REGLAS GENERALES

- Los grupos deben trabajar cada uno sin ayuda de otro grupo. Los grupos que presenten código sospechosamente parecido suspenderán el proyecto y la asignatura en su totalidad.
- Se debe subir a eGela la última versión que se tenga del trabajo por lo menos una vez por semana. Tened en cuenta de que hacer grandes cambios en poco tiempo es sospechoso para el profesorado. Para evitar problemas, subid el código a menudo para que se pueda ver la evolución de vuestro trabajo (ver sección Subir el Proyecto).

OBJETIVO DEL PROYECTO

Nuestra compañía desarrolla varios modelos de GPS, que graban las pistas con diferentes precisiones (diferente frecuencia de muestreo o recogida de datos). La tabla 1 resume las características principales de cada modelo de GPS.

Modelo de GPS	Pantalla	Redes	Precisión	Consumo (e_g)	Precio
Heimdl	Color	GPS+GLONASS	Más alta	0.008 mAh/s	550€
Sleipnir	B/W	GPS+GLONASS	Alta	0.006 mAh/s	400€
Yggdrasil	B/W	GPS	Baja	0.005 mAh/s	300€
Loki	B/W	GPS	Más baja	0.004 mAh/s	200€

Cuadro 1: Modelos de GPS y principales características

Dado que los modelos tienen diferente hardware (precisión, antena...) y por tanto más o menos consumo de energía, la compañía está probándolos con varios modelos y químicas de batería. La tabla 2 recoge los modelos de batería con sus capacidades y precio.

Modelo de batería	Capacidad	Precio
Asgard	4000 mAh	150€
Midgard	3000 mAh	100€
Helheim	2000 mAh	50€

Cuadro 2: Modelos de batería capacidad

En este proyecto analizaréis el rendimiento de los GPS en cuanto a precisión de la información, autonomía y otros factores.

EVALUACIÓN POR GRUPOS

Cada grupo tendrá una calificación que será la base de la nota de cada alumno (que puede ser mayor o menor que la del grupo). La nota máxima del grupo dependerá del número de tareas finalizadas correctamente por el grupo:

- 100 %: todas las tareas
- 80 %: tasks 1, 2, 3, 4 y 6

DIRECTRICES PARA LA PROGRAMACIÓN

- No imprimir mensajes por pantalla, crear gráficas o pedir entradas dentro de una función, a no ser que sea el objetivo de dicha función. La entrada y salida de datos de las funciones es mediante parámetros y resultado.
- Cuando sea posible, intenta reutilizar código mediante llamadas a funciones. En la mayor parte de las tareas se indicará cómo diseñar las funciones.
- Elige nombres con significativos y con la nomenclatura adecuada para crear variables/scripts/funciones, de modo que cualquiera que lea el código pueda entenderlo y seguir el valor de las variables sin pasar tiempo averiguando qué es cada elemento.
- Usa ciclos (for/while) para iterar (repetir instrucciones) sobre un conjunto de valores en lugar de repetir código ya existente.
- Escribe comentarios (con %) para explicar el objetivo de cada parte de los programas y qué datos almacenan las variables.

SUBID EL PROYECTO

1. Con la intención de facilitar la revisión de vuestro trabajo, os solicitamos que subáis el código a la tarea de eGela habilitada al respecto, **al menos una vez a la semana**. No borréis versiones antiguas.

2. Tan sólo un estudiante del grupo debe subir el código.

3. Cada vez que se suba el código es obligatorio subir un fichero ZIP:

   a) El nombre del fichero debe tener el formato: 'project-*yyyy*-*mm*-*dd*.zip', donde *yyyy* es el año, *mm* el mes y *dd* el día.

   b) El fichero zip debe contener todos los scripts/funciones que se hayan desarrollado hasta la fecha.

   c) El fichero zip también debe contener todos los resultados solicitados en las tareas que hayáis podido desarrollar hasta ese punto (plots, ficheros de texto, salidas por pantalla).

   d) Dentro del fichero también se incluirá un fichero único llamado 'log.txt', donde se glosará lo realizado cada día de desarrollo del proyecto. Este fichero irá creciendo con las aportaciones del trabajo realizado cada semana.

   Por ejemplo:

   log.txt en *project-2020-11-22.zip*

   2020-11-20
   Tarea 1: Comenzamos a programar el script ?myscript.m?, pero todavía no funciona.

   2020-11-22
   Tarea 1: Ya lo hemos terminado de revisar, había un error en el bucle principal.

FICHEROS DE ENTRADA

En este proyecto se procesarán los ficheros que se incluyen en el fichero denominado `gps-track-files.zip`. Dicho fichero se deberá decomprimir y ser extraído en la carpeta del proyecto.

Los ficheros tienen un nombre que sigue el patrón `track-XX-GPS.csv` donde `XX` es un número de dos cifras que representa la pista y `GPS` es el nombre del GPS usado para grabar la pista.

Cada pista ha sido grabada con tres tipos de GPS. Para cada posición que se ha grabado hay una fila en cada fichero. En cada fila, la primera columna representa la longitud, la segunda la latitud, la tercera la altitud, y la cuarta el tiempo (en segundos). El ejemplo siguiente muestra un fragmento de una pista[1].

Ejemplo: *track-06-Heimdl.csv*

```
42.3232...;-3.0118...;869.2000...;33181
42.3232...;-3.0118...;869.400...;33182
42.3232...;-3.0118...;869.5999...;33183
42.3232...;-3.0118...;868.7999...;33185
42.3231...;-3.0119...;869;33187
...
```

[1] Para mejorar la legibilidad, se han reemplazado algunas cifras decimales por '...'

TAREAS

Tarea 1: Examinar las rutas grabadas por los dispositivos GPS

En esta tarea se van a analizar las rutas (*tracks*) grabadas por los dispositivos GPS. Tenemos `m` grabaciones por cada una de las `n` rutas. Para ello debéis crear un *script* llamado *examinaRutas* que genere una gráfica compuesta de dos subgráficas. La de la izquierda mostrará superpuestas las rutas grabadas por cada GPS y la de la derecha contendrá las líneas de altitud de cada dispositivo (ver la gráfica 1). El trazado es el mismo pero se observarán sutiles diferencias entre las rutas grabadas por cada GPS, como se puede comprobar ampliando la imagen.

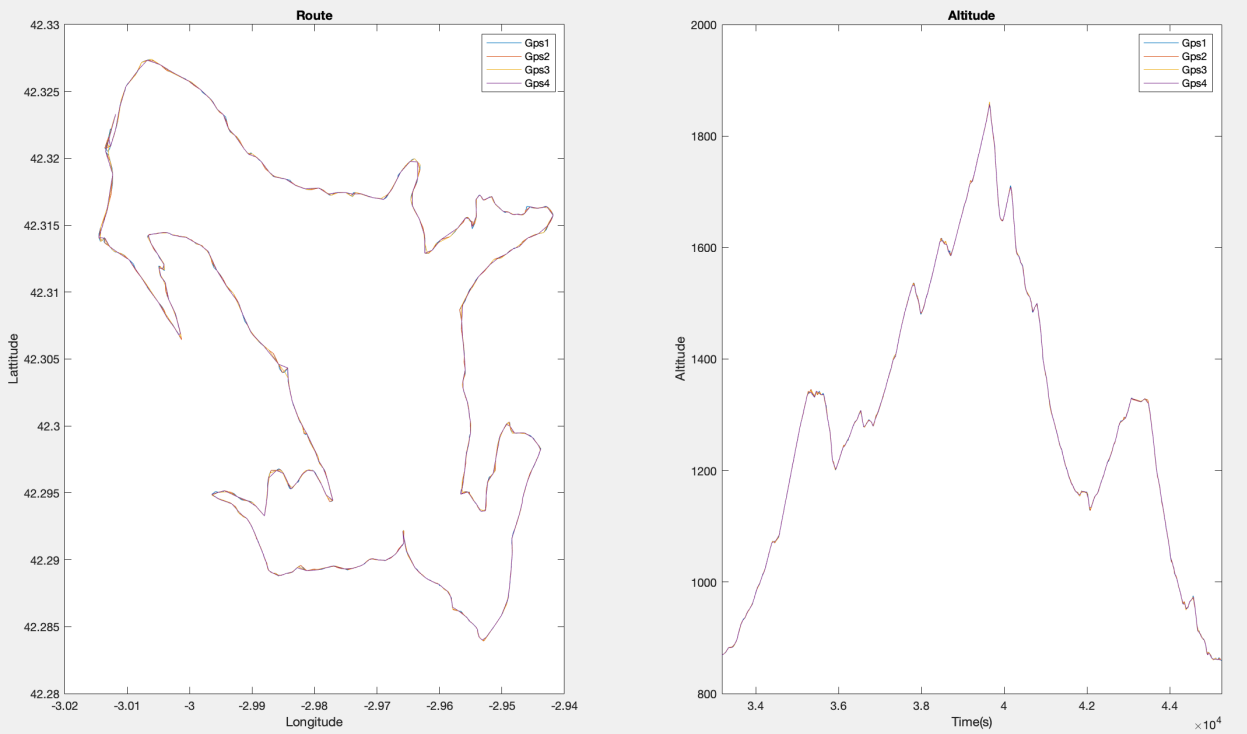

Figura 1: Ejemplo de grabaciones de rutas y altitudes de los diferentes dispositivos

El *script* guardará la gráfica de cada una de las 15 rutas en un fichero llamado `track-XX.png`, donde `XX` es un número de dos cifras que representa la pista. El fichero debe ser guardado en la carpeta *informes*.

Tarea 2: Crear un informe de las rutas grabadas

Para comparar el rendimiento de los diferentes modelos de GPS vamos a realizar análisis estadísticos de las rutas. Debéis cumplir estas 4 fases:

1. Implementa una función llamada `distancia` en la que, dados los vectores que representan las longitudes y latitudes de los puntos de origen y de destino, devuelva un vector que contenga la distancia entre cada par de puntos, calculada en metros.

   ```
   function dist = distance(longOrig, latOrig,
     longDest, latDest)
   ```

   Puedes comprobar la función ejecutándola con el siguiente formato, y comparando con el resultado obtenido:

   ```
   distancia([-3.011 -3.012 -3.13], [42.323 42.324 42.325],
   [-3.012 -3.13 -3.014],[42.324 42.325 42.326])

   [138.286698155807 9701.56094445445 9536.99792232943]
   ```

   Para calcular la distancia usaremos el método pitagórico, que obtiene la distancia en radianes entre dos puntos en una esfera mediante sus coordenadas.

$$x = \Delta\lambda * cos\left(\frac{\varphi_1 + \varphi_2}{2}\right)$$
$$y = \Delta\varphi$$

   donde:

   - φ_1 y φ_2 son las *latitudes* de origen y destino **en radianes**
   - $\Delta\lambda$ es la distancia entre las *longitudes* de destino y origen **en radianes**
   - x es la diferencia entre las *longitudes* de origen y destino **en radianes**
   - y es la diferencia entre las *latitudes* de origen y destino **en radianes**

Posteriormente el resultado hay que multiplicarlo por el radio medio de La Tierra (R=6371km) para calcular la distancia entre los dos puntos usando la siguiente fórmula (y pasar a metros).

$$distancia = \sqrt{x^2 + y^2} * R$$

2. Escribe una función llamada `computeTrackLength` en la que se devuelva la distancia total recorrida en metros a partir delos vectores que contienen las latitudes y longitudes de los puntos de ruta.

```
function totalDistancia = computeTrackLength(
    longitudes, latitudes)
```

Podéis comprobar que la función es correcta mediante el ejemplo siguiente, comparando el resultado que se obtiene:

```
dist = trackLength([42.323 42.324 42.325 42.326],
                   [-3.011 -3.012 -3.13 -3.014]);
fprintf('%.4f', dist);

26177.7054
```

3. Escribe una función llamada *computeCumulativeAscentDescent* que calcule ascenso y descenso acumulados a partir del vector de altitudes de una grabación.

```
function [asc,des] =
    computeCumulativeAscentDescent(altitudes)
```

Podéis comprobar que la función es correcta mediante el ejemplo siguiente, comparando el resultado que se obtiene:

```
[asc, desc] = computeCumulativeAscentDescent(
                [869.2 869.4  869.1 868.9 900])
asc  = 31.3000
desc = 0.5000
```

4. Escribe un *script* llamado *gpsTrackRecordingReport.m* en el que imprima un informe para cada ruta. En el informe se debe mostrar con el siguiente formato la información extraída de cada grabación de GPS y calculada con las funciones programadas anteriormente.

```
Track 1
=======
Modelo: Heimdl
        Altitud máxima: 2565.00   Altitud mínima: 1879.02
        Ascenso acumulado: 785.33 Descenso acumulado: 705.32
        Distancia: 10.05 km       Velocidad media: 2.05 km/h
Modelo: Sleipnir
        Altitud máxima: 2565.00   Altitud mínima: 1881.02
        Ascenso acumulado: 714.93 Descenso acumulado: 634.92
        Distancia: 7.51 km        Velocidad media: 1.53 km/h
Modelo: Yggdrasil
        Altitud máxima: 2565.00   Altitud mínima: 1881.02
        Ascenso acumulado: 703.01 Descenso acumulado: 623.00
        Distancia: 7.30 km        Velocidad media: 1.49 km/h
Modelo: Loki
        Altitud máxima: 2563.98   Altitud mínima: 1884.00
        Ascenso acumulado: 689.92 Descenso acumulado: 609.91
        Distancia: 7.08 km        Velocidad media: 1.44 km/h
...
```

Tarea 3: Informe de consumo de energía

Todos los modelos de GPS se prueban con la batería de más capacidad, pero la compañía quiere analizar el consumo de energía de los diferentes modelos. En esta tarea debéis:

1. Escribe una función llamada *computeEnergyConsumption* que calcule el uso de la batería realizado a partir de dos vectores (que representan las altitudes grabadas y los tiempos grabados en por cada GPS en la ruta, respectivamente) y de la energía que consume el GPS en condiciones ideales (a 400m de altitud.

   ```
   function consumo = computeEnergyConsumption(
                        altitudes, times, eg)
   ```

   La compañía ha observado que la energía que consume el GPS cada segundo puede estimarse con la siguiente fórmula:

   $$e_t = e_g * \left(1 + \frac{|400 - h_t|}{100}\right)$$

   donde e_t es la energía consumida por el GPS en el instante t, e_g es la energía que consume el GPS en un funcionamiento bajo condiciones ideales (ver la Tabla 1), y h_t representa la altura en el tiempo t. Se observa pues que la altitud afecta a la calidad de recepción del GPS[2]. Grabar cada uno de los puntos de la ruta tiene un consumo adicional de 0.0005 mAh por cada punto. Para tener una estimación más ajustada, usar una interpolación de $\Delta t = 0{,}1$.

   Comprobación de la función:

   ```
   computeEnergyConsumption([400 410 380 400],[0 120 240 480], 0.0005)
   0.2620
   computeEnergyConsumption([400 410 380 400],[0 120 240 480], 0.0008)
   0.4180
   computeEnergyConsumption([400 410 380 400],[0 120 240 480], 0.0006)
   0.3140
   ```

[2]Esto es una suposición de la práctica, no un hecho tecnológico.

2. Escribe una función llamada *getAllGPSEnergyConsumptions* en la que, dado el número de rutas, el vector con los nombres de los modelos de GPS y el vector que contiene los consumos de energía (e_g) de los GPS, procese todos los ficheros de ruta y devuelva los consumos de energía de los dispositivos.

```
function consumeMatrix = getAllGPSEnergyConsumptions(
        numTracks, gpsNames, egs)
```

   Esta función devuelve una matriz en la que cada fila representa la energía consumida para grabar la correspondiente ruta y cada columna representa la energía que consume un GPS para grabar una ruta (ver el Cuadro 3).

   En otras palabras, e_{ij} es la energía consumida por el GPS j cuando graba la pista i.

	Heimdl	*Sleipnir*	*Yggdrasil*	*Loki*
$ruta_1$	e_{11}	e_{12}	e_{13}	e_{14}
$ruta_2$	e_{21}	e_{22}	e_{23}	e_{24}
$ruta_3$	e_{31}	e_{32}	e_{33}	e_{34}
...				
$ruta_m$	e_{m1}	e_{m2}	e_{m3}	e_{m4}

Cuadro 3: Estructura de la matriz de consumos de energía de los dispositivos GPS en cada ruta

3. Escribe un *script* llamado *analyseGPSEnergyConsumption.m* para generar una gráfica que muestre el consumo de energía por ruta y GPS (ver la Figura 2).

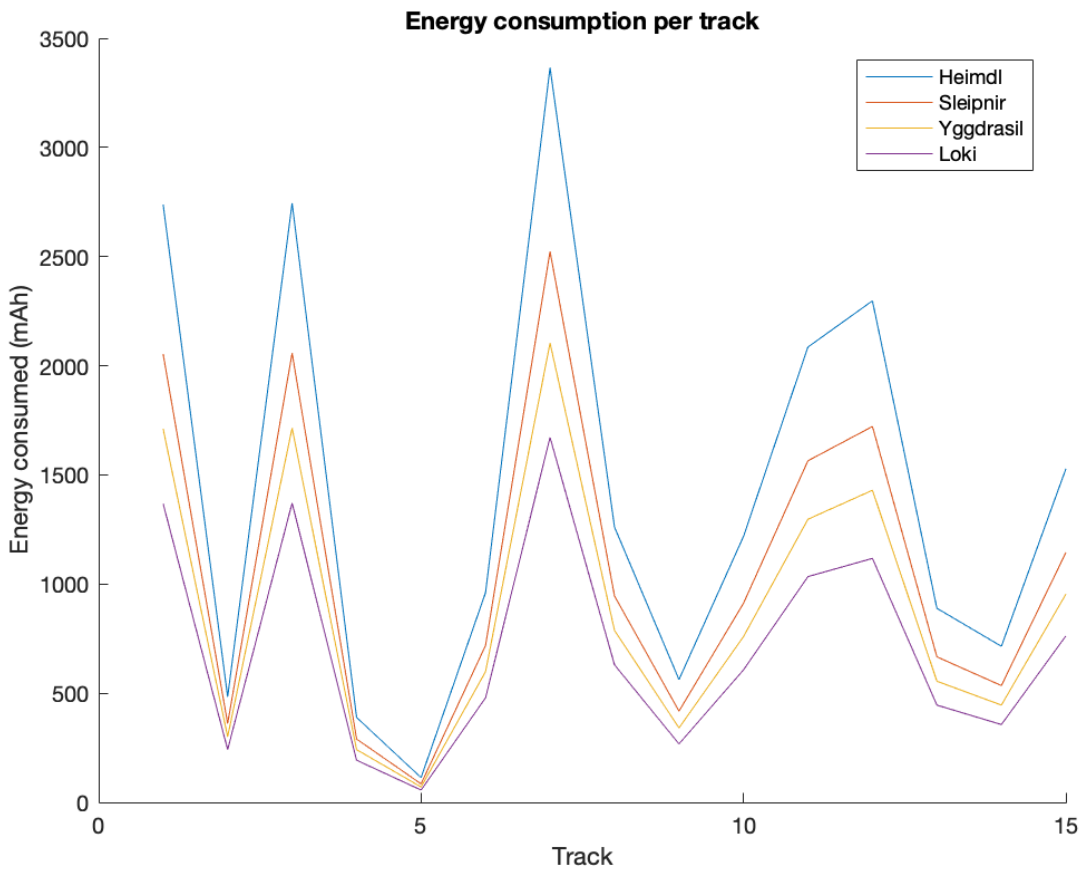

Figura 2: Gráfica que muestra el consumo de energía por ruta y GPS

Tarea 4: Determinar las configuraciones más económicas de los GPS

Para ofrecer modelos de GPS atractivos a un precio asequible, la compañía quiere determitar cuál es la configuración de los dispositivos más económica (modelo + batería). Escribe un *script* que elija la batería(var la Tabla 2) más asequible que asegure la grabación de todas las rutas grabadas en el proceso de evaluación que se ha estado realizando. Con este objetivo debéis realizar las dos siguientes subtareas.

1. Escribe una función que tenga las siguientes entradas y salidas. Como entradas tendrá un vector con la energía consumida por un GPS para grabar las rutas, y otro vector que contiene las capacidades en mAh de los modelos

de batería. Como salida devolverá la posición de la batería más barata que asegure que ese dispositivo GPS puede grabar todas las rutas, es decir, la capacidad es mayor que la energía requerida para grabar cada ruta. Como se puede observar en la Tabla 2, cuanto más baja es la capacidad, más barata es la batería. Vmos a suponer que el vector *batteryCapacities* de capacidades de baterías está ordenado de mayor a menor capacidad.

```
function batteryId = getCheapestBatteryId(
        gpsEnergyConsumes, batteryCapacities)
```

Comprueba la corrección del código:

```
getCheapestBatteryId([1600 1500 3200],[4000 3000 2000])
1
getCheapestBatteryId([1600 1500 2800],[4000 3000 2000])
3
getCheapestBatteryId([1600 1500 1900],[4000 3000 2000])
3
```

2. Escribe un programa o *script* que tenga el nombre *determineCheapestGPSSetUps.m* para imprimir un informe que muestre las configuraciones más económicas. El informe deberá tener este formato:

```
Configuraciones de los GPS
==========================
Modelo de GPS: Heimdl    Batería: Asgard   Precio: 700€
Modelo de GPS: Sleipnir  Batería: Mildgard Precio: 500€
Modelo de GPS: Yggdrasil Batería: Mildgard Precio: 400€
Modelo de GPS: Loki      Batería: Helheim  Precio: 250€
```

Tarea 5: Informe y categorización de las configuraciones de GPS

Se puede elegir un determinado dispositivo GPS en función de varios aspectos: hay quien puede preferir mayor precisión y hay quien elija el más barato. Por ello:

1. Escribe la función *computeGPSTrackAccuracy* que calcule la precisión de la ruta grabada a partir de la longitud de la misma (distancia recorrida) y la

longitud real de la ruta. La operación que va a generar el valor de la precisión será la división de la distancia grabada entre la longitud real.

```
function accuracy = computeGPSTrackAccuracy(
    recordedDistance, realDistance)
```

Para comprobar la corrección:

```
computeGPSTrackAccuracy(12.34, 12.34))
1
computeGPSTrackAccuracy(11.54, 12.34))
0.9352
computeGPSTrackAccuracy(10.43, 12.34))
0.8452
```

2. Escribe la función *computeAverageGPSAccuracy* que compute la precisión media a partir de dos vectores que contienen las longitudes de las rutas grabadas y las distancias reales de las rutas.

```
function avgAccuracy = computeAverageGPSAccuracy(
    recordedDistances, realDistances)
```

Para comprobar la corrección:

```
computeAverageGPSAccuracy([12.34 35.78 9.34], [12.34 35.78 9.34])
1
computeAverageGPSAccuracy([11.45 33.97 9.11], [12.34 35.78 9.34])
0.9509
computeAverageGPSAccuracy([10.89 31.34 8.98], [12.34 35.78 9.34])
0.9066
```

3. Escribe una función que tenga como nombre *computeGPSRatingAndAccuracy* que compute la puntuación final de cada GPS y su precisión media. Las entradas serán la lista de nombres de los GPS, el vector con las distancias reales de las rutas y el precio de los GPS.

```
function [rating, avgAccuracy] =
    computeGPSRatingAndAccuracy(gpsName,
    realDistances, gpsPrice)
```

Para calcular la puntuación final debéis usaar la siguiente fórmula:

$$rating = 0{,}8 * avgAccuracy_g + *{,}3 * \left(\frac{refPrice}{price_g} \right)$$

donde $avgAccuragy_g$ es la precisión media del modelo de GPS, $refPrice$ es un precio de referencia (625€), y $price_g$ es el precio del GPS (ver la Tabla 1).

4. Escribe un *script* llamado *gpsRankings.m* que imprima en pantalla un informe con las mejores opciones de compra. Las opciones se ordenarán por la puntuación final como en el ejemplo siguiente. Para generar el informe se van a considerar como rutas reales las grabadas con el dispositivo GPS *Heimdl*, el de mayor precisión.

```
Mejores opciones de GPS
=======================
Modelo de GPS: Loki      Batería: Helheim
Precisión: 0.801 Precio: 250? Puntuación: 1.141
Modelo de GPS: Heimdl    Batería: Asgard
Precisión: 1.000 Precio: 700? Puntuación: 0.979
Modelo de GPS: Yggdrasil Batería: Mildgard
Precisión: 0.825 Precio: 400? Puntuación: 0.972
Modelo de GPS: Sleipnir  Batería: Mildgard
Precisión: 0.868 Precio: 500? Puntuación: 0.945
```

Taks 6: Menú del programa principal

Programa el script `mainMenu.m`, de manera que a la persona usuaria se le muestre el siguiente menú:

```
#######  MENÚ  ########:
1. Generar las gráficas de las ruttas
2. Informes de las rutas
3. Informe del consumo de energía
4. Determinar las configuratione de los GPS más económicos
```

```
5. Clasificación de las configuraciones de los GPS
0. Finalizar
Selecciona una opción:
```

Según la opción elegida, se ejecutará el script de la tarea correspondiente. Cada entrada del menú se corresponde con cada una de las tareas realizadas para el siguiente proyecto. Si el número introducido como entrada no es correcto, se mostrará un mensaje de error: `La opción elegida no es correcta: elige una opción entre 1 y 5, o 0 para salir`.

Antes de mostrar el menú, el script limpiará la consola. Al elegir una de las opciones, se mostrarán los resultados de la tarea correspondiente y un mensaje "`Pulsa una tecla para continuar...`" para poder ver los resultados con la instrucción `pause`. A continuación se mostrará de nuevo el menú. Esta operación se repetirá mientras no se elija la opción 0.