Ejercitación Módulo 2 - 3ra parte

La siguiente notebook contiene ejercitación del Módulo 2 de la Unidad Curricular **"*Programación Digital Avanzada*"** de la carrera de *Ingeniería Biomédica* de la UTEC.

La *primera parte* y la *segunda parte* de esta notebook serán utilizadas para ejercitación. Por otro lado, la *tercera parte* será utilizada para el laboratorio del módulo y **se debe entregar** resuelta en una Jupyter Notebbok (puede ser esta misma). Además debe subirse la resolución a su repositorio de PDA en github.

Profesor Adjunto: *Mag. Bioing. Baldezzari Lucas*

V2022

Tercera parte

Esta parte corresponde a ejercicios obligatorios que deben ser entregados.

Se trabajará sobre estos problemas en la clase de Laboratorio.

Ejercicio 3.1

Implemente una función que reciba como parámetro una muestra de números (enteros o flotantes) y retorne una lista con los valores *outliers* o *atípicos*. En caso de que no existan valores atípicos, la función debe retornar None.

Los outliers se calculan de la siguiente manera.

$$outlier < Q_1 - 1.5RIC \vee outlier > Q_3 + 1.5RIC$$

Donde RIC representa el *Rango Intercuartil* y esta dado por $RIC = Q_3 - Q_1$.

In [1]:
```
### Resolución ejercicio 3.1
##TO DO
```

Ejercicio 3.2

La función *sum()* de Python permite sumar una secuencia de números.

Su implementación es,

```
sum(iterable, start)
```

Como se ve, solo recibe dos parámetros. Si hacemos,

```
sum([1,2,3]) #obtenemos 6 de resultado.
```

Pero si quisieramos hacer *sum(1,2,3,4)* obtendríamos un error.

Se pide que implemente una función suma de tal manera que pueda recibir cualquier cantidad de parámetros (1,2,...,n) y retorne la suma de los mismos. Los parámetros solo pueden ser números enteros o flotantes separados por coma.

In [2]:
```
### Resolución ejercicio 3.2
### TO DO
```

Ejercicio 3.3

Implemente una función que devuelva el índice de masa corporal. La función recibe como parámetros una lista con las alturas en cm y una lista con los pesos en kg de una cierta cantidad de personas.

Implemente la solución utilizando comprensión de listas e/o intente dar una solución usando funciones lambdas con la función map() de Python.

In [3]:
```
### Resolución ejercicio 3.3
### TO DO
```

Ejercicio 3.4

Implemente una función llamada *mathOperations* que reciba como parámetros,

- Un string con la operación a realizar, "sumar" para sumar, "restar" para restar, "dividir" para dividir y "multiplicar" para multiplicar.
- Y dos variables numéricas que se usarán para realizar la operación matemática seleccionada.

La función debe devolver un resultado dependiendo de la operación seleccionada.

La implementación de la función debe contener anotaciones y documentación apropiada.

In [4]:
```
### Resolución ejercicio 3.4
### TO DO
```

Ejercicio 3.5

El número π puede aproximarse con la siguiente fórmula de aproximación,

$$\pi = 3 + 4 \sum_{n=2,4,6...;k=0,1,2..}^{N,K} (-1)^k \left[\frac{1}{n(n+1)(n+2)} \right]$$

Implemente una función que reciba como argumento la cantidad de aproximaciones N a realizar y retorne el valor de la aproximación.

La implementación de la función debe contener anotaciones y documentación apropiada.

NOTA: El valor de k sirve para alternar el signo del término $(-1)^k$.

In [5]:
```
## Resolución Ejercicio 3.5
##TO DO
```

Ejercicio 3.6

En criptografía, el cifrado César, también conocido como cifrado por desplazamiento, código de César o desplazamiento de César, es una de las técnicas de cifrado más simples conocidas. Es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, con un desplazamiento de 3, la A sería sustituida por la D (situada 3 lugares a la derecha de la A), la B sería reemplazada por la E, etc. Este método debe su nombre a Julio César, que lo usaba para comunicarse con sus generales. Fuente Wikipedia.

Debe implementar,

- Una función que se llame *codificar()* que reciba un mensaje y la cantidad de desplazamientos a realizar (la función debe tener este parametro con un valor por defecto para el caso de que el usuario no otorge un valor). La función debe retornar un mensaje encriptado.
- Una función que se llame *decodificar* que reciba como argumento la cantidad de desplazamientos a realizar (la función debe tener este parametro con un valor por defecto para el caso de que el usuario no otorge un valor). La función debe retornar un mensaje desencriptado.

La implementación de la función debe contener anotaciones y documentación apropiada.

In [6]:
```
## Resolución Ejercicio 3.6
##TO DO
```

Ejercicio 3.7

Una ruleta posee una rueda con 38 espacios. De estos espacios, 18 son negros, 18 son rojos y dos son verdes. Los espacios verdes están numerados cómo 0 y 00 (por simplicidad tomar sólo 0 como posible apuesta).

Los espacios rojos en la ruleta están numerados 1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30 32, 34 y 36. El resto de los números son de color negro.

Es posible realizar muchas apuestas en un juego de ruleta convencional, no obstante, para este ejercicio considere las siguientes:

- Números simples (1 a 36, y 0)
- Negras vs Rojas
- Pares vs Impares (el 0 no debe considerarse dentro de esta apuesta)
- 1 a 18 vs 19 a 36.

Implemente una función que reciba como argumento una apuesta (número entero) entre 0 y 36. La función debe generar un número entero aleatorio (use las funciones de la librería *random*) y devolver un mensaje donde se informe que apuestas deben pagarse al jugador, por ejemplo,

```
print(juegoRuleta(apuesta = 14))
```

Si el número aleatorio generado es el 14, el mensaje retornado debería ser similar a:

- Pagar al 14
- Pagar rojas
- Pagar pares
- Pagar 1 a 18

Si el número aleatorio generado es el 23, el mensaje retornado debería ser similar a:

- Pagar rojas

La implementación de la función debe contener anotaciones y documentación apropiada.

In [7]:
```
## Resolución Ejercicio 3.7
##TO DO
```

Ejercicio 3.8

1. Implemente una función que reciba como parámetro una lista de palabras y devuelva una lista de las posibles combinaciones de a dos. La lista retornada debe contener las tuplas correspondientes a la combinación de dos palabras. Por ejemplo, si la lista es,

```
palabras = ["ing", "bio", "bci"]
```

La función debe retornar algo similar a:

```
>> [("ing","bio"), ("ing","bci"), ("bio","bci")]
```

Recuerde que en combinación **no** importa el orden.

1. Luego intente implementar lo mismo utilizando *list comprehension* y utilizando la función *combinations* de la librería *itertools*. Para usar esta herramienta debe importar la librería o bien el método, haciendo,

```
from itertools import combinations
```

La implementación de la función debe contener anotaciones y documentación apropiada.

In [8]:
```
## Resolución Ejercicio 3.8
##TO DO
```

Ejercicio 3.9

Implemente una función que reciba una lista de elementos enteros, o flotantes, o strings (podría ser una lista con elementos del mismo tipo o bien una mezcla de estos) y retorne una lista con los elementos sin repetir.

Como ejemplo, si la lista es *[1,2,3,4,5,5,6,7,7,1,2,8,9,9]* la función debe retornar *[1,2,3,4,5,6,7,8]*.

La implementación de la función debe contener anotaciones y documentación apropiada.

In [9]:
```
## Resolución Ejercicio 3.9
##TO DO
```

Ejercicio 3.10

Implemente una función que retorne un diccionario con los datos de un equipo médico. La función debe tener como parámetros fijos un nombre genérico del equipo y el área donde se encuentra, además de ser necesario, se le pasarán a la función una mayor cantidad de datos referentes al equipo. A modo de ejemplo el pasaje de argumentos puede ser,

```
equipo1 = datosEquipo(equipo = "Respirador", area = "Neonatología")

equipo2 = datosEquipo(equipo = "Respirador", area = "UCI",
                      coment = "Alarma de presión de O2 no se apaga",
                      responsable = "deja el equipo Mengano")

equipo3 = datosEquipo(equipo = "Bomba infusión", area = "uti",
                      comentario = "Necesitamos el equipo urgente",
                      falla = "equipo no enciende o se apaga luego de algunas horas de
uso",
                      personaResponsable = "Dra. Sierra",
                      infoAdicional = "No había nadie en biomédica")
```

La implementación de la función debe contener anotaciones y documentación apropiada.

In [10]:
```
## Resolución Ejercicio 3.10
##TO DO
```

Ejercicio 3.11

Implemente una función que calcule la raíz cuadrada de un número entero utilizando el método de aproximaciones de Newton, el cual es,

$$\sqrt{a} = \frac{x + \frac{a}{x}}{2}$$

Donde,

- *x* es una *estimación* o *aproximación* conocida de la raíz cuadrada. En general puede asumirse *x = 1* o *x = a*.

La función a implementar debe recibir,

- Un número entero *a* para estimar su raíz cuadrada.
- Un número entero *n* que representa la cantidad de iteraciones a realizar para aproximar la raíz de *a*. Por defecto hacer n = 10.

La función debe retornar

- Una lista con cada uno de los cálculos realizados en cada iteración
- El valor calculado de la última iteración.
- El valor de la raiz cuadrada de *a* utilizando el método *sqrt* de la librería *math* (para comparación).

La implementación de la función debe contener anotaciones y documentación apropiada.

In [11]:
```
## Resolución Ejercicio 3.11
##TO DO
```

Ejercicio 3.12

Implemente una función que reciba como parámetros un número *n* de diccionarios. Asuma que los diccionarios pasados como parámetros poseen un *key* y elementos numéricos (enteros o flotantes).

La función debe retornar un diccionario nuevo con la suma de los elementos correspondientes a los keys iguales.

Por ejemplo,

```
dic1 = {'key1': 10, 'key2': 20, 'key3':30}
dic2 = {'key1': 30, 'key3': 20, 'key4':40}

dic3 = miFunc(d1,d2)
print(dic3)
# El dic3 debe ser de la forma {'key1': 40, 'key2': 20, 'key3': 50, 'key4': 40}
```

In [12]:
```
## Resolución Ejercicio 3.12
##TO DO
```

Ejercicio 3.13

Implemente una función que reciba como parámetro el nombre de un archivo de texto *.txt* y retorne un diccionario en donde los *keys* estarán formados por cada palabra dentro del archivo y los *elementos* serán las veces que la palabra en cuestión se repite dentro del texto.

Considere lo siguiente.

- Descartar símbolos y caracteres. Asuma que el texto solo puede contener los siguientes caracteres y símbolos:
  - _ (guión bajo), - (guión medio).
  - , (coma) y . (punto).
  - ¿, ?, ¡, ! (signos de interrogación y exlamación).
- Todas las palabras retornadas en el diccionario deben estar en minúsculas.

Ejemplo

```
## supogamos que el archivo texto.txt contiene lo siguiente
## "Hola mundo. Hola mundo!!!"

diccionario = makeDict("texto.txt")
#El diccionario retornado debe ser similar a {"hola": 2, "mundo": 2}
```

NOTA: Utilice la función *open()* para cargar el archivo.

La implementación de la función debe contener anotaciones y documentación apropiada.

In [13]:
```
## Resolución Ejercicio 3.13
##TO DO
```

FIN