Initiation à la programmation en Python

Nom :

Prénom :

I-Conventions

- Une commande Python sera écrite en caractère gras.

Exemples : **print**("Bonjour")
max=**input**("Nombre maximum autorisé :")

- Le résultat de l'exécution d'un programme sera précédé du symbole ↳ .

Exemple : **print**("Bonjour")
↳ Bonjour

- ↳ ..... signifie qu'il faut compléter en recopiant ce qui apparaît sur l'écran.

II-Entrées, Sorties et Variables

1. Sortie

Pour permettre au programme en cours d'exécution d'afficher un texte ou un nombre on utilise la commande **print**.

Exemples :

print("Bonjour")	**print**(2)	**print**(Au revoir)
↳	↳	↳ File "<stdin>", line 1 print(Au revoir) SyntaxError : invald syntaxe

Le dernier exemple correspond à une erreur. Les textes (chaînes de caractères) que l'on souhaite afficher doivent être écrits entre des guillemets(" ou ').

2. Entrées

Afin de pouvoir dialoguer avec un programme en cours d'exécution, il est parfois nécessaire de donner une valeur (en utilisant le clavier) que demande le programme.

Exemple 1 : n=**input**("Entrer un nombre :")
↳ .....

A ce niveau, le programme attend que l'utilisateur entre un nombre au clavier. Si, par exemple on tape 4, alors dans toute la suite du programme, la variable n sera égale à 4.

Exemple 2 : n=**input**("Entrer votre nom :")
↳ Entrer votre nom :

Que se passe-t'il lorsque vous rentrez votre nom ? .....
En fait, vous rentrez une chaîne de caractères, il faut donc la rentrer avec des guillemets. Réessayez avec des guillemets.
À la place de **input()**, on peut utiliser **raw_input()**, qui renvoie toujours une chaîne de caractères. Faites l'essai en rentrant votre nom sans guillemets cette fois.

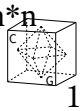
3. Variables

Dans tout programme informatique, on utilise des lettres. Une lettre peut être égale à un nombre, un texte. On les appelle les variables.

Exemple 1 : Le programme suivant demande à un utilisateur de rentrer une valeur et affiche en sortie le carré de cette valeur.

```
n=input("Entrer un nombre :")
p=n*n
print("Le carré de ce nombre est :"),p
```

n et p sont deux variables, la première est égale au nombre que choisit l'utilisateur. L'instruction p=n*n affecte à la variable p le nombre $n \times n = n^2$.



Exemple 2 : Le programme suivant demande à un utilisateur de rentrer son prénom et en retour lui dit bonjour.

```
prenom=raw_input("Entrer votre prenom :")
print("Bonjour"),prenom
```

prenom est une variable qui contient le prénom que choisit de rentrer l'utilisateur.

Exemple 3 : Taper la séquence suivante :

```
n=3
phrase="Bonjour tout le monde"
pi=3.14159
print n
print phrase
print pi
```

A la variable *n*, on affecte l'entier *3*

A la variable *phrase*, on affecte la chaîne de caractères *Bonjour tout le monde.*

A la variable *pi*, on affecte le nombre réel 3,14159

Exemple 4 : Taper la séquence suivante :

```
a=23
a=a+1
print a
↳ .....
a=a-10
print a
↳ .....
```

A quoi correspond l'instruction a=a+1 ?

.....

.....

.....

b=4 b+=1 **print** b ↳	Que constate-t'on ?	c=6 c-=3 **print** c ↳	Que constate-t'on ?
m,n=0,4 **print** m **print** n	Python permet les affectations multiples.	a,b,c=-6,5,3 a=a+b c=b-c b+=a	Prévoir les valeurs de a, b et c et vérifier avec l'instruction **print** a,b,c. a=..... b=..... c=.....

III-Calcul avec Python

Taper les calculs suivants :

```
3+5          5*12         5-12         5-5*3        (5-5)*3      20/7         20.0/7
↳ .....      ↳ .....      ↳ .....      ↳ .....      ↳ .....      ↳ .....      ↳ .....
```

Attention : La division est une division entière. Le calcul 20/7 donne la partie entière du quotient 20 ÷ 7. Pour obtenir une valeur approchée, l'un des deux nombres doit comporter une virgule : 20.0/7 ou 20/7.0 .

Taper les calculs suivants : `2**3` ↳ ..... `4**2` ↳ .....

A quoi correspond l'opérateur ** ?

.....

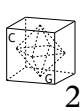
En résumé :

Addition : 4+6	Soustraction : $10-23$	Multiplication :5*44
Division entière : 16/5	Division approchée : 16.0/5	Calcul de puissances : 2**3

Exercice : Effectuer les calculs suivants :

$(4-21)^2-4\times(-4)+(-3)^2-5\times0,25$ (réponse : 312,75)

$34\div(4-12)-(16-(-2)^3)\div(1-(-4))$ (réponse : -9,05)



IV-Boucle for ... in

1. Commande range()

La commande **range**(n) retourne sous la forme d'une liste les n premiers entiers.

Exemple :

```
range(5)
```

↳ .....

2. Instruction for ... in ...

La commande **for ... in ...** est une instruction itérative qui répète les mêmes instructions plusieurs fois.

Exemples :

```
for x in [1,2,3]:
    print x
```

↳ .....

```
for n in range(5):
    print n,
```

↳ .....

```
for k in range(5):
print k
```

↳ .....

Remarque 1 : Noter ci-dessus la différence d'affichage entre les deux premiers exemples, différence provoquée par la virgule après n.

Remarque 2 : Le dernier exemple correspond à une erreur d'indentation : la commande **print** k, doit être décalée afin d'être considérée comme faisant partie du bloc **for ... in**. Pour cela on utilise la touche de tabulation.

Exercices : Taper les programmes dans un éditeur comme SciTE (les enregistrer avec l'extension **.py**).

1. Écrire un programme qui demande le nombre de notes, puis calcule la moyenne.
2. Écrire un programme qui demande un nombre n, puis qui affiche tous les nombres pairs inférieurs ou égaux à n.

V-Test if

1. Instruction if(...):

La commande **if (...):** permet de tester le contenu d'une variable et exécute une série d'instructions si les conditions sont remplies.

Exemples :

```
if (3>0):
    print("3 est supérieur à 0")
```

↳ .....

décalage obligatoire

```
n=raw_input("Choisissez un nombre :")
n=float(n)
if (n>0):
    print("Le nombre choisi est positif")
print("Fin du programme")
```

Remarque : Dans le deuxième exemple, **raw_input** renvoyant une chaîne de caractères, l'instruction **float()** convertit cette chaîne en un nombre réel. On aurait pu écrire encore plus rapidement : n=**float**(**raw_input**("Choisissez un nombre :")). On peut utiliser l'instruction **int()** pour les entiers. Faire un essai avec un nombre positif, puis avec un nombre négatif et observer ce qui se passe.

2. Critère de divisibilité

Le résultat de l'opération a**%**b est le reste de la division euclidienne de a par b.

Exemples :

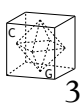
	`5%2`	`24%4`	`321%13`
	↳	↳	↳ .....
car	$5 = 2 \times 2 + \mathbf{1}$	$24 = 4 \times 6 + \mathbf{0}$	$321 = 13 \times 24 + \mathbf{9}$

Tester le programme suivant :

```
if(234%6==0):
    print("234 est un multiple de 6")
```

↳ .....

Que fait ce programme ?
.....
.....
.....



3. Tester la valeur d'une variable contenant un nombre

Si n désigne une variable contenant un nombre alors :

Test en français	*Écrit en langage Python*
Si n est égal à zéro	**if** (n==0):
Si n est positif	**if** (n>0):
Si n est différent de 34	**if** (n!=34):
Si n est compris strictement entre 0 et 10	**if** (n>0) **and** (n<10):
Si n est divisible par 5	**if** (n**%**5==0):

Exemple :

```
for n in range(1001):
    if(n%5==0):
        print n,
```

Que fait ce programme ? .....

4. Tester plusieurs valeurs d'une variable

Il est parfois utile de tester plusieurs valeurs d'une même variable pour poursuivre l'exécution d'un programme.

Exemple : Le programme suivant demande à un utilisateur de choisir un nombre. En fonction du nombre choisi, le programme affiche différents messages.

```
n=input("Entrer un nombre :")
if (n<0):
    print("Le nombre est négatif")
elif (n==0):
    print("Le nombre est égal à zéro")
else:
    print("Le nombre est positif")
```

Si (**if**) n est négatif, alors on l'affiche

Sinon si (**elif**) n est égal à zéro, alors ...

Sinon (**else**) forcément n est positif.

Exercices :

1. Écrire un programme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif. Attention, on ne doit pas calculer ce produit.

2. Écrire un programme qui demande à un client le montant de ses derniers achats dans un supermarché. En fonction de la somme dépensée, il l'informe de sa remise et la calcule :
   Si la somme dépensée est inférieure à 50 €, il obtient 3 % de remise.
   Si la somme dépensée est comprise entre 50 et 100 €, il obtient 5 % de remise.
   Si la somme dépasse 100 €, il obtient 7% de remise.

VI-Boucle While

Voici un exemple de programme utilisant la boucle **while** :

```
a=3
while (a<11):
    print a,
    a=a+1
```

Ce programme affiche tous les nombres entiers compris entre 3 et 10.
L'instruction a=a+1 remplace la valeur entière de a par le nombre entier suivant. On dit que la variable a est incrémentée d'une unitée.
Le mot **while** signifie "tant que" en anglais. Cette instruction utilisée à la deuxième ligne signifie que Python doit répéter le bloc d'instruction tant que a est plus petit que 11.

Exercices :

1. Écrire le programme suivant en utilisant une boucle **while** au lieu de la boucle **for** ... **in** :

```
for n in range(1001):
        p=1000-n
        if (p%5==0):
                print n,
```

2. Écrire et exécuter le programme suivant en entrant des nombres positifs et négatifs :

```
n=-1
while (n<0):
        n=input("Entrer un nombre positif : ")
        if (n<0):
                print("Le nombre que vous avez saisi n'est pas positif !")
print ("Ok, le nombre est positif !")
```

A quoi sert ce programme ?

.....

.....

3. Écrire un programme qui demande un nombre compris entre 123 et 773, puis détermine si ce nombre est un multiple de 9. Ce programme doit prévoir le cas où l'utilisateur ne respecte pas ce qu'on lui demande.

4. Écrire un programme calculant le volume d'un cône de révolution à partir du rayon de la base et de la hauteur. Ce programme doit prévoir le cas où l'utilisateur saisie des données absurdes (rayon ou hauteur négative).

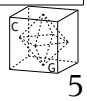
5. Le programme suivant permet d'afficher les dix premiers termes de la suite de Fibonacci. Il s'agit d'une suite de nombre, dont chaque terme est égal à la somme des deux termes qui le précèdent. Analyser ce programme et compléter le tableau d'états qui suit.

```
a,b,c=1,1,1
while c<11:
        print b,
        a,b,c=b,a+b,c+1
```

Tableau d'états

Variables	a	b	c
Valeurs initiales	1	1	1
Valeurs prises successivement, au cours des itérations	1	2	2
	2	...	...
	...	...	...
	...	...	...
	...	...	...
	...	...	...
	...	...	...
	...	...	...
	...	...	...
Expression de remplacement	b	a+b	c+1

Taper le programme et l'exécuter, puis vérifier avec le tableau.



VII-Listes en Python

Une liste est une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets.

1. Un exemple

Dans IDLE, taper : `jour=['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']`

```
print jour[0]
```
↪ .....

```
print jour[2]
```
↪ .....

```
print jour[4]
```
↪ .....

Les éléments de la liste sont ordonnés. Leur numéro est appelé *indice*. Attention, la numérotation commence à 0 !

On peut remplacer, ou modifier certains éléments d'une liste :

```
jour[3] = 'juillet'
print jour
```
↪ .....

2. Fonctions intégrées

La fonction intégrée **len()** renvoie le nombre d'éléments d'une liste :

```
len(jour)
```
↪ .....

La fonction **del()** permet de supprimer un élément quelconque à partir de son indice :

```
del(jour[4])
print jour
```
↪ .....

3. Méthodes

On peut utiliser des *méthodes* de l'*objet* liste. Par exemple le méthode *append()*. Append signifie "ajouter" en anglais :

```
jour.append('samedi')
print jour
```
↪ .....

Une méthode est "appliquée" par un point. Il existe d'autres méthodes pour les listes. Entre autres : *sort()* qui trie les éléments dans l'ordre croissant, *reverse()* qui inverse l'ordre des éléments de la liste, *index()* qui retrouve l'indice d'un élément, *remove()* qui enlève un élément, etc.

```
jour.index('mardi')
```
↪ .....

```
jour.remove('mercredi')
```
↪ .....

4. Exercices

a. Soient les listes : t1=[31,28,31,30,31,30,31,31,30,31,30,31] et t2=['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'].
Écrire un programme qui crée une nouvelle liste t3. Celle-ci devra contenir tous les éléments des deux listes en les alternant, de telle manière que chaque nom de mois soit suivi du nombre de jours correspondant : ['Janvier', 31, 'Février', 28, ...].

b. Écrire un programme qui affiche "proprement" tous les éléments d'une liste. Par exemple, pour la liste t2, on devrait obtenir : Janvier Février Mars Avril ...