Lorsque l'on veut tester plusieurs fois de suite un script, il faut exécuter à nouveau ce dernier, ce qui peut être très contraignant. Pour contourner ce problème, on utilise des fonctions. Les fonctions permettent aussi d'organiser le travail de développement lorsque les scripts deviennent de taille conséquente.

Partie I : Découverte

A. Définition et utilisation de fonction

Une fonction est un programme qui prend en **argument** un ensemble de variables et qui **renvoie** un résultat. La notion de fonction en informatique est comparable à la notion de fonction en mathématiques.

Exemple : Avec la fonction y(x) = 3x+2, pour une valeur donnée de x, nous aurons une valeur de y.

La fonction en informatique est basée sur la même idée :

En Python, on distingue la **définition** d'une fonction (au moyen du mot-clef *def*) de son **utilisation**. Quand on définit une fonction, celle-ci ne s'exécute pas immédiatement, mais sa définition est stockée en mémoire pour être utilisée quand on appellera la fonction plus tard dans le programme.

Le **prototype** de la fonction correspond à la déclaration et aux arguments de la fonction.

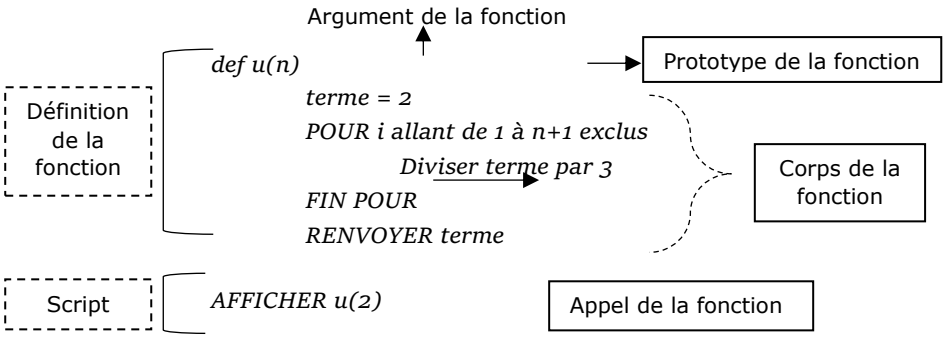
```
def NomFonction(arguments)
```

Une fonction peut avoir 0, 1, 2 ou autant d'arguments que l'on veut. C'est le prototype de la fonction qui indique combien elle en prend. Les arguments d'une fonction sont des noms de variables. Lors de l'appel d'une fonction, on précise les valeurs que l'on veut que ces arguments prennent, en respectant l'ordre dans lequel ces arguments sont écrits dans la définition de la fonction. Ces valeurs lors de l'appel sont aussi appelées arguments de la fonction.

L'avantage de l'utilisation d'une fonction est qu'on peut utiliser cette fonction au milieu d'un autre programme pour obtenir un résultat intermédiaire.

Exemple :

La fonction ci-dessous permet de calculer le terme de rang n d'une suite telle que $u_n = u_{n-1} \times 3$, avec $u_0 = 2$ (qui donne donc les termes $u_0 = 2$, $u_1 = 6$, $u_2 = 18$, $u_3 = 54$…), où n est choisi par l'utilisateur.



Lorsqu'on appelle u(2) dans la console, cela équivaut à exécuter le script à l'intérieur de la fonction u en affectant la valeur 2 à la variable n.

On peut aussi appeler la fonction dans une autre fonction. Par exemple, pour calculer la somme $u_0+u_1+\ldots+u_n = \sum_{k=0}^{n} u_k$ *où* u_n *est la suite de l'exemple précédent:*

```
def S(n)
    somme = 2
    POUR k allant de 1 à n+1 exclus
        Ajouter u(k) à somme
    FIN POUR
    RENVOYER somme
```

Sans utiliser de fonction, ce programme serait bien plus difficile à écrire !

La présence d'argument n'est pas obligatoire : une fonction peut ne prendre aucun argument et tout de même renvoyer un résultat (ce dernier sera alors « constant »).

Elle peut aussi ne prendre ou ne renvoyer aucun résultat, mais être utile lors de la répétition d'une instruction dans un code. Dans ce cas, soit la fonction se termine par un RENVOYER seul, soit à la fin du corps de la fonction si aucun RENVOYER n'est inclus.

Note : Dans certains langages, on utilise les termes méthode ou procédure pour qualifier une fonction "qui ne renvoie rien".

Exemple : une fonction qui tire un trait

```
def TirerTrait()
    AFFICHER "----------"
```

Attention : si la fonction ne renvoie rien, l'appel de cette fonction renvoie None.

Exemple : print(TirerTrait()) affichera *None*

Il est aussi intéressant de noter que, lors de l'utilisation d'une **boucle** de répétition d'instruction (POUR ou TANT QUE) dans une fonction, la commande RENVOYER peut permettre **d'arrêter brutalement la boucle** quand le résultat désiré est obtenu.

A noter qu'il est possible d'inclure une documentation sur la fonction, accessible dans la console avec *help(NomFonction)*. Le texte de documentation en défini à la ligne suivant le prototype de la fonction, placé entre triples guillemets.

Exemple : help(TirerTrait) affichera « Cette fonction permet de tirer un trait »

```
def TirerTrait()
    """ Cette fonction permet de tirer un trait"""
    AFFICHER "----------"
```

Lorsqu'on appelle une fonction, les valeurs d'entrée doivent, bien entendu, être du même type que les paramètres de la fonction, dans l'ordre de leur apparition dans le prototype de la fonction.

Exemple : La fonction ci-dessous prend une chaine de caractères et un nombre en arguments

```
def LongueurVol(destination, duree)
    SI duree inférieure à 2
        AFFICHER "Le vol pour", destination, "est un vol court"
    SINON
        AFFICHER "Le vol pour", destination, "est un vol long"
    FIN SI
```

Appeler la fonction avec LongueurVol("Los Angeles",11) affecte la valeur "Los Angeles" à la variable destination et la valeur 11 à la variable duree. Mais si on appelle la fonction en écrivant LongueurVol(11,"Los Angeles"), on aura un message d'erreur car les variables destination et duree n'auront pas le bon type.

De plus, les valeurs d'entrée peuvent avoir certaines conditions imposées.

Exemple : La fonction RacineCarre(x) qui renvoi $\sqrt{x}$
L'argument de cette fonction, x, doit être un nombre (de type entier par exemple) positif.
Les conditions sur x sont donc : nombre et positif. La valeur renvoyée sera alors un nombre positif.

Les conditions (type, valeur, etc…) sur les arguments s'appellent les **préconditions**, et imposent des conditions sur le résultat renvoyé, appelée **postconditions**.

Une manière de tester les pré- et postconditions et d'utiliser la méthode assert. Cette instruction est une aide au débogage qui teste une condition : si la condition est vraie, le programme continu de s'exécuter normalement ; si la condition est fausse, une exception AssertionError (avec un message d'erreur facultatif) est levée.

La syntaxe est la suivante : *assert condition, "message d'erreur"*

Exemple : Dans la fonction RacineCarre(x) qui renvoi $\sqrt{x}$, on peut ajouter les tests suivants :

```
assert int(x)==x and x>=0, "Erreur : x n'est pas un entier positif"
```

B. Bibliothèque logicielle

Une bibliothèque logicielle (*library* en anglais) est une collection de fonctions pré-écrites et prêtes à être utilisées par des programmes. L'intérêt des bibliothèques réside dans le fait qu'elles contiennent du code utile que l'on ne désire pas avoir à réécrire à chaque fois**.**

Les bibliothèques sont enregistrées dans des fichiers semblables aux fichiers de programmes, et sont accompagnées d'un index permettant de retrouver facilement chaque fonction. Les fonctions sont regroupées de par leur appartenance à un même domaine conceptuel (mathématique, graphique, etc).

Pour que le code exécutable puisse accéder aux instructions d'une fonction d'une bibliothèque qu'il utilise, il faut amener en mémoire les fonctions des bibliothèques utilisées, de sorte que lorsqu'une fonction est appelée, son code soit disponible en mémoire. La bibliothèque est alors qualifiée de dynamique.

I.1. Dans la capture d'écran ci-dessous, déterminer :

a. Où est le prototype de la fonction et quels sont ses arguments.

b. Où cette fonction est-elle appelée (2 appels) et, dans chaque appel, quelle valeur prend chaque argument de la fonction.

main.py saved

```
1  def SommePuissances(x,n):
2      somme = 0
3      for i in range(n+1):
4          somme += x**n
5      return somme
6
7  res = SommePuissances(2,5)
```

https://IntroPython.mgendre.repl.run

```
> print(res)
192
> SommePuissances(3,3)
108
>
```

I.2. Décrire précisément ce que font les algorithmes suivants. Quel est le résultat affiché dans la console lorsque le code est exécuté ?

• Algorithme A

```
IMPORTER BIBLIO math

def AlgoA(x)
    RENVOYER x²

AFFICHER RacineCarrée(AlgoA(1) + AlgoA(2))
```

• Algorithme B

```
def AlgoB(x)
    RENVOYER x + 3

n =0
POUR i allant à 3 exclus
    n=AlgoB(n)
FIN POUR
AFFICHER n
```

• Algorithme C

```
def AlgoC(n)
    POUR i allant de 0 à 100 exclus
        AFFICHE i
        SI i est égal à n
            RENVOYER
        FIN SI
    FIN POUR

AFFICHER AlgoC(5)
```

• Algorithme D

```
def AlgoD(texte)
    N = 0
    POUR les caractères de la chaine un par un
        SI le caractère est "a"
            Ajouter 1 à N
        FIN SI
    FIN POUR
    RENVOYER N
```

Partie II : A vos ordinateurs !

La syntaxe Python (=CQFR !)

Fonction
def nom_de_la_fonction(arguments): corps de la fonction return resultat S'il y a plusieurs arguments, on les sépare par une virgule. Il n'est pas nécessaire pour une fonction d'avoir des arguments, ni de renvoyer un résultat (pas de return ou un return non-suivi). Pour appeler la fonction, on utilise dans le script ou la console : nom_de_la_fonction(valeurs des arguments)

APPEL D'UNE BIBLIOTHÈQUE	
from BIBLIO import *	Importer toutes les fonctions d'une bibliothèque
from BIBLIO import Fonction	Importer une fonction particulière venant du module
Bibliothèque math Fonctions mathématiques telles que sin, cos, la racine carrée sqrt ou la constante pi	

sqrt(x)	Racine carrée	pi	Constante π
sin(x)	Sinus	cos(x)	Cosinus
abs(x)	Valeur absolue		
Bibliothèque random Générateur de variables aléatoires			
randint(A,B)		Entier aléatoirement choisi entre A et B (inclus)	

Les variables utilisées dans une fonction sont appelées **variables locales** : elles n'existent qu'à l'intérieur de la fonction et sont détruites une fois sorti de la fonction.
Pour utiliser des variables accessibles à l'intérieur comme à l'extérieur d'une fonction, il faut utiliser des **variables globales**, à définir à l'intérieur de la fonction.
*Attention : l'utilisation de variables globales peut être risquée et causer des bugs – **il vaut mieux l'éviter autant que possible***

Exemple :
On considère le script ci-dessous.

```
1  def EquDroite(x) :
2      A = 4
3      y = A*x
4      return y
5
6  print(EquDroite(2), A)
```

Quand on exécute le script, on a un message d'erreur car l'ordinateur ne connaît pas la variable A en dehors de la fonction EquDroite.

Avec le script suivant, le problème est résolu car A est déclarée comme variable globale

```
1  def EquDroite(x) :
2      global A
3      A = 4
4      y = A*x
5      return y
6
7  print(EquDroite(2), A)
```

Rappelez-vous : le code d'une fonction ne tourne QUE si cette fonction est appelée (en donnant des valeurs pour ses paramètres).

A. Pour débuter ensemble

II-A.1. Écrire le code qui réalise la tâche demandée (dont l'algorithme est donné dans le sujet), sans oublier les commentaires, et le tester.

a. [Algorithme d'exemple] On considère une suite géométrique de raison $r = 1/3$ et de premier terme $u_0 = 2$:

$$u_0+u_1+\ldots+u_n = \sum_{k=0}^{n} u_k \text{ où } u_n = u_{n-1} \times r$$

Étant donné un rang n donné en argument, écrire une première fonction u(n) qui calcule le terme de rang n, puis une seconde fonction S(n) qui calcule la somme.

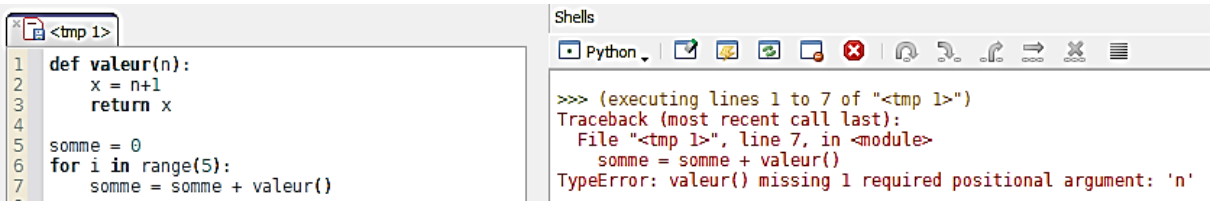
b. [Algorithme C] Etant donné un entier n, écrire une fonction TrouveN dans laquelle une boucle POUR affiche les entiers de 0 à 100, mais qui s'arrête si on tombe sur la valeur n.

Vérifions que nous avons compris : Exercice flash !

Débogage

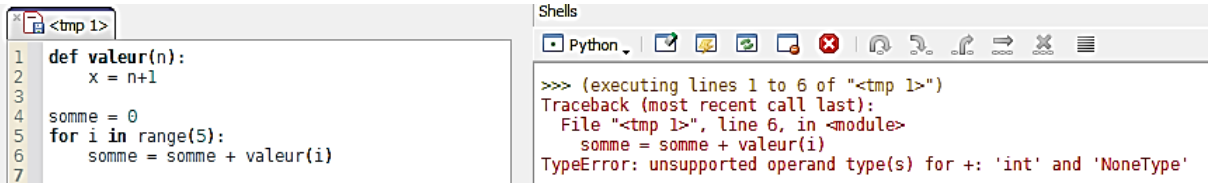
Lorsqu'une fonction est appelé mais qu'il n'y a pas le bon nombre ou type d'argument, le compilateur affiche un message.

Exemple : On veut utiliser la fonction valeur dans le code, mais elle a mal été appelée.



Lorsqu'une fonction est utilisée mais qu'il n'y a pas de renvoi de résultat (pas de `return`), le compilateur donne une erreur de type (la valeur « renvoyée » par la fonction est alors de type `Null` – il n'y en a pas !)

Exemple : On veut utiliser la fonction `valeur` dans le code pour calculer une somme, mais comme `valeur` ne renvoie pas de résultat, le code ne peut pas faire la somme d'un élément entier et d'un élément `null`.



B. Exercices pour s'entrainer

Pour chacun des exercices suivants, écrire le code qui réalise la tâche demandée et le tester. Ne pas hésiter à écrire l'algorithme avant de coder, et penser aux commentaires !!!

II.B.1. Les suites de Syracuse sont définies de la manière suivante :

u_0 est un entier naturel non nul et, pour tout $n \in \mathbb{N}$, $u_{n+1} = \begin{cases} \frac{u_n}{2} \text{ si } u_n \text{ est pair} \\ 3u_n+1 \text{ si } u_n \text{ est impair} \end{cases}$

Étant donné un entier u_0, écrire une fonction `Syracuse` qui renvoie le rang n du premier terme égal à 1 de la suite de Syracuse associée.

II.B.2. On cherche à connaitre la vitesse moyenne d'un trajet en avion, connaissant l'heure de départ (ex : 09h10), l'heure d'arrivée (ex :13h50) et la distance parcourue. On considère ici que le départ et l'arrivée se font le même jour et que les horaires sont donnés pour la plage horaire de la zone de départ uniquement (pas de décalage horaire).

a. Ecrire une première fonction `CalculDuree` qui prend en arguments l'heure *Hd* et les minute *Md* du temps de départ ainsi que l'heure *Ha* et les minutes *Ma* du temps d'arrivée et qui calcule la durée *Duree* entre les deux temps. *Rappel : Il y a 60 minutes dans 1 heure*

b. Ecrire ensuite une seconde fonction `CalculVitesse` qui prend en argument l'heure *Hd* et les minute *Md* du temps de départ, l'heure *Ha* et les minutes *Ma* du temps d'arrivée ainsi que la distance parcourue *D* et qui affiche la vitesse moyenne en km/h en utilisant `CalculDuree`.

Aide : La vitesse moyenne est donnée par $vitesse = \frac{distance}{durée}$

c. Quelles sont les préconditions sur les arguments *Hd*, *Md*, *Ha* et *Ma* de la fonction `CalculDuree` ? Quelles sont les postconditions sur le résultat de cette fonction ? Introduire les tests nécessaires dans le code.

II.B.3. On peut générer des entiers pseudo-aléatoires compris entre A et B inclus en python au moyen de la fonction `randint(A,B)`, que l'on importe depuis la bibliothèque `random`.

a. Écrire une fonction `LanceDe` qui simule 6 lancers d'un dé à 6 faces et renvoyer la moyenne des résultats obtenus.

b. Écrire une fonction `VingtLances` qui simule 20 expériences dont chacune consiste à lancer 6 fois le dé, afficher la moyenne à chaque fois ainsi que la plus grande moyenne obtenue lors des 20 expériences.

Pour aller un peu plus loin…

Réécrire les fonctions des lancés de dés en utilisant uniquement des boucles TANT QUE

Pour aller encore plus loin…

En mathématique, *factorielle*(n) est le nombre entier défini par *factorielle*(1)=1 et, pour tout $n > 1$, *factorielle*(n) = $1 \times 2 \times \ldots \times n-1 \times n$.

Ecrire une fonction `Factorielle` qui renvoi le résultat $1 \times 2 \times \ldots \times n-1 \times n$ si la condition est vérifiée, de telle sorte à ce qu'elle s'appelle elle-même (sans boucle).

On appelle cela une fonction récursive.

Ce qu'il faut retenir

❖ Prototype d'une fonction

Une fonction est un programme qui prend en argument un ensemble de variables et qui renvoie un résultat. Le prototype de la fonction correspond à de la déclaration et aux arguments de la fonction.
On distingue la définition d'une fonction de son utilisation. Quand on définit une fonction, celle-ci ne s'exécute pas immédiatement, mais sa définition est stockée en mémoire pour être utilisée quand on appellera la fonction plus tard dans le programme.

❖ Arguments et résultats

Une fonction peut demander ou non des arguments, et renvoyer ou non des résultats : ni l'un, ni l'autre ne sont obligatoires.
Attention : une fonction ne renvoyant rien renvoie un type `None` lorsqu'elle est appelée.
Les conditions (type, valeur, etc…) sur les arguments s'appellent les préconditions, et imposent des conditions sur le résultat renvoyé, appelées postconditions.

❖ Variables locales et variables globales

Les variables utilisées dans une fonction sont appelées variables locales : elles n'existent qu'à l'intérieur de la fonction et sont détruites une fois sorti de la fonction.
Pour utiliser des variables accessibles à l'intérieur comme à l'extérieur d'une fonction, il faut utiliser des variables globales, à définir à l'intérieur de la fonction.

❖ Bibliothèque logicielle

Une bibliothèque logicielle est une collection de fonctions pré-écrites et prêtes à être utilisées par des programmes.
Pour que le code exécutable puisse accéder aux instructions d'une fonction d'une bibliothèque qu'il utilise, il faut amener en mémoire les fonctions des bibliothèques utilisées, de sorte que lorsqu'une fonction est appelée, son code soit disponible en mémoire. La bibliothèque est alors qualifiée de dynamique.

❖ Fonctions et bibliothèques : Syntaxe python *(Voir sujet)*