Exercice 1 : utilisation de la commande print

Affectez les variables temps et distance par les valeurs 6.892 et 19.7.
Calculez et affichez la valeur de la vitesse. V= ?
Améliorez l'affichage en formatant avec un chiffre après le point décimal.

```
vitesse = 2.8583865351131745
----------------------------

 vitesse = 2.86 m/s
>>>
```

Exercice 2 : saisie au clavier (input)

Saisissez un flottant. S'il est positif ou nul, affichez sa racine, sinon affichez un message d'erreur.

```
donnez x ? 24.6
La racine de 24.60 est 4.960

 A bientôt
>>>
```

Exercice 3 : input()

On désire sécuriser une enceinte pressurisée.
On se fixe une pression seuil et un volume seuil : **pSeuil = 2.3, vSeuil = 7.41**.
On demande de saisir la pression et le volume courant de l'enceinte et d'écrire un script qui simule le comportement suivant :
– si le volume *et* la pression sont supérieurs aux seuils : arrêt immédiat ;
– si seule la pression est supérieure à la pression seuil : demander d'augmenter le volume de l'enceinte ;
– si seul le volume est supérieur au volume seuil : demander de diminuer le volume de l'enceinte ;
– sinon déclarer que « tout va bien ».
Ce comportement sera implémenté par une alternative multiple.

```
Pression courante = 5
Volume courant = 8
     pression ET volume elevés. ALARME Stoppez !
>>>
```

```
Pression courante = 3
Volume courant = 5
     Il faut augmenter le volume
```

```
Pression courante = 2
Volume courant = 6
     Tout va bien !
```

```
Seuil pression : 2.3    Seuil volume ; 7.41

Pression courante = 2
Volume courant = 8
     Vous pouvez diminuer le volume
```

Ex 3bis :Un gardien de phare va aux toilettes cinq fois par jour or les WC sont au rez-de-chaussée.
. .
Écrire une procédure (donc sans retour) ***hauteurparcourue*** qui reçoit deux paramètres le nombre de marches du phare et la hauteur de chaque marche (en cm), et qui affiche :

`Pour x marches de y cm, il parcourt z.zz m par semaine.`

On n'oubliera pas :
– qu'une semaine comporte 7 jours ;
– qu'une fois en bas, le gardien doit remonter ;
– que le résultat est à exprimer en m.

```
Combien de marches ? 100
Hauteur d'une marche (cm) ? 20
100 marches de 20 cm. Il parcourt 1400.00 m par semaine.
>>>
```

Exercice 4 : range()

Affichez les entiers de 0 à 15 non compris, de trois en trois, en utilisant une boucle for et l'instruction range().

```
--------- Instruction <range> ----------


-------- <range> dans un <for> ---------

0 3 6 9 12
>>>
```

Exercice 5 : range()

Utilisez l'instruction break pour interrompre une boucle for d'affichage des entiers de 1 à 10 compris, lorsque la variable de boucle vaut 5.

```
1 2 3 4 arrêt

>>>
```

5-1. Utilisez l'instruction continue pour modifier une boucle for d'affichage de tous entiers de 1 à 10 compris, sauf lorsque la variable de boucle vaut 5.

```
1 2 3 4 6 7 8 9 10
>>>
```

5-2 . Utilisez une *exception* pour calculer, dans une boucle évoluant de -3 à 3 compris, la valeur de $\sin(x)/x$.

```
0.047 0.455 0.841 1.000 0.841 0.455 0.047
>>>
```

Calculer la factorielle d'un nombre

Exercice 6 : fonction()

Écrire une fonction cube qui retourne le cube de son argument.

Écrire une fonction volumeSphere qui calcule le volume d'une sphère de rayon r fourni en argument et qui utilise la fonction cube.
Tester la fonction volumeSphere par un appel dans le programme principal.

```
Rayon : 5

Volume de la sphere de rayon 5.0 : 523.599
```

Exercice 7 : listes

Définir la liste : liste =[17, 38, 10, 25, 72], puis effectuez les actions suivantes :

- triez et affichez la liste ;
- ajoutez l'élément 12 à la liste et affichez la liste ;
- inversez et affichez la liste ;
- affichez l'indice de l'élément 17 ;
- enlevez l'élément 38 et affichez la liste ;
- affichez la sous-liste du 2ème au 3ème élément ;
- affichez la sous-liste du début au 2ème élément ;
- affichez la sous-liste du 3ème élément à la fin de la liste ;
- affichez la sous-liste complète de la liste ;
- affichez le dernier élément en utilisant un indiçage négatif.

Bien remarquer que certaines méthodes de liste ne retournent rien.

```
---------- Liste initiale ----------
[17, 38, 10, 25, 72]

---------- Tri ----------
[10, 17, 25, 38, 72]

---------- Ajout d'un element ----------
[10, 17, 25, 38, 72, 12]

---------- Retournement ----------
[12, 72, 38, 25, 17, 10]

---------- Indice d'un element ----------
4

---------- Retrait d'un element ----------
[12, 72, 25, 17, 10]
```

```
---------- Indicage ----------
nombres[1:3] = [72, 25]
nombres[:2] = [12, 72]
nombres[2:] = [25, 17, 10]
nombres[:] = [12, 72, 25, 17, 10]
nombres[-1] = 10
>>>
```

Afficher les nombres impairs dans une liste

Exercice 8 : module et import

Nombres parfaits et nombres chanceux.

- On appelle nombre premier tout entier naturel supérieur à 1 qui possède exactement deux diviseurs, lui-même et l'unité.
- On appelle diviseur propre de n, un diviseur quelconque de n, n exclu.
- Un entier naturel est dit parfait s'il est égal à la somme de tous ses diviseurs propres.
- Un entier n

tel que : $(n+i+i2)$ est premier pour tout i dans $[0,n-2]$

- est dit chanceux.

Écrire un module (parfait_chanceux_m.py) définissant quatre fonctions : somDiv, estParfait, estPremier, estChanceux et un autotest.

- La fonction somDiv retourne la somme des diviseurs propres de son argument.
- Les trois autres fonctions vérifient la propriété donnée par leur définition et retourne un booléen. Si par exemple la fonction estPremier vérifie que son argument est premier, elle retourne True, sinon elle retourne False.

La partie de test doit comporter quatre appels à la fonction verif permettant de tester somDiv(12), estParfait(6), estPremier(31) et estChanceux(11).

```
Somme des diviseurs propres de 12 :  16
6 est-il parfait :  True
31 est-il premier :  True
11 est-il chanceux :  True
```

Exercice 9 :

Puis écrire le programme principal (parfait_chanceux.py) qui comporte :

- l'initialisation de deux listes : parfaits et chanceux ;
- une boucle de parcours de l'intervalle [2,1000]

- incluant les tests nécessaires pour remplir ces listes ;
- enfin l'affichage de ces listes.

```
---------- Nombres remarquables dans [2 .. 1000] ----------

Parfaits :  [6, 28, 496]

Chanceux :  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449,
457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727,
733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
>>>
```

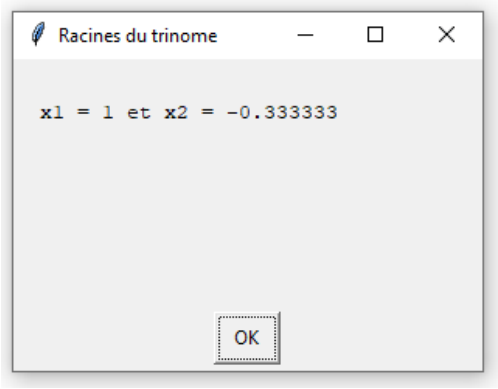
Exercice 10 : modules

1. Écrire un module de calcul des racines du trinôme réel : *ax2 + bx + c*.
Le module définit une fonction trinôme avec les trois paramètres du trinôme, *a*, *b* et *c*. La fonction doit retourner un tuple dont le premier élément est le nombre de racines du trinôme (0, 1 ou 2), et les autres éléments sont les racines éventuelles.
Testez votre fonction avec les trois jeux de valeurs suivantes : 1,/3,2, 1,/2,1 et 1,1,1.

```
(2, 1.0, 2.0)
(1, 1.0)
(0,)
```

2. Écrire un programme principal utilisant le module précédent.
Les trois paramètres seront saisis et les résultats seront affichés dans une msgbox du module easygui.

a= 1, b= -3, c= 2



Exercice 11 : orienté objet

1. Définir une classe MaClasse possédant les attributs suivants :
données : deux attributs de classes : x = 23 et y = x + 5.
méthode : une méthode affiche contenant un attribut d'instance z = 42 et les affichages de y et de z.
Dans le programme principal, instanciez un objet de la classe MaClasse et invoquez la méthode affiche.

2. Définir une classe Vecteur2D avec un constructeur fournissant les coordonnées par défaut d'un vecteur du plan (par exemple : x = 0 et y = 0).
Dans le programme principal, instanciez un Vecteur2D sans paramètre, un Vecteur2D avec ses deux paramètres, et affichez-les.

3. Enrichissez la classe Vecteur2D précédente en lui ajoutant une méthode d'affichage et une méthode de surcharge d'addition de deux vecteurs du plan.
Dans le programme principal, instanciez deux Vecteur2D, affichez-les et affichez leur somme.

Ex12 Conversion décimale binaire

- l'opérateur **//** donne le quotient de la division euclidienne : **5 // 2** donne **2**;
- l'opérateur **%** donne le reste de la division euclidienne : **5 % 2** donne **1** ;
- **append** est une méthode qui ajoute un élément à une liste existante.
  Soit **T=[5,2,4]**, alors **T.append(10)** ajoute **10** à la liste **T**. Ainsi, **T** devient **[5,2,4,10]**
- **reverse** est une méthode qui renverse les éléments d'une liste.
  Soit **T=[5,2,4,10]**. Après **T.reverse()**, la liste devient **[10,4,2,5]**

On remarquera qu'on récupère la représentation binaire d'un entier **n** en partant de la gauche en appliquant successivement les instructions :
b = n % 2
n = n // 2