



Version 1.3 (07 mars 2022)

Démarrer Python en Lycée Professionnel

A propos de ce document : Ce document est une version PDF d'un parcours Moodle réalisé dans l'objectif d'accompagner des enseignants dans leurs premiers pas en algorithmique en maths/sciences.

Depuis la transformation des programmes de 2019, nos élèves doivent travailler des bases d'algorithmique à travers l'utilisation du langage Python. Ce document propose des retours et idées pour travailler avec les élèves.

Il n'a pas pour vocation à transformer l'enseignant en un développeur. Mais plutôt à réfléchir "global" lors de l'approche de l'algorithmique pour mettre en parallèle les notions vues en maths et le langage Python. L'objectif n'est pas d'injecter coûte que coûte du Python pour faire joli.

L'un des objectifs en fil rouge de ce papier est de privilégier une approche sans print/input quand cela peut être évité.

Table des matières

Chapitre 1 Travailler sur Python et bases d'algorithmique

Environnement de travail	3
Algorithmique	4

Chapitre 2 Langage Python - Bases

Premier programme décrypté	5
Retour à Pythagore	7
Opérations et Variables	9
Premier exercice	11
Boucles	12
Deuxième application	14
Tests (if, elif, else)	15
Listes	17
Application 3	20
Listes en compréhension (avancé)	21

Chapitre 3 Des bons réflexes

Conventions sur les variables	23
Type de variables et manipulations	24
L'importance de l'indentation	26
Modules externes	27

Chapitre 4 Des idées d'activité

Rappel des BO (programmes)	29
Probas - 2nd - Lancer de pièces avec comptage	32
Probas - Term - Lancer de pièces avec comptage	33
Suites - 1ere/Term - Générer une suite numérique	34
Géométrie - Tous niveaux - Calculer des aires	35
Géométrie - 1ere/Term - Vecteurs	38
Suites - Term - Trouver le rang max pour ne pas dépasser une valeur	40
Prof - Générateur d'équations	42
Fonctions - Term - Encadrement d'une solution à $f(x)=g(x)$	45
Géométrie - Tous niveaux - Construction d'une pyramide	48

Chapitre 5 Pour aller plus loin

Décomposer un programme en modules	50
Utilisation de CAPYTALE avec les élèves	52

Chapitre 6 Scripts Python

Travailler sur Python et bases d'algorithmique

Environnement de travail

Python est un langage de programmation nécessitant un environnement de travail et un moteur (qui n'est pas visible).

On peut installer sur son ordinateur Python (cela peut parfois être complexe) ou utiliser un service en ligne qui propose une interface sur le navigateur et un serveur distant pour exécuter le code.

En local, le serveur est l'ordinateur, la puissance de ce dernier va donc influencer sur l'aisance de Python pour réaliser ce qu'on lui demande.

Le conseil : travailler en ligne pour éviter l'installation permet de prendre des habitudes qui pourront être utilisées de partout (dans un établissement, en changeant de machine ...)

Quelques idées d'éditeurs en ligne :

- [Basthon](#)
- [Trinket](#)
- [CAPYTALE](#) (basée sur Basthon, permet à l'enseignant de gérer une classe et de récupérer les travaux)

Et quelques exemples d'installation locales :

- [EduPython](#) : noyau simple et libre avec une version de Python simplifiée
- [Anaconda](#) : solution complète et complexe

Remarque : les calculatrices supportent maintenant Python :

- [Numwork](#) (on peut télécharger et tester les scripts)
- Casion 35+ et supérieures.
- TI-83 Python et supérieures.

Pour avoir testé uniquement Numworks : c'est utilisable pour recopier un script simple mais sinon privilégier le téléchargement depuis le PC.

Algorithmique

Avant de vouloir apprendre un langage comme Python, se posent plusieurs questions :

1. Par quoi commencer ?
2. Quelles pourraient-être ensuite les applications avec les élèves ?

On pourrait se dire qu'être capable d'écrire un algorithme, c'est déjà une compétence mobilisée : avoir une idée de la route à emprunter pour résoudre un problème.

Un exemple pour mettre en place un algorithme

Je veux vérifier si n'importe quel triangle dont on connaît les mesures des côtés est rectangle.

Ce problème sera traité en guise de fil rouge pour le début de cette présentation.

Écrivons une suite d'étapes permettant de résoudre le problème (sans utiliser de pseudo code) :

1. Trouver le plus grand côté ;
2. Calculer le carré de cette valeur et la stocker.
3. Calculer le carré des deux autres côtés ;
4. Faire la somme des résultats précédents et la stocker.
5. Comparer les deux valeurs stockées.
6. Conclure en utilisant la réciproque de Pythagore (SI, SINON ALORS)

Ce problème "complexe" au départ a été décomposé en sous problèmes. Si on suit cette recette, alors on pourra conclure quant à la question originelle.

La place de Python là dedans ?

Python est un langage qui permet de mettre en œuvre cet algorithme en utilisant un langage. On va devoir traduire depuis le langage naturel (ou pseudo code selon le cas) vers le langage Python puis faire fonctionner le programme.

On peut faire exactement la même chose avec un autre langage (exemple : Scratch et la programmation par blocs)

La grande difficulté est de transformer un problème en algorithme "à la main".

Langage Python - Bases

Premier programme décrypté

C'est l'heure de se jeter dans le bain. Plutôt que de voir trop grand pour le moment, entrons dans Python avec un exemple moins ambitieux :

```
1 a=5
2 b=-2
3 c=1
4 x=3
5 f=a*x**2+b*x+c
6 print("L'image de",x,"par la fonction f est",f)
```

Posons-nous quelques questions :

1. A quoi sert ce script ? A calculer l'image d'un nombre par une fonction.
2. Quel est le problème de base qui a pu conduire à l'écriture de ce script ? Créer un programme permettant de calculer l'image d'un nombre par une fonction de la forme $f(x) = ax^2 + bx + c$.
3. Quel a été le choix d'algorithme fait ?
 - Définir les coefficients du polynôme a , b et c ;
 - Définir la valeur pour laquelle on calcule l'image x ;
 - Faire le calcul et stocker le résultat dans une variable f .
 - Afficher le résultat sous la forme d'une phrase.
4. Que changer dans ce programme pour le réutiliser ? La ligne 4 : c'est la variable d'entrée. On verra qu'on peut ne pas la figer !

Quelques résultats :

```
>>> # script executed
L'image de 3 par la fonction f est 40
>>> # script executed
L'image de -2 par la fonction f est 25
>>> # script executed
L'image de 0 par la fonction f est 1
>>>
```

Expliquons quelques points du langage Python utilisé dans ce script :

- Ligne 1 à 4 : on a créé des **variables**. Elles stockent des valeurs qu'on peut ensuite réutiliser.
- Ligne 5 : une nouvelle variable f mais cette fois **définie à partir d'autres variables**.
- Toujours ligne 5 : pour mettre au carré en Python on peut utiliser l'expression $x**2$ (on aurait pu écrire $x \times x$). **En python, l'opérateur $**$ correspond à la puissance.**
- Remarque : en Python, l'affectation des variables se fait via un symbole $=$.
- Ligne 6 : la commande `print()` permet d'afficher quelque chose à l'écran. A l'intérieur des `()` on a :
 - Du texte entouré de guillemets " " (remarque, dans la phrase on a mis un `'` devant le guillement simple `'` pour que Python ne le traite pas comme un caractère spécial)
 - Des variables
 - Les éléments sont séparés par des virgules ,
 - Dans cet exemple, il y a 4 éléments dans le `print` (deux zones texte et deux variables).
- ★ L'utilisation de `print` est confortable mais pour ma part, je considère cette fonction comme s'éloignant trop de l'esprit "fonction" mathématique. Je l'utilise beaucoup pour de l'affichage intermédiaire et du "debugage" mais j'essaye de limiter cette utilisation.

Retour à Pythagore

Retour à mon premier problème !

Un exemple pour mettre en place un algorithme

Je veux vérifier si n'importe quel triangle dont on connaît les mesures des côtés est rectangle.

Ce problème sera traité en guise de fil rouge pour le début de cette présentation.

Écrivons une suite d'étapes permettant de résoudre le problème (sans utiliser de pseudo code) :

1. Trouver le plus grand côté ;
2. Calculer le carré de cette valeur et la stocker.
3. Calculer le carré des deux autres côtés ;
4. Faire la somme des résultats précédents et la stocker.
5. Comparer les deux valeurs stockées.
6. Conclure en utilisant la réciproque de Pythagore (SI, SINON ALORS)

Dans un monde idéal, on pourra envisager **plusieurs modules différents** pour traiter les différents problèmes. On pourra même imaginer faire travailler des élèves en binôme sur chaque module et à la fin assembler les travaux en un programme fonctionnel. C'est aussi l'intérêt de Python : on peut (et on verra comment) facilement intégrer des modules !

1. : Déterminer le plus grand côté,
2. : Calculer le carré d'un nombre et le stocker dans une variable
3. : Comparer deux valeurs et conclure en utilisant la réciproque de Pythagore.

Le module 2 serait utilisé plusieurs fois, d'où l'idée de ne l'écrire qu'une fois !

A ce stade, vos connaissances ne permettent pas de réaliser un programme en python à partir de l'algorithme, voici donc une proposition :

```

1 def rectangle(a,b,c):
2     """vérifions si le triangle est rectangle"""
3     hypotenuse = ""
4     c_1=""
5     c_2=""
6     if a>b:
7         hypotenuse=a
8         c_1,c_2=b,c
9     else:
10        hypotenuse=b
11        c_1,c_2=a,c
12    if hypotenuse<c:
13        hypotenuse=c
14        c_1,c_2=b,a
15
16    h_2=hypotenuse**2
17    somme=c_1**2 + c_2**2
18    if h_2 == somme:
19        verdict="Le triangle est rectangle"
20    else:
21        verdict="Le triangle n'est pas rectangle"
22    return(verdict)
23

```

Ce programme renvoie :

```

>>> # script executed
>>> rectangle(3,4,5)
'Le triangle est rectangle'
>>> rectangle(3,4,6)
"Le triangle n'est pas rectangle"
>>>

```

Incompréhensible ? C'est normal ! Et déprimant à la fois ! L'idée est d'aller pas à pas en expliquant :

- Ligne 1 : On crée **une fonction** nommée rectangle qui admet 3 paramètres en entrée ;
- Ligne 2 : L'utilisation de texte entre "''" "''" permet de commenter l'usage de la fonction (on pourra lire ce texte en tapant help(rectangle) dans ce cas)
- Ligne 3 à 5 : définition de variables vides à ce stade.
- Ligne 6 à 14 : on cherche le maximum entre a, b et c en réalisant des **tests** (et on remplit les variables définies). Pour cela on compare d'abord a avec b, puis avec c.
- Ligne 16 : calcul du carré de l'hypoténuse.
- Calcule les autres carrés et les additionne.
- Ligne 18 à 21 : comparaison et conclusion en ligne 22.
- La commande "verdict" aurait pu être remplacée par du print mais ici, on cherche à garder en tête l'esprit de fonction : une entrée / une sortie (via le return).

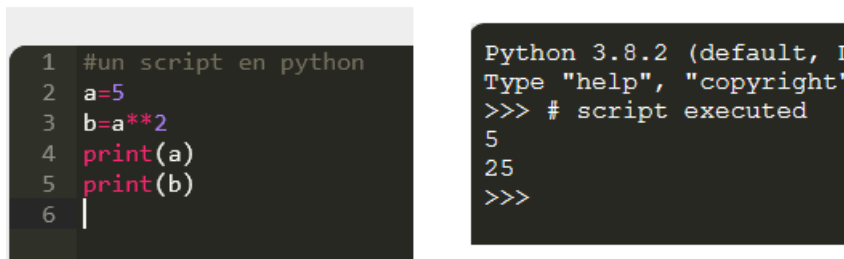
Opérations et Variables

Avant toute chose voici un éditeur en ligne pour tester en direct les commandes :

<https://console.basthon.fr/>

On peut travailler directement dans l'interface sans compléter la partie gauche de l'écran qui sert à écrire des scripts (ou programmes).

Un **script** python est un fichier possédant une extension .py qui peut être stocké, envoyé, utilisé par un autre programme. Le script doit être chargé dans la **console** pour être utilisé.



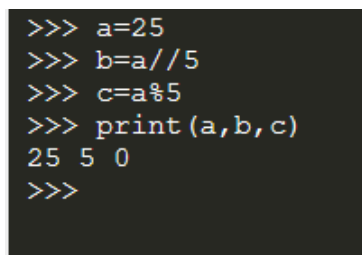
```
1 #un script en python
2 a=5
3 b=a**2
4 print(a)
5 print(b)
6 |
```

```
Python 3.8.2 (default, D
Type "help", "copyright"
>>> # script executed
5
25
>>>
```

La console est une interface sur laquelle tourne le moteur de Python et qui peut exécuter les commandes du langage. On peut travailler directement dans la console mais notre travail n'est pas enregistré.

Liste des opérations mathématiques courantes :

- $a*b$ (multiplication)
- $a+b$ (addition)
- a/b (division)
- $a-b$ (soustraction)
- $a**b$ (puissance)
- $a\%b$ (reste de la division de a par b)
- $a//b$ (partie entière de la division de a par b)



```
>>> a=25
>>> b=a//5
>>> c=a%5
>>> print(a,b,c)
25 5 0
>>>
```

Pour d'autres opérations (comme par exemple la racine carrée) le chargement d'un module externe pourra être requis.

Affectation de variables

En Python, une variable est désignée par une lettre (ou une suite de caractères) et va contenir un élément (qui peut être un nombre, une chaîne de caractère, une liste ...). **On affecte une variable avec le signe =**. Voici trois affectations :

```
>>> a=5
>>> type(a)
<class 'int'>
>>> a=5.0
>>> type(a)
<class 'float'>
>>> a="5"
>>> type(a)
<class 'str'>
```

La fonction intégrée de Python `type()` retourne le type d'élément que contient la variable. Une même variable peut être redéfinie très simplement (voir capture).

Si on doit réaliser **plusieurs affectations de variables**, on peut le faire en direct (une ligne par variable) ou en parallèle.

```
>>> a=5
>>> b=10
>>> c=7
>>> print(a,b,c)
5 10 7
>>> d,e,f=8,-5,9
>>> print(d,e,f)
8 -5 9
```

Attention : l'affectation en parallèle est globalement néfaste pour la lisibilité du code.

Une autre méthode pour redéfinir les variables peut être la suivante :

```
>>> a=5
>>> a=a+1
>>> print(a)
6
>>> a+=1
>>> print(a)
7
```

Pratique lors de l'utilisation de compteurs par exemple.

Premier exercice

Pour apprendre, il faut mettre les mains dans le moteur, alors voici une proposition.

Exercice

On souhaite définir deux variables a et b contenant des entiers de votre choix et afficher deux nouvelles variables qui retourneraient le produit et la somme.

Et si on veut aller plus loin voici deux suggestions de niveau avancé :

1. Lors de l'exécution du script, le programme demande à l'utilisateur de saisir deux entiers.
2. On transforme le script pour créer une fonction nommée `debut` qui accepte deux paramètres et qui renvoie produit et somme.

En espérant que les liens restent valides voici des corrigés :

1. [Première version](#)
2. [Niveau avancé 1](#)
3. [Niveau avancé 2](#)

Script disponible en annexe à la fin de ce document.

Boucles

Il sera souvent nécessaire d'utiliser la notion de **boucle**. C'est le fameux "tant que" ! Python l'autorise de plusieurs manières avec chacune leurs avantages et inconvénients.

Un point général sur la structure

Python est un langage qui nécessite une indentation parfaite pour fonctionner. On termine la ligne possédant une fonction par le symbole : et on recommence la ligne du dessous avec un décalage (indentation). Voir plus bas.

La boucle WHILE (tant que)

C'est, à mon sens, la boucle la plus simple à comprendre en français : tant que l'objectif fixé n'est pas atteint, alors il se passe quelque chose.

Attention avec While il est très facile de lancer une boucle infinie qui va bloquer la console.

Un exemple : Soit une valeur de départ fixée, tant que la valeur 5 n'est pas atteinte, on affiche la valeur actuelle et on l'incrémente de 1.

```
1 #une boucle avec while
2 i=0
3 while i!=5:
4     print(i)
5     i=i+1
6 |
```

Remarques :

- On déclare la variable *i* AVANT le while sinon Python ne peut pas comprendre, *i* n'existe pas.
- On termine la ligne du while par les fameux : (source d'oubli fréquente, et donc erreur).
- On a indenté (décalé) les instructions avec une tabulation. Il y en a deux (instructions) : la première affiche la valeur et la seconde incrémente.
- Sans la ligne 5, la boucle serait infinie.

```
>>> # script executed
0
1
2
3
4
>>>
```

En passant : attention, si on inverse les lignes 4 et 5, on incrémente la variable avant d'afficher, mais la valeur finale de *i* en fin de boucle sera bien de 5 :

```
>>> # script executed
0
1
2
3
4
>>> print(i)
5
```

Si on veut vraiment afficher le 5, on change la valeur dans le while, ou on inverse les positions des lignes.

La boucle FOR (pour tout)

Deuxième syntaxe de boucle, bornée (elle ne peut pas boucler à l'infini, en théorie). En voici un exemple :

```
1 #une boucle avec for
2 for i in [0, 1, 2, 3]:
3     print(i)
4
```

On définit un ensemble d'éléments (ici les nombres entiers de 0 à 3) et on affiche la valeur pour tout élément dans la liste. On utilisera plutôt la structure :

```
1 #une boucle avec for
2 for i in range(0,4):
3     print(i)
4
5
```

Ou même dans le cas où le départ est 0 :

```
1 #une boucle avec for
2 for i in range(4):
3     print(i)
4
```

On peut vérifier que les trois versions affichent la même chose :

```
>>> # script executed
0
1
2
3
```

Comment choisir entre FOR et WHILE ?

Il n'est pas nécessaire de choisir, il est toujours possible d'utiliser les deux. Si on connaît d'avance le nombre d'itérations à faire, on peut préférer le FOR. Sinon on peut préférer le WHILE si on veut insister sur le test réalisé dans la boucle.

[Pour aller plus loin sur cette notion de boucles.](#)

Deuxième application

Soit une entreprise qui décide d'augmenter chaque année tous les salaires de 1%.

Application

Proposer un script qui :

- Définit une variable salaire (qui aura pour valeurs respectivement 1500 - 1750 et 2000).
- Affiche les salaires des 10 augmentations correspondant aux années d'ancienneté dans l'entreprise (on n'affichera pas le salaire de base que l'on connaît).
- Ne pas tenir compte des arrondis dans cet exercice sauf si on veut apprendre la fonction round().

Comme d'habitude, niveau avancé :

1. Créer un script qui demande le salaire de base mais faisant la même chose ;
2. Faire en sorte que le retour contienne, en plus du salaire, l'année.
3. Et si on imaginait une fonction qui accepte trois entrées : salaire, nombre d'année et pourcentage d'augmentation. Cette fonction permettrait de faire un simulateur.

```
Entrez ici le salaire de base : 1675
Le salaire au bout de 1 année(s) est 1691.75
Le salaire au bout de 2 année(s) est 1708.6675
Le salaire au bout de 3 année(s) est 1725.754175
Le salaire au bout de 4 année(s) est 1743.01171675
Le salaire au bout de 5 année(s) est 1760.4418339175
Le salaire au bout de 6 année(s) est 1778.046252256675
Le salaire au bout de 7 année(s) est 1795.8267147792417
Le salaire au bout de 8 année(s) est 1813.784981927034
Le salaire au bout de 9 année(s) est 1831.9228317463044
Le salaire au bout de 10 année(s) est 1850.242060063767
4
>>> salaire(1675,5,1)
Année n° 1 Salaire : 1691.75
Année n° 2 Salaire : 1708.6675
Année n° 3 Salaire : 1725.754175
Année n° 4 Salaire : 1743.01171675
Année n° 5 Salaire : 1760.4418339175
>>>
```

[correction en ligne](#) - ou en annexe.

Tests (if, elif, else)

Il est très souvent indispensable de réaliser des tests sur des variables (c'est d'ailleurs implicite dans les boucles).

Exemple : si mon nombre est pair, j'affiche pair, sinon, j'affiche impair.

Ce test, écrit en français, est "simple". Pour le réaliser en Python, on va utiliser l'instruction IF (si) et ses acolytes. La structure nécessite aussi un usage de l'indentation :

```
1 #un test pour la parité
2 a=int(input("Entrez un nombre : "))
3 if a%2==0:
4     print("Le nombre est pair")
5 else:
6     print("Le nombre est impair")
```

L'utilisation de # en première ligne permet de commenter. Ce script, lancé, retourne :

```
>>> # script executed
Entrez un nombre : 15888
Le nombre est pair
>>> 
```

Remarques :

- Le IF et le ELSE sont au même niveau d'indentation (aucun en l'occurrence).
- Pour tester une **égalité** on doit écrire un double ==
- Pour tester une **non-égalité**, on doit utiliser !=
- Pour tester une **inégalité** on peut utiliser < (strict) ou =< ou <= (large).
- On peut imbriquer les conditions en utilisant autant d'IF que nécessaire.

Si on souhaite intégrer d'autres conditions de même niveau (si, sinon, alors) on peut utiliser l'instruction ELIF :

```
1 from random import *
2
3 nombre = randint(1,100)
4 essai = int(input("Votre proposition : "))
5 if essai>nombre:
6     print("Trop grand !")
7 elif essai==nombre:
8     print("Gagné !")
9 else:
10    print("Trop petit !")
11
```

Quelques remarques :

- Ligne 1 : nouveauté, on importe un module externe (nommé random) qui contient la partie "choix aléatoire".
- Ligne 3 : on utilise une fonction du module random qui permet de choisir un entier aléatoire (ici entre 1 et 100)
- On utilise la commande "input" (qu'on reverra) pour demander à l'utilisateur d'entrer un nombre. L'utilisation de INT avant input permet de transformer la réponse en entier (et retournera une erreur si je tape "bonbon" par exemple).

- ★ Comme pour le print, l'utilisation de la fonction input() est, selon moi, une mauvaise habitude : le programme se met en pause pour demander quelque chose. Ce n'est pas l'interaction "naturelle" d'une fonction mathématiques.

Avec vos connaissances

Proposer une modification du script précédent pour autoriser un maximum de 10 essais à l'utilisateur.

On peut même envisager un affichage du nombre d'essais restants.

Pour arrêter la boucle quand l'utilisateur trouve, on peut utiliser l'instruction break à un endroit bien placé

```
>>> # script executed
Votre proposition : 50
Trop grand !
Il reste 9 essais.
Votre proposition : 25
Trop petit !
Il reste 8 essais.
Votre proposition : 40
Trop petit !
Il reste 7 essais.
Votre proposition : 45
Trop petit !
Il reste 6 essais.
Votre proposition : 47
Trop grand !
Il reste 5 essais.
Votre proposition : 46
Gagné !
>>>
```

[Correction en ligne](#) ou en annexe.

Listes

En Python, une variable peut définir un conteneur de type "Liste".

C'est quoi une liste ?

Une liste est un ensemble ordonné d'éléments ayant un indice et qui se définit par des crochets [].

Exemple : a=[1,2] est une liste contenant deux éléments, le premier de rang 0 est 1.

```
>>> a=[1,2]
>>> a[0]
1
>>> a[1]
2
>>> a[2]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
IndexError: list index out of range
>>> █
```

On remarque qu'étant donné que la liste ne contient que 2 éléments, l'appel de l'indice 3 retourne une erreur (out of range = hors de portée).

On peut **ajouter un élément** dans une liste avec la commande append() avec la structure a.append(élément) :

```
>>> a.append(5)
>>> a[2]
5
>>> print(a)
[1, 2, 5]
>>>
```

Le nouvel élément se place naturellement à la suite du précédent.

Si on souhaite **insérer un élément** on peut le faire via la commande insert(rang,élément) :

```
>>> print(a)
[1, 2, 5]
>>> a.insert(2,3)
>>> print(a)
[1, 2, 3, 5]
>>>
```

On insère ici un élément "en tant que nouveau rang 2" (le 5 est donc décalé dans la liste !)

On peut supprimer un élément de deux manières différentes :

1. Avec la commande remove(élément). Python parcourt la liste et supprime la première occurrence (peu importe son rang) de l'élément.
2. Avec la commande pop(rang). Python supprime (et affiche) le rang indiqué.

```
>>> print(a)
[1, 2, 3, 5]
>>> a.remove(3)
>>> print(a)
[1, 2, 5]
>>> a.pop(2)
5
>>> print(a)
[1, 2]
>>>
```

Autres opérations sur les listes

Liste non exhaustive ...

- `reverse()` > inverse les éléments d'une liste ;
- `sort()` > trie une liste (si les données sont des nombres, par ordre croissant, si ce sont des chaînes, par ordre alphabétique, si pas du même type renvoie une erreur) ;
- concaténer deux listes. Si `a` et `b` sont deux listes, alors `c=a+b` les additionne.

Quel usage des listes ?

L'idée naturelle par rapport à nos programmes est de relier les listes aux suites numériques. Voici l'exemple d'un script permettant de créer (et d'afficher) un tableau de valeur pour une suite arithmétique :

```
1 u_0=1
2 r=4
3 liste=[]
4 n = 10
5 for i in range(0,n):
6     liste.append(u_0+i*r)
7 print(liste)
```

```
>>> # script executed
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37]
```

Remarques :

- On utilise ici une boucle pour créer une liste, elle est bornée par l'utilisateur (ici $n = 10$) donc une petite préférence pour le FOR.
- Si on indente le `print` pour le placer dans le FOR, on peut obtenir ceci :

```
>>> # script executed
[1]
[1, 5]
[1, 5, 9]
[1, 5, 9, 13]
[1, 5, 9, 13, 17]
[1, 5, 9, 13, 17, 21]
[1, 5, 9, 13, 17, 21, 25]
[1, 5, 9, 13, 17, 21, 25, 29]
[1, 5, 9, 13, 17, 21, 25, 29, 33]
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37]
>>>
```

Cela permet de bien voir les étapes de construction de la suite.

Quand on a rempli une liste avec les valeurs de la suite, on peut imaginer les utiliser pour extraire le terme de rang n souhaité, calculer une somme ...

Pour l'enseignant qui travaille sur ce thème et qui cherche à générer des suites pour des exercices/exemples, une idée pour remplacer le tableau est d'utiliser un script python !

```

1 def arithmetique(u_0,r,n):
2     liste=[]
3     for i in range(n):
4         liste.append(u_0+i*r)
5     return(liste)
6
7 def geometrique(v_0,q,n):
8     liste=[]
9     for i in range(n):
10        liste.append(v_0*q**i)
11    return(liste)

```

```

>>> # script executed
>>> geometrique(100,1/2,10)
[100.0, 50.0, 25.0, 12.5, 6.25, 3.125, 1.5625, 0.78125, 0.390625, 0.1953125]
>>> arithmetique(100,-5,10)
[100, 95, 90, 85, 80, 75, 70, 65, 60, 55]
>>>

```

Ce script crée deux fonctions que l'on peut appeler et utilise à chaque fois une boucle FOR pour remplir la suite selon les paramètres entrés par l'utilisateur. La structure des fonctions sera revue plus tard.

Application 3

Comme dernière application de cette introduction aux bases, je vous propose un nouveau saut dans les probabilités. A ce stade, vous savez générer une liste de nombres, importer le module random.

Dé à 6 faces

On souhaite simuler le lancer d'un dé non truqué à 6 faces et que le programme affiche le nombre de fois où une certaine face (choisie par l'utilisateur) apparaît.

Quelques instructions :

- Il faudra importer le module random (from random import);
- On pourra stocker les lancers dans une liste et ensuite parcourir la liste pour compter.

On peut aussi imaginer deux niveaux de réussite pour cette activité :

1. Le programme fait ce qui est demandé, on définit dans le script la face recherchée et le nombre de dés lancés et il compte ;
2. Le programme permet à l'utilisateur de spécifier à la fois le nombre n de lancers et la face à compter.

Bon courage !

```
>>> # script executed
[2, 4, 1, 3, 4, 4, 4, 3, 3, 6, 3, 3, 6, 4, 1, 6, 1, 5, 2, 6, 2, 6, 3, 3, 4, 5,
5, 3, 5, 6, 3, 6, 1, 5, 5, 1, 4, 5, 6, 3, 3, 1, 6, 6, 2, 5, 1, 3, 5, 2, 2, 2, 4
, 5, 1, 2, 2, 1, 1, 4, 4, 2, 5, 3, 6, 2, 6, 2, 4, 5, 4, 5, 6, 2, 6, 6, 5, 5, 2,
2, 6, 4, 2, 6, 6, 4, 5, 6, 6, 4, 3, 1, 5, 2, 3, 1, 4, 6, 2, 5]
Sur 100 lancers, il y a eu 16 fois le nombre 4
>>> simulateur(10,4)
[6, 6, 4, 1, 2, 6, 6, 2, 6, 5]
Il y a eu 1 fois le chiffre 4 sur 10 lancers.
>>>
```

C'est un exemple directement utilisable avec des élèves, on peut imaginer leur faire modifier quelques lignes pour adapter la situation à un problème donné.

[Correction en ligne](#) - et en annexe.

Listes en compréhension (avancé)

Une utilisation avancée des listes est ce qu'on appelle la liste en compréhension (list comprehension en anglais). L'objectif derrière cette notion est de simplifier l'écriture du code (dans le sens : le réduire).

Pour comprendre ce principe, un exemple support :

```
1 ma_liste=[1,2,3,4,5,6,7,8,9,100,145,1288,2000]
>>> ma_liste
[1, 2, 3, 4, 5, 6, 7, 8, 9, 100, 145, 1288, 2000]
```

Cette liste est définie explicitement et contient 13 éléments. Je veux en extraire les nombres pairs. (Je n'ai pas encore utilisé de liste en compréhension). Comment faire ?

- Récupérer la taille de la liste ;
- Créer une liste vide ;
- Parcourir la première liste pour trouver le premier nombre pair ;
- Le recopier dans la liste 2 ;
- Répéter jusqu'à la fin
- Afficher la nouvelle liste.

Voici donc une possibilité classique utilisant les connaissances à ce stade :

```
1 ma_liste=[1,2,3,4,5,6,7,8,9,100,145,1288,2000]
2 print(ma_liste)
3 x=0
4 taille=len(ma_liste)
5 liste2=[]
6 while x!=taille:
7     if ma_liste[x]%2 == 0:
8         liste2.append(ma_liste[x])
9     x=x+1
10 print(liste2)
11
```

Qui affichera :

```
>>> # script executed
[1, 2, 3, 4, 5, 6, 7, 8, 9, 100, 145, 1288, 2000]
[2, 4, 6, 8, 100, 1288, 2000]
>>>
```

Cette première version est longue. Si on omet la ligne 2 (qui sert à réafficher la liste du départ), on a besoin de 7 lignes de code.

En mode "compréhension" le code devient :

```
1 ma_liste=[1,2,3,4,5,6,7,8,9,100,145,1288,2000]
2 liste3=[x for x in ma_liste if x%2==0]
3 print("résultat par le second script : ", liste3)
```

```
>>> # script executed
résultat par le second script : [2, 4, 6, 8, 100, 1288, 2000]
```

En français la ligne 2 se lit : "on définit les éléments de la liste 3 comme étant égaux à la valeur de x pour chaque élément x de la liste nommée ma_liste à la condition que le reste de la division par 2 soit nul."

Python passe tous les éléments de la liste à la division et ne garde que ceux remplissant la condition qu'il intègre dans la liste3.

Un autre exemple :

```
1 pairs=[x for x in range(100) if x%2==0]
2 print(pairs)
```

On crée une liste des nombres pairs de 0 à 100 et on l'affiche.

Remarque concernant le print

Ici (et beaucoup ailleurs!) j'utilise la fonction print pour afficher des résultats pour "expliquer" le fonctionnement : même si j'ai un discours visant à éviter le print, je trouve au contraire que c'est une très bonne idée de s'en servir régulièrement pour afficher ce qu'il se passe dans le programme.

Des bons réflexes

Conventions sur les variables

Quelques bons réflexes concernant les variables.

Règles sur les noms

Si elles ne sont pas respectées, vous aurez des erreurs :

- On ne commence pas par un chiffre ;
- Pas d'espaces ;
- Caractères de A à z, de 0 à 9 et `_` uniquement ;
- Attention : les variables sont sensibles à la casse.

De plus, des conventions sont généralement utilisées pour faciliter le partage ou la relecture entre pairs :

- Privilégier les lettres en minuscules ;
- Séparer les lettres des nombres par un `_`

Pour ma part j'ai encore une habitude récurrente par exemple de donner une seule lettre pour les compteurs dans les boucles. J'essaye aussi de limiter la taille des variables pour ne pas avoir de noms à rallonge.

Type de variables et manipulations

A savoir qu'en Python les données dans les variables sont généralement de 4 types :

- Une chaîne de caractère (string, str) : elle se place entre guillemets "". Exemple : "Bonjour" ;
- Un nombre entier (integer,int). Exemple : 5 ;
- Un nombre décimal (float,float). Exemple : 5.0 (le séparateur est un point) ;
- Un booléen (boolean, bool). Exemple : True / False (attention : sensible à la casse).

En Python, 25 est différent de "25". Il est parfois nécessaire de convertir un type de donnée en un autre (par exemple pour calculer).

La fonction input() par exemple renvoie toujours un contenu de type "string", on ne peut donc faire aucun calcul dessus :

```
1
2 a=input("merci de donner votre nombre préféré")
3 b=a+2
4 print(b)
```

```
>>> # script executed
merci de donner votre nombre préféré
Traceback (most recent call last):
  File "<input>", line 3, in <module>
TypeError: can only concatenate str (not "int") to str
```

Python a ici essayé de fusionner deux chaînes (car le a, premier, est une chaîne) or ce n'est pas possible. Il n'a pas tenté de faire l'addition, il y aurait aussi eu une erreur. Quelle solution ? La conversion ou un changement d'approche pour demander une entrée par l'utilisateur.

Par la fonction de conversion int()

```
>>> c="10"
>>> c+c
'1010'
>>> d=int(c)
>>> d+d
20
>>>
```

Ce qui permet de faire fonctionner le script du début :


```
1
2 a=int(input("merci de donner votre nombre préféré"))
3 b=a+2
4 print(b)
```

```
>>> # script executed
merci de donner votre nombre préféré3
5
>>>
```

Par une fonction

Plutôt que d'utiliser `input()`, on peut créer une fonction dans laquelle le paramètre entré est reconnu comme un nombre directement :

```
1 def prefere(n):
2     """Merci d'entrer votre nombre préféré pour le doubler"""
3     return n+n
4
```

```
>>> # script executed
>>> prefere(2)
4
```

L'importance de l'indentation

L'indentation est une notion vitale en Python, il faut la prendre en compte sous peine d'erreurs en permanence :

```
1 a=2
2 if a==2:
3     print("bouh")
4
```

```
>>> # script executed
File "<unknown>", line 3
    print("bouh")
    ^
IndentationError: expected an indented block
>>>
```

Alors que :

```
1 a=2
2 if a==2:
3     print("bouh")
4
```

```
>>> # script executed
bouh
>>>
```

L'indentation est une manière de repérer les blocs dans un script. Il faut retenir que dès qu'une ligne se termine par les " : " alors la partie suivante sera indentée (décalée). On doit parfois cumuler ces indentations :

```
1 from random import *
2 ete=["fraise","framboise","abricot","prune","peche","haricot","tomate","courgette"]
3 legume=["haricot","tomate","courgette"]
4
5 a=randint(1,20)
6
7 if a <=7:
8     print("C'est une espèce de l'été")
9     if ete[a] in legume:
10         print("Et en plus c'est un légume !")
11     else:
12         print("Et c'est un fruit")
13 else:
14     print("Ce n'est pas une espèce de l'été")
15
16
```

Le IF de la ligne 7 provoque de nouvelles indentations à l'intérieur.

Modules externes

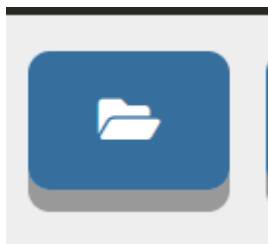
Python permet de ne pas réinventer la roue à chaque fois en mutualisant le travail en important des modules externes.

Un module Python est en fait un autre script .py (ou plusieurs) qui sont appelés selon les besoins. On a déjà vu le module random, le module maths au fil et à mesure des besoins.

Pour illustrer ce fonctionnement, je vais créer un module simple que je vais appeler puissance.py et dans lequel je vais placer trois fonctions :

```
1 def carre(n):
2     return(n**2)
3
4 def cube(n):
5     return(n**3)
6
7 def exposant(x,a):
8     return (x**a)
9
10
```

Ce script est sauvegardé en tant que puissance.py, je vais le charger dans Basthon via le bouton :



Remarque : La plupart des modules classiques sont déjà connus et je n'aurai pas besoin de les charger. Exemple random.

Je peux charger une fonction du module si je la connais et obtenir :

```
1 from puissance import cube
2
3
4 a=2
5 b=cube(a)
6 print(b)
7
```

```
Python 3.8.2 (default, Dec 25
Type "help", "copyright", "cr
>>> # script executed
8
>>>
```

Mais j'ai aussi la possibilité :

- D'importer tout le module (import puissance)
- D'importer toutes les fonction (from puissance import *)
- D'importer le module en lui donnant un alias (import puissance as pui)

Ce qui peut donner :

```

1 import puissance as pui
2
3 a=2
4 b=pui.cube(a)
5 c=pui.carre(b)
6 print(b)
7 print(c)

```

```

Python 3.8.2 (default, Dec
Type "help", "copyright",
>>> # script executed
8
64
>>>

```

On remarque ici que donner un alias n'est pas très pertinent, mais c'est une possibilité. L'intérêt des alias est surtout, à mon sens, de repérer ce qui est chargé en externe de ce qui est dans le script en cours de rédaction pour retrouver plus facilement les problèmes.

Parmi les nombreux modules existants voici une petite liste non exhaustive de modules utiles en maths/sciences :

- Le module math (contient les fonctions trigo, la racine carré, pi ...);
- Le module random (déjà vu dans ce document);
- Le module turtle qui permet de dessiner comme dans scratch;
- Le module numpy pour le calcul scientifique
- Le module matplotlib pour les graphiques (mais pour nos usages, Python sera peu pertinent face aux autres outils graphiques)

Un exemple tiré de [la banque d'exemples](#) de Basthon pour Turtle :



Des idées d'activité

Rappel des BO (programmes)

L'objectif de cette partie est de donner quelques activités et un rappel des éléments du BO pour l'algorithmie.

En seconde

- Mathador (on donne un nombre cible et un nombre de paramètres, l'élève doit programmer pour trouver le nombre cible en exécutant le programme)
- Créer un dé
- Créer un calculateur de surfaces/volumes, convertisseur.
- Créer un tableau de valeurs pour une fonction.
- ...

En première :

- Calcul des termes d'une suite arithmétique
- Affichage des termes d'une suite arithmétique
- Second degré : calcul de delta, résolution d'équations
- ...

En terminale :

- Travail sur la notion d'ensemble en probabilités en comparant des listes
- Suites géométriques
- Générateur d'exercice pour l'étude de fonctions
- ...

Rappels du programme de seconde :

Capacités	Connaissances
Analyser un problème.	
Décomposer un problème en sous-problèmes.	

Repérer les enchaînements logiques et les traduire en instructions conditionnelles et en boucles.	Séquences d'instructions, instructions conditionnelles, boucles bornées (for) et non bornées (while).
Choisir ou reconnaître le type d'une variable. Réaliser un calcul à l'aide d'une ou de plusieurs variables.	Types de variables : entiers, flottants, chaînes de caractères, booléens. Affectation d'une variable.
Modifier ou compléter un algorithme ou un programme. Concevoir un algorithme ou un programme simple pour résoudre un problème.	
Comprendre et utiliser des fonctions. Compléter la définition d'une fonction. Structurer un programme en ayant recours à des fonctions pour résoudre un problème donné.	Arguments d'une fonction. Valeur(s) renvoyée(s) par une fonction.

Rappels du programme de première :

Capacités	Connaissances
Analyser un problème. Décomposer un problème en sous-problèmes.	
Repérer les enchaînements logiques et les traduire en instructions conditionnelles et en boucles.	Séquences d'instructions, instructions conditionnelles, boucles bornées (for) et non bornées (while).
Choisir ou reconnaître le type d'une variable. Réaliser un calcul à l'aide d'une ou de plusieurs variables.	Types de variables : entiers, flottants, chaînes de caractères, booléens. Affectation d'une variable.
Modifier ou compléter un algorithme ou un programme. Concevoir un algorithme ou un programme simple pour résoudre un problème.	
Comprendre et utiliser des fonctions. Compléter la définition d'une fonction. Structurer un programme en ayant recours à des fonctions pour résoudre un problème donné.	Arguments d'une fonction. Valeur(s) renvoyée(s) par une fonction.
Générer une liste. Manipuler des éléments d'une liste (ajouter, supprimer, extraire, etc.). Parcourir une liste. Itérer une ou plusieurs instructions sur les éléments d'une liste.	Liste.

Rappels du programme de terminale :

Capacités	Connaissances
Analyser un problème. Décomposer un problème en sous-problèmes.	
Repérer les enchaînements logiques et les traduire en instructions conditionnelles et en boucles.	Séquences d'instructions, instructions conditionnelles, boucles bornées (for) et non bornées (while).
Choisir ou reconnaître le type d'une variable. Réaliser un calcul à l'aide d'une ou de plusieurs variables.	Types de variables : entiers, flottants, chaînes de caractères, booléens. Affectation d'une variable.
Modifier ou compléter un algorithme ou un programme. Concevoir un algorithme ou un programme simple pour résoudre un problème.	
Comprendre et utiliser des fonctions. Compléter la définition d'une fonction. Structurer un programme en ayant recours à des fonctions pour résoudre un problème donné.	Arguments d'une fonction. Valeur(s) renvoyée(s) par une fonction.

Générer une liste. Manipuler des éléments d'une liste (ajouter, supprimer, extraire, etc.). Parcourir une liste. Itérer une ou plusieurs instructions sur les éléments d'une liste.	Liste.
--	--------

Probas - 2nd - Lancer de pièces avec comptage

Simuler des lancers

Objectifs :

- Simuler un lancer de n pièces ;
- Compter le nombre de pile (en prenant pile=0) et face (1) ;
- Afficher la liste des pièces pour contrôler.

Version proposée :

```
1 from random import *
2
3 def piece(n,valeur):
4     """lance n pièce et compte la face indiquée (0 pour pile,
5     1 pour face """
6     liste=[]
7     compteur = 0
8     if valeur==0:
9         mot="pile"
10    else:
11        mot="face"
12    for i in range (n):
13        liste.append(randint(0,1))
14    for x in liste : #on parcourt la liste et on compte
15        if x==valeur:
16            compteur+=1
17    print(liste)
18    print("Il y a eu {} {} sur un total de {} lancers soit {}".format(c
19
20
```

```
Python 3.8.2 (default, Dec 25 2020 21:20:57)
Type "help", "copyright", "credits" or "license" for mor
e information.
>>> # script executed
>>> piece(100,1)
[1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1
, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0
, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
0, 0, 0, 1, 1, 1, 1]
Il y a eu 59 face sur un total de 100 lancers soit 59.0%
>>>
```

[Version en ligne](#) et en annexe.

Avec les élèves on pourrait :

- Modifier pour lancer un dé à la place ;
- Modifier pour lancer plusieurs objets en même temps ;
- Utiliser en CCF comme un support TICE pour simuler des lancers à la place d'un tableur
- ...

Probas - Term - Lancer de pièces avec comptage

Reprise de la version seconde mais cette fois avec des objectifs différents

Simuler des lancers

Objectifs :

- Simuler un lancer de n pièces ;
- Utiliser les listes et la compréhension de liste dans un cas simple ;
- Afficher la liste des pièces pour contrôler.

Version proposée :

```
1 from random import *
2
3 def piece(n):
4     """lance n pièce et affiche la sortie """
5     liste=[randint(0,1) for x in range(n)]
6     compteur = [x for x in liste if x == 0]
7     print(liste)
8     print("Pile > 0 et face > 1")
9     print(len(compteur),"pile(s) et",n-len(compteur),"face(s).")
10
11 |
```

Python 3.8.2 (default, Dec 25 2020)
Type "help", "copyright", "credits"
>>> # script executed
>>> piece(10)
[1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
Pile > 0 et face > 1
2 pile(s) et 8 face(s).
>>>

[Version en ligne](#) et en annexe.

Avec les élèves on pourrait :

- Modifier pour lancer un dé à la place ;
- Comprendre le processus de fonctionnement de l'algorithme ;
- Améliorer la sortie.
- ...

Suites - 1ere/Term - Générer une suite numérique

Générer une suite numérique

Mettre en oeuvre un algorithme permettant de :

- Générer automatiquement une suite (arithmétique ou géométrique) de raison et premier terme connus pour n termes ;
- Afficher éventuellement des paramètres comme la somme de ces n termes.

```
1- def suite(u_0,r,n):
2-     """génère une suite arithmétique"""
3-     U=[]
4-     somme=0
5-     for i in range (n):
6-         U.append(u_0+r*i)
7-         print(U)
8-         somme=somme+U[i]
9-     print("La somme des {} premiers termes vaut {}".format(n,somme))

I
```

```
Python 3.8.2 (default, Dec 25 2020 21:20:57)
Type "help", "copyright", "credits" or "license"
ion.
>>> # script executed
>>> suite(0,2,10)
[0]
[0, 2]
[0, 2, 4]
[0, 2, 4, 6]
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8, 10]
[0, 2, 4, 6, 8, 10, 12]
[0, 2, 4, 6, 8, 10, 12, 14]
[0, 2, 4, 6, 8, 10, 12, 14, 16]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
La somme des 10 premiers termes vaut 90
>>>
```

Idée de modification possible avec les élèves :

- Créer une seconde fonction pour l'autre type de suites ;
- N'afficher la liste qu'une fois la génération terminée ;
- Afficher sur une deuxième ligne l'indice.

[Version en ligne](#) - ou en annexe

Géométrie - Tous niveaux - Calculer des aires

Un programme Python peut être installé dans une calculatrice et utilisé quotidiennement par des élèves. On peut ici imaginer facilement l'intérêt d'un programme permettant de calculer des aires.

Calculateur d'aires

Cahier des charges :

- L'utilisateur choisit parmi une liste de figure connue du programme ;
- Il spécifie ensuite les dimensions utiles ;
- Le programme calcule la surface de la figure et renvoie le résultat

Le travail derrière ce problème est complexe d'un point de vue algorithmique (pour les élèves) et il faut insister là dessus car l'intérêt n'est pas de programmer en python le calcul de la surface d'un carré !

Voici une proposition utilisant la notion de fonction (et bien adaptée pour un enseignement en seconde avec une mise en parallèle de la notion de fonction)

1. On crée autant de fonctions que de figure ;
2. Chaque fonction accepte le nombre de paramètre utile et renvoie la surface associée

Ou une autre version :

1. On choisit la figure ;
2. (On demande l'unité)
3. On demande les dimensions (à adapter selon le 1/);
4. On calcule la surface avec la bonne formule ;
5. On renvoie le résultat.

La complexité de mise en œuvre va dépendre de la voie choisie. Le module `input()` de dialogue est peut être plus "simple" à envisager pour les élèves mais s'éloigne de la notion "fonctionnelle".

Version 1 (avec des fonctions)

D'un point de vue programmation c'est la plus simple à mettre en œuvre :

```
1 def carre(c):
2     return (c*c)
3 def rectangle(l,L):
4     return (L*L)
5 def cercle(R):
6     return(R*R*3.14)
7 def triangle(b,h):
8     return(0.5*b*h)
9
```

```
>>> carre(5)
25
>>> cercle(14)
615.44
>>> rectangle(4,5)
25
>>> triangle(5,2)
5.0
>>>
```

Version 2 (avec interaction)

```

12 def surface():
13     type=0
14     valeurs_autorisees=[1,2,3,4]
15     print("1 pour triangle")
16     print("2 pour carré")
17     print("3 pour cercle")
18     print("4 pour rectangle")
19     while type not in valeurs_autorisees:
20         type=int(input("Choix : "))
21     if type==1:
22         base=int(input("Base ? "))
23         hauteur=int(input("Hauteur? "))
24         surface=base*hauteur*0.5
25     elif type==2:
26         cote=int(input("Côté ? "))
27         surface=cote**2
28     elif type==3:
29         rayon=int(input("Rayon ? "))
30         surface=rayon**2*3.14159
31     else:
32         longueur=int(input("Longueur ? "))
33         largeur=int(input("Largeur ? "))
34         surface=longueur*largeur
35     unite=input("unité (ex: m2) : ")
36     print("S={}{}".format(surface,unite))

```

Qui donnera :

```

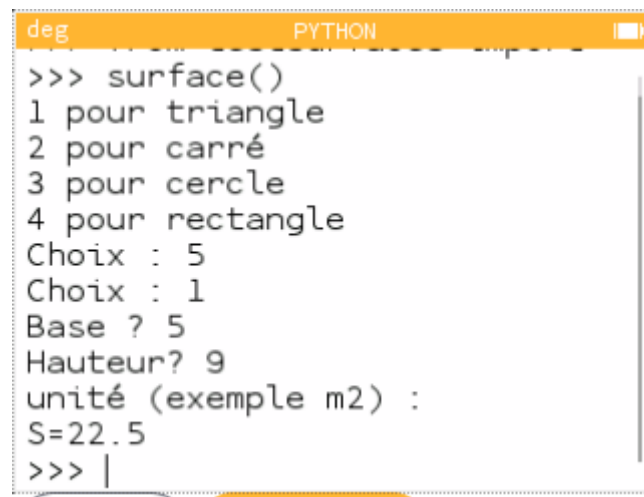
>>> surface()
1 pour triangle
2 pour carré
3 pour cercle
4 pour rectangle
Choix : 2
Côté ? 7
unité (ex: m2) : cm2
S=49cm2
>>>

```

Quelques remarques :

- Dans la version 2, sur cette proposition, les outils à connaître des élèves en 3 années de bac sont utilisés :
 - Une fonction pour lancer le programme (elle ne sert qu'à ça ici) ;
 - Plusieurs listes (valeurs_autorisees est là pour anticiper une mauvaise entrée de l'utilisateur)
 - Une boucle ;
 - Du conditionnel
- Dans la version 1, on peut imaginer renvoyer plus d'information mais en l'état une entrée donne une sortie sans habillage.

- Ce genre de programme s'adapte totalement à une calculatrice :



```
>>> surface()
1 pour triangle
2 pour carré
3 pour cercle
4 pour rectangle
Choix : 5
Choix : 1
Base ? 5
Hauteur? 9
unité (exemple m2) :
S=22.5
>>> |
```

Pistes pour travailler avec les élèves :

- On peut donner un bout du programme et demander de le continuer (V1)
- On peut rajouter des figures
- On peut (en V2) adapter le programme pour ajouter une unité par défaut qui sera utilisée si l'utilisateur ne rentre rien
- Et le calcul de volume ?
- Gestion des arrondis.

[versions en ligne](#) ou en annexe

Géométrie - 1ere/Term - Vecteurs

Vecteurs

Proposer un algorithme puis un script en Python permettant de calculer les coordonnées puis la norme d'un vecteur \overrightarrow{AB}

Quelques points de vigilance :

- Attention aux arrondis ;
- Il sera compliqué de gérer la notation du vecteur avec une flèche ;
- Bien définir la partie algo avant de se lancer car plusieurs voies possibles ;
- Le module math comporte la fonction SQRT() et sera à importer ici ;
- En première on peut faire écrire l'algorithme aux élèves ;
- On peut privilégier la notion de fonction en première.

Je vous propose encore ici deux versions fonctionnelles :

```
1 from math import *
2 def vecteur(xa,ya,xb,yb):
3     """calcule la norme et les coordonnées de vecAB"""
4     coo_x = xb-xa
5     coo_y = yb-ya
6     norme=round(sqrt(coo_x**2+coo_y**2),2)
7     print("x_AB = ",coo_x)
8     print("y_AB = ",coo_y)
9     print("||AB|| = ",norme)
10
11
```

```
>>> vecteur(4,3,0,7)
x_AB = -4
y_AB = 4
||AB|| = 5.66
```

Ou la version avec input() :

```
12 def vec():
13     print("Coord/Norme de vec(AB)")
14     xa=int(input("Abscisse du point A : "))
15     ya=int(input("Ordonnée du point A : "))
16     xb=int(input("Abscisse du point B : "))
17     yb=int(input("Ordonnée du point B : "))
18     coo_x = xb-xa
19     coo_y = yb-ya
20     norme=round(sqrt(coo_x**2+coo_y**2),2)
21     print("x_AB={}".format(coo_x))
22     print("y_AB={}".format(coo_y))
23     print("Norme ||AB|| = {}".format(norme))
24
```

```
>>> vec()
Coord/Norme de vec(AB)
Abscisse du point A : -5
Ordonnée du point A : 4
Abscisse du point B : 9
Ordonnée du point B : 3
x_AB=14
y_AB=-1
Norme ||AB|| = 14.04
>>>
```

Remarque : encore une fois c'est un exemple de programme qu'on peut envoyer sur une calculatrice :

```
deg PYTHON
>>> from test import *
>>> vec()
Coord/normes de vec(AB)
Abscisse du point A : -5
Ordonnée du point A : 4
Abscisse du point B : 9
Ordonnée du point B : 3
x_AB=14
y_AB=-1
Norme ||AB|| = 14.04
>>> |
```

Pour faire travailler les élèves on peut imaginer :

- Leur faire rédiger la partie python après avoir posé l'algorithme (ce n'est pas très compliqué vu la structure) sur papier ;
- Rajouter des fonctionnalités (tester si deux vecteurs sont colinéaires...)
- Tracer les coordonnées avec un module externe (mais dans ce cas, se poser la pertinence d'utiliser Python, Geogebra le fait très bien !)

[version en ligne](#) ou en annexe.

Suites - Term - Trouver le rang max pour ne pas dépasser une valeur

Valeur max d'une suite

Objectifs :

- Mettre sur papier un algorithme pour trouver le rang max d'une suite pour ne pas dépasser une valeur fixée ;
- Proposer une version Python permettant de réaliser cet algorithme.

Une proposition :

1. Générer les termes de la suite selon les informations fournies (par défaut, en générer 50 ou 100 si pas préciser)
2. Stocker les termes dans une liste ;
3. Parcourir la liste avec une boucle et tant que la valeur lue est plus petite que la valeur à atteindre, recopier cette valeur dans une seconde liste
4. Quand la boucle est terminée, récupérer le nombre d'itération qui donnera le rang (à +-1 prêt!)

Cette version est "brute" : on calcule un nombre de termes fixé au départ mais on ne sait pas si on en a calculé assez, il se peut que le script ne donne rien. Ici, l'utilisation d'un "tant que" n'est pas optimale. On peut préférer une boucle avec un FOR et tout faire en même temps :

```
1 def suitemax(v0,q,valeurmax):
2     """Cherche le rang à partir duquel les termes
3     de la suite sont supérieurs à valeurmax"""
4     suite=[v0*q**n for n in range(1000) if v0*q**n<=valeurmax]
5     print(suite)
6     print("""Les termes de la suite seront supérieurs à {}
7     si n est plus grand que {}.""".format(valeurmax,len(suite)+1))
```

```
>>> suitemax(1,2,1000)
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
Les termes de la suite seront supérieurs à 1000
si n est plus grand que 11.
>>>
```

La liste est ici générée en compréhension de liste ce qui économise beaucoup de lignes de codes.

[Version en ligne de cette proposition](#) (ou en annexe).

Pour illustrer la différence entre les deux modes, une version plus "simple" du programme Python :


```

11 - def suitemax2(u0,r,valeurmax):
12     """Cherche le rang à partir duquel les termes
13     de la suite sont supérieurs à valeurmax"""
14     suite=[]
15     i=0
16     #génération de la suite
17 - while i<=100: #valeur fixée pour le nombre de termes à générer
18         suite.append(u0+i*r)
19         i+=1
20     #création de la seconde suite
21     suite2=[]
22     j=0
23 - while suite[j]<=valeurmax:
24         suite2.append(suite[j]) #on remplit la liste 2
25         j+=1
26     print("""Les termes de la suite seront supérieur à {}
27     pour n plus grand que {}.""").format(valeurmax,len(suite2))

```

```

>>> # script executed
>>> suitemax2(0,1,10)
Les termes de la suite seront supérieur à 10
pour n plus grand que 11.
>>>

```

Avec les élèves on peut envisager :

- D'écrire un algorithme papier permettant la résolution du problème ;
- Expliquer les 2 versions proposées et comparer les différences ;
- Préciser les choix faits (range 1000, $i \leq 100$) ;
- Modifier pour déterminer le rang minimum pour dépasser une certaine valeur.

[version en ligne](#) ou en annexe

Prof - Générateur d'équations

Une petite gourmandise pour l'enseignant qui parfois à besoin de générer des équations pour faire travailler les élèves. J'utilise ce script en "automatismes" et en AP (à la fois en seconde, premier degré, ou plus tard).

cahier des charges

- Permettre de générer des équations de degré 1 et 2 ;
- Gérer les conflits de double signes (+-) sur les variables aléatoires

Voici une transcription de l'algorithme que j'ai utilisé :

1. Définir premier ou 2nd degré ;
2. Définir 4 variables a , b , c et d aléatoires comprises entre [bornes] (entiers ou pas ?)
3. SI second degré ALORS renvoyer $ax^2 + bx + c = 0$
4. SI second degré ET que $a = 0$ ALORS retirer a tant que $a \neq 0$;
5. SI premier degré ET qu'une des 4 variables = 0, retirer ;
6. SI premier degré ALORS renvoyer $ax + b = c + dx$ sauf SI :
7. SI premier degré ET que $a = d$ OU $b = c$ ALORS retirer ;
8. Gestion des conflits de signes (avec des SI)
9. Répéter n fois.

Ce programme peut être écrit ainsi : (ce n'est probablement pas le plus simple !)

```

1  """Générateur d'équations"""
2  from random import *
3  def equations(degre,nombre):
4      x="x"
5      for n in range(nombre):
6          a=randint(-10,10)
7          b=randint(-10,10)
8          c=randint(-10,10)
9          d=randint(-10,10)
10
11         if degre==1:
12             while a==0 or a==d:
13                 a=randint(-10,10)
14             while b==0 or b==c:
15                 b=randint(-10,10)
16             while c==0:
17                 c=randint(-10,10)
18             while d==0:
19                 d=randint(-10,10)
20             if b<0 and c>0:
21                 print("{}x{}={}+{}x".format(a,b,d,c))
22             elif b<0 and c<0:
23                 print("{}x{}={}{}x".format(a,b,d,c))
24             elif b>0 and c<0:
25                 print("{}x+{}={}{}x".format(a,b,d,c))
26             else:
27                 print("{}x+{}={}+{}x".format(a,b,d,c))
28         else:
29             while a==0:
30                 a=randint(-10,10)
31             if b<0 and c<0:
32                 print("{}x^2{}x{}=0".format(a,b,c))
33             elif b<0 and c>0:
34                 print("{}x^2{}x+{}=0".format(a,b,c))
35             elif b>0 and c<0:
36                 print("{}x^2+{}x{}=0".format(a,b,c))
37             else:
38                 print("{}x^2+{}x+{}=0".format(a,b,c))
39
40

```

On obtient le résultat :

```

>>> equations(1,10)
-3x+1=10+2x
9x+1=3-3x
2x+8=4-10x
-7x-2=-4+7x
-4x-4=-6+7x
1x+2=8+1x
9x-5=6+4x
-1x+2=-3-9x
1x+2=-9-2x
1x+3=2-7x

```

Qui permet de copier depuis la console :

$$\begin{aligned}
 -3x + 1 &= 10 + 2x \\
 9x + 1 &= 3 - 3x \\
 2x + 8 &= 4 - 10x \\
 -7x - 2 &= -4 + 7x \\
 -4x - 4 &= -6 + 7x \\
 1x + 2 &= 8 + 1x \\
 9x - 5 &= 6 + 4x \\
 -1x + 2 &= -3 - 9x \\
 1x + 2 &= -9 - 2x \\
 +3 &= 2 - 7x
 \end{aligned}$$

Avec un simple copier collé depuis Python, je me retrouve avec mes équations prêtes à mettre en forme. Le seul bémol est en second degré l'affichage de la puissance que Python ne gère pas :

```
>>> equations(2,10)
-1x^2+2x+9=0
9x^2+4x+0=0
-2x^2-4x+1=0
6x^2-7x+10=0
-6x^2-1x-4=0
7x^2-9x+2=0
8x^2+8x+6=0
-4x^2+9x-9=0
-6x^2+10x-4=0
-3x^2+9x-4=0
>>>
```

Pour le second degré on pourrait s'amuser à rajouter une condition qui éliminerait éventuellement les équations sans solutions ou à solution unique.

Intérêt avec les élèves : aucun, outil pour l'enseignant.

[Version en ligne](#) - ou en annexe

Fonctions - Term - Encadrement d'une solution à $f(x)=g(x)$

Le programme suggère cette activité :

- Déterminer un encadrement ou une valeur approchée par balayage d'une solution d'une équation du type $f(x) = g(x)$ lorsqu'on sait qu'elle existe dans un intervalle donné.

La complexité ici résulte de l'intégration de la fonction $f(x) = ax^3 + bx^2 + cx + d$ en Python (idem pour l'autre fonction). C'est une bonne occasion d'utiliser la modularité pour ne pas rendre le script trop complexe.

Version globale :

1. Spécifier la précision du pas,
2. Spécifier les bornes de l'intervalle,
3. Prévoir des valeurs par défaut au cas où.
4. Récupérer l'expression de f ;
5. Récupérer l'expression de g ;
6. Calculer les images $f(x)$ et $g(x)$ pour des valeurs de x incrémentées d'après le pas et dans l'intervalle.
7. Conclure.
8. Prévoir un cas où il n'y a pas de solutions pour éviter une boucle infinie.

Sous module pour récupérer les fonctions

Dans cet algorithme, on pourrait imaginer que les étapes 4 et 5 sont identiques et peuvent être traitées en "externe" en écrivant un bout de code à répéter (module!). Un algorithme de ce module pourrait être :

1. Demander à l'utilisateur les coefficients des fonctions de type $f(x) = ax^3 + bx^2 + cx + d$;
2. Vérifier qu'il n'y ait pas de valeurs vides sinon les transformer en 0.
3. Gérer le cas d'une valeur non numérique ;
4. Renvoyer ces coefficients pour la fonction principale.

Ce que donne cette idée transformée en langage Python : (l'utilisation de la fonction `float()` transforme un terme en nombre décimal).

```
1 def expressionfonction(n):
2     """écrit l'expression d'une fonction"""
3     a=input("Fonction {} : valeur devant x^3 (vide si zéro) : ".format(n))
4     b=input("Fonction {} : valeur devant x^2 (vide si zéro) : ".format(n))
5     c=input("Fonction {} : valeur devant x (vide si zéro) : ".format(n))
6     d=input("Fonction {} : valeur sans x (vide si zéro) : ".format(n))
7     liste=[a,b,c,d]
8     liste2=[float(x) if x!=" " else 0 for x in liste]
9     return (liste2[0],liste2[1],liste2[2],liste2[3])
10
```

Cette fonction prend un paramètre qui sera uniquement le nom à afficher dans la boîte de dialogue qui demande les coefficients (pour que l'utilisateur ne se perde pas). Il renvoie 4 valeurs. L'utilisation de deux listes est pour traiter la transformation d'une chaîne de caractère donnée

par le input en nombre décimal (float) et pour évaluer le cas où l'utilisateur n'entre rien. (""). Il manque ici à traiter le cas où la valeur ne peut être transformée en flottant (exemple si je tape le mot "banane").

Module principal

Dans la proposition suivante, la fonction est documentée de manière explicite afin d'utiliser éventuellement la commande help(encadrement) pour l'utilisateur :

```
>>> help(encadrement)
Help on function encadrement in module __main__:
encadrement(n, borneinf=-10, bornesup=10, detail=0)
    Détermine une valeur approchée à n près par balayage
    de f(x)=g(x) sur l'intervalle [borneinf;bornesup]
    par défaut -10,10
    mettre une valeur pour détail pour afficher tous les calculs
```

Et le script :

```
13 def encadrement(n, borneinf=-10, bornesup=10, detail=0):
14     """Détermine une valeur approchée à n près par balayage
15     de f(x)=g(x) sur l'intervalle [borneinf;bornesup]
16     par défaut -10,10
17     mettre une valeur pour détail pour afficher tous les calculs"""
18
19     a1,b1,c1,d1=expressionfonction("f")
20     a2,b2,c2,d2=expressionfonction("g")
21     x=borneinf
22     f_x=-100
23     g_x=100
24     while f_x != g_x:
25         f_x=round(a1*x**3+b1*x**2+c1*x+d1,3)
26         g_x=round(a2*x**3+b2*x**2+c2*x+d2,3)
27         if detail!=0:
28             print(f_x,g_x,x)
29
30         x=round(x+n,2)
31
32         if x>=bornesup:
33             print("""Je n'ai pas trouvé d'égalité entre f(x) et g(x).
34 Il faut peut être changer le pas ou l'intervalle""")
35             break
36     print("J'ai trouvé f(x)=g(x) pour x={} ({}={})".format(x-n,f_x,g_x))
37
```

Dans les paramètres on trouve un qui est obligatoire (n), les autres sont facultatifs et ont donc une valeur par défaut (qui se définit dans la fonction). Autres remarques :

- Lignes 19 et 20 : on appelle la fonction du sous module précédent ;
- Ligne 27 : ce petit bout permet d'afficher le détail de toutes les comparaisons pour tester le programme ;
- Ligne 32 : gestion de la sortie de la boucle infinie du WHILE par l'utilisation de l'instruction break

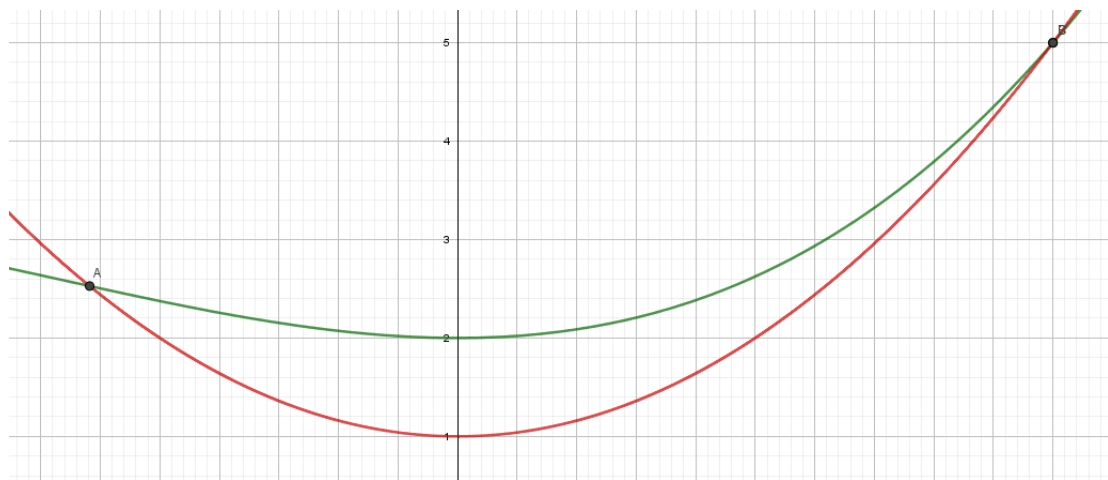
Ce programme utilise une recherche en force, méthode opposée à l'observation graphique des intersections. Il peut donc, selon le pas et l'intervalle, nécessiter beaucoup de ressources de calculs

et prendre un certain temps ... Il faudrait en plus gérer l'arrondi des valeurs.

Avec les fonctions $f(x) = x^3 + 2x^2 + 2$ et $g(x) = 4x^2 + 1$ sur l'intervalle $[-10; 10]$:

```
>>> encadrement(0.2)
Fonction f : valeur devant x^3 (vide si zéro) : 1
Fonction f : valeur devant x^2 (vide si zéro) : 2
Fonction f : valeur devant x (vide si zéro) :
Fonction f : valeur sans x (vide si zéro) : 2
Fonction g : valeur devant x^3 (vide si zéro) :
Fonction g : valeur devant x^2 (vide si zéro) : 4
Fonction g : valeur devant x (vide si zéro) :
Fonction g : valeur sans x (vide si zéro) : 1
J'ai trouvé f(x)=g(x) pour x=1.0 (5.0=5.0)
```

Si on prend $n = 0.2$ la première solution est trouvée pour $x = 1$ alors que graphiquement on observe une autre solution proche de -0.6 :



Pour la trouver, il faudrait changer le pas, et vérifier les arrondis éventuellement.

[Version en ligne](#) ou en annexe

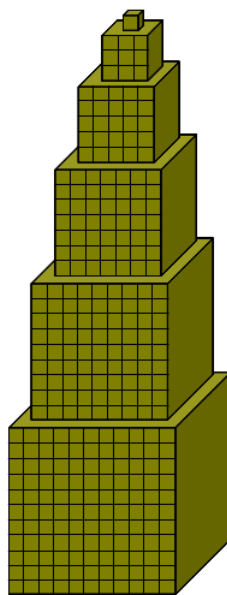
Géométrie - Tous niveaux - Construction d'une pyramide

Cette activité m'a été inspirée par le [parcours Algo du site france-ioi](#) (qui est une mine d'or pour débiter en algorithmique et en python!).

Défi

On souhaite construire une pyramide telle qu'illustrée ci-dessous où chaque étage est composé de petits cubes.

- Proposez aux élèves de mettre en place un algorithme pour déterminer le nombre de cubes nécessaires selon le nombre d'étages ;
- Traduire cet algorithme en Python ;



Exemple d'une tour allant de 1×1×1 cubes à 11×11×11 cubes

Le programme devra :

- (Prendre en compte le nombre d'étages demandé)
- Compter les cubes ;
- Afficher le résultat ;
- (Pour aller plus loin : afficher par étage le nombre nécessaire)

Une proposition fonctionnelle pour résoudre le puzzle de manière "simple" pour une pyramide de 11 étages :

```
1 somme = 0
2 etages=[11,9,7,5,3,1]
3 for i in etages:
4     nombre_pour_etage = i**3
5     somme = somme + nombre_pour_etage
6     print("Pour l'étage {} il faut {} cubes.".format(i,nombre_pour_etage))
7
8 print("Il faut au total {} petits cubes.".format(somme))
```



```
>>> # script executed
Pour l'étage 11 il faut 1331 cubes.
Pour l'étage 9 il faut 729 cubes.
Pour l'étage 7 il faut 343 cubes.
Pour l'étage 5 il faut 125 cubes.
Pour l'étage 3 il faut 27 cubes.
Pour l'étage 1 il faut 1 cubes.
Il faut au total 2556 petits cubes.
>>>
```

Travail avec les élèves

Pour aller plus loin, on peut envisager de demander à l'utilisateur le nombre d'étages, de remplir automatiquement la liste etages etc

[version en ligne](#) ou en annexe de ce script.

Pour aller plus loin

Décomposer un programme en modules

Comme il a déjà été évoqué, Python se prête très bien au découpage d'un problème en sous problèmes. C'est d'ailleurs le principe de l'algorithmie et il est compliqué d'imaginer utiliser cet outil sans avoir cet objectif dans un coin de la tête.

Utiliser Python doit avoir un intérêt visible et celui là en est un, à mon sens.

La modularité

Pour éviter qu'un code devienne vite trop long et illisible, on peut rendre les modules indépendants entre eux.

Cela permet aussi une meilleure gestion dans le temps du code, un meilleur dépannage et partage.

Ce découpage avait déjà été évoqué pour Pythagore, mais illustrons avec un autre exemple, celui du calcul des surfaces. Plutôt que de tout écrire dans un seul programme on pourrait :

- Premier module : Calcul d'une surface pour une figure
- Second module : La même pour une autre figure
- ...
- Module global : utilise au besoin les autres modules.

Les élèves peuvent travailler de manière séparée et se regrouper à la fin.

```

1  """Calculateur de surfaces"""
2  def carre(c):
3      return (c*c)
4  def rectangle(longueur,largeur):
5      return(longueur*largeur)
6  def cercle(rayon):
7      return(rayon**2*3.14)
8  def triangle(base,hauteur):
9      return(base*hauteur*0.5)
10
11 def surface(figure,dimension_1,dimension_2=0,unite="m2"):
12     """Calcule la surface d'une figure"""
13     liste=["rectangle","cercle","carre","triangle"]
14     liste2=["cercle","carre"]
15     if figure not in liste:
16         print("Je ne connais pas cette figure, attention aux guillemets")
17     else:
18         if dimension_2==0 and figure not in liste2:
19             print("erreur, il me manque une valeur")
20         else:
21             if figure=="rectangle":
22                 s=rectangle(dimension_1,dimension_2)
23             elif figure=="carre":
24                 s=carre(dimension_1)
25             elif figure=="cercle":
26                 s=cercle(dimension_1)
27             elif figure=="triangle":
28                 s=triangle(dimension_1,dimension_2)
29             print("La surface est {}".format(s,unite))

```

Remarques :

- Lignes 22, 24, 26 et 28 ce sont les appels aux modules définis au début ;
- La partie entre les lignes 15 et 20 sert à vérifier l'intégrité des informations saisies par l'utilisateur.

Si on voulait pousser, on pourrait continuer d'utiliser les premiers modules pour le volume :

```

31 def volume(figure,d1,d2,d3,unite="m3"):
32     if figure=="rectangle":
33         s=rectangle(d1,d2)
34         v=s*d3
35         print(v,unite)

```

```

>>> # script executed
>>> volume("rectangle",5,7,5)
175
>>> # script executed
>>> surface("carre",5)
La surface est 25m2
>>> volume("rectangle",5,7,5)
175 m3
>>>

```

[version en ligne](#) ou en annexe.

Utilisation de CAPYTALE avec les élèves

CAPYTALE est un service de partage d'activités de codage en ligne. Il est propulsé par Basthon, déjà utilisé sur ce support. Pour les élèves, il est accessible depuis leur identifiant académique ENT pour la plupart des régions.

On peut, sous CAPYTALE, mettre en place des activités et superviser / récupérer le travail des élèves :



Et on obtient pour une classe ou un groupe :

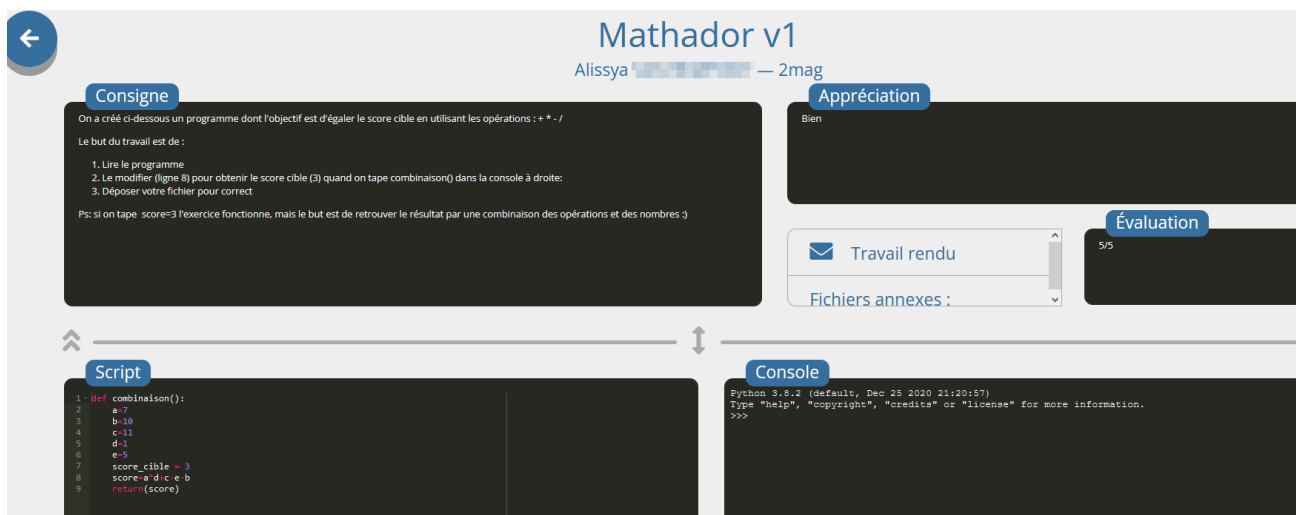
Mathador v1

-- Choisir un état -- Appliquer à la sélection

Rechercher

<input type="checkbox"/>	Dernière modif.	Nom	Classe	Mode / État	Appréciation	Évaluation
<input type="checkbox"/>	10/12/21 à 11:44		2mag			
<input type="checkbox"/>	10/12/21 à 11:57		2mag			
<input type="checkbox"/>	10/12/21 à 11:45		2mag			
<input type="checkbox"/>	10/12/21 à 11:52		2mag			
<input type="checkbox"/>	10/12/21 à 11:50		2mag			
<input type="checkbox"/>	10/12/21 à 12:00		2mag			
<input type="checkbox"/>	10/12/21 à 12:01		2mag			
<input type="checkbox"/>	10/12/21 à 12:02		2mag			
<input type="checkbox"/>	10/12/21 à 11:35		enseignants			
<input type="checkbox"/>	10/12/21 à 11:44		2mag			
<input type="checkbox"/>	10/12/21 à 12:00		2mag			
<input type="checkbox"/>	10/12/21 à 12:02		2mag		Bien	5/5

Quand l'élève a envoyé son travail, on peut l'ouvrir et surtout on peut exécuter son code directement dans sa console et observer le fonctionnement du programme.















Avec cette interface, on a tout au même endroit, et surtout la connexion est facile pour les élèves :

- Partage du lien ;
- Connexion ;
- Travail et envoi ;

- Correction / Commentaire / Retour vers l'élève.

Sans parler des possibilités de partage entre enseignants :

Vous pouvez consulter en cliquant sur le titre ou télécharger un fichier .ipynb pour le retravailler.

Afficher 10 lignes		Rechercher LP				
Type	Titre	Description	Ens.	Like	Copie	Modifié
	SUITES1 Suites arithmetiques niveau1	Activité de découverte des suites arithmétiques avec python : les variables (affectation et affichage) et la boucle for	math, math-PC_en_LP	0 	10 	18/02/2022
	Suites et Python en spé math Partie 1	Fonctions de base en Python en lien avec les suites : Calculer un terme avec une forme explicite, avec une forme récurrente, fonction de seuil	math	0 	10 	18/02/2022
	Sens de variation d'une fonction du 2nd degré	Contexte: modélisation d'un nombre de visite Observation du sens de variation Détermination du maximum Utilisation d'une liste	math-PC_en_LP	0 	9 	06/02/2022
	Etude des suites arithmétiques	Modifier les paramètres pour calculer les termes (rang 1,5,17,25,53,102 et 845) de la suite arithmétique de premier terme 4 et de raison 2,5 Compléter la ligne 8 du programme pour calculer la somme des n premiers termes Modifier la ligne 9 pour afficher en plus du rang et du terme, le résultat de ce[...]	math-PC_en_LP	0 	5 	01/02/2022

Scripts Python

Avant Propos

Tous les scripts présentés ici sont perfectibles. Ils n'ont pas vocation à être une référence absolue mais à proposer une piste de réflexion.

Premier exercice :

```
1  #version 1
2  a=5
3  b=4
4  somme=a+b
5  produit = a*b
6  print(somme)
7  print(produit)
8
9  #version niveau 2 avec input
10 a=int(input("Entrer le premier nombre"))
11 b=int(input("Entrer le second nombre"))
12 somme=a+b
13 produit = a*b
14 print("La somme est :",somme)
15 print("Le produit est :",produit)
16
17
18 #version niveau 3 avec une fonction
19 def debut(a,b):
20     """Soient deux nombres a et b, renvoie la somme puis le produit
21     """
22     somme=a+b
23     produit = a*b
24     return(somme, produit)
```

Ex 2

```
1  salaire = 1500 #remplacer ici la valeur
2  i=0
3  while i!=10:
4      i+=1
5      salaire = salaire *1.01
6      print(salaire)
7
8
9  #niveau avancé 1
10
11  salaire = int(input("Entrez ici le salaire de base : ")) #remplacer
    ici la valeur
12  i=0
13  while i!=10:
14      i+=1
15      salaire = salaire *1.01
16      print(salaire)
17
18
19  #niveau avancé 2
20
21  salaire = int(input("Entrez ici le salaire de base : ")) #remplacer
    ici la valeur
22  i=0
23  while i!=10:
24      i+=1
25      salaire = salaire *1.01
26      print("Le salaire au bout de ",i,"année(s) est",salaire)
27
28  #niveau avancé 3 (sans input ni print)
29  def salaire(s,n,a):
30      """Affiche le salaire s selon l'année n avec une augmentation
    annuelle de a%"""
31      i=0
32      while i!=n:
33          i+=1
34          s=s*(1+a/100)
35          print("Année n",i,"Salaire :",s)
```

Exercice d'application sur les tests

```
1  from random import *
2
3  nombre = randint(1,100)
4  nb=0
5  while nb<10:
6      essai = int(input("Votre proposition : "))
7      if essai>nombre:
8          print("Trop grand !")
9      elif essai==nombre:
10         print("Gagné !")
11         break
12     elif essai>100:
13         print("Petit malin, c'est plus petit que 100")
14     else:
15         print("Trop petit !")
16     print("|| reste ",9-nb," essais.")
17     nb+=1
```


Exercice 3

```
1  from random import *
2
3  #niveau 1
4  nombre=100 #on lance 100 dés
5  liste=[] #on crée une liste vide pour le moment
6  i=0 #compteur pour arrêter la boucle de création des dés
7  cible=4 #ce qu'on veut compter
8  compteur=0 #compteur qui va compter le nombre
9
10 while i!=nombre: #création de la liste de n dés
11     liste.append(randint(1,6)) #on ajoute avec append() un nombre aléatoire en 1 et 6
12     i=i+1 #pour ne pas boucler à l'infini
13 print(liste) #non obligatoire mais permet de vérifier
14
15 for x in liste:
16     if x==cible:
17         compteur=compteur+1
18
19 print("Sur",nombre,"lancers, il y a eu",compteur,"fois le nombre",
20       cible)
21
22 #niveau 2
23 def simulateur(n,cible): #on crée une fonction pour que l'utilisateur rentre les paramètres
24     liste=[]
25     compteur=0
26     for i in range(n): #on remplit une liste de dés
27         liste.append(randint(1,6))
28     for x in liste : #on parcourt la liste et on compte
29         if x==cible:
30             compteur+=1
31     print(liste) #affichage de la liste si on veut vérifier à la main
32     print("Il y a eu {} fois le chiffre {} sur {} lancers.".format(
33         compteur,cible,n))
34     """dans ce dernier print, petite nouveauté avec la fonction
35     format()
36     Cette dernière permet une plus grande souplesse
37     """
```

2nd - simuler des pièces

```
1 from random import *
2
3 def piece(n,valeur):
4     """lance n pièce et compte la face indiquée (0 pour pile ,
5     1 pour face """
6     liste=[]
7     compteur = 0
8     if valeur==0:
9         mot="pile "
10    else:
11        mot="face"
12    for i in range (n):
13        liste.append(randint(0,1))
14    for x in liste : #on parcourt la liste et on compte
15        if x==valeur:
16            compteur+=1
17    print(liste)
18    print("Il y a eu {} {} sur un total de {} lancers soit {}%".
19    format(compteur , mot , n , round((compteur/n)*100,1)))
```

Term - simuler des pièces

```
1 from random import *
2
3 def piece(n):
4     """lance n pièce et affiche la sortie """
5     liste=[randint(0,1) for x in range(n)]
6     compteur = [x for x in liste if x == 0]
7     print(liste)
8     print("Pile > 0 et face > 1")
9     print(len(compteur) ,"pile(s) et" ,n-len(compteur) ,"face(s).")
```

Générer une suite

```
1 def suite(u_0,r,n):
2     """génère une suite arithmétique"""
3     U=[]
4     somme=0
5     for i in range (n):
6         U.append(u_0+r*i)
7         print(U)
8         somme=somme+U[i]
9     print("La somme des {} premiers termes vaut {}".format(n,somme))
```

```

1  def carre(c):
2      return (c*c)
3  def rectangle(l,L):
4      return (L*L)
5  def cercle(R):
6      return (R*R*3.14)
7  def triangle(b,h):
8      return (0.5*b*h)
9
10
11
12 def surface():
13     type=0
14     valeurs_autorisees=[1,2,3,4]
15     print("1 pour triangle")
16     print("2 pour carré")
17     print("3 pour cercle")
18     print("4 pour rectangle")
19     while type not in valeurs_autorisees:
20         type=int(input("Choix : "))
21     if type==1:
22         base=int(input("Base ? "))
23         hauteur=int(input("Hauteur? "))
24         surface=base*hauteur*0.5
25     elif type==2:
26         cote=int(input("Côté ? "))
27         surface=cote**2
28     elif type==3:
29         rayon=int(input("Rayon ? "))
30         surface=rayon**2*3.14159
31     else:
32         longueur=int(input("Longueur ? "))
33         largeur=int(input("Largeur ? "))
34         surface=longueur*largeur
35     unite=input("unité (ex: m2) : ")
36     print("S={}\{}".format(surface , unite))

```

```

1  from math import *
2  def vecteur(xa, ya, xb, yb):
3      """ calcule la norme et les coordonnées de vecAB """
4      coo_x = xb-xa
5      coo_y = yb-ya
6      norme=round(sqrt(coo_x**2+coo_y**2),2)
7      print("x_AB = ", coo_x)
8      print("y_AB = ", coo_y)
9      print(" ||AB|| = ", norme)
10
11
12 def vec():
13     print("Coord/Norme de vec(AB)")
14     xa=int(input("Abscisse du point A : "))
15     ya=int(input("Ordonnée du point A : "))
16     xb=int(input("Abscisse du point B : "))
17     yb=int(input("Ordonnée du point B : "))
18     coo_x = xb-xa
19     coo_y = yb-ya
20     norme=round(sqrt(coo_x**2+coo_y**2),2)
21     print("x_AB={} ".format(coo_x))
22     print("y_AB={} ".format(coo_y))
23     print("Norme ||AB|| = {}".format(norme))

```

```

1  def suitemax(v0,q,valeurmax):
2      """Cherche le rang à partir duquel les termes
3      de la suite sont supérieurs à valeurmax"""
4      suite=[v0*q**n for n in range(1000) if v0*q**n<=valeurmax]
5      print(suite)
6      print("""Les termes de la suite seront supérieurs à {}
7  si n est plus grand que {}.""".format(valeurmax,len(suite)+1))
8
9
10
11 def suitemax2(u0,r,valeurmax):
12     """Cherche le rang à partir duquel les termes
13     de la suite sont supérieurs à valeurmax"""
14     suite=[]
15     i=0
16     #génération de la suite
17     while i<=100: #valeur fixée pour le nombre de termes à générer
18         suite.append(u0+i*r)
19         i+=1
20     #création de la seconde suite
21     suite2=[]
22     j=0
23     while suite[j]<=valeurmax:
24         suite2.append(suite[j]) #on remplit la liste 2
25         j+=1
26     print("""Les termes de la suite seront supérieur à {}
27 pour n plus grand que {}.""".format(valeurmax,len(suite2)))

```

```

1  """Générateur d'équations"""
2  from random import *
3  def equations(degre , nombre):
4      x="x "
5      for n in range (nombre):
6          a=randint(-10,10)
7          b=randint(-10,10)
8          c=randint(-10,10)
9          d=randint(-10,10)
10
11         if degre==1:
12             while a==0 or a==d:
13                 a=randint(-10,10)
14             while b==0 or b==c:
15                 b=randint(-10,10)
16             while c==0:
17                 c=randint(-10,10)
18             while d==0:
19                 d=randint(-10,10)
20             if b<0 and c>0:
21                 print("{}x{}={}+{}x".format(a,b,d,c))
22             elif b<0 and c<0:
23                 print("{}x{}={}{}x".format(a,b,d,c))
24             elif b>0 and c<0:
25                 print("{}x+{}={}{}x".format(a,b,d,c))
26             else:
27                 print("{}x+{}={}+{}x".format(a,b,d,c))
28         else:
29             while a==0:
30                 a=randint(-10,10)
31             if b<0 and c<0:
32                 print("{}x^2{}x{}=0".format(a,b,c))
33             elif b<0 and c>0:
34                 print("{}x^2{}x+{}=0".format(a,b,c))
35             elif b>0 and c<0:
36                 print("{}x^2+{}x{}=0".format(a,b,c))
37             else:
38                 print("{}x^2+{}x+{}=0".format(a,b,c))

```

```

1  def expressionfonction(n):
2      """écrit l'expression d'une fonction"""
3      a=input("Fonction {} : valeur devant x^3 (vide si zéro) : ".
4      format(n))
5      b=input("Fonction {} : valeur devant x^2 (vide si zéro) : ".
6      format(n))
7      c=input("Fonction {} : valeur devant x (vide si zéro) : ".format(
8      n))
9      d=input("Fonction {} : valeur sans x (vide si zéro) : ".format(n)
10     )
11     liste=[a,b,c,d]
12     liste2=[float(x) if x!=" " else 0 for x in liste]
13     return (liste2[0],liste2[1],liste2[2],liste2[3])
14
15 def encadrement(n, borneinf=-10, bornesup=10, detail=0):
16     """Détermine une valeur approchée à n près par balayage
17     de f(x)=g(x) sur l'intervalle [borneinf;bornesup]
18     par défaut -10,10
19     mettre une valeur pour détail pour afficher tous les calculs"""
20
21     a1,b1,c1,d1=expressionfonction("f")
22     a2,b2,c2,d2=expressionfonction("g")
23     x=borneinf
24     f_x=-100
25     g_x=100
26     while f_x != g_x:
27         f_x=round(a1*x**3+b1*x**2+c1*x+d1,3)
28         g_x=round(a2*x**3+b2*x**2+c2*x+d2,3)
29         if detail!=0:
30             print(f_x,g_x,x)
31
32         x=round(x+n,2)
33
34         if x>=bornesup:
35             print("""Je n'ai pas trouvé d'égalité entre f(x) et g(x).
36             Il faut peut être changer le pas ou l'intervalle""")
37             break
38         print("J'ai trouvé f(x)=g(x) pour x={} ({}={})".format(x-n,f_x,
39         g_x))

```

Pyramide

```
1 somme = 0
2 etages=[11,9,7,5,3,1]
3 for i in etages:
4     nombre_pour_etage = i**3
5     somme = somme + nombre_pour_etage
6     print("Pour l'étage {} il faut {} cubes.".format(i,
7         nombre_pour_etage))
8 print("Il faut au total {} petits cubes.".format(somme))
```



```

1  """Calculateur de surfaces"""
2  def carre(c):
3      return (c*c)
4  def rectangle(longueur , largeur):
5      return(longueur*largeur)
6  def cercle(rayon):
7      return(rayon**2*3.14)
8  def triangle(base , hauteur):
9      return(base*hauteur*0.5)
10
11 def surface(figure , dimension_1 , dimension_2=0 , unite="m2") :
12     """Calcule la surface d'une figure"""
13     liste=["rectangle","cercle","carre","triangle"]
14     liste2=["cercle","carre"]
15     if figure not in liste:
16         print("Je ne connais pas cette figure , attention aux
17 guillemets")
18     else:
19         if dimension_2==0 and figure not in liste2:
20             print("erreur , il me manque une valeur")
21         else:
22             if figure == "rectangle":
23                 s=rectangle(dimension_1 , dimension_2)
24             elif figure == "carre":
25                 s=carre(dimension_1)
26             elif figure=="cercle":
27                 s=cercle(dimension_1)
28             elif figure=="triangle":
29                 s=triangle(dimension_1 , dimension_2)
30             print("La surface est {}".format(s , unite))
31
32 def volume(figure , d1 , d2 , d3 , unite="m3") :
33     if figure=="rectangle":
34         s=rectangle(d1 , d2)
35         v=s*d3
36         print(v , unite)

```