1.- Escribe una función en Python de nombre **ecuación_grado_2** que reciba como argumento tres números reales: `a, b` y `c` ($a \neq 0$) y devuelva una lista que contiene las raíces de la ecuación de segundo grado:

$$ax^2 + bx + c = 0$$

Nota: Es necesario discriminar entre todos los casos posibles.

- Si la ecuación tiene dos raíces reales devolverá una lista con dos elementos ordenados de menor a mayor.
- Si la ecuación tiene una raíz devolverá una lista con un único elemento.
- Si la ecuación tiene dos raíces complejas conjugadas devolverá una lista con dos elementos. Cada elemento será a su vez una lista de dos elementos: parte real y parte imaginaria ordenados de manera que el primer elemento de la lista sea el que tenga parte imaginaria positiva.

2.- Escribe una función en Python de nombre **funcion_trigonometrica** que reciba como argumento dos números reales a y b y un número entero N. La función generará una partición de N+1 puntos en el intervalo $[a, b]$ con $x_0 = a$ y $x_N = b$.

Devolverá una lista con los valores de la función $f(x) = \begin{cases} \cos(\pi x) & x \leq 0 \\ 1 + sen(\pi x) & x > 0 \end{cases}$ en los puntos de la partición.

3.- Escribe una función en Python de nombre **primos** que reciba un número natural N y devuelva una lista con los números primos menores o iguales que el dado.

Nota: Para $N < 2$ devolverá una lista vacía.

4.- Escribe una función de nombre **palabras** que recibe dos cadenas de texto y devuelve una lista con las letras que aparecen en ambas cadenas.

Por ejemplo:

- Para 'melón' y 'tomate' devolverá ['m', 'e'] (Distingue entre vocales con o sin tilde).
- Para 'Violín' y 'lava' devolverá ['v', 'l'] (No distingue entre mayúsculas y minúsculas).

5.- Escribe una función de nombre **letra_en_palabra** que recibe una letra y una cadena de texto. Devolverá un entero con el numero de veces que aparece la letra en la cadena y una lista con las posiciones en las que aparece. (El computo de las posiciones se hará considerando que la primera letra de la cadena está en la posición 1. Python empieza en 0)

Ejemplo:

- Letra_en_palabra('a', 'La casa está cara') → 5 , [2,5,7,15,17] (Distingue tildes)
- Letra_en_palabra('n', 'No nada') → 2 , [1,4] (No distingue entre mayúsculas y minúsculas)

6.- La expresión general de un polinomio $p(x)$ de grado N es:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N$$

Se pide crear una función llamada **eval_poly(x, coeffs)** que devuelva el valor del polinomio evaluado en el punto *x*.
Los coeficientes a_i del polinomio se dan a la función a través del argumento de entrada *coeffs*, que los almacena en una lista de la siguiente manera *coeffs* $= [a_0, a_1, \ldots, a_N]$.
La función debe ser válida para un número arbitrario de coeficientes y el argumento de entrada *x* es un valor real.

Ejemplo:

- Si coeffs = [1, 2, 3]; x = 2 → 1 + 2 * 2 + 3 * 2 ** 2 = 17.

7.- Escribe una función en Python de nombre **lista_diff** que recibe como argumento dos listas y devuelve una lista con los elementos que están en, solo, una de las dos listas.

Ejemplo:

- lista1 = [1,3,5,7,9]; lista2 = [1,2,3,4,5] → [2,4,7,9]
- lista1 = [‘pera’, ‘pulpo’, ‘pasta’]; lista2 = [‘manzana’, ‘pera’] → [‘pulpo’,’pasta’,’manzana’]

8.- Escribe una función de nombre **quitar_elem_pos**(lista, pos) que recibe dos argumentos:

- *lista* = lista de números
- *pos* = número entero

Devolverá la lista ordenada de menor a mayor eliminando el elemento que ocupa la posición *pos* (orden natural) en ese orden. Si la posición pedida no existe devolverá la lista ordenada sin eliminar ningún elemento.

Ejemplo:

- lista = [2.3, 1.4, 9.1, -2.2, 7.0, 43.2] pos = 3 → [-2.2, 1.4, 7.0, 9.1, 43.2]
- lista = [3, 4, 2, -1] pos =5 → [-1, 2, 3, 4]

9.-Escribe una función de nombre **perfectos** que recibe un numero entero N y devuelve una lista con los primeros N números perfectos.
Nota: Un numero es perfecto si es la suma de todos sus divisores sin contar el mismo. Ej 6 es un número perfecto 6 = 1 + 2 + 3; 28 es un número perfecto 28 = 1 + 2 + 4 + 7 + 14.

Ejemplo:

- **perfectos**(3) = [6, 28, 496]

10.- Escribe dos funciones en Python:

- La primera de nombre **divisores** recibe un numero natural N y devuelve una lista con sus divisores.
  - Ejemplo divisores(26) = [1, 2, 13, 26]
- La segunda de nombre **mcd** recibe un número indeterminado de números naturales y, mediante llamadas a la función **divisores**, calcula el máximo común divisor de los números introducidos.

Ejemplo:

- **mcd**(24, 30, 72) = 6
- **mcd**(180, 30, 45, 60) = 15

11.- Escribe una función en Python de nombre **particion** `(a, b, N)` que reciba como argumento dos variables reales `a,b`, que representan los extremos de un intervalo, y otra variable entera `N`. Devolverá una lista `[x]` que contiene una partición equiespaciada del intervalo `[a,b]` con `N+1` puntos de manera que: $x_0 = a;\ x_N = b$

12.- Escribe una función en Python de nombre **gaussian** `(x, m = 0, s = 1)` que reciba como argumento una variable real `x`, otra variable real `m` cuyo valor por defecto será 0 y otra variable real `s` cuyo valor por defecto será 1. Devolverá el resultado de la operación:

$$f(x) = \frac{1}{\sqrt{2\pi}s} exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]$$

En el programa principal, se llamará a la función **partición** (definida en el ejercicio 11) para, a partir de un número natural N, generar una lista `x[]` con $x_0 = m - 5s$ y $x_N = m + 5s$. A continuación usará la función **gaussian** para generar una lista `y[]` con las imágenes en los puntos de la lista `x[]`.

13.- Escribe una función en Python de nombre **Taylor_seno** `(x, tol)` que reciba como argumento una variable real `x`, otra variable real `tol`. Devolverá la aproximación del valor del seno en `x`, usando la aproximación mediante el polinomio de Maclaurin.

$$seno(x) \sim T(x) = \sum_{n=1}^{M} \frac{(-1)^{n+1} x^{2n-1}}{(2n-1)!}$$

El número de términos en el sumatorio vendrá determinado por la variable `tol` de manera que se cumpla: $|seno(x) - T(x)| < tol$

14.- Escribe una función en Python de nombre **Taylor_coseno** `(x, tol)` que reciba como argumento dos variables reales `x`, `tol`. Devolverá la aproximación del valor del coseno en `x`, usando la aproximación mediante el polinomio de Maclaurin.

$$coseno(x) \sim T(x) = \sum_{n=0}^{M} \frac{(-1)^{n} x^{2n}}{(2n)!}$$

El número de términos en el sumatorio vendrá determinado por la variable `tol` de manera que se cumpla: $|coseno(x) - T(x)| < tol$

15.- Una partícula se mueve en el plano describiendo un movimiento circular de radio `R` centrado en `(x,y)=(0,0)`. Escribir una función en Python de nombre **posicion** `(R, t)` que recibe el radio de giro `R` y el instante temporal `t` como variables reales y devuelve la posición: `[x(t), y(t)]` como una tupla real usando la fórmula:

$$f: \mathbb{R} \longrightarrow \mathbb{R}^2$$

$$\vec{r}(t): t \longrightarrow (Rcos(t), Rsen(t))$$

16.- Escribir una función en Python **distancia** `(x, y)`, que recibe dos tuplas: [x] e [y] que almacenan la posición en $\mathbb{R}^3$de 2 puntos. Devolverá la distancia euclídea entre ambos, usando la fórmula: (no se puede usar la función *math.dist*)

$$distancia = \sqrt{\sum_{i=1}^{3}(x_i - y_i)^2}$$

17.- Escribir una función en Python **posicion_relativa** `(x, y, R)`, que recibe dos tuplas: [x] e [y] que almacenan la posición en $\mathbb{R}^3$de 2 puntos y un número real `R`. Devolverá una variable lógica que tomará el valor True si el punto `x` se encuentre en el interior o superficie de la esfera de centro `y` y radio `R`.

18.- Crear un módulo de nombre funciones.py que contendrá al menos 5 funciones reales de variable real. Cada una de ellas recibirá un argumento real `x` y devolverá un número real.

19.- Escribir una función en Python de nombre **derivada** `(x,h,funcion)`, que recibe como argumentos dos valores reales `x,h` y el nombre de una función real de variable real `funcion`. Devolverá el valor de la derivada de `funcion` en `x` de acuerdo con la fórmula:

$$f'(x) \approx \frac{funcion(x+h) - funcion(x)}{h}$$

20.- Juntar los dos ejercicios anteriores en un único código con dos ficheros. El ejercicio 19 forma parte de un fichero principal.py que importa el módulo funciones.py donde están definidas las funciones del ejercicio 18. En el programa principal se llamara a la función **derivada** `(x,h,funcion)` para calcular la derivada de la función deseada en el punto indicado.

21.- Completar el ejercicio anterior con la función **particion** del ejercicio 11 para evaluar la función derivada en una lista de `N+1` puntos equiespaciados dentro de un intervalo definido por el usuario.

22.- Escribe una función de nombre **divide_lista**(lista) que recibe como argumento una lista de números naturales. Devolverá 3 listas: una con los números pares de la lista original; otra con los números impares y una última con los números primos. Estas listas deben estar ordenadas de menor a mayor.

23.- Escribe una función de nombre collatz(N) que reciba como argumento un numero natural N y devuelva una lista con la sucesión $\{a_1, a_2, \dots, a_i, \dots\}$ generada a partir de $a_1 = N$ usando la fórmula:

$$a_{i+1} = f(a_i);$$

Con:

$$f(x) = \begin{cases} \dfrac{x}{2} & \text{si x es par} \\ 3x+1 & \text{si x es impar} \end{cases}$$

Devolverá la lista cuando se alcance $a_i = 1$
Ej collatz(7) = [7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]