TP Python en classe de 2nde

Table des matières

1 Console et éditeur

1. Dans la fenêtre du bas (console), tester les instructions suivantes :

```
>>> 2 + 5
>>> 8 - 2
>>> 6*3
>>> 5**2
>>> 9 / 2
>>> 16 // 3
>>> 16 % 3
>>> a=3
>>> a
>>> a==3
>>> a==5
>>> a=a+2
>>> a
>>> a!=3
```

Signification des instructions - à retenir :

`+` .....

`-` .....

`*` .....

`**` .....

`/` .....

`//` .....

`%` .....

`=` .....

`==` .....

`!=` .....

2. Même consigne avec les commandes suivantes :

```
>>> type(4)
>>> type(1.2)
>>> type(''hello'')
>>> type(True)
```

Les 4 types (ou classes) de variables - à retenir :

.....

.....

.....

.....

3. Dans chacun des cas, deviner le résultat de la dernière commande, puis vérifier en tapant les instructions :

```
>>> longueur = 10
>>> largeur = 5
>>> longueur*largeur
```

```
>>> masse = 50
>>> vitesse = 12
>>> energie = 0.5*masse*vitesse**2
>>> energie
```

```
>>> x = 4
>>> x = x+6
>>> x = x**2
>>> x
```

```
>>> b = 3
>>> c = 2*b
>>> d = c-1
>>> d == 5
```

4. a) Dans la fenêtre d'édition, écrire votre premier programme sous Python. **Attention à bien respecter la syntaxe et l'indentation !**

```
def hello():
    for i in range(4):
        print(”Bonjour!”)
```

b) Après avoir « compilé » (en cliquant sur la flèche verte), exécuter le programme dans la console : `>>> hello()` puis Entrer

c) Modifier le programme précédent pour qu'il écrive « Bonjour ! » 10 fois.

d) Afin que l'utilisateur puisse choisir le nombre de « Bonjour ! » écrits sans avoir à modifier le programme à chaque fois, on modifie le programme une fois pour toute. Compléter les parenthèses :

```
def hello(n):
   for i in range(  ):
       print(”Bonjour!”)
```

Quelle instruction faut-il entrer dans la console pour afficher « Bonjour ! » 12 fois ?

2 Les fonctions

1. On écrit et on compile dans l'éditeur la fonction Python suivante :

```
def f(x):
    y = 2*x+3
    return y
```

a) Êtes-vous capables de deviner les résultats quand on exécute ces commandes dans la console ?

```
>>> f(1)
>>> f(5)
>>> f(-1)>0
>>> f(2) + f(3)
```

La syntaxe des fonctions - à retenir :

def : .....

.....

entre parenthèses : .....

.....

return : .....

.....

b. Comment note-t-on cette fonction en mathématique ? Quelle est sa nature ?

c. Compléter pour obtenir une version plus courte de cette fonction, sans la variable y :

```
def f(x):
    return ........
```

d. Quelle différence entre print et return ? Testons :

```
def f(x):
    print(2*x+3)
```

```
>>> f(1)
>>> f(-1)>0
```

print : .....

.....

2. On considère les deux fonctions Python suivantes :

```
def aire_rectangle(largeur,longueur):
    return largeur*longueur
```

```
def est_pair(n):
    return n%2 == 0
```

a) Devinez le résultat affiché lorsqu'on compile puis qu'on exécute les commandes suivantes :

```
>>> aire_rectangle(2 , 3)
>>> aire_rectangle(1.2 , 0.5)
```

```
>>> est_pair(24)
>>> est_pair(5)
```

b) Pour chacune des deux fonctions, précisez :

Nom de la fonction		
Nom des variables en entrée et leur type		
Sortie (que renvoie la fonction ?) en précisant le type		
Utilité de la fonction (à quel problème répond à la fonction ?)		

3. *Appeler votre professeur pour valider votre travail sur l'ordinateur. Écrire vos fonctions dans le cahier d'exercices.*

a) Programmer une fonction Python appelée `volume_pavé` qui prend en entrée la longueur, la largeur et la hauteur d'un pavé droit, et qui renvoie son volume.

b) Programmer une fonction Python appelée `est_impair` qui prend en entrée un entier, et qui renvoie vrai si l'entier est impair, faux sinon.

c) Programmer une fonction Python appelée `vitesse` qui prend en entrée la distance et le temps de parcours d'un mobile, et qui renvoie la vitesse de ce mobile.

4. *Nombre pi et arrondi*

Programmer la fonction suivante :

```
from math import pi
def longueur_cercle(R):
    return 2*pi*R
```

a) À quoi sert l'instruction de la première ligne ?

b) Que renvoie la fonction quand on exécute : `>>> longueur_cercle(5)` ?

c) On modifie la dernière ligne :

```
from math import pi
def longueur_cercle(R):
    return round(2*pi*R,3)
```

Devinez alors : `>>> longueur_cercle(5).`

d) Programmer une fonction `Python` appelée `aire_disque` qui prend en entrée le rayon d'un disque et qui renvoie l'aire de ce disque arrondie à 10^{-2}.

5. On programme les deux fonctions suivantes :

`def carré(x):` `  return x**2`	`def double(x):` `  return 2*x`

a) Prévoir les réponses renvoyées quand on exécute les commandes :

`>>> carré(3)`	`>>> double(-6)`
`>>> carré(-6)`	`>>> carré(double(3))`
`>>> double(3)`	`>>> double((carré(3))`

b) Sauriez-vous expliquer à l'aide du calcul littéral pourquoi les deux derniers résultats ne sont pas identiques ?

3 Boucles Pour et Turtle

1. a) Dans l'éditeur, écrire puis compiler la fonction suivante :

```
from turtle import*
def essais():
    forward(60)
    left(45)
    forward(28)
    right(135)
    forward(45)
    mainloop()
```

b) Tester ensuite dans la console : `>>> essais()`

Remarques :

- La première ligne de commande `from turtle import*` permet d'utiliser toutes les commandes du module turtle de Python (une petite tortue munie d'un stylo).
- La commande `mainloop()` permet de garder la fenêtre graphique ouverte : *il faudra garder cette commande à la fin de vos fonctions et fermer la fenêtre graphique avant de modifier vos programmes !*

Les principales commandes de la tortue :

Actions		Commandes Python
Faire avancer la tortue de n pixels (en ligne droite).		`forward(n)`
Faire reculer la tortue de n pixels (en ligne droite).		`backward(n)`
Faire tourner la tortue sur elle-même vers la gauche d'un angle de a degrés.		`left(a)`
Faire tourner la tortue sur elle-même vers la droite d'un angle de a degrés.		`right(a)`
Déplacer la tortue vers le point de coordonnées.		`goto(x,y)`
Lever le crayon pour que la tortue ne laisse pas de trace.		`up()`
Baisser le crayon pour que la tortue laisse une trace à nouveau.		`down()`
Déterminer la couleur du tracé.		`color(`red')`
Choisir l'épaisseur n du tracé (n est compté en nombre de pixels).		`width(n)`
Remplir un contour fermé à l'aide d'une couleur.	Choisir une couleur	`color(`blue')`
	Encadrer le script par	`begin_fill()` … `end_fill()`

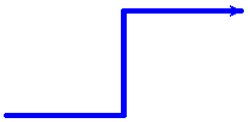
2. Modifier la fonction `essais` :

a) Pour déplacer la tortue de 100 pixels vers la droite et laisse la trace ci-contre.

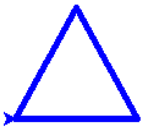
b) Pour déplacer la tortue de 100 pixels vers la droite et laisser une trace bleue de 5 pixels d'épaisseur.

c) Pour obtenir le tracé ci-contre (les 3 segments ont une longueur de 100 pixels).

Appeler votre professeur pour valider ce programme.



d) Pour obtenir le tracé d'un triangle équilatéral de 100 pixels de côté.



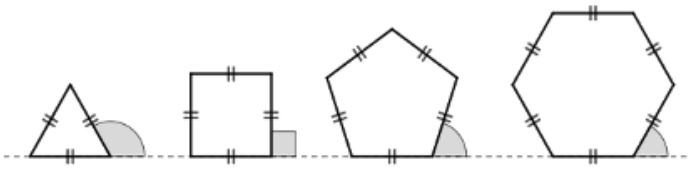
3. a) Parmi les deux fonctions suivantes, laquelle permet d'obtenir le tracé d'un carré ? Tester et expliquer brièvement.

```
from turtle import*
def poly1():
    for i in range(4):
        forward(100)
    left(90)
    mainloop()
```

```
from turtle import*
def poly2():
    for i in range(4):
        forward(100)
        left(90)
    mainloop()
```

b) Modifier la fonction choisie pour obtenir un triangle équilatéral de 100 pixels de côté.

c) Donner le nom de ces polygones réguliers convexes, ainsi que la mesure des angles :



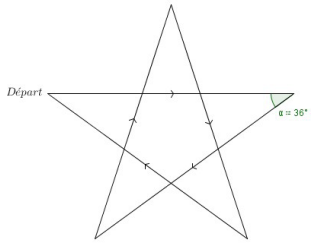
d) On note n le nombre de côtés d'un polygone régulier. Trouver la formule qui permette de calculer la mesure de l'angle précédent.

e) Ecrire un fonction polygone_regulier qui prend en entrée le nombre de côtés n (choisi par l'utilisateur) et qui de trace le polygone régulier à n côtés de longueur 30. **Appeler le professeur pour valider votre programme.**

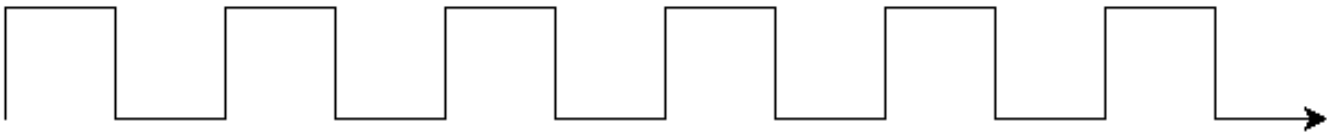
f) Quelle remarque peut-on formuler quand on trace un polygone régulier avec « un grand nombre » de côtés ? (on pourra réduire la taille des côtés pour tester)

4. Pour aller plus loin :

a) Le pentagramme est une étoile à 5 branches inscrite dans un pentagone régulier. Écrire une fonction qui affiche un pentagramme constitué de segments de 200 pixels.



b) Écrire une fonction, la plus courte possible, qui affiche ces créneaux faits de segments de 50 pixels :



c) Écrire une fonction qui affiche ces 5 hexagones réguliers verts de côté 30 pixels :

4 Boucles Pour - longueurs variables

1. On rappelle la fonction `carré` et on donne la fonction `frise`.

```
from turtle import*
def carré():
    for i in range(4):
        forward(30)
        left(90)

def frise():
    carré()
    up()
    forward(60)
    down()
    carré()
    mainloop()
```

a) Ecrire les deux fonctions dans un même fichier, puis exécuter la fonction frise.

b) Modifier la fonction frise afin d'obtenir le dessin suivant :

c) Avez vous donné la version la plus courte de cette fonction ?

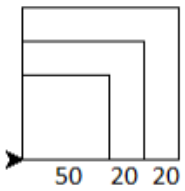
d) Modifier encore la fonction frise pour que l'utilisateur puisse choisir en entrée le nombre de carrés tracés à la suite.

2. a) Modifier à présent la fonction carré pour que l'utilisateur puisse choisir en entrée la longueur du côté.

b) À la suite, écrire puis exécuter la fonction `agrandissement` :

```
def agrandissement():
    carré(50)
    carré(70)
    mainloop()
```

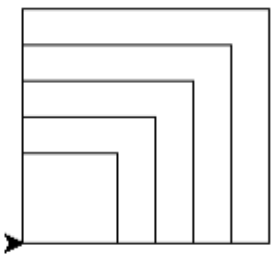
c) Modifier la fonction agrandissement pour obtenir la figure suivante :



d) On donne la version suivante de la fonction `agrandissement` :

```
def agrandissement():
    x = ...
    for i in range(...):
        carré(...)
        x = x + ...
    mainloop()
```

Compléter la afin d'obtenir la figure ci-dessous :



e) Modifier enfin la fonction `agrandissement` afin que l'utilisateur puis choisir le nombre de carrés.

5 Tests et instructions conditionnelles

1. Exécuter les commandes suivantes dans la console, et traduire ces commandes et la réponse en Français :

Instructions	En Français
`>>> 6>5`	
`>>> 5>5`	
`>>> 6>=5`	
`>>> 5>=5`	

2. Tester les commandes suivantes puis compléter les cases des tableaux par VRAI ou FAUX :

`>>> 6 > 5 and 7 > 5`
`>>> 6 > 5 and 5 > 5`
`>>> -5 < -10 and 2**3 == 8`
`>>> -1 < -3 and 2*3 == 8`

ET	VRAI	FAUX
VRAI		
FAUX		

`>>> 6 > 5 or 7 > 5`
`>>> 6 > 5 or 5 > 5`
`>>> -5 < -10 or 2**3 == 8`
`>>> -1 < -3 or 2*3 == 8`

OU	VRAI	FAUX
VRAI		
FAUX		

3. On compile et on exécute la fonction suivante :

```
def f(x):
    if x > 0: return x**2
    else: return 2*x
```

a) Quelles sont les valeurs renvoyées quand on saisit les commandes suivantes ?

`>>> f(5)`	`>>> f(-6)`	`>>> f(0)`	`>>> f(10)`	`>>> f(-10)`	`>>> f(0.1)`

b) Traduire la fonction en Français.

4. Un cinéma propose les tarifs suivants :

   – moins de 18 ans et plus de 60 ans : 5€

   – entre 18 et 60 ans : 8€

   Programmer une fonction `prix` qui prend en entrée l'âge du spectateur et qui renvoie le prix à payer.

5. Un automobiliste roule à 90 km/h pendant deux heures puis à 120 km/h pendant une heure.

   Programmer une fonction `distance` qui prend en entrée le temps de parcours entre 0 et 3 en heures et qui renvoie la distance parcourue par l'automobiliste.

6 Affichage de courbes point par point - tableau de valeurs

1. a) Ecrire puis compiler les fonctions suivantes dans l'éditeur :

```
import matplotlib.pyplot as plt
def f(x):
    return 2*x + 1

def tableau(début):
    x = début
    for i in range(11):
        y = f(x)
        print(''f('',x,'')='', y)
        x = x + 1

def points(début):
    x = début
    for i in range(11):
        y = f(x)
        plt.plot(x, y,'b+')
        x = x + 1
    plt.show()
```

b) Tester :

`>>> tableau(0)`	`>>> tableau(-5)`
`>>> points(0)`	`>>> points(-5)`

c) Modifier la fonction `points` afin d'afficher 21 croix × rouge avec un pas de $0,5$.

d) Modifier la fonction `points` afin que l'utilisateur puisse aussi choisir en entrée le pas et le nombre de points à afficher.

2. Modifier la fonction `f` afin qu'elle corresponde à la fonction distance de l'exercice :

« Un automobiliste roule à 90 km/h pendant 2 heures puis à 120 km/h pendant une heure. »

Afficher alors la courbe entre 0h et 3h avec un pas de $0,1$.

3. a) Modifier la fonction `f` avec la fonction ci-dessous puis afficher la courbe entre 0 et 7 avec un pas de $0,5$.

```
def f(x):
    if 0<=x<2: return 2*x
    elif 2<=x<5: return 4
    else: return -x+9
```

b) Traduire la fonction en français.

c) Modifier la fonction pour programmer la fonction f définie par : $f(x)=\begin{cases} -x-2 & \text{si} \quad x\leqslant -2 \\ x+2 & \text{si} \quad -2<x\leqslant 0 \\ -x+2 & \text{si} \quad 0<x\leqslant 2 \\ x-2 & \text{si} \quad x>2 \end{cases}$.

Afficher la courbe sur l'intervalle $[-5;5]$ avec un pas de $0,5$.

7 Listes - Moyenne - Médiane

En Python, il existe principalement cinq types de variables : les entiers (int), les décimaux ou flottants (float), les chaînes de caractères (string), les booléens (True et False) et les listes (list).

1. a) Pour définir une liste, on met ses éléments entre crochets séparés par des virgules. Dans la console, tester :

```
>>> L = [12, 11, 18, 7, 15, 3]
```

b) Tester alors dans la console les principales commandes associées aux listes :

`>>> len(L)`	Pour connaître la longueur de la liste.
`>>> L[1]`	Renvoie le ..... terme de la liste.
`>>> L[0]`	Renvoie le ..... terme de la liste.
`>>> L[-1]`	Renvoie le ..... terme de la liste.
`>>> L.reverse()` `>>> L`	Pour inverser l'ordre des élèments d'une liste.
`>>> L.sort()` `>>> L`	Pour ranger une liste dans l'ordre croissant.
`>>> L.append(20)` `>>> L`	Ajoute l'élément 20 en fin de liste.
`>>> L.remove(20)`	Supprime le premier élément 20 de la liste.

`>>> M = [23, 25]`
`>>> N = L + M`
`>>> N`
	Pour « rassembler » deux listes entre elles (on dit concaténer).

`>>> for i in range(len(L)):` `...  print(L([i]))`	Pour parcourir les élèments d'une liste.
`>>> for x in L:` `...  print(x)`	Idem mais plus rapide !

2. a) On peut également créer des listes de la façon suivante (essayer de prévoir les listes créées puis tester) :

`>>> list(range(10))`	`>>> [2*x for x in range(10)]`
`>>> list(range(1, 11)`	`>>> [x**2 for x in range(10)]`

b) Quelle instruction permet d'obtenir la liste des 10 premiers multiples de 5 ?

c) Quelle instruction permet d'obtenir la liste des entiers impairs de 1 à 29 ?

3. Deviner les résultats renvoyés puis vérifier en exécutant les instructions suivantes dans la console :

`>>> P = [16, 7 , 8, 14, 11]`	`>>> P.append(18)`	`>>> R = [15, 5]`	`>>> [2*x for x in R]`
`>>> len(P)`	`>>> P`	`>>> Q = P + R`	`>>> 2*R`
`>>> P[1]`	`>>> len(P)`	`>>> Q`	
`>>> P[len(m)-1]`	`>>> P.sort()`	`>>> Q.sort()`	
`>>> P[0]`	`>>> P`	`>>> Q.reverse()`	
`>>> P[-1]`	`>>> P[len(P)-1]`	`>>> Q`	

4. Ecrire et compiler la fonction `mystère` suivante dans l'éditeur :

```
def mystère(liste):
    S = 0
    for x in liste:
        S = S + x
    return(S/len(liste))
```

a) On exécute alors les commandes suivantes :

```
>>> L = [16, 7 , 8, 14, 11]
>>> mystère(L)
```

Compléter le tableau afin d'expliquer le résultat renvoyé :

Variables	Initialisation					
x	#########					
S						

b) Faire de même avec les commandes suivantes :

```
>>> N = [10, 6 , 11, 16, 20, 9]
>>> mystere(N)
```

Variables	Initialisation					
x	#########					
S						

c) Comment devrait s'appeler la fonction `mystère` ?

5. Ecrire, à l'aide de la fonction mystère, une fonction permettant de calculer l'écart-type d'une liste de valeurs.

6. Ecrire une fonction médiane qui prend en entrée une liste de nombres et qui renvoie la valeur médiane de cette liste.

8 Boucles Tant que

1. On considère l'algorithme suivant :

En français	**En Python**
Initialisation	`def algo(A, B):`
Choisir A un nombre entier	`Q = 0`
Choisir B un nombre entier	`while (Q+1)*B<=A :`
Q Prend la valeur 0	`Q = Q+1`
Traitement	`R = A - B*Q`
Tant que $(Q+1)\times B \leqslant A$ Faire :	`return(Q,R)`
Q Prend la valeur $Q+1$	
Fin Tant que	
R Prend la valeur $A-B\times Q$	
Renvoyer $(Q;R)$	
Fin	

a) Tester la fonction puis compléter les tableaux d'état des variables pour expliquer le résultat en choisissant $A=23$ et $B=4$.

Variables	Initialisation						
A							
B							
Test (Q+1)*B<=A	#######						
Q							
R							

b) Même question en choisissant $A=30$ et $B=5$.

Variables	Initialisation							
A								
B								
Test (Q+1)*B<=A	#######							
Q								
R								

c) À quoi sert cet algorithme ? Connaissez-vous des commandes `Python` qui permettent d'obtenir ces résultats directement ?

2. On place 100€ sur un compte en banque au taux d'intérêt simple de 2% par mois, c'est-à-dire que chaque mois les intérêts sont les mêmes, calculés à partir du capital de départ.

   a) À combien s'élèvent les intérêts chaque année ?

   b) Ecrire une fonction qui permet de connaître le nombre de mois nécessaires pour doubler le capital de départ.

3. Mêmes questions avec un taux d'intérêt composé de 2% par mois (chaque mois, les intérêts sont recalculés avec le nouveau capital obtenu)

4. On considère le processus de calcul suivant :

   Choisir un nombre entier. Si l'entier est pair, le diviser par 2, sinon le multiplier par 3 et ajouter 1.

   Recommencer avec le nombre obtenu.

   a) Tester ce processus en choisissant au départ 6, puis en choisissant 7, enfin 22. Quelle conjecture peut-on faire ?

   b) Programmer une fonction `Python` qui demande le nombre départ et calcule :

   - la liste des nombres obtenus jusqu'au premier 1. Tester alors votre conjecture pour d'autres valeurs choisies au départ.
   - le «temps de vol» : c'est à dire le nombre d'étapes nécessaires jusqu'au premier 1.
   - un graphique avec la liste des nombres obtenus en ordonnées et le numéro du nombre en abscisse.
   - l'altitude maximale : c'est-à-dire le plus grand nombre obtenu.

9 Simuler le hasard

1. Tester les instructions dans la console :

`>>> from random import*`	Le module random permet de générer des nombres pseudo-aléatoires.
`>>> randint(0, 10)`	Renvoie un nombre ..... aléatoirement entre ... et ... .
`>>> random()`	Renvoie un nombre ..... aléatoirement entre ... et ... .
`>>> uniform(0, 10)`	Renvoie un nombre ..... aléatoirement entre ... et ... .

2. a) Pile ou Face : on lance une pièce bien équilibrée 10 fois. Compléter la fonction suivante afin de simuler cette expérience aléatoire.

```
from random import*

def pileouface():
    for i in range(...):
        a = randint(0,1)
        if a == ... : print("Pile")
        else: print(........)
```

   b) Modifier afin que l'utilisateur puisse choisir le nombre de lancers.

3. On tire une boule dans une urne contenant 3 boules vertes et 7 boules rouge.

   Ecrire une fonction `urne` permettant de simuler cette expérience aléatoire.

4. a) Soit n un nombre entier choisi par l'utilisateur. On lance un dé n fois et on s'intéresse aux apparitions du 6.

   Compléter l'algorithme suivant afin de simuler cette expérience et de compter le nombre de fois où on obtient 6 :

```
from random import*

def fréquence(n):
    X = 0      #X est le nombre de 6 obtenus
    for i in range(1,n+1):
        a = randint(...,...)
        if a ==6 : X = ...
    return(...)
```

   b) Modifier l'algorithme afin qu'il calcule et affiche dans un graphique la fréquence observée d'apparition du 6 après chaque nouveau lancer. Tester pour de grandes valeurs de n.

   c) De quelle valeur cette fréquence semble-t-elle se rapprocher quand n devient grand ?