«Programmation et algorithmique avant de rentrer en prépa», un résumé

1 Utilisation du langage Python

1.1 Instruction de base

1. Opérations arithmétiques : l'addition, la soustraction, la multiplication, la division et l'exponentiation (calculer la puissance d'un nombre) sont notées respectivement : $+, -, *, /$ et $**$.

2. Division euclidienne :

   - le reste obtenu avec l'opérateur modulo %
   - le quotient avec `//`

   Astuces :

   (a) un entier a divise b ssi b modulo a vaut 0.

   (b) le chiffre des unités d'un entier a est `a % 10`. Celui des dizaines s'obtient en calculant d'abord le quotient de a par 10, par exemple `b = a // 10` puis en calculant le reste modulo 10 : `b % 10`.

3. Import de modules :

   on tape `import nom_module`.

   (a) Module math : contient la plupart des fonctions usuelles de mathématiques. Citons par exemple : sqrt, floor, ceil. Attention à préfixer votre fonction du nom du module : `math.floor(4.6)` vaut 4.

   (b) Module random : randint(1,6) génère un entier aléatoire entre 1 et 6 (modélise un dé).

1.2 Affectation, différents types de variable

1. Affectation : espaces des noms Vs espace des objets

   L'instruction `a = 5` est une affectation. L'objet 5 de type entier est stocké en mémoire sous le nom `a` à une adresse référencée par `id(a)`. En Python, l'affectation est un **étiquetage**. Par exemple, la deuxième affectation `b = a`, ne crée pas de copie de `a`. On crée seulement un deuxième nom `b` qui pointe aussi vers 5. Autrement dit l'entier 5 possède deux étiquettes, `a` et `b`.

2. Les principaux types de données à connaître :

   entier, flottant (`5.63`), chaîne de caractères (`"coucou"`), booléen (`True` ou `False`), liste (`[4,2.1, 6]`), tuple (`(4,6,'re')`)

   On obtient le type d'un objet `truc` en tapant `type(truc)`.

3. «Peu ou pas utilisé en prépa» : intéraction avec l'utilisateur

   On utilise la fonction `input`. Attention elle renvoie une chaîne de caractères, et il faut ainsi penser à convertir si par exemple, on veut récupérer un entier ou un nombre flottant.

```
a = input("Quel est ton âge?")
a = int(a) # on convertit la chaîne de caractère a en un nombre entier
a = float(a)# on convertit a en nombre flottant
a = str(a) on convertit a en chaîne de caractère
```

1.3 Structures de contrôle

1. Structures conditionnelles

   La syntaxe de base :

```
a = 10
if a > 1 and a < 2:
    print("a est strictement compris entre 1 et 2")
elif a == 1 or a == 2:
    print('a est égal à 1 ou à 2')
else:
    print('a est inférieur à 1 ou supérieur à 2 ')
```

   Notion de booléen et connecteurs logiques `and, or, not`. Tests d'égalité `==, !=, >, >=`

2. Structures itératives

   - Boucles «tant que» (on doit pouvoir rentrer dans une boucle et en sortir). Par exemple, ce script détermine le plus petit entier n tel que $2^n \geqslant 1000$.

```
N = 1000
p = 1
n = 0
while p < N:
    p = 2 * p
    n = n + 1
print(n)
```

   - Boucles «pour»
     Attention à la syntaxe Python : `for i in range(3,23)` correspond à pour i de 3 à 22 (et non pas 23). Par exemple, ce script calcule la somme des entiers de 1 à 100.

```
s = 0
for k in range(1,101):
    s = s + k
print(s)
```

La fonction range est un cas particulier d'itérateur. On peut aussi itérer des chaînes de caractère ou des listes :

```
for lettre in 'bonjour':
    print(lettre)
```

1.4 Fonctions Vs procédures

Elles vont constituer des briques élémentaires des programmes, par exemple :

```
def factorielle(n):
    p = 1
    for k in range(1, n+1):
        p = p*k
    return p
```

```
def facto(n):
    p = 1
    for k in range(1, n+1):
        p = p*k
    print(p)
```

Pour obtenir, 4!, on exécute alors `factorielle(4)`. Cela renvoie 20

`factorielle` et `facto` se ressemblent beaucoup, la première est un «vraie fonction», elle renvoie (à l'aide du mot clé return) la valeur de $n!$, tandis que la deuxième va afficher sa valeur mais ne renvoie rien, on dit que c'est une procédure.

En particulier `factorielle(3) + factorielle(4)` renvoie bien 30 tandis que l'exécution de `facto(3) + facto(4)` provoque une erreur. En effet l'addition est impossible à réaliser puisque facto(3) et facto(4) ne «valent rien», ce sont des objets de type `NoneType`.

Morale : «une fonction renvoie quelque-chose, une procédure fait quelque-chose.»

A noter enfin que par exemple la variable p de la fonction `factorielle` est une variable locale, elle n'existe pas à l'extérieur du code de `factorielle`. C'est la problématique de la portée des variables.

1.5 Notion de tableau (structure de données) et de liste Python

1. Un tableau est une suite finie de valeurs d'un même type (entier, flottant, booléen ...), stockées dans des cases mémoires contigües. Dans le langage Python, les tableaux sont implémentés par des objets que l'on appelle **liste**. On obtient leur longueur avec la fonction `len` (diminutif de length). Par exemple `t = [4,6,5,8,7]` est un tableau d'entiers de longueur 5.

   On peut accéder aux éléments d'un tableau et les modifier (contrairement par exemple à une chaîne de caractère que l'on ne peut pas modifier). Ceux-ci sont repérés par leur indice : attention le premier élément est celui d'indice 0, (penser à un ascenceur). Par exemple, `t[0]` vaut 4, et le dernier élément `t[4]` vaut 7. Si l'on tape `t[2] = 14`, la liste `t` est modifiée et vaut `t = [4,6,14,8,7]`.

2. Comment construire des listes ?

   (a) la méthode `append` : on peut ajouter un élément à la fin d'une liste à l'aide de la **méthode** `append`. L'instruction `t.append(4)` ne renvoie pas une nouvelle liste, elle

modifie la liste existante (une liste Python est un objet dynamique). A privilégier par rapport à la concaténation (opérateur +) (on voit ainsi que les listes Python implémentent aussi la notion de pile (notion que l'on abordera).

(b) les comprehension list : l'instruction ci-dessous crée la liste `t = [1,9,25,49,81]` :

```
t= [k**2 for k in range(10): if k % 2 ==1]
```

(c) Le tableau vide est noté `[]`, et on peut créer un tableau `t` longueur n (ne contenant par exemple que des 0) à l'aide de `t = [0] * n` (concaténation) ou de `t = [0 for i in range(n)]` (comprehension list).

3. Quelques pythonneries :

(a) on peut itérer une liste : `for element in liste:`.

(b) le slicing

2 Quelques exercices modèles

1. calcul de la somme des entiers de 1 à n

```
n = 100
s = 0
for k in range(1, n+1):
    s = s + k
print(s)
```

2. fonction factorielle

```
def factorielle(n):
    p = 1
    for k in range(1, n+1):
        p = p*k
    return p

print(factorielle(4))
```

3. Programmer un temps d'attente : par exemple, calcul du plus petit entier n tel que 2^n dépasse 1000

```
cible = 1000
p = 1
n = 0
while p < cible:
    p = 2 * p
    n = n + 1
print(n)
```

4. Savoir programmer une suite récurrente d'ordre 1. Par exemple, afficher les 11 premières valeurs de la suite u définie par $u_0 = 1$ et $u_{n+1} = u_n(4 - u_n)$.

```
n = 10
u = 1
print(u)
for k in range(n): # n itérations
    u = 2*u +3
    print(u)
```

5. Savoir programmer une suite récurrente d'ordre 2 : exemple modèle la suite de Fibonacci notée (F_n) définie par $F_0 = 0, F_1$ et $F_n = F_{n-1} + F_{n-2}$ pour $n \geqslant 2$.

```
def fibo(n):
    if n == 0 or n == 1:
        return n
    f1 = 0
    f2 = 1
    for i in range(0,n-1): # n-1 itérations
        temp = f1 + f2
        f1 = f2
        f2 = temp
    return(f2)

print(fibo(50))
```

6. Écrire une fonction `maxi` qui renvoie le plus grand élément d'une liste.

```
def maxi(t):
    """Données: t un tableau d'entiers non vide
        Résultat: le maximum des éléments de t"""
    n =len(t) # la longueur du tableau t
    maximum = t[0]
    for k in range(1,n):
        if t[k] >  maximum:
            maximum = t[k]
    return maximum


t= [24, 12, 45, 13,5]
print(maxi(t))
```