

Algorithmique et programmation : les structures de base

I. Qu'est-ce qu'un algorithme ?

1. Un exemple d'algorithme : parité d'un entier

Décrivons un algorithme permettant de comparer deux nombres a et b et de donner le plus grand des deux :

- On réalise la différence $a - b$.
- On compare cette différence à 0.
- Si $a - b \geq 0$ alors a est le plus grand des deux nombres et $a - b \leq 0$ alors b est le plus grand des deux nombres.

En étudiant ce processus de plus près on peut distinguer trois parties :

✚ **Une ou des entrées** : ici deux entrées : les nombres entiers a et b .

✚ **Une suite d'instructions** : on pourrait résumer la suite des instructions de cet algorithme comme suit :

instruction 1 : réaliser la différence $a - b$.

instruction 2 : comparer cette différence à 0.

✚ **Une ou des sorties** : ici une sortie : a ou b suivant les cas.

Remarque : les instructions successives ne s'appliquent pas uniquement aux entrées a et b . Par exemple, l'instruction 2 ne s'applique ni à a ni à b mais à la différence $a - b$ « produit » par cet algorithme.

2. Définition de l'algorithme

Un **algorithme** peut être considéré comme une « processus » formée de **trois blocs** :

- Les **ENTREES** qui peuvent être des nombres, des caractères, des listes de nombres, des chaînes de caractères... En seconde, les entrées seront essentiellement des **nombres** de type entier ou décimal comme on le verra plus bas.
- Une suite **finie** d'**INSTRUCTIONS** qui s'appliquent, dans un ordre déterminé, à des entrées ou à des données produites par les instructions précédentes.
- Les **SORTIES** qui sont les résultats obtenus par l'application de la suite, **finie** et **ordonnée**, des instructions

Cette description peut être schématisée comme suit :



Remarques

- Le mot « algorithme » vient du nom du mathématicien persan al-Khuwarizmi (9^{ème} siècle) dont le traité d'algèbre décrit des procédés de calcul à suivre étape par étape pour résoudre des problèmes qui se ramènent souvent à la résolution d'équations.
- Divers algorithmes sont en fait connus dès l'Antiquité, par exemple l'algorithme d'Euclide donnant le plus grand diviseur commun de deux nombres entiers, ou encore le calcul d'approximations du nombre π ...

II. Variables et affectations

Pour mémoriser les données initiales, ou les résultats intermédiaires des calculs, au cours de l'exécution d'un algorithme, on utilise des **variables**.

- Du point de vue de l'ordinateur, une variable est une zone de **mémoire** au contenu de laquelle on accède via une **adresse**.
- Du point de vue algorithmique, une variable a un nom fixe et une **valeur** qui peut changer au cours de l'exécution de l'algorithme.
- La nature et le rôle des variables en informatique et en mathématique sont donc différents, bien qu'on utilise le même mot.
- Pour signifier que l'on affecte la **valeur** 5 à la **variable** nommée a , on écrira par convention : $a \leftarrow 5$

Exercice 1 corrigé : affectations

Considérons l'algorithme ci-dessous :

algorithme 1

a, b et c sont de type entier

$a \leftarrow 1$

$b \leftarrow 2$

$c \leftarrow a$

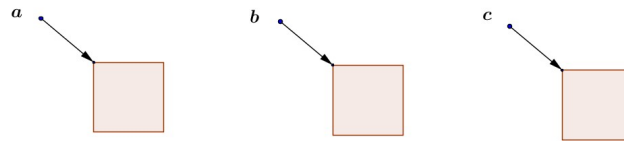
$c \leftarrow a + b$

$b \leftarrow 3b - 10$

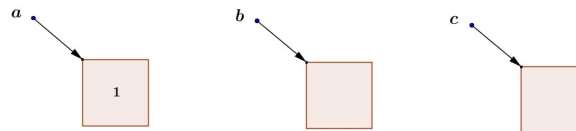
Afficher *b*

Résumons l'état des variables après chaque instruction :

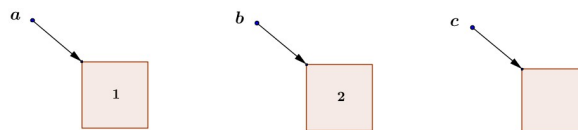
- Dans un premier temps, trois variables sont réservées mais ne contiennent aucune donnée :



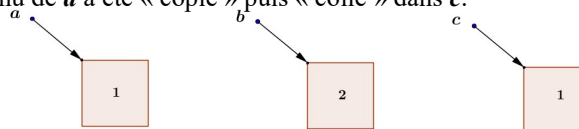
- Après l'instruction 1 :



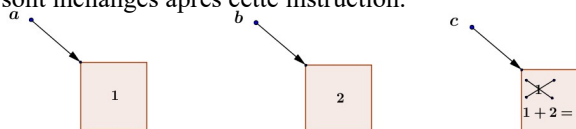
- Après l'instruction 2 :



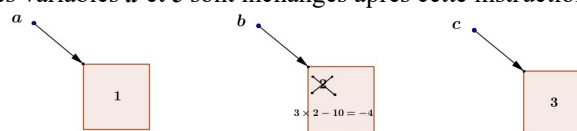
- Après l'instruction 3 : remarquons que le contenu de la variable *a* ne change pas après cette instruction. Tout se passe comme si le contenu de *a* a été « copié » puis « collé » dans *c*.



- Après l'instruction 4 : le contenu de la variable *c* est **remplacé** par la somme des contenus des variables *a* et *b*. Ce remplacement est dû au fait qu'une même variable ne peut contenir des données distinctes. Les contenus des variables *a* et *b* sont inchangés après cette instruction.



- Après l'instruction 5 : le contenu de la variable *b* est **remplacé** par $3 \times 2 - 10 = -4$ où 2 est le contenu de la variable *b* avant l'instruction. Les contenus des variables *a* et *c* sont inchangés après cette instruction.



- Après l'instruction 5 : le contenu de la variable *b* est affiché, autrement dit : -4.

Il sera plus pratique, à l'avenir, de présenter l'état des variables comme suit :

Numéro d'instruction	<i>a</i>	<i>b</i>	<i>c</i>
1	1		
2	1	2	
3	1	2	1
4	1	2	3
5	1	- 4	3

Exercice 2 : à vous

1. Dans l'algorithme suivant, préciser les entrées, les instructions et les sorties :

algorithme 2

a, b et c sont de type entier
Saisir le nombre ***a***
 $b \leftarrow a^2$
 $c \leftarrow 2b$
 $a \leftarrow b - 5a$
 $b \leftarrow a + b + c + 3$
Afficher ***b***

2. Exécuter l'algorithme 2 après saisie de l'entier 6 pour la variable ***a*** en utilisant le tableau de mémoires ci-dessous :

Numéro d'instruction	<i>a</i>	<i>b</i>	<i>c</i>
1			
2			
3			
4			
5			

Exercice 3 corrigé : saisies et affichages

En langage Python, la commande : « saisir A » se code :

- Lorsque ***A*** est une chaîne de caractères :

```
A= input("A=") #si A est une chaîne de caractères
```

- Lorsque ***A*** est un entier :

```
A=int(input("A=")) #si A est un entier
```

- Lorsque ***A*** est un réel :

```
A=float(input("A=")) #si A est un réel
```

la commande : « afficher A » se code :

```
print(A) #afficher A
```

la commande : « affecter à A la valeur B » se code :

```
8 A=int(input("A="))
9
0 B=int(input("B="))
1
2 A=B #affecte le contenu de la variable A dans la variable B
```

Point Python 1

Après la touche # (dièse) on peut écrire un commentaire qui ne sera évidemment pas considéré comme une instruction lors de l'exécution du programme.

On considère l'algorithme :


A, B et C sont de type réel
Saisir ***A***
 $B \leftarrow 2$
 $C \leftarrow A$
 $C \leftarrow A + B$
 $B \leftarrow 3B - 10$
Afficher ***B***

En langage Python, il se code :

```
A=float(input("A="))#saisir le réel A
B=2# affecter à B la valeur
C=A #affecter à C la valeur A
A=A+B#affecter à A la valeur A+B
B=3*B-10#affecter à B la valeur 3*B-10
print('La valeur de B est',B)#afficher B
```

Point Python 2

On peut réaliser un affichage composé d'un « mélange » de caractères et de contenu de variables. La partie « message » s'écrit entre guillemets et le message est séparé de la variable par une virgule.

Pour exécuter ce programme, on appuie sur la touche . Dans la console, on obtient :

```
1 A=float(input("A="))#saisir le réel A
2 B=2# affecter à B la valeur
3 C=A #affecter à C la valeur A
4 A=A+B#affecter à A la valeur A+B
5 B=3*B-10#affecter à B la valeur 3*B-10
6 print('La valeur de B est',B)#afficher B
```

A=

On saisit, par exemple, la valeur 6,7 puis on appuie sur la touche « Entrée » du clavier. Dans la console, on obtient :

```
7
8 A=float(input("A="))#saisir le réel A
9 B=2# affecter à B la valeur
10 C=A #affecter à C la valeur A
11 A=A+B#affecter à A la valeur A+B
12 B=3*B-10#affecter à B la valeur 3*B-10
13 print('La valeur de B est',B)#afficher B
14
```

A=6.7
La valeur de B est -4

Point Python 3 : en langage Python, a^n se code : $a**n$.

Exercice 4 : vitesse d'arrêt

Par sécurité, un véhicule doit respecter une distance minimale avec le véhicule qui le précède, afin d'avoir le temps de freiner avant une collision. Ce temps correspond à celui de la perception puis de la réaction du conducteur, ainsi que des possibilités de freinage du véhicule.

Ce temps est fonction de la vitesse v du véhicule. Des études statistiques ont montré que cette distance D peut être calculée par la formule : $D = 8 + 0,2v + 0,003v^2$, où v est en km/h et D en mètres.

1. Calculer la distance de freinage pour une vitesse de 100 km/h puis de 200 km/h

Vitesse en km/h		
Distance d'arrêt en m		

2. Ecrire un algorithme qui affiche cette distance D en sortie lorsqu'est fournie la vitesse v en entrée.

Traduire cet algorithme en langage Python. On attend un message final du type : « Pour une vitesse de, v km/h , la distance de freinage est :, D , mètres ».

Algorithme « vitesse d'arrêt »	Programme en langage Python

Exercice 5 : indice de masse corporelle avec deux variables

On souhaite écrire un algorithme permettant de calculer l'indice de masse corporelle (IMC) d'un individu. Cet indice se calcule à l'aide de la formule : $IMC = \frac{masse}{taille^2}$ (en kg/m²).

1. Quelles sont les variables qu'il va falloir utiliser dans cet algorithme ?

.....
.....

2. Quelle sera l'instruction de sortie ?.....

3. Ecrire cet algorithme et son codage en langage Python.

Algorithme IMC	Programme en langage Python

4. Utiliser le programme pour obtenir l'IMC d'un individu de taille 1,70m et de poids 65kg

Exercice 6 : coût d'un séjour avec deux variables

Une association organise un séjour balnéaire dans un hôtel. Le coût journalier par personne de la pension à l'hôtel est 80€ et celui du voyage est de 300€.

1. Quel est le coût total d'un séjour d'une semaine pour 10 personnes ?

2. Ecrire un algorithme qui affiche le coût total du séjour selon le nombre J de jours réservés et le nombre N de participants.
Coder cet algorithme en langage Python.

Algorithme coût	Programme en langage Python

Exercice 7

Point Python 4

En langage Python :

- \sqrt{a} s'écrit `sqrt(a)`.
- L'utilisation des fonctions usuelles : racine, sinus, cosinus... nécessite le chargement d'un module nommé « math ».
- Ce chargement s'effectue au début du programme en écrivant : `from math import*`. Voir ci-dessous.

```

from math import* #chargement du module "math"

#####Exercice 7 : aire, périmètre et diagonale d'un rectangle#####
#####ENTREES#####
l=float(input('l='))
L=float(input('L='))
#####

```

1. Ecrire un algorithme qui prend en entrées la largeur l et la longueur L d'un rectangle et qui, en sorties, donne l'aire, le périmètre et la longueur de la diagonale de ce rectangle.
2. Traduire cet algorithme en langage Python avec des messages finaux adaptés à la situation.

Algorithme aire périmètre	Programme en langage Python

III. La structure Si-Alors

- Un algorithme ne contient pas seulement des instructions de manipulation des données à exécuter les unes après les autres, mais aussi des instructions dites de contrôle ou de structure : **conditions** et **boucles**, qui ont un effet sur l'exécution des autres instructions.
- Le premier type de telles instructions est celle permettant une **exécution conditionnelle**. Elle a pour but d'exécuter des instructions seulement dans le cas où une certaine condition est réalisée. On peut la symboliser comme suit :

```

Instructions...
Si condition alors
    instructions à réaliser si
    la condition est réalisée
fin Si
Instructions...

```

- La condition doit pouvoir être évaluée à **vrai** ou **faux**. Il s'agit donc d'une **expression logique**, plus ou moins compliquée.
 - Si la **condition** est vérifiée les instructions situées entre le **Si** et le **fin Si** sont exécutées, celles situées après le **fin Si** le sont ensuite. Si la **condition** n'est pas vérifiée, les instructions situées entre le **Si** et le **fin Si** ne sont pas exécutées et « on passe » à l'instruction qui suit immédiatement le « **fin Si** ».
- L'organigramme ci-contre permet de visualiser cette structure : les rectangles représentent des instructions.

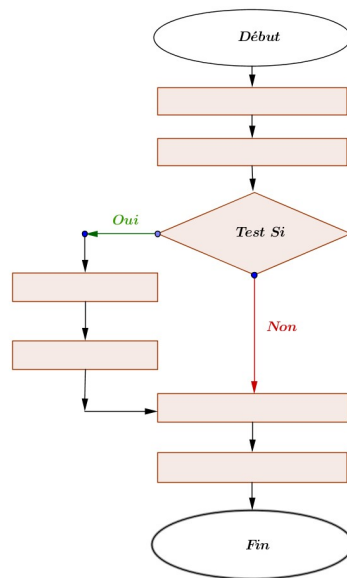
Point Python 5 : en langage Python la structure **Si-alors** se code :

```

1
2
3
4
5
6
7
8 from math import* #chargement du module "math"
9
10
11 instructions...
12
13 if (condition):
14     instructions..
15
16 instructions

```

On remarquera qu'il n'est pas nécessaire, en Python, d'indiquer la « fin-si ». L'indentation, autrement dit le décalage vers la droite des instructions suivant le Si, suffisent à indiquer la « fin-si ».



Exercice 8 : la structure Si-alors : corrigé

Point Python 6 : en langage Python on compte les tests suivants :

- Tester le fait que deux nombre A et B sont égaux : **if ($A=B$)** :. On remarquera le double « = » plutôt qu'un seul « = » destiné, lui, à l'affectation.
- Tester si deux nombres A et B sont différents : **if ($A \neq B$)** :.
- Tester si le nombre A est strictement supérieur au nombre B : **if ($A > B$)** : ou **if ($B < A$)** :.
- Tester si le nombre A est supérieur ou égal au nombre B : **if ($A \geq B$)** : ou **if ($B \leq A$)** :.

1. Ecrire un algorithme qui prend deux nombres réels A et B en entrées et qui affiche le plus grand des deux en sortie.

A et B sont de type réel
 Saisir A et B
Si $A < B$ **alors**
 $A \leftarrow B$
fin Si
 Afficher A

Commentaire :

- si $A < B$, il faut afficher B d'où l'affectation de B à A . Si $A \geq B$, il n'y a rien à changer puisque A est le plus grand.
- Si $A = B$, on peut considérer que l'un quelconque de ces deux nombres est le plus grand au sens large.

2. Traduire cet algorithme en langage Python. On affichera un message adapté à la situation.

```

from math import* #chargement du module "math"

#####Exercice corrigé 8 : max de deux nombres###

A=float(input('A='))
B=float(input('B='))
print(A)
if (A<B):
    A=B
print('le plus grand des deux nombres est : ',A)
  
```

Exercice 9 : la structure si-alors

1. On considère l'algorithme ci-dessous :

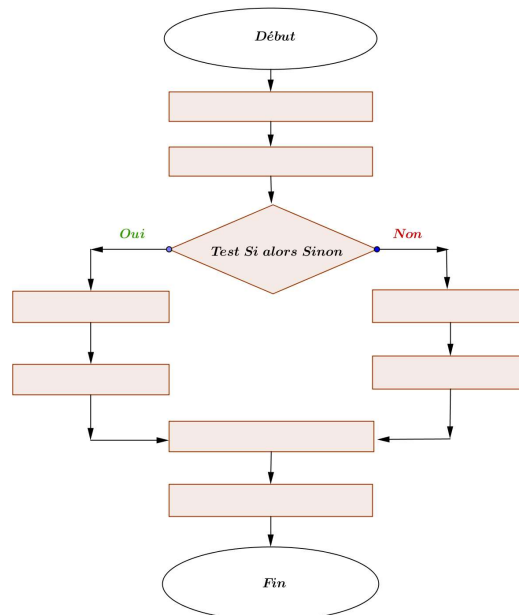
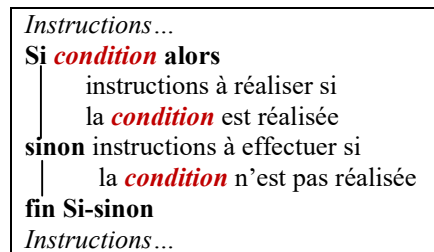
- 1.a. Tester cet algorithme à plusieurs reprises, sur des entrées de votre choix.
- 1.b. Que permet-il d'obtenir ?

2. Traduire cet algorithme en langage Python. On affichera un message adapté à la situation.

L'algorithme	Le programme python
<p>A, B, C et m sont de type réel Saisir A, B et C $m \leftarrow A$ Si $A \geq B$ alors $m \leftarrow B$ fin Si Si $m \geq C$ alors $m \leftarrow C$ fin Si Afficher m</p>	

IV. La structure Si-Alors-Sinon

- Pour exécuter certaines instructions dans le cas où une condition est réalisée et d'autres dans le cas où elle ne l'est pas, on utilise la structure **Si-alors-Sinon**. On peut la symboliser comme suit :



- Si la *condition* est vérifiée les instructions situées entre le **Si** et le **sinon** sont exécutées, celles situées entre le **sinon** et le **fin Si-sinon** ne le sont pas.
- Si la *condition* n'est pas vérifiée, les instructions situées entre le **Si** et le **sinon** ne sont pas exécutées, celles situées entre le **sinon** et le **fin Si-sinon** le sont.
- Dans tous les cas, les instructions situées après le **fin Si-sinon** sont exécutées. L'organigramme ci-dessus permet de visualiser cette structure.

Point Python 7 : en langage Python la structure *Si-alors-sinon* se code :

```

13
14 instructions...
15
16 if (condition):
17     instructions... #exécutées si la condition est vérifiée
18
19 else:
20     instructions...#exécutées si la condition n'est pas vérifiée
21
22 instructions...
23

```

On remarquera, après le else, la présence du « : » puis un passage à la ligne avec indentation.

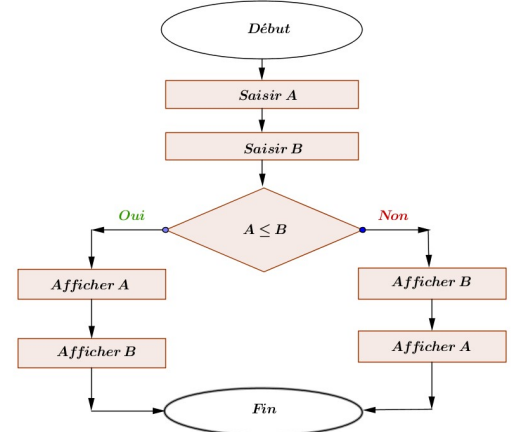
Exercice 10 : la structure *Si-alors-sinon* : corrigé

1. Ecrire un algorithme qui prend deux nombres réels en entrées et qui les affiche dans l'ordre croissant.

```

A et B sont de type réel
Saisir A et B
Si A ≤ B alors
    Afficher A
    Afficher B
sinon
    Afficher B
    Afficher A
fin Si-sinon

```



2. Traduire cet algorithme en langage Python. On affichera un message adapté à la situation.

```

14 A=float(input('A='))
15 B=float(input('B='))
16
17 if (A<=B):
18     print (A, '<=', B)
19
20 else:
21     print (B, '<=', A)

```

Exercice 11 : structure *si-alors-sinon* : contrat

Une maison d'édition recrute des vendeurs à domicile pour placer des collections de livres d'art. Elle propose, au choix, deux types de contrats :

contrat 1 : un salaire mensuel fixe de :3150 € + 165 € par collection vendue.

contrat 2 : pas de salaire fixe, mais 240 € par collection vendue.

1. Compléter l'algorithme ci-dessous qui prend le nombre x de collections vendues en entrée et qui renvoie, en sorties, le numéro du contrat le plus intéressant puis le revenu qu'il permettra de gagner pour ces x collections vendues.

En cas de revenus identiques, le contrat 1 sera privilégié.

2. Traduire cet algorithme en langage Python. On affichera les messages adaptés à la situation.

L'algorithme contrat	Le programme python
x , R_1 et R_2 sont de type entier Saisir x $R_1 \leftarrow 3150 + 65x$ $R_2 \leftarrow$ Si alors Afficher : « choisir le contrat ... et vous gagnerez sinon Afficher : « choisir le contrat... et vous gagnerez Fin Si-sinon	

3. Tester le programme successivement pour une vente de 10 collections puis pour une vente de 50

Exercice 12 corrigé : réciproque du théorème de Pythagore

Point Python 8 : structure *elif* :

- Il s'agit d'une structure propre à Python et qui évite une imbrication de **Si-Alors-sinon** lorsque l'on est confronté à de multiples conditions. Voir la capture d'écran ci-dessous.
- La condition 1 est testée, si elle est vérifiée le bloc d'instructions 1 est exécuté puis on passe au bloc d'instructions 5. Si la condition 1 n'est pas vérifiée, la condition 2 est testée.
- Si la condition 2 est vérifiée, le bloc d'instructions 2 est exécuté puis on passe au bloc d'instructions 5. Si la condition 2 n'est pas vérifiée, la condition 3 est testée.
- Si la condition 3 est vérifiée, le bloc d'instructions 3 est exécuté puis on passe au bloc d'instructions 5. Si la condition 3 n'est pas vérifiée, le bloc d'instructions 4 est exécuté puis on passe au bloc d'instructions 5.
- On peut évidemment utiliser autant de blocs **elif** : qu'on le souhaite.

```
14 instructions...
15
16 if (condition 1):
17     bloc d'instructions 1
18
19 elif (condition 2):
20     bloc d'instructions 2
21
22 elif (condition 3):
23     bloc d'instructions 3
24
25 else:
26     bloc d'instructions 4
27
28 bloc d'instructions 5
29
```

Ecrire un programme Python qui prend, en entrées, les longueurs $BC = a$, $AC = b$, $AB = c$ des trois côtés d'un triangle ABC et qui renvoie un message adapté précisant si le triangle est rectangle ou non. Si le triangle est rectangle, le message doit préciser en quel point.

```
a=float(input('a='))
b=float(input('b='))
c=float(input('c='))

if (b**2+c**2==a**2):
    print ('ABC est rectangle en A')

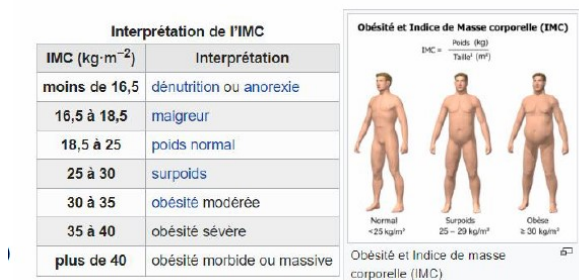
elif (a**2+c**2==b**2):
    print ('ABC est rectangle en B')

elif(a**2+b**2==c**2):
    print ('ABC est rectangle en C')

else:
    print ("ABC n'est pas rectangle")
```

Exercice 13 : IMC avec elif

On rappelle que l'indice de masse corporelle (IMC) d'un individu se calcule à l'aide de la formule : $IMC = \frac{masse}{taille^2}$ (en kg/m^2) où M est la masse en kg et T la taille en m. Ci-dessous figure l'interprétation de l'IMC :



1. Ecrire un programme Python que permet la saisie de la masse en kg et la taille en m d'un individu et de lui préciser dans quel cas de figure il se situe.
2. Tester ce programme sur vos propres paramètres de masse et de taille.

Exercice 14 : fonction affine par morceaux

Antoine, passionné de modélisme, casse au moins une dizaine d'hélices de son drone à chaque fois qu'il l'utilise sur les palges de Normandie. Il achète donc régulièrement sur Internet des hélices en grande quantité. Son site favori lui propose les tarifs dégressifs suivants :

Quantité q d'hélices	$q < 10$	$10 \leq q < 50$	$50 \leq q < 100$	$q \geq 100$
Prix en euros	1 € par hélice et 5 € de frais de port.	0,8 € par hélice et 6 € de frais de port	0,75 € par hélice et 6 € de frais de port	0,5 € par hélice et 7 € de frais de port

1. Ecrire un programme en langage Python qui permet la saisie du nombre d'hélices qu'Antoine souhaite acheter et qui affiche le prix qu'il aura à payer pour cet achat.
2. Tester ce programme sur les valeurs de q suivantes : 5, 20, 60 et 200.

Point Python 9 : le connecteur logique **and** (et) se présente comme suit :

```

14
15 instructions...
16
17 if(condition 1 and condition 2):
18     bloc d'instructions 1
19
20 bloc d'instructions 2
21

```

Il faut lire :

- si les conditions 1 et 2 sont réalisées **en même temps** alors on exécute le bloc d'instructions 1 puis le bloc d'instructions 2.
- si l'une au moins des deux conditions n'est pas réalisée, le bloc d'instructions 1 n'est pas exécuté et on passe au bloc d'instructions 2

Point Python 10 : le connecteur logique **or** (ou) se présente comme suit :

```

13
14
15 instructions...
16
17 if(condition 1 or condition 2):
18     bloc d'instructions 1
19
20 bloc d'instructions 2
21

```

Il faut lire :

- si **l'une au moins** des conditions 1 et 2 est réalisée alors on exécute le bloc d'instructions 1 puis le bloc d'instructions 2.
- si **aucune** des deux conditions n'est réalisée, le bloc d'instructions 1 n'est pas exécuté et on passe au le bloc d'instructions 2

Exercice 15 : connecteurs logiques

1. Recopier et compléter le programme python ci-dessous qui permet la saisie de l'âge (en années) de l'utilisateur :
 - s'il a moins de 18 ans (au sens large) ou plus de 60 ans (au sens large), le message : « l'entrée au musée est gratuite pour vous » est affiché en console ;
 - s'il a plus 18 ans et moins de 25 ans (au sens large), le message : « l'entrée au musée vous coûtera 10€ » est affiché en console ;
 - dans tout autre cas, le message : « l'entrée au musée vous coûtera 20€ » est affiché en sortie.
2. Tester ce programme sur les valeurs 5 ans, 20 ans, 30 ans et 70 ans.

```

age=int(input('âge='))

if(..... or .....):
    print ("L'entrée au musée est gratuite pour vous")

elif(..... and .....):
    print(".....")

else:
    print(".....")

```

V. La boucle bornée « Pour »

- Pour répéter un bloc d'instructions un nombre de fois donné, on dispose de la boucle **Pour**, que l'on peut symboliser comme suit :

```
instructions...
Pour i de 1 à n faire
|   instructions à effectuer
fin Pour
instructions...
```

- la variable **i** est un compteur, elle augmente automatiquement de 1 à chaque « tour ».

Point Python 11 : en langage Python la boucle **Pour** se code

```
14 instructions...
15
16 for i in range(n):
17     instructions...
18
19 instructions...
```

- On remarquera qu'il n'est pas nécessaire, en Python, d'indiquer la « **fin-Pour** ». Encore une fois, l'indentation suffit à indiquer la « **fin-Pour** ».
- Il est capital de comprendre que le codage « **for i in range(n)** » signifie que le compteur **i** prend les valeurs de 0 à $n-1$. Il s'agit d'ailleurs bien de n valeurs.
- Si l'on souhaite que **i** prenne les valeurs de 1 à n , il faudra coder : « **for i in range(1,n+1)** ».

Exercice 16 : somme d'entiers consécutifs

On considère l'algorithme ci-dessous :

```
S et i sont de type entier
Affecter à S la valeur 0
Pour i de 1 à 5 faire
|   S ← S + i
fin Pour
Afficher S
```

- 1.a. En complétant le tableau ci-dessous, déterminer les valeurs de S et de i à la fin du traitement.

Numéro d'instruction	S	i
<u>Initialisation</u>	0	1
1	1	2
2	3	3
3		
4		
5		

- 1.b. Recopier et compléter le programme Python ci-dessous qui code l'algorithme étudié ci-dessus :

```
S=..... #on initialise S
for i in range(.....):
    S=.....
    print('La somme des',n,'premiers entiers est S =',S)
```

- 2.a. Ecrire un algorithme qui prend un entier naturel n en entrée et affiche, en sortie, la somme des entiers de 1 à n .
- 2.b. Traduire cet algorithme en langage Python.

L'algorithme somme	Le programme python

- 3.a. Ecrire un algorithme qui prend un entier naturel n non nul en entrée et affiche, en sortie, le produit des entiers de 1 à n .
- 3.b. Traduire cet algorithme en langage Python.
- 3.c. Tester ce dernier programme sur les valeurs 5, 10 et 30. Que constate-t-on ?

Exercice 17 : table de multiplication d'un entier

1. Ecrire un algorithme qui prend un entier N en entrée et qui renvoie, en sortie, la table de N .
2. Traduire cet algorithme en langage Python.
- Par exemple, pour l'entrée $N = 7$, on obtiendra l'affichage :

```
>>>
7 x 0 = 0
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Exercice 18 : inflation

Un produit se vend 300 € en 2014. On considère que son prix augmente de 2% par an.

- En 2015 son prix sera $P_1 = 300 + \frac{2}{100} \times 300 = 306$ €.
 - En 2016 son prix sera $P_2 = 306 + \frac{2}{100} \times 306 = 312,12$ €.
1. Calculer le prix P_3 du produit en 2017 puis le prix P_4 en 2018 au centime près.
2. Ecrire un algorithme qui prend le nombre N d'années après 2014 et donne, en sortie, le prix du produit en l'année $2014 + N$.
- 3.a. Traduire cet algorithme en langage Python.
- 3.b. Que suffirait-il de modifier dans ce programme pour que s'affichent les prix du produit chaque année entre les années 2015 et $2014 + N$.

Exercice 19 corrigé : le hasard

Point Python 12 : nombres aléatoires :

- L'obtention de nombres aléatoires en Python, nécessite le chargement d'un module nommé «random». Ce chargement s'effectue au début du programme en écrivant : `from random import*`. Voir ci-dessous.
- La commande « **random()** » renvoie un réel aléatoire (au hasard) entre 0 et 1.
- Si n est un entier naturel, la commande « **randint(1,n)** » renvoie un entier aléatoire entre 1 et n .
Par exemple : `randint(1,3)` renvoie, aléatoirement, l'un des entiers 1, 2 ou 3.

Le programme ci-dessous simule un jeu entre opposant Vincent à Axel. Décrivez ce jeu.

```

• 9 from math import* #chargement du module "math"
• 10 from random import* #chargement du module "random"
11
12 #####Exercice 21 : connecteurs Logiques#####
• 13 de_un=randint(1,6)
• 14 de_deux=randint(1,6)
• 15 if (de_un>de_deux):
• 16     print('Vincent a gagné en obtenant un ',de_un,' et Axel a obtenu un ',de_deux)
• 17 else:
• 18     print('Axel a gagné en obtenant ',de_deux,' et Vincent a obtenu un ',de_un)

```

On a simulé le jeu suivant : Vincent fait rouler le dé 1 et Axel le dé 2. Si le numéro obtenu avec le dé 1 est supérieur à celui obtenu par le dé 2 c'est Vincent qui gagne sinon c'est Axel.

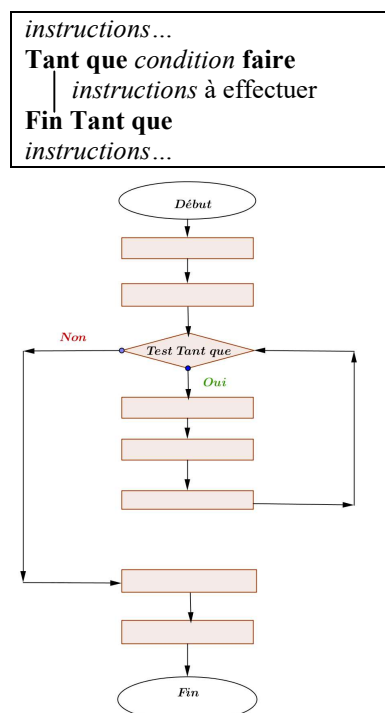
Exercice 20 : nombre de 6

1. Compléter l'algorithme ci-dessous prenant en entrée un entier N représentant le nombre de lancers d'un dé et affichant le nombre de 6 obtenus ainsi que la fréquence d'apparition des numéros 6 dans l'échantillon.
2. Traduire cet algorithme en langage Python.

L'algorithme nombre de 6	Le programme python
N , S et i sont de type entier Saisir N $S \leftarrow 0$ Pour i de 1 à N faire $d = \text{aléa}(1,6)$ Si $d = \dots$ alors $S \leftarrow \dots$ Fin si fin Pour Afficher	

VI. La boucle non bornée «Tant que »

- Pour répéter un bloc d'instructions tant qu'une condition est vérifiée, on dispose de la structure **Tant que** que l'on peut symboliser comme suit :



- Le test de la condition est effectué avant d'entrer dans la **boucle**.
- Par conséquent, si la condition n'est pas vérifiée avant l'entrée dans la boucle, on n'y entre pas, les instructions à l'intérieur de la boucle ne sont pas effectuées, et on passe à l'instruction suivant la boucle.

Point Python 13 : en langage Python la boucle *Tant que* se code :

```
11 instructions...
12
13 while (condition):
14     instructions...
15
16 instructions...
```

On remarquera qu'il n'est pas nécessaire, en Python, d'indiquer la « *fin-Tant que* ». Encore une fois, l'indentation y suffit.

Exercice 21 corrigé : exemple de boucle Tant que

On souhaite simuler le jeu suivant : on lance un dé équilibré à 6 faces jusqu'à obtenir un 6. On veut obtenir l'affichage :

- du nombre de lancers qu'il a fallu réaliser jusqu'à l'obtention du 6 ;
- la liste des résultats obtenus jusqu'à l'obtention du 6.

1. Ecrire un algorithme correspondant à cette situation.

Ci-dessous, n est le nombre aléatoire entre 1 et 6 représentant le dé et N le nombre de lancers nécessaires à obtenir un 6.

```
n et N sont de type entier
Affecter à n la valeur 0
Affecter à N la valeur 0
Tant que  $n \neq 6$  faire
    |  $n \leftarrow \text{aléa}(1,6)$ 
    |  $N \leftarrow N + 1$ 
    | Afficher n
fin Tq
Afficher N
```

2. Traduire cet algorithme en langage Python.

Le programme commenté

```
from math import*
from random import*

#####Exercice 21 : tant qu'un 6 n'est pas apparu

N=0#initialisation du nombre de lancers à 0
d=0#variable représentant le dé, on l'initialise à 0 pour rentrer une première
    #fois dans la boucle Tant que

while (d!=6):#Le numéro du dé n'est pas encore un six
    d=randint(1,6)
    N=N+1
    print('le résultat courant est un :',d)

print ('le premier 6 a été obtenu au bout de',N,'tentatives')
```

Un affichage possible

```
>>>
le résultat courant est un : 4
le résultat courant est un : 1
le résultat courant est un : 3
le résultat courant est un : 5
le résultat courant est un : 6
le premier 6 a été obtenu au bout de 5 tentatives
```

Exercice 22 : boucle tant que

En prévision d'une course de vélo, Fanny suit le programme d'entraînement suivant tous les samedis : elle parcourt 30 km le premier samedi, puis augmente chaque samedi de 5 km la distance parcourue. Elle arrêtera cet entraînement lorsque qu'elle aura parcouru et cumulé une distance totale supérieure à 1000 km.

1. Calculer la distance D parcourue le deuxième samedi et la distance totale T parcourue au bout de deux samedis d'entraînement.

.....

.....

2. Nous allons écrire un algorithme afin de connaître le nombre de samedis nécessaires à l'entraînement de Fanny.
Préciser les variables qui seront utiles à l'écriture de cet algorithme ?
3. Recopier et compléter l'algorithme 1 ci-dessous afin que qu'il affiche le nombre de samedis nécessaires à l'accomplissement de l'entraînement de Fanny.

```

D ← ....
T ← ....
N ← ....
Tant que..... faire
    | D ← ....
    | T ← ....
    | N ← ....
fin Tq
Afficher .....

```

4. Traduire cet algorithme en langage Python et donner le nombre de samedis nécessaires à l'accomplissement de l'entraînement de Fanny.
5. Modifier et programmer le programme afin qu'il affiche aussi la distance totale parcourue et la distance du dernier samedi.

Exercice 23 : somme des entiers impairs

On considère l'algorithme suivant :

```

Variables :  $S$  et  $i$  sont de type entier

Traitement
    Affecter à  $S$  la valeur 0
    Affecter à  $i$  la valeur 1
    Tant que  $i \leq 10$  faire
        |  $S \leftarrow S + i$ 
        |  $i \leftarrow i + 2$ 
    Fin Tant que
Sortie : Afficher  $S$ 

```

1. En complétant le tableau ci-dessous, déterminer les valeurs de S et de i à la fin du traitement.

<i>Numéro d'instruction</i>	S	i
<u>Initialisation</u>	0	1
1	1	3
2		
3		
4		
5		

- 2.a. Ecrire un algorithme qui prend un entier N en entrée et affiche, en sortie, la somme des entiers **impairs** entre 1 et N .
- 2.b. Traduire cet algorithme en langage Python.
- 2.c. Tester ce programme pour $N = 1000$. **Indication** : la somme attendue est 250 000

Exercice 24

Une ville de 10 000 habitants en 2014 voit sa population augmenter de 3% par an.

- 1.a. Montrer que chaque année sa population est multipliée par 1,03.
- 1.b. Calculer, à l'habitant près, le nombre d'habitants en 2015, 2016 et 2017.
2. Ecrire un algorithme qui renvoie, en sortie, le nombre d'années au bout duquel la population dépasse 13 400 habitants.
3. Traduire cet algorithme en langage Python.
4. Utiliser ce programme pour déterminer le nombre d'années au bout duquel la population a dépassé 13 400 habitants.

VII. Les fonctions

1. Un exemple pour comprendre

On peut simplifier l'écriture des programmes en utilisant des **fonctions**, sur le modèle des fonctions numériques étudiées en mathématiques.

On rappelle l'énoncé de l'exercice 4 :

Par sécurité, un véhicule doit respecter une distance minimale avec le véhicule qui le précède, afin d'avoir le temps de freiner avant une collision. Ce temps correspond à celui de la perception puis de la réaction du conducteur, ainsi que des possibilités de freinage du véhicule.

Ce temps est fonction de la vitesse v du véhicule. Des études statistiques ont montré que cette distance D peut être calculée par la formule : $D = 8 + 0,2v + 0,003v^2$, où v est en km/h et D en mètres. La vitesse d'arrêt D est donc **fonction** de la vitesse v .

- Pour programmer ce calcul de manière « systématique », on va définir une fonction, comme en mathématiques : on donne un nom, **arret**, à cette fonction et on dit qu'elle a pour **argument** v .
- En langage Python, on définit cette fonction par la commande **def** suivi du nom de la fonction, puis de son argument v , suivi de deux points.
- Le bloc d'instructions, indentées par rapport à la première ligne, permettent le calcul de D .
- Après le calcul de D , la renvoie le contenu de la variable D par la commande **return**.

```
def arret(v):  
    D=8+0.2*v+0.003*v**2  
    return(D)
```

- Pour calculer la distance d'arrêt correspondant à une vitesse de 100 km/h , il suffit « d'appeler » la fonction **arret** dans la console de la manière suivante :

```
>>> arret(100)  
58.0  
>>> |
```

2. Quelques applications de la notion de fonction

Exercice 25 corrigé : une fonction avec deux arguments

La surface S en m^2 de la peau d'un adulte est donnée approximativement par la formule : $S = \frac{\sqrt{L \times M}}{6}$ où L est la taille de la personne exprimée en m et M sa masse exprimée en kg .

1. Ecrire une fonction nommée **surface** qui prend deux arguments L et M et retourne la surface corporelle S .

```
from math import*  
  
def surface(L,M):  
    S=sqrt(L*M)/6  
    return(S)
```

2. Appeler cette fonction dans la console pour déterminer la surface corporelle d'un adulte qui mesure 1,80 m et pèse 75 kg .

```
>>> surface(1.80,75)  
1.9364916731037083  
>>>
```

Exercice 26 : fonction

Aux urgences, on recouvre les grands brûlés d'un médicament liquide afin de calmer les douleurs et de favoriser la cicatrisation. Pour recouvrir une surface de $1m^2$ de peau, on estime que 5L de médicament suffisent.

1. Programmer une fonction **soin** permettant d'obtenir la quantité de médicament suffisante pour calmer un individu connaissant sa taille et sa masse. La fonction **surface** de l'exercice 25 sera appelée dans ce programme.
2. Un individu brûlé se présente aux urgences. Il mesure 1,90 m et pèse 92 kg .
Quelle quantité de médicament sera-t-il nécessaire pour soigner ce patient ?

Exercice 27 : maximum-minimum

1. Programmer une fonction, nommée **maximum**, qui prend trois réels en arguments et renvoie le plus grand des trois ;
2. Programmer une fonction, nommée **minimum**, qui prend trois réels en arguments et renvoie le plus petit des trois.
3. Ecrire un programme qui permet la saisie de trois réels et affiche le plus grand des trois et le plus petit des trois.

Exercice 28

1. Programme une fonction nommée **fact** qui prend un entier $n \geq 1$ en argument et retourne le produit des entiers de 1 à n .
Info : en mathématiques, le produit des entiers de 1 à n se note $n!$ et s'appelle factorielle de l'entier n .
2. Ecrire un programme qui prend un entier $N \geq 1$ en entrée et qui, pour tout $1 \leq i \leq N$, affiche le nombre $i!$.

Pour la saisie de l'entier $N = 5$, un affichage possible serait :

```
1 != 1
2 != 2
3 != 6
4 != 24
5 != 120
```

3. Que remarque-t-on pour l'entrée $N = 30$?

Point Python 14 : opérations et fonctions de base en langage Python :

- Avec les entiers et les réels, il est possible de réaliser sept opérations de base.

Opération	symbole	Exemple
l'addition	+	2+3 donne 5
la soustraction	-	2-3 donne -1
la multiplication	*	2*3 donne 6
le calcul du quotient	/	9/8 donne 1.125
le quotient de la division euclidienne	//	29//8 donne 3
le reste de la division euclidienne	%	29%8 donne 5
les puissances	**	2**5 donne 32

- L'utilisation des fonctions usuelles : racine, sinus, cosinus... nécessite le chargement d'un module nommé « math ». Ce chargement s'effectue au début du programme en écrivant : `from math import*`. Voir ci-dessous.

```
10 from math import* #chargement du module "math"
11
12
13 a=float(input('a='))#saisir a
14 print(sqrt(a))#affiche la racine carrée de a
15 print(cos(a))#affiche le cosinus de a
16 print(sin(a))#affiche le sinus de a
17 print(tan(a))#affiche la tangente de a
18 print(abs(a))#affiche la valeur absolue de a
19
```