EJERCICIOS PYTHON NIVEL 3 - FUNCIONES

AÑO BISIESTO

Tu tarea es escribir y probar una función que toma un argumento (un año) y devuelve *True* si el año es un año bisiesto, o *False* si no lo es. Puedes y debes utilizar el ejercicio ya realizado en la primera serie.

Tu programa debe incluir la realización de las pruebas del código: utiliza dos listas, una con los datos de prueba y la otra con los resultados esperados.

Tu propio programa debe decirte si alguno de tus resultados no es válido.

```
def es_bisiesto(year):
    #
    # Escribe tu código aquí.
    #

test_data = [1900, 2000, 2016, 1987]   # Rellena esta lista con más años
test_results = [False, True, True, False]  # Rellena esta lista con los resultados esperados para esos años
#
#  codifica aquí la prueba de tu función
```

DIAS DEL MES.

Tu tarea es escribir y probar una función que toma dos argumentos (un año y un mes) y devuelve el número de días del mes de ese año. Aunque solo el mes de febrero es sensible al valor del año, tu función debería ser universal.

Haz que la función devuelva ***None*** si los argumentos no tienen sentido.

Recomendación: si utilizas una lista con los meses dentro de la función, se acortará significativamente el código.

Tienes que incluir un código de prueba.

```
def es_bisiesto(year):
    #
    #

def dias_del_mes(year, month):
    #
    #

test_years = [1900, 2000, 2016, 1987]
test_months = [2, 2, 1, 11]
test_results = [28, 29, 31, 30]
#
# codifica aquí la prueba de tu función
```

NÚMERO DE DÍA

Tu tarea es escribir y probar una función que toma tres argumentos (un año, un mes y un día del mes) y devuelve el número del día dentro del año, o devuelve None si cualquiera de los argumentos no es válido.

Utilizar las funciones anteriores, ya escritas y probadas.

Agrega casos de prueba al código.

```
def es_bisiesto(year):
    #
    # Tu código anterior.
    #

def dias_del_mes(year, month):
    #
    # Tu código anterior.
    #

def numero_dia(year, month, day):
    #
    # Escribe tu código nuevo aquí.
    #

print(day_of_year(2000, 12, 31))
```

NÚMEROS PRIMOS

Un número natural es primo si es mayor que 1 y no tiene divisores más que 1 y sí mismo. Tu tarea es escribir una función que verifique si un número es primo o no.

La función recibe de argumento el valor a verificar y devuelve *True* si el argumento es un número primo, y *False* de lo contrario.

Salida esperada con los números primos entre 1 y 20:

2 3 5 7 11 13 17 19

TRIÁNGULO RECTÁNGULO

Tu tarea es escribir y probar una función para verificar si, dadas las longitudes de los tres lados de un triángulo, el triángulo es o no rectángulo.

Debes utilizar primeramente una función (*es_triangulo*) para comprobar si los tres argumentos pasados pueden formar un triángulo: " la suma de dos lados tiene que ser mayor que la longitud del tercer lado".

Una vez confirmado que se trata de un triángulo, utiliza el teorema de Pitágoras para comprobar que se trata de un triángulo rectángulo.

En una primera versión, lee del teclado el valor de los tres lados del triángulo.

En una segunda versión, utiliza una batería de casos de prueba.

AREA DEL TRIÁNGULO

Tu tarea es escribir y probar una función para, dadas las longitudes de los tres lados de un triángulo, aplicar la fórmula de Herón para calcular el área del triángulo.

$$Área = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

donde **s** es el semiperímetro del triángulo:

$$s = \frac{a+b+c}{2}.$$

Utiliza primero la función *es_triangulo* para comprobar si se trata de un triángulo.

```
def es_triangulo(lado1, lado2, lado3):
  #
  # Tu código anterior.
  #

def heron(lado1, lado2, lado3):
  #
  # Escribe tu código
  #

def area_triangulo(lado1, lado2, lado3):
  #
  # Escribe tu código aquí.
  #

print(area_triangulo(3, 4, 5))
```

Agrega casos de prueba al código

FACTORIAL DE UN NÚMERO (n)

Tu tarea es escribir y probar una función para obtener el factorial de un número n.

```
0! = 1 (¡Si!, es verdad)
1! = 1
2! = 1 * 2
3! = 1 * 2 * 3
4! = 1 * 2 * 3 * 4
```

En una primera versión, el valor del número n se ingresa por teclado.

En una segunda versión, utiliza una batería de casos de prueba (factorial de 1 a 10, por ejemplo)

Modifica tu función, si es necesario, para incluir recursividad. n! = n * (n-1)!