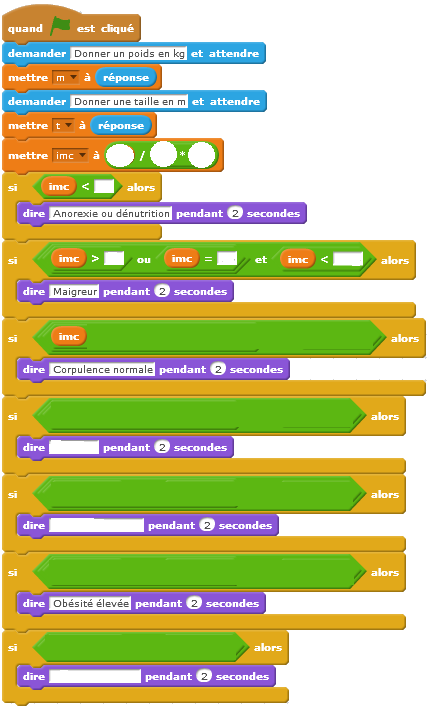
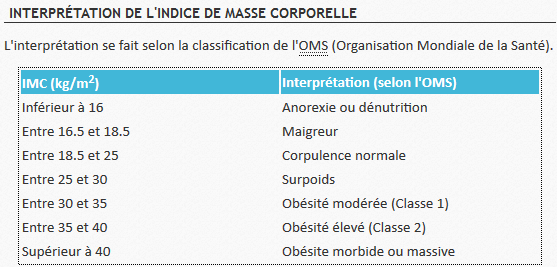
**EXERCICE 1 : IMC**

l’IMC : créer un script qui demande le poids en kg et la taille en m d’un adulte et qui lui renvoie son IMC avec l’interprétation selon l’OMS.



OBJECTIF : Découvrir l’algorithmique en langage naturel, programmer une boucle if… elif.

1°) Travail avec Scratch

Compléter l’algorithme ci-contre pour qu’il réponde au problème posé :

Réaliser ou vérifier votre travail en travaillant dans Scratch.

2°) Langage naturel : En utilisant le tableau de correspondance ci-dessous, on souhaite traduire en langage naturel le script précédent.

|  |  |
| --- | --- |
| Instruction Scratch | Langage naturel |
|  | 2 x (3 – 4) |
|  | (2 + 3) / 4 = |
|  | a b |
|  | Afficher le message : « Donner………. »  Lire a  Il peut être intéressant de préciser, en début d’algorithme le type de la donnée a utilisée : nombre (entier ou quelconque), chaine de caractère , liste …) |
|  | Début algorithme  Il sera pratique de noter fin algorithme lorsque celui-ci doit s’arrêter. |
|  | Si « condition(s) » alors « instructions »  Si le bloc est complété par un « **sinon** » il suffit de rajouter dans la traduction en langage naturel : **alors**  Afin de « dessiner » l’intégralité du bloc, on pourra ajouter **fin si** lorsque la boucle est terminée. |
|  | Afficher « phrase » |
|  | Affecter 0 à a |

Compléter l’algorithme en langage naturel proposé, afin qu’il réponde à la question posée :

Début algorithme :

Afficher : « Donner un poids en kg : »

Lire m

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

Affecter ……………………………………. à imc

Si imc 16

alors afficher : « Anorexie ou dénutrition »

Fin Si

Si …………………………………………………………………………………………..

……………………………………………………………………………………………………….

Fin Si

Si ……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

Si ……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

……………………………………………………………………………………………………….

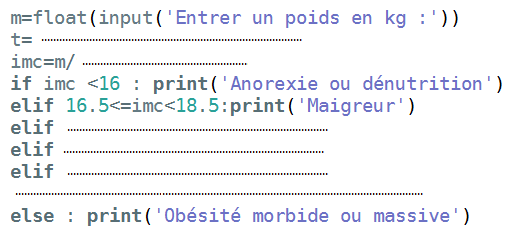
Fin algorithme

3°) Langage Python

Reprendre le travail précédent en programmant l’algorithme avec Python, pour cela on pourra utiliser le tableau de correspondance suivant :

|  |  |
| --- | --- |
| Langage naturel | Python |
| 2 x (3 – 4) |  |
| 23 |  |
| a ≤ b |  |
| Afficher le message : « Donner………. »  Lire a |  |
| Si « condition(s) » alors « instruction(s) » |  |
| Afficher « phrase » |  |
| Affecter 0 à a |  |

Compléter le programme en python ci-dessous afin qu’il réponde au problème posé. On pourra implanter le programme dans l’éditeur afin de vérifier qu’il fonctionne correctement.



[Fichiers imc.sb2 et imc.py](https://ent2d.ac-bordeaux.fr/disciplines/mathematiques/wp-content/uploads/sites/3/2017/08/Fichiers-IMC.zip)

**EXERCICE 2 : Tirelire**

Matthieu souhaite s’acheter un smartphone qui coute 150€. Pour cela il décide d’économiser sur son argent de poche.

Chaque semaine il reçoit 15€, il en dépense les deux tiers et économise le reste.

Actuellement il a 23€ dans sa tirelire.

Établir un script qui permette de déterminer le nombre de semaines à attendre avant de pouvoir s’acheter le smartphone de ses rêves ?

OBJECTIF : Travail en langage naturel – Découvrir une boucle while

1°) Travail avec Scratch

On pourra s’aider des briques suivantes pour écrire le script Scratch correspondant.



2°) Langage naturel

Réutiliser le tableau de correspondance Scratch – langage naturel auquel on ajoute la correspondance :

|  |  |
| --- | --- |
| Scratch | Langage naturel |
|  | Pour i allant de 1 à 10  Instruction  Fin pour |
|  | Tant que condition non réalisée  Instruction  Fin tant que |

3°) Python

|  |  |
| --- | --- |
| Langage naturel | Python |
| Pour i allant de 1 à 10  Instruction  Fin pour |  |
| Tant que condition non réalisée  Instruction  Fin tant que |  |

En python il n’est pas nécessaire de noter une instruction pour signifier que la boucle est finie, c’est l’indentation (retrait par rapport au bord de l’éditeur).

En déduire le script Python correspondant au problème donné.

[Fichiers tirelire.sb2 et tirelire.py](https://ent2d.ac-bordeaux.fr/disciplines/mathematiques/wp-content/uploads/sites/3/2017/08/Fichiers-Tirelire.zip)

**EXERCICE 3 : Table de 7**

Écrire un script qui donne la table de 7 en commençant par 7 x 1 et en finissant par 7 x 10.

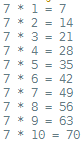
OBJECTIF : Langage naturel – Découvrir une boucle For

1°) Travail sous scratch

Réaliser un script permettant au lutin de réciter la table de 7.

2°) Langage naturel

Adapter le script précédent pour écrire un algorithme en langage naturel qui affiche la table de 7 de 7\*1 à 7\*10.

 3°) Python :

Écrire le programme en Python correspondant à l’algorithme précédent.

On souhaite que l’affichage dans la console soit :

[Fichiers table de 7.sb2 et table de 7.py](https://ent2d.ac-bordeaux.fr/disciplines/mathematiques/wp-content/uploads/sites/3/2017/08/Fichiers-table-de-7.zip)

**EXERCICE 4 : le problème du Duc de Toscane** (D’après IREM de la réunion : atelier épreuve pratique)

L’objectif est de comparer les probabilités d’obtenir 9 et 10 lorsqu’on fait la somme des numéros sortis lors du lancer de trois dés. Ce problème fait référence au paradoxe historique du Duc de Toscane dont voici une brève présentation :

Galilée (1554-1642) est surtout connu pour ses travaux en astronomie, faisant suite à son invention de la lunette astronomique. Cependant, il rédigea vers 1620 un petit mémoire sur les jeux de dés pour répondre à une demande du Duc de Toscane (Galilée est alors Premier Mathématicien de l’Université de Pise et Premier Philosophe du Grand Duc à Florence). Galilée est ainsi l’un des premiers avec Cardan à avoir écrit sur le « calcul des hasards », mais leurs écrits n’ont été publiés qu’après la célèbre correspondance entre Pascal et Fermat qui marque « officiellement » le début de la théorie des probabilités.

Le mémoire de Galilée n’a été édité qu’en 1718.

À la cour de Florence, de nombreux jeux de société étaient alors pratiqués. Parmi ceux-ci, l’un faisait intervenir la somme des numéros sortis lors du lancer de trois dés. Le Duc de Toscane, qui avait sans doute observé un grand nombre de parties de ce jeu, avait constaté que la somme 10 était obtenue légèrement plus souvent que la somme 9. Le paradoxe, que le Duc avait exposé à Galilée, réside dans le fait qu’il y a autant de façons d’écrire 10 que 9 comme sommes de trois entiers compris entre 1 et 6 :

10 = 6 + 3 + 1 = 6 + 2 + 2 = 5 + 4 + 1 = 5 + 3 + 2 = 4 + 4 + 2 = 4 + 3 + 3 (6 possibilités)

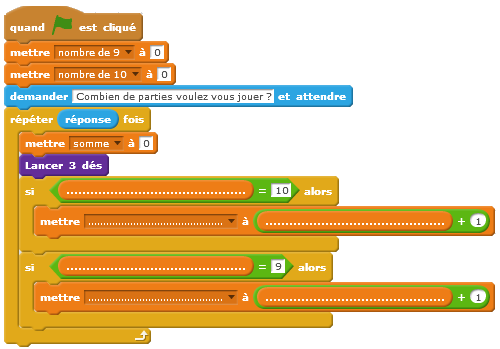
9 = 6 + 2 + 1 = 5 + 3 + 1 = 5 + 2 + 2 = 4 + 4 + 1 = 4 + 3 + 2 = 3 + 3 + 3 (6 possibilités)

Simuler un grand nombre de parties afin de vérifier ce paradoxe.

OBJECTIF : Découverte d’un problème historique – Travail en langage naturel – Découverte de la notion de fonction informatique (sans paramètre).

1°) Travail avec scratch

1. Définir un bloc qui simule le lancer de 3 dés cubiques numérotés de 1 à 6, parfaitement équilibrés et qui renvoie la somme des 3 faces.
2. Utiliser ce bloc pour simuler un nombre de parties proposé par l’utilisateur, et dénombrer le nombre de fois où 10 et 9 apparaissent. On pourra s’aider du script incomplet ci-dessous :



2°) Langage naturel

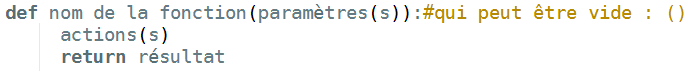
Traduire le bloc « lancer 3 dés » puis le script en entier en langage naturel.

Pour appeler le bloc on pourra écrire : **Faire : lancer 3 dés**

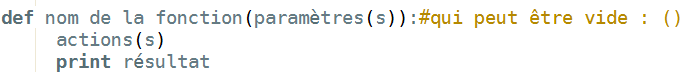
3°) Python

Pour définir un bloc on va utiliser l’outil qui nous permet de définir des **fonctions**, ici nous avons une fonction qui n’a pas de **paramètre**. En effet, dès que le bloc « lancer 3 dés » est activé le script effectue le lancer de 3 dés il n’a pas besoin d’une donnée supplémentaire pour « faire le travail ».

Pour définir une fonction en python nous utilisons :



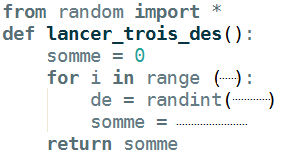
Dans ce cas, la fonction renvoie une **donnée exploitable** dans un autre script : résultat, l’appel de ce résultat se fait grâce à l’instruction **return**

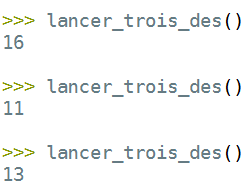


Dans ce cas, la fonction renvoie une **donnée non exploitable** dans un autre script : résultat, en effet l’instruction **print** se contente d’afficher une information, cette information n’a pas de type, elle ne peut dons pas être utilisée dans un calcul, par exemple, où le type doit être un entier (int) ou un nombre quelconque (float).

Pour simuler le lancer d’un dé on choisit de demander au programme de choisir, de manière aléatoire, un entier entre 1 et 6. Pour cela nous allons utiliser l’instruction **randint(1,6**). Cette instruction n’existe pas par défaut, il faut l’appeler, pour cela on commence notre script par : **from random import \*** (on importe toutes les instructions relatives à la bibliothèque random, donc entre autres l’instruction randint).

Compléter le script de la fonction lancer\_trois\_dés pour qu’elle réalise le même travail que le bloc lancer 3 dés.

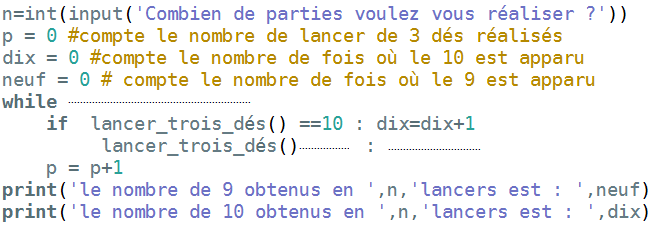


Pour contrôler la justesse de la fonction on pourra la lancer dans la console :

Finaliser le programme pour qu’il renvoie le nombre de fois où le 10 et le 9 sont apparus au cours de n parties (le n étant choisi par l’utilisateur).

**Le test « être égal à » se note : ==,** ainsi pour tester si la valeur de a vaut 2 on note :

Compléter le script ci-dessous pour que le programme demande le nombre de parties et affiche le nombre de fois ou 10 et 9 sont apparus.



4°) Explication

Conjecturer vous le même paradoxe que le Duc de Toscane ?

Pouvez-vous expliquer ce paradoxe ?

[Fichiers duc de toscane.sb2 et duc de toscane.py](https://ent2d.ac-bordeaux.fr/disciplines/mathematiques/wp-content/uploads/sites/3/2017/08/Fichiers-Duc-de-Toscane.zip)