **Création du fichier README.md**
  - Ouvrez README.md et saisissez le contenu suivant. Vous pouvez personnaliser cela si vous le souhaitez.

```
# Example Package

This is a simple example package. You can use
[Github-flavored Markdown](https://guides.github.com/features/mastering-markdown/)
to write your content.
```

- **Création d'une LICENCE**
  - Il est important que chaque package téléchargé dans l'index des packages Python inclue une licence.
  - Cela indique aux utilisateurs qui installent votre package les conditions dans lesquelles ils peuvent utiliser votre package.

```
Copyright (c) 2018 The Python Packaging Authority

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
………………………………….
```

Génération d'archives de distribution

L'étape suivante consiste à générer des packages de distribution pour le package. Ce sont des archives qui sont téléchargées dans le PyPI et peuvent être installées par pip

- Exécutez maintenant cette commande à partir du même répertoire où se trouve pyproject.toml

```
py -m build
```

- Cette commande devrait générer beaucoup de texte et une fois terminée, elle devrait générer deux fichiers dans le répertoire dist

```
dist/
  example_package_YOUR_USERNAME_HERE-0.0.1-py3-none-any.whl
  example_package_YOUR_USERNAME_HERE-0.0.1.tar.gz
```

Téléchargement des archives de diffusion

- Commencer par créer un compte sur TestPyPI, qui est une instance distincte de l'index de package destiné aux tests et à l'expérimentation.
- Pour accéder à un compte, accédez à **https://test.pypi.org/account/register/** et suivez les étapes sur cette page.
- Vous devrez également vérifier votre adresse e-mail avant de pouvoir télécharger des packages.
- Pour télécharger en toute sécurité votre projet, vous aurez besoin d'un jeton d'API PyPI. **Créez-en un sur https://test.pypi.org/manage/account/#api-tokens**

Copyright - Tout droit réservé - OFPPT

Téléchargement des archives de diffusion

- Maintenant que vous êtes enregistré, vous pouvez utiliser **Twine** pour télécharger les packages de distribution.
- Vous devrez installer **Twine** :

```
py -m pip install --upgrade twine
```

- Exécutez Twine pour télécharger toutes les archives sous dist

```
py -m twine upload --repository testpypi dist/*
```

- Un nom d'utilisateur et un mot de passe vous seront demandés. Pour le nom d'utilisateur, utilisez __token__. Pour le mot de passe, utilisez la valeur du jeton, y compris le préfixe pypi-
- Une fois la commande terminée, vous devriez voir une sortie semblable à celle-ci :

```
Uploading distributions to https://test.pypi.org/legacy/
Enter your username: [your username]
Enter your password:
Uploading example_package_YOUR_USERNAME_HERE-0.0.1-py3-none-any.whl
100%|████████████████████| 4.65k/4.65k [00:01<00:00, 2.88kB/s]
Uploading example_package_YOUR_USERNAME_HERE-0.0.1.tar.gz
100%|████████████████████| 4.25k/4.25k [00:01<00:00, 3.05kB/s]
```

- Une fois téléchargé, votre package doit être visible sur TestPyPI, par **exemple, https://test.pypi.org/project/example-pkg-YOUR-USERNAME-HERE**

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
DÉPLOYER UNE SOLUTION PYTHON

1-Outils de déploiement de solution Python

2- Création de fichiers d'installation de solution Python

3- Documentation du programme

PyInstaller

- **PyInstaller** est un paquet de PyPI gestionnaire de librairies pour Python
- **PyInstaller** regroupe une application Python et toutes ses dépendances dans un seul package. L'utilisateur peut exécuter l'application packagée sans installer d'interpréteur Python ou de modules
- **PyInstaller** est directement fournit avec les versions de Python3.5 ou plus et avec Python2

Etapes de création de fichiers exécutable

1. Exécutez :

```
pip install pywin32
```

2. Ajoutez C:\Program Files\Pyton\PythonX.X\Scripts au PATH de Windows
3. Exécutez:

```
pip3 install pyinstaller
```

4. Placez-vous dans le dossier où se trouve le fichier et exécuter:

```
pyinstaller votre_script_en_python.py
```

→ L'exécutable est alors placé dans le dossier Dist à l'emplacement du script

Copyright - Tout droit réservé - OFPPT

CHAPITRE 2
DÉPLOYER UNE SOLUTION PYTHON

1-Outils de déploiement de solution Python

2- Création de fichiers d'installation de solution Python

3- Documentation du programme

Sphinx

- Sphinx est un outil de génération de documentation automatique utilisant le format ReStructured Text. Après une phase de configuration, il permet de générer assez facilement la documentation d'un projet python en un format pdf ou html.
- Installation de Sphinx en utilisant la commande:

```
py -m pip install -U sphinx
```

- Créez un répertoire docs dans votre projet pour contenir votre documentation

```
cd /path/to/project
mkdir docs
```

- Exécutez **sphinx-quickstart** ou **python -m sphinx.cmd.quickstart** dans le répertoire docs

```
cd docs
sphinx-quickstart
```

- Cela vous guide à travers certaines configurations de base à créer un fichier index.rst ainsi qu'un fichier conf.py

Copyright - Tout droit réservé - OFPPT