Chapitre 1 : Prise en main du langage Python

I Quelques opérations simples

I.1 Les types numériques int et float

Q1. Dans un code écrit en Python, comment apporte-t-on un commentaire ?

Solution: Tout ce qui est après le caractère # sur la ligne est ignoré par l'interpréteur, on y place ainsi un commentaire.

Q2. À l'aide de Python, faire le travail suivant et commentez :

(a) Calculer la valeur de : $\mathbf{2}$, $\mathbf{0}^{13\times21}$ et de $\mathbf{2}^{13\times21}$.

(b) L'instruction `2.4**5` renvoie-t-elle le même résultat que `2.4*2.4*2.4*2.4*2.4` ?

(c) Déterminer si le nombre 271+515 est divisible par 3 ;

(d) Écrire le nombre 6 210 sous la forme $b.q + r$ où $b = 12$ et $0 \leq r < b$;

(e) Afficher le texte suivant : Bonjour, je m'appelle Jean.

Solution:

```
>>> 2.0**(13*21)
1.517710072051351e+82
>>> 2**(13*21)
151771007205135083665582961470587414581438034300948400097797844510851897281656913 92
>>> 2.4**5 ; 2.4*2.4*2.4*2.4*2.4
79.62623999999998
79.62624
>>> (271+515) % 3
0
>>> divmod(6210,12)
(517, 6)
>>> print("Bonjour, je m ' appelle Jean")
Bonjour, je m ' appelle Jean
```

(a) L'exponentiation est prioritaire devant la multiplication, il faut donc placer des parenthèses autour du produit 13*21 avant le passage à la puissance. On remarque également que les flottants (de type float) sont représentés jusqu'à 15 chiffres après la virgule alors que les entiers (de type int) sont calculés avec une précision qui n'a de limite que celle de l'ordinateur ! Tous les chiffres sont calculés et représentés.

(b) Les résultats sont différents à cause des erreurs d'arrondis dus à la représentation des flottants dans la mémoire de l'ordinateur.

(c) On utilise l'opération %, prononcez « modulo », pour obtenir le reste de la division euclidienne de a par b lorsque l'on exécute le code a%b.

(d) La fonction divmod(a,b) prend deux nombres et donne leur quotient et reste de leur division entière sous forme d'une paire de nombres. Pour deux entiers le résultat est le même que (a // b, a % b). Pour des nombres à virgule flottante le résultat est (a / b, a % b)

(e) Pour inclure du texte dans le code, il faut marquer ses extrémités par des apostrophes simples ou doubles.

Q3. Afficher la période T d'un pendule de longueur l= 50 cm. Puis, arrondir le résultat à 2 chiffres après la virgule (pour cela utiliser la fonction round(x,n) où x est le nombre à arrondir et n le nombre de chiffres retenus après le virgule)).

On rappelle que $T = 2.\pi.\left(\frac{l}{g}\right)^{0.5}$ où g = 9.81 m.s^{-2} l'accélération de la pesanteur.

Solution:

```
>>> T = 2*3.14*(.5/9.81)**0.5
>>> print("la période du pendule est ", round(2*3.14*(.5/9.81)**0.5, 2), " secondes.")
la période du pendule est  1.42  secondes.
```

Q4. Exprimer la durée τ qu'il faut à la lumière émise par le Soleil pour nous parvenir. On rappelle la distance Terre-Soleil : $D = 150 * 10^6$ **km**. On exprimera le résultat sous la forme d'un nombre entier (valeur plancher) et on affichera la phrase suivante : « La lumière émise par le Soleil met environ ... minutes à nous parvenir »

Solution:

```
>>> D=150e6                # la distance
>>> c= 3e5                 # célérité de la lumière dans le vide
>>> tau = D/(c*60)         # on convertit la durée en minutes
>>> print("La lumière émise par le Soleil met environ ", int(tau), "minutes à nous parvenir")
La lumière émise par le Soleil met environ  8 minutes à nous parvenir
```

Q5. Quelles erreurs produisent ces lignes de code. Expliquer.

```
1 13%0
2 float('bonjour')
```

Solution:

```
>>> 13%0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>> float('bonjour')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'bonjour'
```

L'expression 13%0 provoque une erreur car la division par zéro n'est pas possible.
Une chaîne de caractère ne peut pas être convertie en une constante à virgule flottante , d'où l'erreur de valeur sur l'argument de la fonction float().

I.2 L'affectation : identificateur = valeur

Q6. En vous appuyant sur les exemples suivants, et en réalisant les tests nécessaires à l'aide de la fonction type(), commentez la phrase « Le type d'une constante ou d'une variable définit les opérations qu'elle supporte ».

```
1 a = 23.6 + 87
2 b = 'bonjour ' + 'le monde'
3 print('*o' * 10 + ' Bienvenue ' + '*o' * 10)
4  c="bonjour" * 3.0
```

Solution:

```
>>> a = 23.6 + 87
>>> a ; type(a)
110.6
<class 'float'>
>>> b = 'bonjour ' + 'le monde'
>>> b ; type(b)
'bonjour le monde'
<class 'str'>
>>> print('*o' * 10 + ' Bienvenue ' + '*o' * 10)
*o*o*o*o*o*o*o*o*o*o Bienvenue *o*o*o*o*o*o*o*o*o*o
>>> c="bonjour" * 3.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
```

Ces exemples montrent qu'on peut **concaténer** des chaînes de caractères grâce à l'opérateur « + », on peut même les multiplier par un nombre entier, mais en revanche, une multiplication par un nombre à virgule lève une erreur de type (`TypeError`). À l'aide de tests, on vérifie bien que les types des variables a et b sont déterminés après avoir exécuté l'affectation, il n'y a pas d'annonce particulière pour exprimer que a sera un flottant par exemple (contrairement à d'autres langages de programmation). Le typage des variables est dynamique.

Q7. Déterminer les valeurs de x et y au cours des instructions suivantes et repérer les affectations simultanée, multiple et la syntaxe de la permutation (propre au langage Python).

1 *# suite d' instructions 1*	1 *# suite d' instructions 2*	1 *# suite d' instructions 3*
2 x = y=3	2 x=3	2 x=3
3 x = x+2*y	3 x, y = x + 2, x*2	3 y=5
4 y = x-y	4 x *= 2	4 x, y = y, x

Solution:

- La suite d'instructions 1 commence par une affectation **simultanée** : les variables x et y prennent toutes les deux, simultanément la valeur entière 3. On peut modifier la valeur d'une variable suite à l'exécution de la ligne 3, x prend désormais la valeur 3 + 2*3 soit 9. Ainsi, la nouvelle valeur de y est 9-3 soit 6.
- La suite d'instructions 2 met en œuvre par une affectation **multiple** : les variables x et y prennent respectivement la valeur entière 5 et 6. On reconnaît la syntaxe particulière : variable @= expression où @

désigne n'importe quel opérateur arithmétique : +,-,*,/,//,%,** qui produit le même résultat que l'instruction : variable = variable @ expression. Très utile pour l'incrémentation ou pour générer des suites arithmétique ou géométrique.

- Et enfin, la suite d'instructions 3 met en œuvre une **permutation**. Les variables ont permuté leur valeur. C'est désormais x qui vaut 5 et y qui vaut 3

Q8. Quelles erreurs produisent ces lignes de code. Expliquer.

```
1 largeur = 5. ; longueur = 10 ; hauteur = 2
2 aire = largeur * longeur
3 volume = Aire *  hauteur
```

Solution:

```
>>> largeur = 5.                # 5. est équivalent à 5.0
>>> longueur = 10
>>> hauteur = 2
>>> aire = largeur * longeur
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'longeur' is not defined
>>> volume = Aire * hauteur
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Aire' is not defined
```

Il faut faire attention à l'orthographe précise des **noms** de variables ! En effet, il faut bien écrire longueur et non longeur idem il faut écrire aire et non Aire. Python distingue la casse (les lettres minuscules et majuscules sont distinctes).

I.3 Manipuler les fonctions

Pour toute fonction, il faudra préciser **sa signature**. La signature est l'ensemble des paramètres (avec leurs noms, positions, valeurs par défaut et annotations), ainsi que l'annotation de retour d'une fonction. C'est-à-dire toutes les informations décrites à droite du nom de fonction lors d'une définition.

Q9. Écrire une fonction **superficie** qui a comme argument le rayon r d'un disque et retourne l'aire du disque. En employant superficie, écrire une nouvelle fonction **cylindre** qui, à partir du rayon de base r et de la hauteur h retourne l'aire de la base et le volume du cylindre. Commentez *le type retourné* par chaque fonction. On prendra π **= 3,1415**.

Q10. Calculer l'aire d'un disque de rayon 2,22 cm. Calculer l'aire et le volume d'un cylindre r **= 1,5 cm** et h **= 10 cm**. Stockez dans les variables aire et volume les grandeurs calculées et retournées par superficie.

Solution:

```
>>> def superficie(r):
...     '''
...     retourne l'aire du disque de rayon r
...     '''
...     return 3.14*r**2 # de type float
...
>>> def cylindre(r,h):
...     '''
...     retourne l'aire de la base de rayon r et le volume du cylindre de hauteur h
...     '''
...     aire = superficie(r)
...     # une fois qu'une fonction est définie, on peut l'appeler par d'autres fonctions !
...     volume = aire * h
...     return aire, volume
...     # la virgule est un séparateur qui construit une structure de données :
...     # ici il s'agit d'un tuple
...
>>> superficie(2.22e-2)
0.0015475176000000002
>>> cylindre(1.5e-2,10e-2)
(0.0007065, 7.065e-05)
>>> type(cylindre(1.5e-2,10e-2))
<class 'tuple'>
>>> type(cylindre(1.5e-2,10e-2))
<class 'tuple'>
>>> aire, volume = cylindre(1.5e-2,10e-2)
>>> aire; volume
0.0007065
7.065e-05
```

Pour les calculs, on fait appel aux fonctions en passant les valeurs ou expressions en paramètre. Pour extraire les grandeurs calculées dans les variables, on peut **dépaqueter le tuple.**

Q11. Soient les variables x et y. Écrire une fonction **norme** qui retourne le résultat de l'expression $(x^2 + y^2)^{1/k}$ où x, y et k peuvent être un nombre entier ou à virgule. Faites-en sorte que par défaut, k prenne la valeur 2.

```
>>> def norme (x:nbre,y:nbre,k:nbre=2)->float:
...     a = 1/k
...     n = (x**2 + y**2)**a
...     return n
...
>>> norme(1,1) # par défaut k=2
1.4142135623730951
>>> norme(2,3.2,4) # ici, k prend la valeur 4
1.9425736672833394
```

Q12. Trouver l'erreur !

```
1 print ("5 * 3 =", 5*3)
2 #calcul du nombre de secondes dans une journée
3 print ("une journée a une durée égale a", 60 * 60 * 24, "secondes"
4 print("bonjour le monde")
```

Solution:

```
>>> print ("5 * 3 =", 5*3)
5 * 3 = 15
>>> # calcul du nombre de secondes dans une journée
>>> print ("une journée a une durée égale à", 60 * 60 * 24, "secondes"
... print("bonjour le monde")
  File "<stdin>", line 2
    print("bonjour le monde")
    ^
SyntaxError: invalid syntax
```

Il ne faut pas oublier de fermer la parenthèse de chaque parenthèse ouvrante ! voir la fin de ligne 3.

Q13. Trouver l'erreur !

```
1 def carre(x) :
2     '''
3     calcule et affiche le carre de x
4     '''
5     print(x**2)
6 a = carre(2.6+45*3) * 5
```

Solution:

```
>>> def carre(x) :
...     III
...     retourne le carre de x
...     III
...     print(x**2)
...
>>> a = carre(2.6+45*3) * 5
18933.76
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'
>>> type(carre(2.6+45*3))
18933.76
<class 'NoneType'>
```

Ici, la fonction carre ne retourne rien (**None** seule constante de type NoneType) car il n'y a pas le mot clé **return**. Par conséquent, on ne peut pas définir le produit de "rien" par 5. .. On a bien le résultat de carre(2.6+45*3) qui s'affiche car rien n'empêche la fonction print de s'exécuter.

I.4 Manipuler les tests et le type booléen

Q14. Taper les lignes de codes suivantes et à l'aide de tests expliquer ce que fait chaque série d'instructions.

```
# suite 1
aff1 = a = 3
aff2 = a == 3
```

```
# suite 2
a = 1234 ; x=0
x !=0 and a/x>1
```

```
# suite 3
'chou' > 'fleur'
'o' in 'hello'
```

```
# suite 4
a = 2+3.001
a == 5.001
```

```
# suite 5
e = bool(20%4)
True * 3
```

Solution:

```
>>> # suite 1
>>> aff1 = a = 3 ; aff1 ; a
3
3
>>> aff2 = a == 3 ; aff2 ; a
True
3
```

La 1er ligne est une affectation simultanée : les variables a et aff1 prennent toutes les deux la valeur 3.
La 2nde ligne est une affectation : la variable aff2 prend la valeur `True` résultat de l'expression a == 3 (test d'égalité).

```
>>> # suite 2
>>> a = 1234 ; x=0
>>> x !=0 and a/x >1
False
```

Seule la première clause est évaluée et d'après la table de vérité de `and`, son évaluation à `False` suffit pour donner le résultat sans avoir à évaluer a/x. On remarque que si a/x avait été évaluée, cela aurait provoquée une erreur puisque x vaut 0 !

```
>>> # suite 3
>>> a = 'chou' > 'fleur' ; a
False
>>> b = 'o' in 'hello' ; b
True
```

Les chaines de caractères peuvent être ordonnées dans l'ordre alphabétique et comme la lettre 'c' arrive avant la lettre 'f' dans l'alphabet, le test de comparaison retourne `False`.
Le mot clé `in` permet de tester l'appartenance de 'o' (un élément) dans la chaîne de caractère 'hello' (séquence)

```
>>> a = 2+3.001
>>> a == 5.001
False
```

Problèmes d'arrondis avec les nombres flottants ! On retiendra qu'il ne faudra jamais tester l'égalité entre les nombres flottants mais on utilisera plutôt une syntaxe du type :

```
>>> # suite 4 bis
>>> a = 2+3.001
>>> abs(a-5.001) < 10e-10 # précision arbitraire
True
```

```
>>> e = bool(20%4)
>>> True * 3
3
```

On illustre ici la notion de véracité d'une expression. Un entier nul peut être converti en un booléen de valeur `False`. Et par transtypage dynamique, `True` est évaluée à 1 pour l'opération `True` * 3.

Q15. Écrire une fonction **triangle**(a,b,c) où a,b,c sont des longueurs codées sous forme d'entiers. À l'aide de ces trois longueurs, déterminer s'il est possible de construire un triangle. Déterminer ensuite si ce triangle est rectangle, isocèle, équilatéral ou quelconque. Attention : un triangle rectangle peut être isocèle.

Solution:

```
>>> def triangle(a,b,c) :
...     flag=False
...     if a<b+c and b<a+c and c<a+b :
...         print("Ces trois longueurs peuvent construire un triangle.")
...         flag=True
...     else:
...         print("Impossible de construire un  triangle.")
...     if flag :
...         if a==b and b==c:
...             print("Ce triangle est équilatéral")
...         elif a==b or a==c or b==c :
...             if a*a + b*b == c*c or b*b + c*c == a*a or c*c + a*a == b*b :
...                 print("Ce triangle est isocèle rectangle")
...             else:
...                 print("Ce triangle est isocèle")
...         elif a*a + b*b == c*c or b*b + c*c == a*a or c*c + a*a == b*b :
...             print("Ce triangle est rectangle")
...         else:
...             print("Le triangle est quelconque").
```

Q16. Écrire une fonction **appreciation** qui convertit une note scolaire N entrée par l'utilisateur sous forme de points (par exemple 27 sur 85 , N vaut alors 32%) passée en argument, en une note standardisée suivant le code ci-après :

La note	$N \geq 80$	$60 \leq N < 80$	$50 \leq N < 60$	$40 \leq N < 50$	$N < 40$
Appreciation	*A*	*B*	*C*	*D*	*E*

Appreciation(27, 85) renvoie E

Taper les lignes de code suivantes :

```
A = 51
If a> 0 :
        print('positif')
elif a<0 :
        print('négatif')
else :
        print('nul')
```

Solution:

```
>>> def appreciation(note, total):
...     pourcentage = note*100/total
...     if pourcentage < 40 :
...         categorie = "E"
...     elif 50 > pourcentage >= 40 :
...         categorie = "D"
...     elif 60 > pourcentage >= 50 :
...         categorie = "C"
...     elif 80 > pourcentage >= 60 :
...         categorie = "B"
...     elif pourcentage >= 80 :
...         categorie = "A"
...     return categorie
...
```

II Les boucles

Un type très utilisé que nous verrons ultérieurement en détail est le type liste :

Taper les lignes de code suivantes :
L= [‘a’, 1, 5 , ‘bc’] # création d’une liste de 4 termes
print(len(L))
for elt in L :
print(L)
L2 =[] # création d’une liste vide
for k in range(len(L)-1,-1,-1) : # parcourt dans le sens décroissant des indices
print (k, L[k])
L2.append(L[k]) # rajout dans L2 de l’élément L[k]
print(L2)

Q17. Écrivez un programme qui parcourt un par un tous les éléments d’une liste de mots (par exemple : [’chat’, ’chien’, ’pomme’, ’poire’, ’bonjour’, ’bonsoir’]) pour générer deux nouvelles listes. L’une contiendra les mots comportant un maximum de 5 caractères, l’autre les mots comportant au moins 6 caractères.

Solution:

```
>>> L = ['chat', 'chien', 'pomme', 'poire', 'bonjour', 'bonsoir']
>>> L5 = []
>>> L6=[]
>>> for mot in L :
...     if len(mot) <= 5 :
...         L5.append(mot)
...     else :
...         L6.append(mot)
...
>>> print(L5, L6)
['chat', 'chien', 'pomme', 'poire'] ['bonjour', 'bonsoir']
```

Q18. À l’aide de la fonction range :

(a) énumérer les entiers de 2 à 60 par pas de 3 ;

(b) énumérer les entiers de 0 à 5 (exclu) par pas de 1 ;

(c) énumérer les entiers de 10 à 0 (inclus) par pas de -1 ;

(d) énumérer les entiers de 5 à 25 par pas de 5 (les deux bornes incluses) ;

Solution:

```
>>> print(list(range(2,61,3)))
[2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, 53, 56, 59]
>>> print(list(range(0,5,1)))
[0, 1, 2, 3, 4]
>>> print(list(range(10,-1,-1)))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> print(list(range(5,30,5)))
[5, 10, 15, 20, 25]
```

Q19. Construire les listes suivantes par compréhension de liste :

Exemple : taper les lignes

```
L3 = [x**3 for x in range(7) if x%3==0]
print(L3)
L4 = [(a, b) for a in range(3) for b in range(4)]
print(L4)
```

(a) Liste des carrés des entiers pairs compris entre 0 et 9.

(b) Liste des multiples de 12 compris entre 0 et 100.

(c) Liste des diviseurs d'un entier naturel *N (prendre N=20 pour tester)* .

(d) Donner la liste des couples **(***a, b***)** tel que $a + b = 10$.a et b sont chacun compris entre 0 et9.

(e) Résoudre l'équation $x, y, z \in \mathbb{N}, 0 < x, y, z \leq 100, x^2 + y^2 = z^2$

Solution:

```
>>> Lcarre = [x**2 for x in range(10) if x%2==0]
>>> print(Lcarre)
[0, 4, 16, 36, 64]
>>> Lmul = [x for x in range(12,100) if x%12==0]
>>> print(Lmul)
[12, 24, 36, 48, 60, 72, 84, 96]
>>> Ldiv = [x for x in range(1,20) if 20%x==0] # N=20
>>> print(Ldiv)
[1, 2, 4, 5, 10]
>>> Lcouple = [(a,b) for a in range(10) for b in range(10) if a+b==10]
>>> print(Lcouple)
[(1, 9), (2, 8), (3, 7), (4, 6), (5, 5), (6, 4), (7, 3), (8, 2), (9, 1)]
>>> L = [(x,y,z) for x in range(100) for y in range(100) for z in range(100) if x**2 + y**2      == z**
>>> #print(L)
```

Q20. Pour cette question, on considère que la liste étudiée est une liste de nombres (entiers et/ou à virgule).

(a) Écrire une fonction **somme**(liste) qui calcule et retourne la somme de tous les termes contenus dans la liste de nombre passée en paramètre.

(b) Écrire une fonction **moyenne**(liste) qui calcule et retourne la moyenne des nombres contenus dans la liste.

Solution:

```
>>> def somme(L):
...     s = 0
...     for nombre in L :
...         s += nombre
...     return somme
...
>>> def moyenne(L):
...     return somme(L)/len(L)
```

Q21. Soit une liste L, définie par la commande suivante : >>> L = [10.4, 20.21, 30.0, 6, 2, 4, 201.3, 0.98, 0] Testez les différentes opérations suivantes. Regardez à chaque fois le contenu de L.

Solution:

```
>>> L = [10.4, 20.21, 30.0, 6, 2, 4, 201.3, 0.98, 0]
>>> len(L)
9
>>> L.append(30.0) ; print(L) # on ajoute 30.0 à la fin de L
[10.4, 20.21, 30.0, 6, 2, 4, 201.3, 0.98, 0, 30.0]
>>> L[0] # accès à l'élément placé en position 0 dans L
10.4
>>> L[ :5] # vue des 5 derniers éléments dans L
[10.4, 20.21, 30.0, 6, 2]
>>> L[ 5 :] # vue des 5 premiers éléments dans L
[4, 201.3, 0.98, 0, 30.0]
>>> L[ 2 : 7 ] # vue des éléments d'indice 2 (inclus) à 6 (7 exclu)
[30.0, 6, 2, 4, 201.3]
>>> L[: : 2] # vue de tous les éléments de L par pas de 2
[10.4, 30.0, 2, 201.3, 0]
>>> L[3] = 1 # modification de l'élément positionné en troisième place
>>> print(L) # L a été modifiée
[10.4, 20.21, 30.0, 1, 2, 4, 201.3, 0.98, 0, 30.0]
```

Q22. Trouvez l'erreur !

```
1 >>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
2 >>>print(jour[7])
```

Solution:

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> print(jour[7])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

jour est une liste qui ne contient que 7 éléments, ils sont indicés entre 0 et 6. Ainsi jour[7] n'existe pas.