Démarche de Résolution de Problèmes

I- Introduction :

De nos jours l'informatique est présente dans tous les domaines de notre vie courante, en effet, tous les gens réalisent leurs travaux grâce ou à l'aide de l'ordinateur, mais chacun a son propre besoin. D'où la nécessité de créer divers programmes pour satisfaire les besoins de chacun.

- Comment alors est-il créé un programme ?
- Quelles sont les étapes à suivre pour arriver à écrire un programme résolvant un problème donné ?

II- Les étapes à suivre ;

La résolution d'un problème passe impérativement par 5 étapes essentielles.

1- Position du problème :

Le problème est souvent posé par un demandeur de solution informatique. C'est le cas du pharmacien, d'un élève, d'un médecin, …

Parfois, ces demandeurs ne savent pas exprimer leur besoin avec précision. L'objectif de cette étape est "**bien formuler le problème pour pouvoir le résoudre correctement.**"

Activité :

Un élève rencontre le problème suivant : ➔Comment je peux convertir 450 m en cm ???

Aidez-le à bien exprimer son besoin en reformulant son problème !

⇨ Le problème est : "**Écrire un programme qui permet de convertir une distance DM donnée en mètre en sa valeur Dcm exprimée en centimètres.**"

2- Spécification et Analyse du problème :

L'objectif de cette étape est de bien comprendre le problème, déterminer les formules de calcul, les règles de gestion, …

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

- Les données nécessaires à la réalisation du traitement (…….) ;
- Les actions réalisées pour atteindre le résultat (…….) ;
- Les résultats souhaités (……..

C'est-à-dire :

Donnée = ……..

Traitement = ……..

Résultat = ……..

Figure 1: analyse du problème Conversion.

Une fois les données et les résultats sont précisés, il est temps de définir pour chaque objet son type (entier, réel, caractère, …) et sa nature (constante ou variable) pour leurs réserver les emplacements appropriés en mémoire (***RAM***). On dresse alors le **T**ableau de **D**éclaration des **O**bjets (**TDO**) ayant la forme suivante :

Objet	Type/Nature	Rôle

Le TDO correspondant à notre problème de Conversion sera comme suit :

Objet	Type/Nature	Rôle
.....	.....	.....
.....	.....	.....

3- Écriture de l'Algorithme :

Après avoir terminé l'analyse, il faut mettre les instructions dans leur ordre logique d'exécution. On obtient ainsi un algorithme.

Définition : "Un algorithme est une suite structurée et finie d'actions ou d'instructions permettant de résoudre un problème".

L'algorithme relatif à notre problème de la conversion est le suivant :

Algorithme : .....
Début
.....
.....
.....
Fin.

Un algorithme utilise les conventions suivantes :

- Le verbe "**Lire**" est utilisé pour la **saisie** des données.
- Le verbe "**Écrire**" est utilisé pour **l'affichage** des résultats sur l'écran.
- Le signe "←" est utilisé pour **mettre** une valeur dans une variable.

4- Traduction en langage machine :

Une fois l'algorithme est établi, on doit penser à son exécution par l'ordinateur. Mais pensez-vous qu'un algorithme tel que vous avez rédigé soit directement **compréhensible** par l'ordinateur ? La réponse est bien "**Non**". Mais pourquoi ? Car tout simplement, votre algorithme est écrit en Français qui est une langue Humaine, or l'ordinateur ne comprend aucune langue Humaine, il ne peut comprendre que le langage machine écrit en bits (0 et 1).

On est donc face à cette situation :

On a une solution écrite sous forme d'algorithme exprimé en langue Humaine non compréhensible par l'ordinateur.

Un ordinateur qui doit exécuter cette solution mais en langage machine.

Algorithme : Conversion
Début
Lire (DM)
Dcm←DM*100
Écrire (Dcm)
Fin.

Comment passer de l'algorithme au langage machine ???

00001011111010111110
00011111111010101110
10101000001110101010
00001011111010111110
00011111111010101110
10101000001110101010

⇨ ***Il faut traduire (.....) l'algorithme en un langage de programmation.***

a. Langage de programmation :

Un langage de programmation sert à la traduction d'un algorithme en un programme source compréhensible par l'ordinateur. Il est composé de mots clés, de symboles obéissant à des règles de syntaxe (la façon d'écrire une instruction) et de sémantique (sens de l'instruction).

Exemples : Pascal, Java, C, C++, Python, Basic, Fortran, …

b. Passage de l'algorithme au programme source :

Dans ce cours, on utilisera le langage Python pour l'implémentation de nos algorithmes.

Algoritme	Programme source en Pascal	Programme source en Python
Algorithme Conversion **Début** **Lire** (DM) Dcm←DM*100 **Écrire** (Dcm) **Fin.**	**program Conversion;** **uses wincrt;** **var DM,Dcm:real;** **begin** **write ('Taper une distance en mètre ');** **readln (DM);** **Dcm:=DM*100;** **writeln (DM,' m = ',Dcm,' cm');** **end.**	DM=float(input('Taper une distance en mètre ')) Dcm=DM*100 print(DM,'m =',Dcm,'cm')

Pour pouvoir exécuter ce programme, il doit être traduit en langage machine, cette étape est appelée *compilation* ou *interprétation*, selon le type du langage de programmation utilisé.

c. Les types de langage de programmation :

Dans le monde de la programmation il existe 2 types de langages de programmation :

- Langage de programmation interprété : le code source sera traduit en langage machine et exécuté instruction par instruction.
- Langage de programmation compilé : le code source sera traduit en langage machine avant l'exécution du programme. Cette tâche est assurée par le compilateur et elle est dite **compilation**.

Langage interprété	Langage compilé
Algorithme → Code source → **Interprétation et Exécution**	Algorithme → Code source → ***Compilation*** → Programme exécutable

Pascal est un langage compilé !

Python est un langage interprété !

5- Exécution et test du programme :

Une fois compilé ou interprété (selon le langage de programmation utilisé), un programme doit être testé pour s'assurer de son bon fonctionnement et qu'il répond bien aux besoins et attentes de son futur utilisateur. Il est testé par différents valeurs de variables.

Exemple d'exécution de notre programme Conversion :

```
>>> %Run conversion_m_cm.py

 Taper une distance en mètre 12
 12.0 m = 1200.0 cm
```

Application 1 :

On se propose d'écrire un programme qui permet de calculer et afficher la somme de 2 entiers a et b saisis par l'utilisateur. On vous demande d'analyser ce problème, dresser son TDO nécessaire et en déduire l'algorithme ainsi que son implémentation en Python.

……

……

……

……

……

……

……

……

……

……

……

……

……

Application 2 :

On se propose d'écrire un algorithme et un TDO d'un programme qui saisit la longueur et la largeur d'un rectangle, calcule et affiche sa surface et son périmètre. On donne :

S = longueur * largeur

P = (longueur + largeur) * 2

Implémentez votre algorithme en python.

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Les structures simples

I- Introduction :

Une structure est dite simple si elle est réduite à :

- Une opération d'entrée ou de lecture (……).
- Une opération de sortie ou d'écriture (……).
- Une opération d'affectation (……).

II- L'opération d'entrée :

1- Activité 1 :

Donner l'instruction qui permet de lire un entier **n** saisi au clavier.

2- Définition :

L'opération d'entrée est l'instruction qui permet à l'utilisateur de saisir des valeurs au clavier pour qu'elles soient utilisées par le programme. Cette opération est la ***lecture*** (***saisie***).

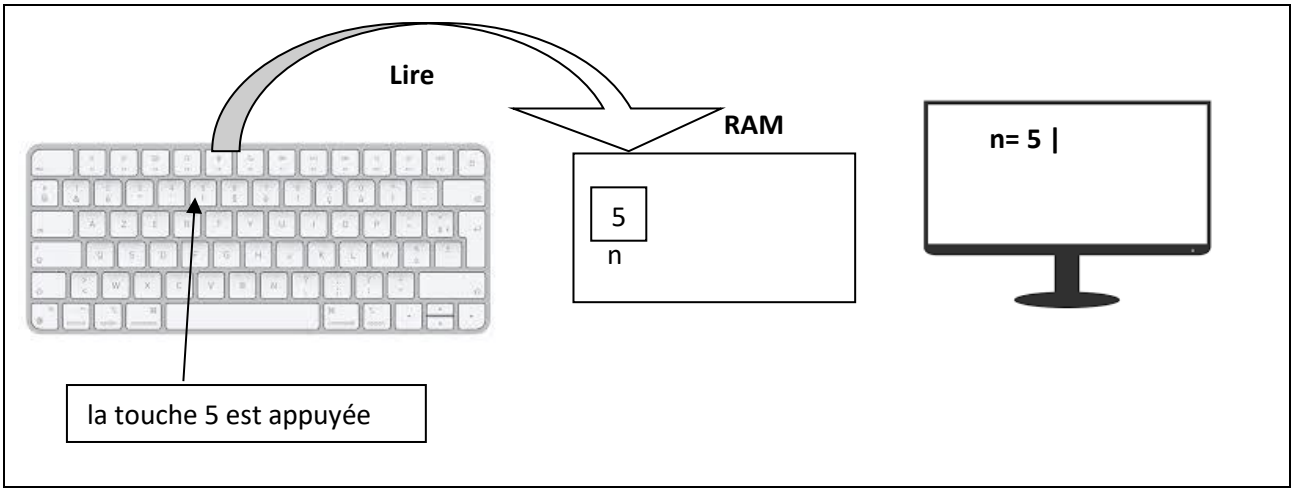
3- Vocabulaire et syntaxe :

Algorithme	Python
Écrire("taper une valeur") **Lire**(nom_variable)	**nom_variable=type_var(input("taper une valeur"))**

4- Exemple : (Réponse de l'Activité 1)

Algorithme	Python
…… ……	……

5- Au niveau de la machine :



6- Remarques :

- Quand on demande à la machine de lire une variable, cela implique que l'utilisateur doit taper cette valeur.
- Les données qui sont lues doivent être compatibles aux variables réservées en mémoire.
- Si on a deux variables **a** et **b** on peut écrire **Lire (a), Lire (b)** ou bien **Lire (a, b)**.

III- L'opération de sortie :

1- Activité 2 :

Donner l'instruction qui permet d'afficher l'entier **n** qui est déjà saisi au clavier à l'activité 1.

2- Définition :

L'opération de sortie est l'instruction qui permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération est l'***écriture*** (***la sortie***).

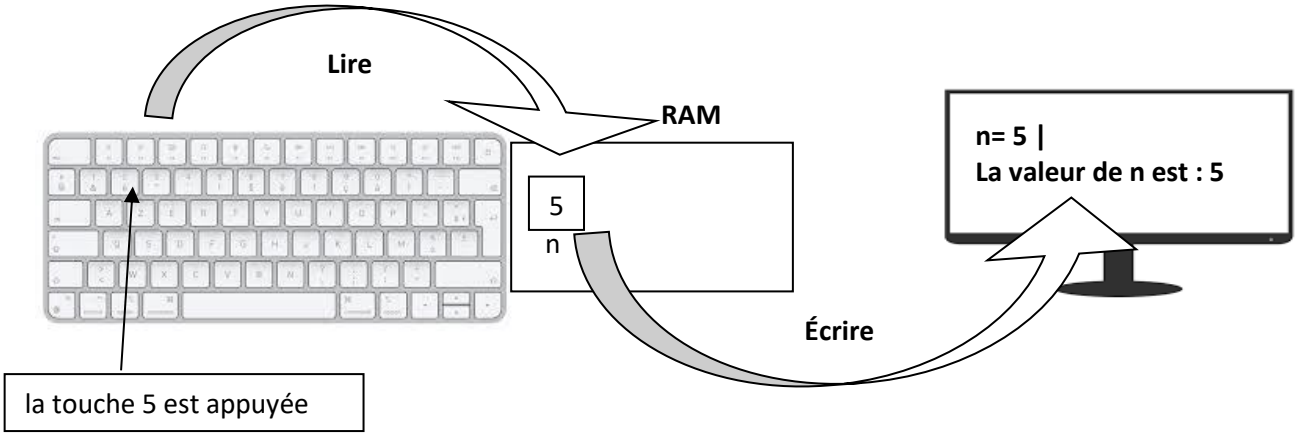
3- Vocabulaire et syntaxe :

Algorithme	Python
Affichage d'un texte : **Écrire** ("message")	***Affichage d'un texte :*** **print** ('message')
Affichage de contenu d'une variable : **Écrire** (nom_variable)	***Affichage de contenu d'une variable :*** **print** (nom_variable)
Affichage mixte (texte et variable) : **Écrire** ("message", nom_variable)	***Affichage mixte (texte et variable) :*** **print** ('message', nom_variable)

4- Exemple : (Réponse de l'Activité 2)

Algorithme	Python
……	……

5- Au niveau de la machine :



6- Remarque :

- Quand on demande à la machine d'écrire une valeur, c'est pour que l'utilisateur puisse la regarder sur l'écran.

IV- L'opération d'affectation :

1- Définition :

L'opération d'affectation est l'action qui permet d'…… (……/……) une valeur à une variable. Elle est représentée par une flèche orientée vers la gauche « ← » en algorithme et par le signe "=" en python.

2- Vocabulaire et syntaxe :

Algorithme	Python
nom_variable ← valeur	nom_variable = valeur

3- Exemples :

Algorithme	Python	Commentaire	Résultat
x←5	x = 5	x reçoit 5	x = 5
a←x+3	a =x+3	a reçoit le résultat de l'expression x+3	a = 5+3 = 8
x←x-2	x =x-2	x reçoit le résultat de l'expression x-2	x = 5-2 =3

4- Remarques :

- Il est évident que la variable située à gauche soit de type numérique.
- Le type des variables situées à droite de l'affectation doit être de même type ou de type compatible que celle située à gauche.

5- Application 1 :

Soit la séquence d'affectations suivante :

x ← 10
y ← 8
z ← x
x ← y
y ← z

- Donner le résultat d'exécution de cette séquence sous forme d'un tableau.

	Instructions				
Variables	1	2	3	4	5
x					
y					
z					

- Quelles sont les valeurs finales de x et de y ?

⇨ ……

- Quel est le rôle de cette séquence ?

⇨ ……

- Quelle est l'utilité de la variable z ?

⇨ ……

6- Application 2 :

a) *Écrire une analyse, un algorithme et son implémentation en Python qui permet de calculer et afficher la surface et le périmètre d'un cercle de rayon r saisi par l'utilisateur.*

Analyse	***Algorithme***	***Implémentation en Python***

b) *Écrire un programme qui saisit et affiche votre nom et prénom, votre classe et votre matière préférée, puis les affiche comme suit :*

Exemple :

Taper votre nom : ben salah
Taper votre prénom : mohamed
Classe : 2TI1
Matière préférée : anglais
Bonjour mohamed ben salah, bienvenu à la classe 2TI1 !
Votre matière préférée est : anglais.

***Algorithme** : bienvenu*

TDO :

objets	*type/nature*

Les structures de contrôle itératives

I- La structure de contrôle itérative complète :

Activité 1 :

Écrire un algorithme qui permet d'afficher le mot "informatique" 10 fois.

Solution :

```
Algorithme affiche
Début
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
        Écrire("informatique")
Fin.
```

Constatation : *répétition de la même instruction plusieurs fois* ➔ ***algorithme très long !!!***

Solution : *utilisation de la boucle pour !!!*

Comment ?

```
Algorithme affiche
Début
        ……..
                ……
        ……
Fin.
```

i : compteur (de type entier)

0 : valeur initiale de i

9 : valeur finale de i

i s'incrémente automatiquement de 1, et prend les valeurs 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Implémentation en python :

```
……
        ……
```

➔ ***En python, i s'arrête toujours à la valeur finale-1 et commence généralement par la valeur 0***

Application 1 : Écrire un programme qui affiche tous les entiers positifs pairs <= 20.

Solution :

Algorithme :	TDO :	Implémentation en python :
	objet : ……. type : ……	

Retenons :

Algorithme	Python	Explication	Exemple

pour i **de** 0 **à** n-1 **faire**
  écrire(i)
fin pour
	for i **in** range(n):
	  print(i)		

pour i **de** m **à** n-1 **faire**
  écrire(i)
fin pour
	for i **in** range(m,n):
	  print(i)		

pour i **de** m **à** n-1 **pas**=p **faire**
  écrire(i)
fin pour
(avec m<n)
	for i **in** range(m,n,p):
	  print(i)		

pour i **de** m **à** n-1 **pas**=-p **faire**
  écrire(i)
fin pour
(avec m>n)
	for i **in** range(m,n,-p):
	  print(i)		

Application 2 :

a- Écrire un programme qui permet de calculer et afficher la somme des n premiers entiers avec n saisie par l'utilisateur.
Exemple : pour n=5 le programme calculera s=1+2+3+4+5=15

b- Écrire un programme qui permet de calculer et afficher le produit des n premiers entiers avec n saisie par l'utilisateur.
Exemple : pour n=6 le programme calculera p=1*2*3*4*5*6=720

Solution :

Algorithme :	TDO :	Implémentation en python :
#saisie #traitement #affichage	Objets / Types/Natures …… / …… …… / ……	

#saisie #traitement #affichage	**Objets** / **Types/Natures** …… / …… …… / ……	

II- Les structures de contrôle itératives à condition d'arrêt :

Activité 2 :

Écrire un algorithme qui permet de saisir un entier n strictement positif.

⇨ Pour résoudre ce problème, on ne peut pas utiliser la boucle "**pour**", car le nombre d'itérations n'est pas connu d'avance.

Pour résoudre ce genre de problèmes, il faut utiliser une structure de contrôle itérative à condition d'arrêt. En effet, l'instruction de saisie de n va se répéter jusqu'à ce que l'utilisateur saisisse un entier strictement positif.

Il existe 2 structures de contrôle itératives à condition d'arrêt : la boucle "**tant que** … **faire**" et la boucle "**répéter** … **jusqu'à**" ayants les syntaxes suivantes :

"tant que … faire"	"répéter … jusqu'à"

{initialisations}
Tant que non (condition d'arrêt) faire
  Traitement
Fin tant que
	Répéter
	  Traitement
	Jusqu'à (condition d'arrêt)

Exemple :

"tant que … faire"	"répéter … jusqu'à"

n←0
Tant que non (n>0) faire
  écrire("taper un entier > 0 ")
  lire(n)
Fin tant que
	Répéter
	  écrire("taper un entier > 0")
	  lire(n)
	Jusqu'à (n>0)

Implémentation en python :

⇨ ***La boucle "répéter … jusqu'à" n'existe pas en python !***

⇨ ***On utilise alors la boucle tant que = "while"***

1ère méthode :

```
n=0
while(n <= 0): #ou bien while not (n > 0):
    n=int(input('taper un entier strictement positif'))
print("n=",n)
```

2ème méthode :

```
n= int(input('taper un entier strictement positif'))
while(n <= 0): #ou bien while not (n > 0):
    n=int(input('n doit être strictement positif !!!'))
print("n=",n)
```

Structures de contrôle conditionnelles

I- Structure de contrôle conditionnelle simple :

Activité 1 :

On se propose d'écrire un programme qui saisit la moyenne M d'un élève puis décide et affiche son résultat comme suit :

- Resultat = "**Admis**" si **M>=10**
- Resultat = "**Redouble**" si **M<10**

Remarque : M est un réel compris entre 0 et 20.

Constatation :

On constate qu'on est face à 2 cas possible :

- Le résultat de l'élève sera "**Admis**" si sa moyenne>=10 ;
- Le résultat de l'élève sera "**Redouble**" sinon.

D'où la nécessité d'une structure qui nous permet de résoudre ce genre de problèmes ; c'est "***la Structure de Contrôle Conditionnelle SI***" ayant la syntaxe suivante :

Algorithme :	Implémentation en python :

si condition **alors**
traitement 1
sinon
traitement 2
fin si.
	`if condition :`
	`  traitement 1`
	`else :`
	`  traitement 2`

Ainsi notre algorithme sera comme suit :

Algorithme :	Implémentation en python :

TDO :

Objet :	Type/Nature

Remarque :

- La clause condition est de type booléen, si elle retourne la valeur Vrai alors le programme exécute le Traitement 1 et si elle retourne la valeur Faux le programme exécute le traitement 2.

Application 1 :

Écrire un programme qui permet de saisir un entier N formé de 3 chiffres, puis affiche si cet entier est cubique ou non.

On rappelle qu'un entier est dit cubique s'il est égal à la somme des cubes de ses trois chiffres.

Exemple :
153 est cubique car $153=1^3+5^3+3^3$
Rappel : En python on écrit : a^3=a**3 ou bien a^3=a*a*a, on peut aussi utiliser la fonction **pow** de la bibliothèque **math** comme suit a^3=pow(a,3)

Algorithme :	Implémentation en python :

TDO :

Objet :	Type/Nature

Application 2 :

Apporter les modifications nécessaires à l'algorithme cubique pour qu'il affiche tous les entiers cubiques qui existent.

Solution :

Algorithme :	Implémentation en python :

Remarque : Dans cet algorithme, on a utilisé ***la structure de contrôle conditionnelle réduite*** ayant la forme suivante :

si condition **alors**
    traitement
fin si

Cette forme est utilisée lorsqu'on a un seul traitement à exécuter à la suite de la vérification d'une condition.

TDO :

Objet :	Type/Nature

Application 3 :

Écrire un programme qui permet de :

- Saisir 2 entiers A et B strictement positifs et tel que A > B ;
- Tester et afficher si A et multiple de B.

Solution :

Algorithme :	Implémentation en python :

TDO :

Objet :	Type/Nature

II- Structure de contrôle conditionnelle généralisée :

Activité 2 :

Revenons à l'activité 1 et soyons plus souples avec nos élèves. Le résultat d'un élève sera alors décidé comme suit :

- Resultat = "**Admis**" si **M>=10**
- Resultat = "**Rachat**" si **9<=M<10**
- Resultat = "**Redouble**" si **M<9**

Constatation :

On est face à 3 cas possibles :

- Le résultat de l'élève sera "**Admis**" si sa moyenne>=10 ;
- Le résultat de l'élève sera "**Rachat**" si sa moyenne est comprise entre 9 et 10
- Le résultat de l'élève sera "**Redouble**" sinon.

➔ Il nous faut une structure de contrôle conditionnelle qui traite ce cas, c'est "***la structure de contrôle conditionnelle généralisée***", ayant la syntaxe suivante :

Algorithme :	Implémentation en python :

si **condition 1** alors
traitement 1
sinon si **condition 2** alors
traitement 2
……
sinon si **condition n-1** alors
	`if condition 1 :`
	`  traitement 1`
	`elif condition 2 :`
	`  traitement 2`
	……
	`elif condition n-1 :`
	`  traitement n-1`

traitement n-1 sinon **traitement n** fin si.	else : traitement n

La solution de notre activité sera comme suit :

Algorithme :	Implémentation en python :

TDO :

Objets	Type/Nature

Application 4 :

On se propose d'écrire un programme qui permet de résoudre dans $\mathbb{R}$ une équation de la forme $ax^2+bx+c=0$.

On rappelle que pour résoudre ce genre d'équations il faut d'abord calculer le discriminant $\Delta=b^2-4ac$ puis selon le signe de Δ on calcule les solutions comme suit :

Si $\Delta > 0$ alors il existe 2 solutions qui sont : $x1 = \frac{-b-\sqrt{\Delta}}{2a}$ et $x2 = \frac{-b+\sqrt{\Delta}}{2a}$

Si $\Delta = 0$ alors il existe 1 seule solution qui est $x = \frac{-b}{2a}$

Si $\Delta < 0$ alors pas de solutions pour cette équation !

On vous demande d'écrire un algorithme d'un programme qui permet de :

- Saisir trois réels a, b et c tel que $a\neq0$;
- Calculer Δ
- Calculer et afficher la/les solution(s) de l'équation si ça existe.

Solution :

Algorithme :	Implémentation en python :

TDO:

Objets	Type/Nature

III- La structure de contrôle conditionnelle à choix multiples :

Activité 3 :

On se propose d'écrire un algorithme du programme intitulé touche permettant d'afficher la nature d'une touche du clavier appuyée par l'utilisateur. La nature d'une touche peut être soit une consonne ou une voyelle dans le cas des lettres, soit un chiffre, soit un symbole.

Voici la solution proposée par une élève :

TDO:

Objets	Type/Nature

Que remarquez-vous ?

Constatation :

On constate, que cet algorithme est très long, il est formé par plusieurs conditions qui portent sur une seule variable dite encore "sélecteur" de type scalaire (caractère). Dans ce cas on peut utiliser ***la structure de contrôle conditionnelle à choix multiples : "selon"***, ayant la syntaxe suivante :

Algorithme :	Implémentation en python :

Selon Sélecteur
Valeur1_1[, Valeur1_2, …] :
Traitement1
Valeur2_1 . . Valeur2_2 :
Traitement2
……
[**Sinon** TraitementN]
Fin Selon
	À partir de la version 3.10 :
	`match Sélecteur :`
	`case Valeur1 :`
	`Traitement1`
	`case Valeur2_1 | Valeur2_2 :`
	`Traitement2`
	`case Sélecteur if V3_1 <=Sélecteur<= V3_2 :`
	`Traitement3`
	`case _ : TraitementN`
	N.B. : Le sélecteur doit être de type scalaire.

Solution : avec selon

Algorithme :	Implémentation en python :

TDO:

Objets	Type/Nature

Application 5 :

On se propose d'écrire un algorithme d'un programme qui saisit le numéro d'un mois (entier compris entre 1 et 12) puis affiche la saison correspondante à ce mois. On vous rappelle que :

Mois	Saison
1, 2, 12	Hiver
3, 4, 5	Printemps
6, 7, 8	Eté
9, 10, 11	Automne

Solution :

Algorithme :	Implémentation en python :

Les chaînes de caractères

I- Définition :

Une chaîne de caractères est une suite ordonnée de caractères. En algorithmique, une chaîne de caractères est considérée comme une variable scalaire. Cependant, en langage C, une chaîne est vue comme un tableau de caractères alors qu'en Java, une chaîne est un objet. En Python, les chaînes de caractères sont classées dans des séquences tout comme les listes et les tuples (les tuples et les chaînes sont des séquences immuables, c'est-à-dire qu'elles ne peuvent pas être modifiées).

➔ *"Une chaîne de caractères est une suite finie de caractères."*

Exemples :

- La chaîne de caractères "**Baobab**" est constituée des caractères "**B**", "**a**", "**o**", "**b**", "**a**" et "**b**".
- La chaîne de caractères "**123**" n'est constituée que de chiffres, on parle d'une chaîne numérique, mais en aucun cas il ne faut la confondre avec la grandeur numérique 123.
- La chaîne de caractères "" représente une chaîne de caractères vide.
- La chaîne de caractères " " est une chaîne de caractères ne contenant qu'un seul caractère qui est ici le caractère espace (à ne pas confondre avec la chaîne de caractères vide).

Les chaînes sont constituées de caractères. Chaque caractère (qui n'est autre qu'un symbole) se trouve à une position donnée dans la chaîne (on parle alors de rang). On admet qu'en algorithmique, le premier caractère se trouve au rang 0, le deuxième caractère se trouve au rang 1 et ainsi de suite... De même pour le langage Python, le premier caractère d'une chaîne se trouve au rang 0, le deuxième caractère se trouve au rang 1 et ainsi de suite...

Exemple : la chaîne "Baobab"

Caractères	"B"	"a"	"o"	"b"	"a"	"b"
Indices	0	1	2	3	4	5

Un caractère peut-être :

- Une lettre (caractère alphabétique – la distinction est faite entre les lettres minuscules et les lettres majuscules),
- Un chiffre (caractère numérique),
- Un signe de ponctuation,
- Ou tout autre symbole (comme l'étoile, le tiret, le "slash"...).

Dans un système informatique, à chaque caractère est associé une valeur numérique : son code **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). L'ensemble des codes est recensé dans une table nommée "**table des codes ASCII**". Quand on stocke un caractère en mémoire (dans une variable), on mémorise en réalité son code **ASCII**. Un code **ASCII** est codé sur un octet (huit bits).

On peut remarquer que les lettres minuscules se suivent dans la table des codes **ASCII** (tout comme les lettres majuscules et les chiffres). Ces trois "**classes**" de caractères forment trois plages de codes **ASCII** (trois domaines de valeurs) qu'il sera possible d'exploiter pour effectuer des conversions.

II- Relation d'ordre

Exemple : Comparaison de caractères

"**B**" > "**A**" car le code **ASCII** du caractère "**B**" (66 en base 10) est supérieur au code ASCII du caractère "**A**" (65 en base 10).

Pour comparer deux chaînes de caractères, on compare les caractères de même rang dans les deux chaînes en commençant par le premier caractère de chaque chaîne (le premier caractère de la première chaîne est comparé au premier caractère de la seconde chaîne, le deuxième caractère de la première chaîne est comparé au deuxième caractère de la seconde chaîne, et ainsi de suite…).

Exemples : Comparaison de deux chaînes

- "**baobab**" < "**sapin**" car le code ASCII de "**b**" est inférieur au code ASCII de "**s**".
- "**baobab**" > "**banania**" car le code ASCII de "**o**" est supérieur au code ASCII de "**n**" (la comparaison ne peut pas se faire sur les deux premiers caractères car ils sont identiques).
- "**1999**" > "**1998**" car le code ASCII de "**9**" est supérieur au code ASCII de "**8**" (la comparaison ne peut pas se faire sur les trois premiers caractères car ils sont identiques). Attention, ici ce ne sont pas des valeurs numériques qui sont comparées, mais bien des caractères.
- "**333**" > "**1230**" car le code ASCII de "**3**" est supérieur au code ASCII de "**1**".
- "**333**" < "**3330**" car la seconde chaîne a une longueur supérieure à celle de la première (la comparaison ne peut pas se faire sur les trois premiers caractères car ils sont identiques).
- "**Baobab**" < "**baobab**" car le code ASCII de "**b**" est supérieur au code ASCII de "**B**".

Application 1 :

Compléter le tableau ci-dessous en indiquant la relation d'ordre (<,=,>) ainsi que le nombre de caractères concernés par la comparaison.

"**python**"	>	"**Python**"	Le code ASCII de "**p**" est supérieur au code ASCII de "**P**".
"**Bonsoi**r"		"**Bonjour**"	
"**Hola**"		"**hola**"	
"**Ohé**"		"**Eho**"	
"**BTS1IG**"		"**BTS2IG**"	
"**27/10/00**"		"**27/10/99**"	
"**C & Java**"		"**C | Java**"	
"**> 15**"		"**< 15**"	
"**toi**"		"**toi**"	
"**5170**"		"**517**"	
"**318**"		"**1273**"	

III- Concaténation de chaînes de caractères

Le but de la concaténation est de créer des chaînes de caractères en juxtaposant deux (ou plusieurs) chaînes de caractères. En algorithmique, l'opérateur de concaténation est représenté par le signe +. Cette notation est utilisée en Java, Python ou encore en JavaScript.

Exemple : Concaténation de deux chaînes

```
Algorithme Concaténation
Début
    categorie ← "développeur"
    categorie ← categorie + "s"
    écrire(catégorie)        # Affiche : développeurs
    categorie ← "des " + categorie
    écrire(categorie)        # Affiche : des développeurs
Fin
```

IV- Manipulation de chaînes de caractères

1- Copie d'un extrait

Algorithme	**SOUS_CHAINE**(ch, d, f)
Python	ch[position] ch[d : f]

sachant que :

- **ch** est une chaîne de caractères ;
- **d** est un entier représentant la position de début ;
- **f** est un entier représentant la position de la fin.

Le rôle de la fonction **SOUS_CHAINE** est de retourner une "sous-chaîne" (une partie de la chaîne **ch** à partir de la position **d** jusqu'à la position **f-1** (**f exclue**)). En Python, une chaîne étant une séquence, l'utilisation des crochets permet d'accéder à un extrait constitué d'un ou plusieurs caractères.

2- Nombre de caractères

Algorithme	**LONG**(ch)
Python	**len**(ch)

La fonction **LONG** a pour rôle de calculer et de retourner le nombre de caractères présents dans la chaîne de caractères **ch** passée en paramètre.

3- Recherche d'un caractère ou d'une sous-chaîne

Algorithme	**POS**(ch2, ch1)
Python	ch2.**find**(ch1)

La fonction **POS** recherche et retourne la première occurrence de la chaîne **ch2** dans la chaîne **ch1**. La recherche commence à partir de la position 0. Cette fonction retourne la position du premier caractère de la chaîne **ch2**, ou la valeur -1 si la chaîne **ch2** n'existe pas dans la chaîne **ch1**. En Python, si aucune occurrence de la chaîne **ch2** n'est trouvée, la méthode **find()** retourne la valeur -1.

4- Conversion en code ASCII

Algorithme	**ORD**(c)
Python	**ord**(c)

- **c** est un caractère

La fonction **ORD** retourne le code ASCII du caractère **c** passé en paramètre.

5- Conversion en caractère

Algorithme	**CHR**(nbr)
Python	**chr**(nbr)

- **nbr** est un entier représentant un code ASCII

La fonction **CHR** retourne le caractère dont le code ASCII est passé en paramètre.

6- Conversion en grandeur numérique

Algorithme	**VALEUR**(ch)
Python	**int**(ch) #pour convertir ch en entier **float**(ch) #pour convertir ch en réel

La fonction **VALEUR** a pour rôle de convertir la chaîne de caractères **ch** passée en paramètre, en grandeur numérique (entier ou réel) si la suite de caractères représente un nombre.

7- Conversion en chaîne de caractères

Algorithme	**CONVCH**(nbr)
Python	**str**(nbr)

- **nbr** peut être entier ou réel

La fonction **CONVCH** a pour rôle de convertir en chaîne de caractères une valeur numérique **nbr** passée en paramètre, puis de retourner la chaîne résultante.

Application 2 :

Que retourne chacune des instructions ci-dessous ? implémenter chaque instruction en langage Python.

Instruction en algorithme	Implémentation en python	Résultat
LONG("Agenda")		
SOUS_CHAINE("Almanach",1,4)		
SOUS_CHAINE("Almanach",4,2)		
SOUS_CHAINE("Almanach",5,8)		
POS("an","Almanach")		
POS("Almanach","an")		
POS("kan","Almanach")		
POS("a","Almanach")		
POS("Almanach","a")		
ORD("Z")		

CHR(118)		

8- Transformer en majuscules :

Algorithme	**MAJUS**(c) ou **MAJUS**(ch)
Python	c.**upper()** ou ch.**upper()**

- **c** est un caractère
- **ch** est une chaîne de caractères

La fonction **MAJUS** transforme un caractère **c** en son équivalent en majuscule, ou une chaîne de caractères **ch** en son équivalente en majuscules.

9- Supprimer des caractères d'une chaîne de caractères :

Algorithme	**EFFACER**(ch,d,f)
Python	**ch[: d] + ch[f :]**

La fonction **EFFACER** supprime de la chaîne de caractères **ch** les caractères à partir de la position **d** jusqu'à la position **f-1** (**f** exclue).

10- Vérification d'une chaîne numérique :

Algorithme	**ESTNUM**(ch)
Python	ch.**isdecimal()**

La fonction **ESTNUM** est une fonction booléenne qui retourne la valeur "vrai" si la chaîne de caractères **ch** (passée en paramètre) est convertible en valeur numérique (c'est-à-dire formée de chiffres seulement) et la valeur "faux" sinon.

Remarques :

En Python, il existe de nombreuses fonctions/méthodes de manipulation des chaînes de caractères comme qui n'ont pas été détaillées dans ce document. Certaines sont applicables aux séquences en général et d'autres sont spécifiques aux chaînes. Elles peuvent être classées en plusieurs catégories :

- Modification de la casse : **lower**(), **swapcase**(), **capitalize**(), **title**()
- Nombre d'occurrences : **count**()
- Suppression de caractères : **lstrip**(), **rstrip**(), **strip**()
- Alignement : **ljust**(), **rjust**(), **center**()
- Découper et reconstituer : **split**(), **join**()
- Remplacement : **replace**()
- Recherche inversée : **rindex**(), **rfind**()
- Valeur minimale et valeur maximale : **min**(), **max**()
- Classe des caractères : **isalnum**(), **isalpha**(), **isdigit**(), **islower**(), **isupper**(), **isspace**(), **istitle**()

Cette liste de fonctions est mentionnée juste pour information, il est strictement interdit de les utiliser à votre niveau !

Les tableaux

I- Introduction :

Imaginons que dans un programme, nous avons besoin d'un grand nombre de variables, il devient difficile de donner un nom à chaque variable.

Exemple :

Écrire un algorithme permettant de saisir cinq notes et de les afficher après avoir multiplié toutes les notes par trois.

Solution :

Algorithme Note
Début
Écrire ("Entrer la valeur de la 1ère note") **Lire** (N1)
Écrire ("Entrer la valeur de la 2ème note") **Lire** (N2)
Écrire ("Entrer la valeur de la 3ème note") **Lire** (N3)
Écrire ("Entrer la valeur de la 4ème note") **Lire** (N4)
Écrire ("Entrer la valeur de la 5ème note") **Lire** (N5)
Écrire ("La note 1 multipliée par 3 est : ", N1 * 3)
Écrire ("La note 2 multipliée par 3 est : ", N2 * 3)
Écrire ("La note 3 multipliée par 3 est : ", N3* 3)
Écrire ("La note 4 multipliée par 3 est : ", N4 * 3)
Écrire ("La note 5 multipliée par 3 est : ", N5 * 3)
Fin

TDO :

Objets	Type/Nature
N1, N2, N3, N4, N5	Réels

Constatation :

- La même instruction se répète cinq fois. Imaginons que si l'on voudrait réaliser cet algorithme avec 100 notes, cela deviendrait fastidieux.
- Comme les variables ont des noms différents, on ne peut pas utiliser de boucle, ce qui allonge considérablement le code et le rend très répétitif.
- Pour résoudre ce problème, il existe un type de données qui permet de définir plusieurs variables de même type.

II- Définition :

Un tableau est une suite d'éléments de même type. Il utilise plusieurs cases mémoire à l'aide d'un seul nom. Comme toutes les cases portent le même nom, elles se différencient par un numéro ou un indice.

Nous pouvons représenter schématiquement un tableau nommé "**Note**" composé de cinq cases, dans la mémoire comme suit :

	Note[0]	Note[1]	Note[2]	Note[3]	Note[4]	
Notes :	12	14	11.5	13	12	← Contenu du tableau

Nous disposons alors de cinq variables. Pour les nommer, on indique le nom du tableau suivi de son indice entre crochets : La première s'appelle Note[0], la deuxième Note[1], etc. jusqu'à la dernière Note[4].

Note[3] représente le 4ème élément du tableau Note et vaut 13.

III- Tableau à une dimension :

1- Déclaration :

La déclaration d'un tableau permet d'associer à un nom une zone mémoire composée d'un certain nombres de cases mémoires de même type.

Syntaxe :

TDO :

Objets	Type/Nature
Nom_tableau	Tableau de N Type _élément

En python :

- On utilisera la bibliothèque numpy pour implémenter les tableaux.
- Un tableau de la bibliothèque numpy est :
  - homogène, c'est-à-dire constitué d'éléments de même type,
  - statique, car sa taille est fixée lors de la création.
- La déclaration d'un tableau se fait en deux étapes :
  - Importation des modules nécessaires de la bibliothèque numpy :

```
from numpy import array ou
from numpy import * ou
import numpy as alias
```

  - Déclaration du tableau

```
T = array ([Type_élément] * N) ou bien
T = array ([valeur_initiale] * N)
```

On peut spécifier le type des éléments d'un tableau avec la syntaxe :

```
Nom_tableau = array ([Valeur_initiale] * N, dtype=Type_élément)
```

Exemples de déclaration de tableaux en python :

Déclaration :	Explication :
T=array([5]*10)	Déclarer un tableau T de 10 entiers et initialiser ses éléments par «5».
T=array([float()]*10)	Déclarer un tableau T de 10 réels et initialiser ses éléments par «0.0 ».
T=array([str]*10)	Déclarer un tableau T de 10 chaînes de caractères.
T=array([str()]*10)	Déclarer un tableau T de 10 caractères et initialiser ses éléments par le caractère vide.
T=array(['']*10,dtype='U20')	Déclarer un tableau T de 10 éléments initialisés par une chaîne vide. Chaque élément peut contenir 20 caractères au maximum.

Remarques :

- Le premier élément d'un tableau porte l'indice zéro ou l'indice 1 selon les langages.
- La valeur d'un indice doit être un nombre entier.
- La valeur d'un indice doit être inférieure ou égale au nombre d'éléments du tableau.

Exemple :

L'instruction suivante déclare un tableau de 30 éléments de type réel :

TDO :

Objets	Type/Nature
Note	Tableau de 30 Réels

- **Note** : c'est le nom du tableau (identificateur).
- **30** : c'est nombre d'éléments du tableau(taille).

Exercice d'application :

Déclarer deux tableaux nommés A et B composés chacun d'eux de 10 éléments de type chaîne de caractères.

Solution :

En algorithme :	En python :
TDO : **Objets** | **Type/Nature** |	

Remarques :

- Lorsqu'on déclare un tableau, on déclare aussi de façon implicite toutes les variables indicées qui le constituent. Nous utiliserons la valeur 1 comme premier indice mais on peut aussi utiliser un autre indice minimum, comme 0. Dans ce cas, l'indice maximum sera égal au nombre d'éléments -1.

- Dans le langage Python, les cases du tableau sont numérotées à partir de 0. En Visual Basic l'indice minimum est 1.

2- Utilisation :

Les éléments d'un tableau sont des variables indicés qui s'utilisent exactement comme n'importe quelle autre variable classique. Elles peuvent faire l'objet d'une affectation, elles peuvent figurer dans une expression arithmétique, dans une comparaison, elles peuvent être affichées et saisies etc.

L'utilisation de ces éléments se fait en suite, via le nom du tableau et son indice. Ce dernier peut être soit une valeur (exemple : **Note**[3]), soit une variable (exemple : **Note**[i]) ou encore une expression (exemple : **Note**[i+1]).

Remarque :

Il ne faut pas confondre l'indice d'un élément d'un tableau avec son contenu. Par exemple, la 4ème maison de la rue n'a pas forcément 4 habitants.

Exemple 1 :

L'instruction suivante affecte à la variable X la valeur du premier élément du tableau "**Note**" :

X ← **Note**[0]

Exemple 2 :

L'instruction suivante affiche le contenu du quatrième élément du tableau "**Note**" :

Écrire(**Note**[3])

Exemple 3 :

L'instruction suivante affecte une valeur introduite par l'utilisateur à l'élément trois du tableau "**Note**" :

Lire(**Note**[2])

Parcours complet d'un tableau :

Le fait que les éléments d'un tableau soient indicés permet de parcourir tous ces éléments avec une boucle, en utilisant une variable qui sert d'indice et s'incrémente à chaque tour de boucle.

Exercice 1 :

Écrire un algorithme permettant de saisir 30 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur :

Solution :

TDO :

Objets	Type/Nature

Exercice 2 :

Écrire un algorithme qui calcule la somme des éléments d'un tableau d'entiers.

Solution :

On suppose que le nombre d'éléments du tableau est compris entre 5 et 100.

TDO :

Objets	Type/Nature

Implémentation en python :

Les sous-programmes : Fonctions et Procédures

Pourquoi les sous-programmes ?

Dans un algorithme (ou programme informatique) on se retrouve souvent avec des bouts de codes qui peuvent se répéter dans plusieurs emplacements. Cette redondance rend le code long, moins lisible et, par-suite, si on souhaite modifier le comportement de ces blocs redondants alors on doit les réécrire tous, ce qui n'est pas du tout pratique.

Les sous-programmes permettent d'éviter cette redondance en factorisant le code qui peut se répéter. De cette façon nous n'aurons qu'un seul bloc de code que l'on peut réutiliser à volonté dans plusieurs endroits sans être obligé de le réécrire.

Les sous-programmes peuvent être des **fonctions** ou des **procédures**. Les fonctions et les procédures sont presque similaires, mais ***une fonction retourne un et un seul résultat de type simple***, alors qu'***une procédure ne retourne rien***.

I- Les Fonctions

1- Déclaration d'une fonction en algorithme et en Python :

Algorithme	Python

Fonction Nom_fonction (pf1 : type, pf2 : type, …) : type_résultat
Début
Traitement
Retourner *resultat*
Fin
	`def Nom_fonction (pf1,pf2,…):`
	`  Traitement`
	`  return resultat`

2- Appel d'une fonction en algorithme et en Python (au niveau du programme principal) :

L'appel d'une fonction, en algorithme, comme en python, peut se faire de deux méthodes :

Appel algorithmique	Appel en Python

Objet←Nom_fonction (pe1, pe2, …, pen)
Ou bien
Ecrire (Nom_fonction (pe1, pe2, …, pen))
	`Objet=Nom_fonction(pe1,pe2,…,pen)`
	Ou bien
	`print(Nom_fonction(pe1,pe2,…,pen))`

NB :

- *pf :* est un **paramètre formel** qui est défini au niveau de la définition de la fonction.
- *pe :* est un **paramètre effectif** utilisé lors de l'appel de la fonction.
- L'instruction **return**, définit ce que doit renvoyer la fonction.

Application 1 : écrire un algorithme et son implémentation en Python d'une fonction **Carre(x)** qui permet de calculer le carré d'un entier "x", ainsi que son programme principal.

Correction :

En algorithme	En Python
TDOG : O | T TDOL : (pas d'objets locaux à déclarer) O | T	

Application 2 : écrire un algorithme et son implémentation en Python d'une fonction **NbVoy(ch)** qui permet de retourner le nombre de voyelles dans une chaine de caractères. Exp : ch= "Salut mes amis" ➔ nb = 5.

Correction :

En algorithme	En Python
TDOL : O | T |	TDOG : O | T |

Application 3 : écrire un code, en algorithme et en Python, d'une fonction **Nbchiff(ch)** qui permet de retourner le nombre de chiffres dans une chaine de caractères. Exp : ch= "Bac2022" nb = 4.

Correction :

En algorithme	En Python
TDOL : O | T |	TDOG : O | T |

II- Les Procédures :

1- Déclaration d'une procédure en algorithme et en Python :

Algorithme	Python
Procédure Nom_procédure (pf1 : type1, pf2 : type2, …) Début Traitement Fin	**def** Nom_procédure (pf1, pf2,…) Traitement

2- Appel d'une procédure :

L'appel d'une procédure, au niveau Algorithme, comme en python se fait par son nom suivi par la liste de paramètres effectifs, comme suit :

En algorithme	En Python
Nom_Procédure (pe1,pe2, …)	Nom_Procédure (pe1,pe2, …)

Application 1 : écrire un code, en algorithme et en Python, d'une procédure **lecture (ch)** qui permet de saisir une chaine de caractères non vide et de longueur maximale =20.

Correction :

En algorithme	En Python
TDOG : Objets | Type/Nature TDOL : Objets | Type/Nature	

Application 2 : écrire un algorithme (+TDO nécessaires) et son implémentation en python qui :

- Saisit 2 entiers n et m compris entre 1 et 10 ;
- Calcule et affiche n^m (n à la puissance m).

NB : une solution modulaire est exigée.

En algorithme	En Python
TDOG : O | T TDOL : O | T	

TDOL :

O	T

Application 3 : écrire un code, en algorithme et en Python, d'une procédure remplir(n,T) qui permet de remplir un tableau T de n caractères alphabétiques.

Application 4 : écrire un algorithme modulaire « traitement » qui permet de :

- Saisir un entier N avec (3 <= N <= 30)
- Remplir un tableau par N caractères alphanumériques.
- Déterminer le nombre de lettres se trouvant dans ce tableau T
- Déterminer la chaine formée par les chiffres existant dans le tableau.
- Afficher le tableau T ainsi que le nombre de lettres et la chaine des chiffres.

Exemple :

Si N= 8 et T :

2	M	0	A	2	T	2	H
0	1	2	3	4	5	6	7

Alors le programme affichera : 2 M 0 A 2 T 2 H

Le nombre de lettres = 4

La chaine des chiffres = 2022