Algorithmes : Tests et boucles

1. Conditions

Une condition est une expression qui peut prendre l'une des deux valeurs suivantes **vrai** ou **faux**. On dit également que c'est une valeur de type **« logique »** ou **« booléen »**.

Les principaux opérateurs de comparaison que vous rencontrerez sont les suivants :

- égal à (= en pseudo code)
- différent de (!= en pseudo code)
- strictement supérieur (> en pseudo code)
- strictement inférieur (< en pseudo code)
- supérieur ou égal (> = en pseudo code)
- inférieur ou égal (< = en pseudo code)

Ces comparaisons n'ont un sens que si les variables que l'on compare sont de même type.

Conditions composées

On peut écrire des conditions plus complexes en reliant des comparaisons à l'aide des opérateurs logiques **ET**, **OU** et **NON**.

- **Condition 1 ET condition 2** sera vraie si les deux conditions sont **toutes les deux vraies**.

  Par exemple, la condition : « **âge supérieur à 5 ET âge inférieur à 10** » sera vraie si la variable âge est **strictement comprise entre 5 et 10**.

- **Condition 1 OU condition 2** sera vraie si **l'une au moins** des deux conditions est vraie.

  Par exemple, la condition « **prénom=Jean OU nom=Dupont** » sera vraie pour :

  - Jean Dupont (conditions 1 et 2 vraies)
  - Jean Durand (condition 1 vraie)
  - Pierre Dupont (condition 2 vraie)

  mais fausse pour

  - Pierre Durand (conditions 1 et 2 fausses)

- **NON (condition 1)** sera vraie si et seulement si **condition 1 est fausse**.

  Par exemple : « **NON (x < 3)** » sera vraie si **x > = 3**

Syntaxe de l'instruction conditionnelle dans 4 langages de programmation :

Langage naturel	Python	Scilab	TI	CASIO

Si *Condition*
Alors
Instructions1
Sinon
Instructions2
	if *condition*:
	
	Instruction1
	else:
	
	Instruction2
		if *condition* **then**
		Instruction1
		else
		Instruction2
		end
			:If *condition*
			:Then
			:*Instruction1*
			:Else
			:*Instruction2*
			:End
				If *condition*↵
				Then
				Instruction1↵
				Else *Instruction2*↵
				IfEnd↵

Exemple

Voici un algorithme écrit en langage naturel et en langage de programmation

Langage naturel
Entrée Saisir A **Traitement des données** Affecter à B la valeur A/13 Affecter à C la valeur arrondie à l'unité de B **Sortie** Si B = C Alors Afficher "A est divisible par 13" Sinon Afficher "A n'est pas divisible par 13"

Python

```
A=int(input('A='))
B=A/13
C=round(B,0)
if B==C:
    print(A,'est divisible par 13')
else:
    print(A,'n est pas divisible par 13')
```

Commentaires :
"==" est le symbole d'égalité ; "=" celui d'affectation.

TI

```
PROGRAM:DIVISIBI
:Input A
:A/13→B
:round(B,0)→C
:If B=C
:Then
:Disp "EST DIV P
AR 13"
:Else
:Disp "N EST PAS
 DIV PAR 13"
:End
```

Exercices

Exercice 1 :

Écrire un programme permettant de vérifier si un nombre donné est divisible par 13 en effectuant un test sur le reste de la division de ce nombre par 13.
Tester ce programme à l'aide d'une calculatrice programmable ou d'un logiciel.

Voici la syntaxe permettant d'afficher le reste d'une division euclidienne :

Langage naturel	Python	Scilab	TI	CASIO
Reste de la division euclidienne de A par B	A%B	reste(A,B)	A–B*ent(A/B)	A–BxInt (A÷B)

Exercice 2 :

Voici un algorithme écrit en langage naturel :

Langage naturel
Entrée Saisir x Saisir y **Traitement des données** Si x < 5y Alors Affecter à x la valeur 10x Sinon Affecter à y la valeur 10y **Sortie** Afficher xy

Dans chacun des programmes ci-dessous traduisant l'algorithme précédent, les instructions conditionnelles ont été supprimées.

Python	TI

`x=float(input('x='))`
`y=float(input('y='))`
`x<5*y`
`x=10*x`
`y=10*y`
`print(x*y)`
	Input X
	Input Y
	X<5*Y
	10*X→X
	10*Y→Y
	Disp X*Y

1) Corriger en complétant un des programmes par les instructions conditionnelles manquantes.
2) À l'aide d'une calculatrice programmable ou d'un logiciel, tester ce programme pour x = 5 et y = 9.

Exercice 3 :

On considère l'algorithme suivant donné en langage naturel :

Entrée
Saisir dans l'ordre croissant trois nombres entiers A, B, C

Traitement des données
Affecter à M la valeur de A^2
Affecter à N la valeur de B^2
Affecter à X la valeur de M + N
Affecter à Y la valeur de C^2

Sortie
Si X = Y Alors
Afficher "A, B, C est un triplet de Pythagore"
Sinon
Afficher "A, B, C n'est pas un triplet de Pythagore"

1) Rédiger un programme (langage au choix) traduisant cet algorithme.
2) À l'aide d'une calculatrice programmable ou d'un logiciel, tester ce programme pour trouver quelques triplets de Pythagore.

Exercice 4 :

Écrire un programme qui affiche le plus grand de deux nombres saisis en entrée.
Tester ce programme à l'aide d'une calculatrice programmable ou d'un logiciel.

Exercice 5 :

Écrire un programme qui demande en entrée à un client le montant total de ses achats.
En fonction de la somme dépensée, le programme affiche en sortie le prix à payer :
- Si la somme dépensée est strictement inférieure à 75 €, il obtient 5 % de remise.
- Si la somme dépensée est supérieure à 75 €, il obtient 8 % de remise.

Tester ce programme à l'aide d'une calculatrice programmable ou d'un logiciel.

2. Boucle

Définition

Une **boucle** permet de répéter un traitement un certain nombre de fois.

a. Première forme : Boucle « Pour » (boucle bornée)

Exemple

```
variables
i : nombre
...
début algorithme
...
pour i variant de 1 à 10
    instructions
fin pour
...
fin algorithme
```

L'algorithme ci-dessus va exécuter **dix fois** les *instructions* situées dans la boucle.

Plus précisément :

- **La première fois** que l'algorithme va rencontrer l'instruction *« pour i variant de 1 à 10 »*, il va affecter la valeur 1 à i; comme i est strictement inférieur à 10, il passe ensuite aux *instructions* situées **à l'intérieur** de la boucle.

- après les avoir exécutées, la ligne *« fin pour »* va faire boucler l'algorithme et le faire revenir à l'instruction *« pour i variant de 1 à 10 »*
- **La seconde fois (et les fois suivantes…)** que l'algorithme va exécuter l'instruction *« pour i variant de 1 à 10 »*, il va :
  - ajouter 1 à i (on dit **incrémenter i**)
  - si i est inférieur ou égal à 10, il passe aux *instructions* situées **à l'intérieur** de la boucle
  - si i est supérieur à 10, il passe aux instructions situées **après** la ligne *« fin tant que »*

Remarque

Si l'on souhaite incrémenter l'indice avec une valeur différente de 1 on utilise l'instruction : `pour i variant de ... à ... avec un pas de ...` Par exemple : `pour i variant de 2 à 8 avec un pas de 2` *i* va prendre successivement les valeurs : 2; 4 ; 6; 8 (et il quittera la boucle lorsqu'il vaudra 10)

Exemple

L'algorithme ci-dessous affiche les carrés des 21 premiers nombres entiers naturels (de 0 à 20) **variables**

```
n : nombre
c : nombre
début algorithme
pour n variant de 0 à 20
   c prend la valeur n*n
   afficher "Le carré de ", n, " est ", c
fin pour
fin algorithme
```

b. Deuxième forme : Boucle « Tant que » (boucle non bornée)

```
tant que condition
    instructions
fin tant que
```

L'algorithme ci-dessus effectuera les instructions tant que la condition sera vraie.
Dès que la condition devient fausse, on se branchera sur l'instruction suivant le fin tant que.

Exemple

```
variables
nombre, somme: nombres
continuer: texte

début algorithme
continuer prend la valeur "oui" // initialisation
afficher 'entrez un nombre :'
lire nombre
somme prend la valeur nombre
tant que continuer="oui"
   afficher "entrez le nombre suivant"
   lire nombre
   somme prend la valeur somme+nombre
   afficher "voulez-vous continuer (oui/non)"
   lire continuer
fin tant que
afficher "la somme des nombres entrés est" somme
fin algorithme
```

L'algorithme précédent demande à l'utilisateur d'entrer un premier nombre**.**
Puis il lui demande **s'il veut entrer un autre nombre**.
Tant que l'utilisateur répond *« oui »*, l'algorithme lui **demande un nouveau nombre** qu'il **additionne** au contenu de la variable « somme ».
Dès que l'utilisateur répond autre chose que *« oui »*, l'algorithme **sort de la boucle**, **affiche le total** et se **termine**.

Remarque

On utilise généralement une instruction *« pour »* lorsqu'on connaît le nombre d'itérations à réaliser dès le début de la boucle et une instruction *« Tant que »* lorsque ce nombre est inconnu ou difficile à déterminer.

Syntaxe pour les boucles dans 4 langages de programmation :

Langage naturel	Python	Scilab	TI	CASIO
Tant que *Condition est vraie* **Faire** *Instructions*	**while** *Condition*: *Instructions*	**while** *Condition* *Instructions* **end**	**:While** *Condition* **:***Instructions* **:End**	**While** *Condition*↵ *Instructions*↵ **WhileEnd**↵

Langage naturel	Python (*)	Scilab	TI	CASIO
Pour *i* **allant de** *3* **à** *7* **Faire** *Instructions*	**for** *i* **in range(***3,8***):** *Instructions*	**for** *i =3:7* *Instructions* **end**	**:For**(*i*,*3*,*7*) **:***Instructions* **:End**	**For** *3*→*i* **To** *7*↵ *Instructions*↵ **Next**

(*) En Python, **range**(*a*,*b*) désigne la séquence des entiers *n* vérifiant $a \leq n < b$.
range(*b*) désigne la séquence des entiers 0, 1, …, $b-1$

Syntaxe pour sortir d'une boucle :

Langage naturel	Python	Scilab	TI	CASIO
Sortir de la boucle	break			

Exercices

Exercice 1 :

Voici un algorithme écrit dans différents langages de programmation :

Python	Scilab	TI	CASIO

`S=0`
`for i in range(101):`
`  S=S+i`
`print(S)`
	`S=0`
	`for i=1:100`
	`  S=S+i`
	`end`
	`afficher(S)`
		`:0→S`
		`:For(I,1,100)`
		`:S+I→S`
		`:End`
		`:Disp S`
			`0→S`
			`For 1→I To 100`
			`S+I→S`
			`Next`
			`S`

1) Quel problème permet de résoudre cet algorithme.
2) a) Écrire un programme (langage au choix) permettant de calculer la somme des entiers de 34 à 145. Tester ce programme à l'aide d'une calculatrice ou d'un logiciel.
   b) Même question pour la somme des entiers de 67 à 456.

Exercice 2 :

Rédiger et tester un programme permettant de calculer la somme des entiers naturels pairs inférieure à 1000.

Exercice 3 :

On dépose 25€ dans une tirelire.

L'algorithme suivant, écrit en langage naturel, permet de calculer le nombre de pièces de 1€ ou 2€ ajoutés de façon aléatoire dans la tirelire avant de dépasser 50€.

Langage naturel
Initialisation
Affecter à S la valeur 25
Affecter à D la valeur 0
Traitement des données
Tant que S<50
Faire
Affecter à A la valeur aléatoire 1 ou 2
Affecter à S la valeur S+A
Affecter à D la valeur D+1
Afficher A
Sortie
Afficher D

Rédiger et tester un programme traduisant cet algorithme.

Exercice 3 :

Soit u la suite définie pour tout entier naturel n par $u_n = \frac{1}{4}u_n - 3$

1) Écrire un programme qui permet d'afficher les 10 premiers termes de la suite u.
   Quelles conjectures peut-on faire concernant le comportement de la suite.
2) Écrire un programme qui affiche la première valeur de n pour laquelle