Formulaire Python 3

I. Aide

Obtenir de l'aide sur une fonction ou un module: *help(nom_fonction)* (dans la console) ou *print(help(nom_fonction))* (dans un script)

II. Types et conversions

int (entier sur 32 bits), *long* (entier non limité), *float* (réel), *bool* (True ou False), *str* (chaînes de caractères), *list* (listes), *tuple* (tuple), *dict* (dictionnaire)

Obtenir le type d'une variable : *type(variable)*

Tester si une variable est d'un type donné : *isinstance(variable, nom_du_type)*

Convertir une variable en entier : *int(variable)*

Convertir une variable en float : *float(variable)*

Convertir une variable en chaîne de caractères : *str(variable)*

III. Syntaxe générale

1. Affectation

variable = valeur

Affectation multiple : *variable1, variable2, variable3 = valeur1, valeur2, valeur3*

Echange de deux valeurs: *variable1, variable2 = variable2, variable1*

2. Commentaires

Commentaires sur une seule ligne : # au début

Commentaires sur plusieurs lignes : """ *texte* """

3. Structures de contrôle

Les blocs d'instructions sont délimités en les indentant de 4 espaces.

Conditions	Boucles while	Boucles for
	Si on ne connaît pas le nombre d'itérations	Si on connaît le nombre d'itérations
if condition1 : * instructions1* *elif condition2 :* * instructions2* *else :* * instructions finales*	*while condition :* * instructions*	*for variable in range(a,b) :* * instructions* ou *for variable in range(b) :* * instructions* a et b sont des entiers (a inclus, b exclu)

4. Lecture/écriture dans la console Python 2

Pour entrer une chaîne de caractères : *chaine_saisie = input("message")*

Pour entrer un nombre entier: *nombre_saisi = int(input("message"))*

Pour entrer un nombre réel: *nombre_saisi = float(input("message"))*

Pour afficher dans la console, on utilise la fonction *print* :

print(chaine_de_caracteres) ou *print(nombre)*

print(chaine_de_caracteres, chaine_de_caracteres2, nombre) (l'un au bout de l'autre avec un espace)

print("le résultat est égal à {}".format(nombre)) (remplace les accolades par la valeur de nombre)

print("a est égal à {} et b est égal à {}".format(nombre1, nombre2)) (remplace la 1ère accolade par la valeur de nombre1 et la deuxième par la valeur de nombre2)

5. Importation de modules

   Un module est un fichier qui contient des fonctions. Pour utiliser ces fonctions, il faut importer le module au début du script.

   import nom_module (pour utiliser une fonction avec cette syntaxe, il faut taper *nom_module.nom_fonction()*). On peut changer l'alias du module pour simplifier. Par exemple : *import nom_module as nm* et on tape *nm.nom_fonction().*

   *from nom_module import ** (pour utiliser une fonction avec cette syntaxe, taper *nom_fonction()* suffit. Attention si on a importé deux modules avec des noms de fonctions identiques)

   from nom_module import nom_fonction (on importe uniquement la fonction utile)

IV. Chaînes de caractères

Les caractères sont des chaînes de longueur 1. Pour obtenir la liste des méthodes de la classe *str*, taper *dir(str)*.

Les chaînes de caractères sont non modifiables.

Définir une chaîne vide : *chaine = ""*

Obtenir le caractère dont on connaît le code ASCII : *chr(code_ascii)*

Obtenir le code ASCII d'un caractère : *ord(caractere)*

Concaténer des chaînes : *chaine1 + chaine2*

Concaténer une chaîne et un nombre : *chaine+str(nombre)*

Accéder au caractère d'indice i : *chaine[i]* (les indices commencent à 0)

Accéder au dernier caractère : *chaine[-1]*

Accéder du caractère d'indice i (inclus) au caractère d'indice j (exclu) : *chaine[i :j]*

Accéder du caractère d'indice i jusqu'à la fin : *chaine[i :]*

Longueur d'une chaîne *: len(chaine)*

Mettre une chaîne en minuscules : *chaine.lower()*

Mettre une chaîne en majuscules : *chaine.upper()*

Parcourir une chaîne en utilisant les indices :

```
for i in range(len(chaine)):
    print(chaine[i])
```

Parcourir une chaîne sans utiliser les indices :

```
for caractere in chaine:
    print(caractere)
```

Indiquer un passage à la ligne dans une chaîne : insérer *\n* dans la chaîne

Pour effectuer des *print* l'un au bout de l'autre sans aller à la ligne : *print(chaine, end='')*

V. Listes, tableaux à deux dimension

Attention : si on écrit

```
liste1 = [1,2,3]
liste2 = liste1
liste2[1]=5
```

print(liste1)

On obtient [1,5,3]. En effet si liste2=liste1, les références liste1 et liste2 pointent au même endroit dans la mémoire et modifier liste2 modifie en même temps liste1.

Définir une liste vide : *liste = []* ou *liste = list()*

Longueur d'une liste : *len(liste)*

Créer la liste des entiers de a inclus à b exclu : *list(range(a, b))*

Créer la liste des entiers de 0 à b exclu : *list(range(b))*

Accéder à l'élément d'indice i : *liste[i]* (les indices commencent à 0)

Effacer l'élément d'indice i : *del liste[i]* ou *liste.pop(i)*

Effacer un élément : *liste.remove(element)*

Effacer les éléments d'indices entre i inclus et j exclu : *del liste[i :j]*

Concaténer deux listes : *liste = liste1+liste2*

Ajouter un élément à la fin d'une liste : *liste.append(element)*

Insérer un élément à la position d'indice i : *liste.insert(i , element)*

Renvoyer l'indice d'un élément de la liste : *liste.index(element)*

Trier une liste : *liste.sort()* (trie liste)

Renvoyer une liste triée : *liste_triee = sorted(liste)* (ne trie pas liste)

Savoir si un élément est dans une liste : *element in liste* (renvoie un booléen)

Renvoyer la somme des éléments d'une liste : somme = sum(liste)

Appliquer une fonction à tous les éléments d'une liste : *list(map(nom_fonction, liste))*

Parcourir une liste en utilisant les indices :

```
for i in range(len(liste)):
    print(liste[i])
```

Parcourir une liste sans utiliser les indices :

```
for element in liste:
    print(element)
```

Parcourir une liste pour utiliser les indices et les valeurs:

```
for indice, valeur in enumerate(liste) :
    print("L'indice est {} et la valeur est {}".format(indice, valeur))
```

Pour remplir une liste, on peut utiliser append:

```
liste = []
for i in range(5):
    liste.append(i**2)
```

Les tableaux à deux dimensions sont des listes de listes : *tableau = [[1,2,3], [4,5,6], [7,8,9]]*

[1,2,3] est la première ligne, *[4,5,6]* la deuxième…

Il existe aussi la classe array du module numpy.

Obtenir l'élément sur la ligne i et la colonne j : *element = tableau[i][j]*

Parcourir un tableau à deux dimensions en utilisant les indices :

```
for i in range(len(tableau)):
    for j in range(len(tableau[i])) :
        print(tableau[i][j])
```

Parcourir un tableau à deux dimensions sans utiliser les indices :

```
for ligne in tableau:
    for element in ligne :
```

```
    print(element)
```

Créer un tableau à deux dimensions (5 lignes 5 colonnes) où chaque élément est le produit de la ligne et de la colonne (parmi d'autres méthodes) :

```
tableau = []
for i in range(5):
    tableau.append([])
    for j in range(5) :
        tableau[i].append(i*j)
```

VI. Dictionnaires

Définir un dictionnaire vide : *dictionnaire = {}* ou *dictionnaire = dict()*

Créer un dictionnaire avec des clés/valeurs : *dictionnaire = {clé1 : valeur1, clé2 : valeur2}*

Ajouter une clé/valeur à un dictionnaire : *dictionnaire[clé] = valeur*

Accéder à une valeur à partir de la clé *: variable = dictionnaire[clé]*

Supprimer une valeur connaissant la clé : *del dictionnaire[clé]* ou *dictionnaire.pop(clé)*

Longueur d'un dictionnaire : *len(dictionnaire)*

Parcourir les clés d'un dictionnaire :

```
for cle in dictionnaire.keys() :
    print(cle)
```

Parcourir les valeurs d'un dictionnaire :

```
for valeur in dictionnaire.values() :
    print(valeur)
```

Parcourir un dictionnaire pour utiliser les clés et les valeurs:

```
for cle, valeur in dictionnaire.items() :
    print("la clé est {} et la valeur est {}".format(cle, valeur))
```

VII. Calculs

Addition, soustraction, multiplication, division : + - * /

Reste de la division euclidienne : %

Puissance : 4**2 (4 au carré) ou *pow(4,2)*

Pour la suite, il faut importer le module math (from math import *).

Racine carrée :*sqrt(5)*

Nombre pi :*pi*

Sinus, cosinus, exponentielle, logarithme népérien, valeur absolue, partie entière :*sin, cos, exp, log, abs, floor*

Comparaison de deux valeurs : == (égal à) != (différent de)

Non, et, ou : *not, and, or*

Pour la suite, il faut importer le module random (from random import *).

Générer un entier aléatoire entre a (inclus) et b (inclus) : *randint(a,b)*

Générer un réel dans l'intervalle [0 ;1[: *random()*

VIII. Exceptions

Lorsqu'il y a une erreur d'exécution (division par 0, conversion d'une chaîne de caractères invalide en nombre,…) Python lève une exception et renvoie un message d'erreur. On peut intercepter cette erreur avec la syntaxe suivante :

```
Instructions
try :
   Instructions qui peuvent provoquer une erreur
   instructions si tout se passe bien
except :
   Instructions à exécuter en cas d'erreur
Suite du programme
```

Exemple : saisie d'un nombre entier dans la console.

```
saisie_valide = False
#On boucle tant qu'on n'a pas saisi un nombre entier
while not saisie_valide :
   try :
      saisie = int(input("Entrer un nombre entier: "))
      saisie_valide = True
   except :
      print("La saisie n'est pas un nombre entier")
#On continue le programme dès que la saisie est valide
Print("Le nombre saisi est {}".format(saisie))
```

IX. Fonctions

Les fonctions ayant un rapport entre elles sont généralement regroupées dans un module (fichier).

Définition d'une fonction :

```
def nom_fonction(nom_parametre1, nom_parametre2, nom_parametreN) :
   """ Explications sur la fonction : elles seront incluses automatiquement dans la docstring : que
renvoie la fonction, à quoi correspondent les paramètres ? """
   Instructions
   return resultat
```

return est optionnel (une fonction peut ne rien renvoyer) et interrompt l'exécution de la fonction.

La docstring apparaît si on tape *help(nom_fonction)*

Si les paramètres n'ont pas de nom, il faut respecter l'ordre lors de l'appel de la fonction.

On peut donner des valeurs par défaut aux paramètres (seulement pour les derniers paramètres):

```
def nom_fonction(nom_parametre1, nom_parametre2=5, nom_parametre3=6) :
```

Avec l'appel *nom_fonction(2)*, *nom_parametre1* prend la valeur 2, *nom_parametre2* prend la valeur 5 et *nom_parametre3* prend la valeur 6.

On peut nommer les paramètres lors de l'appel de la fonction. Par exemple, si on a :

```
def nom_fonction(n1, n2, n3):
```

on peut appeler la fonction sous la forme *nom_fonction(n2=5, n1=7,n3=3).*

Pour plus de souplesse, on peut combiner les valeurs par défaut et les paramètres nommés.