Exercice n°4 :

Écrire une fonction somme_cube(n) qui renvoie la somme : $1^3 + 2^3 + \cdots + n^3$

>>>somme_cube(2)
9
>>>somme_cube(4)
100

Exercice n°5 : Calcul mental

a) Écrire la fonction **alea** qui prend en argument un entier n et renvoie un nombre aléatoire compris entre 1 et n (inclus). (Importer la bibliothèque « random » : en début de programme : ***from random import randint***)

b) Écrire la fonction **calcul** qui prend en argument un entier n.
Cette fonction détermine deux entiers aléatoirement choisis entre 1 et n et demande à l'utilisateur de calculer la somme de ces deux nombres.
Si la réponse est bonne le programme affiche bravo et renvoie 1, si la réponse est fausse le programme affiche la bonne réponse et renvoie 0.

c) Écrire la fonction **Serie_calcul** qui prend en argument deux entiers (*n* et *nb*).
Le programme va enchaîner *nb* questions comme celles de la question précédente, *n* étant la valeur maximum des nombres utilisés.
À la fin le programme doit afficher le nombre de bonnes réponses.
Le programme doit fonctionner comme dans l'exemple.

d) Écrire la fonction **main** pour lancer le programme précédent:
Le programme demande à l'utilisateur les valeurs de *n* et de *nb*.

e) Faire une autre version du jeu : il s'agit ici de recommencer avec le même nombre maximum, tant que le joueur n'a pas réussi une série de 5 calculs consécutifs justes.

valeur maximale des nombres : 20
nombre de calculs : 6
calculez 12 + 3 :15
bravo !
calculez 8 + 2 :10
bravo !
calculez 19 + 1 :21
c'est faux, il fallait trouver 20
calculez 12 + 10 :22
bravo !
calculez 8 + 7 :15
bravo !
calculez 7 + 20 :27
bravo !
score total pour les 6 calculs: 5

Exercice n°6 : Jeu du nombre mystérieux

Écrire un programme qui propose une partie de jeux du nombre mystérieux à l'utilisateur (le programme choisit aléatoirement un entier que l'utilisateur doit deviner).
On commencera par une version très simple puis on rajoutera des fonctionnalités (comme un comptage des coups, des indications etc....)

exemple de déroulement du jeu:
Votre proposition ? 50
Le nombre que vous devez trouver est plus grand !
Votre proposition ? 85
Le nombre que vous devez trouver est plus grand !
Votre proposition ? 100
le nombre que vous devez trouver est plus petit !
Votre proposition ? 90
Le nombre que vous devez trouver est plus grand !
Votre proposition ? 95
le nombre que vous devez trouver est plus petit !
Votre proposition ? 93
Le nombre que vous devez trouver est plus grand !
Votre proposition ? 94
Bravo, vous avez trouvé le nombre mystère en 7 coups !!!

Principe du jeu : l'ordinateur choisit un nombre entier entre 0 et 100, appelé nombre mystère. Le joueur doit trouver le plus rapidement possible quel est ce nombre, en faisant des propositions successives à l'ordinateur. L'ordinateur indique au joueur si sa proposition est plus grande ou plus petite que le nombre à trouver.

Vous décomposerez votre programme en différentes fonctions:

- la fonction **generer_nombre** choisit au hasard puis retourne un entier entre 1 et 10, le nombre mystère ;

- la fonction **demande_coup** demande à l'utilisateur d'entrer un nombre entier puis retourne sa valeur ;

- la fonction **compare_nombre** retourne 0, 1 ou -1 selon que le coup proposé est respectivement égal, inférieur ou supérieur au nombre mystère ;

- la fonction **partie** qui choisit un nombre (appel à la fonction **generer_nombre**) puis demande au joueur d'entrer son coup tant qu'il n'a pas gagné. Cette fonction retourne le nombre de coup joués, si le joueur gagne (le nombre mystère sinon).

A vous de voir quels sont les arguments à passer à ces fonctions.

Le programme principal ne devra contenir qu'une seule instruction, l'appel à la fonction partie.

Exercice n°7 : Jeu des allumettes

Il s'agit de réaliser un programme pour jouer au jeu des allumettes ou jeu de Nim.
Ce jeu très connu et dont il existe de nombreuses variantes se jouera de la manière suivante:
on dispose d'un certain nombre d'allumettes. Chacun son tour, un des deux joueurs prend 1 , 2 ou 3 allumettes selon son choix. Celui qui est obligé de prendre la dernière allumette perd la partie.

Il est demandé de faire un programme pour jouer à ce jeu avec deux joueurs “humains”.

1. Écrire la fonction **affiche(n)** : cette fonction qui prend en paramètre un entier n doit afficher sur une ligne n étoiles séparées par des espaces puis aller à la ligne.
2. Écrire la fonction **min(a,b)** qui prend en paramètre deux entiers et qui renvoie le plus petit des deux.
3. Écrire la fonction **saisie(a,b)** qui prend en paramètre deux entiers , demande à l’utilisateur de saisir un entier entre a et b (compris) et recommence tant que la réponse n’est pas satisfaisante. Si la réponse est satisfaisante, l’entier est renvoyé.
   On supposera que l'utilisateur saisie effectivement un entier et on ne testera que le fait qu'il soit compris entre a et b.
4. On fera ensuite le programme principal. On jouera avec 21 allumettes au départ.
   Le programme devra afficher le joueur dont c’est le tour, l’état du jeu, etc.. comme sur l’exemple ci-dessous.
   On appellera les joueurs joueur1 et joueur2.
   A tout moment un joueur doit choisir comme nombre d’allumettes un nombre entre 1 et le minimum de *nb*-1 et de 3 si *nb* est le nombre d’allumettes restantes.

Exemple de jeu :

```
**********************
joueur 1 combien d'allumettes, saisir un entier entre 1 et 3: 6
saisir un entier entre 1 et 3: 3
******************
joueur 2 combien d'allumettes, saisir un entier entre 1 et 3: 3
***************
joueur 1 combien d'allumettes, saisir un entier entre 1 et 3: 3
************
joueur 2 combien d'allumettes, saisir un entier entre 1 et 3: 3
*********
joueur 1 combien d'allumettes, saisir un entier entre 1 et 3 :3
******
joueur 2 combien d'allumettes, saisir un entier entre 1 et 3: 3
***
joueur 1 combien d'allumettes, saisir un entier entre 1 et 3: 1
**
joueur 2 combien d'allumettes, saisir un entier entre 1 et 2: 3
mauvaise saisie , saisir un entier entre 1 et 2: 1
*
victoire du joueur 2
```

Bonus: faire une version où un joueur humain joue contre l'ordinateur.