eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZAKO UNIBERTSITATE ESKOLA
ESCUELA UNIVERSITARIA DE INGENIERÍA DE VITORIA-GASTEIZ

REGLAS GENERALES

- Los grupos deben trabajar por sí mismos sin ayuda de otro grupo. Los grupos que presenten código sospechosamente parecido no sólo van a suspender el proyecto sino también la asignatura.

- 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 puede ser sospechoso para el profesorado. Para evitar problemas, subid el código a menudo para se que pueda ver la evolución de vuestro trabajo (ver sección *Subir el Proyecto*).

OBJETIVO DEL PROYECTO

En este proyecto trabajaremos para una compañía de logística analizando sus datos para hacer las decisiones adecuadas. La compañía tiene dos conductores para transporte entre Donostia-San Sebastián y Vitoria-Gasteiz. Se pueden usar dos rutas distintas entre dichas ciudades: la antiguamente llamada N-1 (Nacional 1) y la autopista, que llamaremos A-1.

Vamos a desarrollar unos programas (*scripts*) y funciones para que la compañía decida entre conductores y rutas.

CALIFICACIONES DEL GRUPO

Cada grupo tendrá una calificación que será la base de la nota de cada alumno (nota 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

- 75%: tareas 1, 2, 3 y 4

DIRECTRICES PARA LA PROGRAMACIÓN

- No imprimir, 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, reutiliza código mediante funciones. En la mayor parte de las tareas se indicará cómo diseñaar las funciones.
- Usa ciclos (*for*/*while*) para iterar (repetir instrucciones) sobre un conjunto de valores en vez de replicar código ya existente.
- Elige nombres con un significado y forma adecuados para identificar variables/scripts/funciones para que cualquiera que lea el código pueda entenderlo y seguir el valor de las variables sin pasar tiempo averiguando qué es cada elemento.
- Escribe comentarios (con **%**) para explicar el objetivo de cada parte de los programas.

SUBIR EL PROYECTO

Con la intención de revisar vuestro trabajo, os solicitamos que subáis el código e la tarea de eGela habilitada al respecto, **al menos una vez a la semana**.

- Tan sólo un estudiante del grupo debe subir el código
- Cada vez deberéis subir un fichero ZIP:
  - 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.
  - El fichero zip debe contener todos los scripts/funciones que hayáis desarrollado hasta la fecha.
  - 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).
  - Dentro del fichero también se incluirá un '*log.txt*', donde añadiréis lo que habéis hecho desde la última vez que subisteis vuestro código/resultados. Por ejemplo:

log.txt dentro de *project-2017-11-22.zip*

```
2017-11-20
Task 1: We started working on 'myscript.m', but it still
doesn't work

2017-11-22
Task 1: We finally made it work, results look reasonable
Task 2: Just started working on function 'myfunction'
```

FICHEROS DE ENTRADA

Trabajaremos con los siguientes ficheros:

1. Registros (*logs*) de los dos conductores en las dos rutas:
   - `a1-driver1-log.csv`
   - `a1-driver2-log.csv`
   - `n1-driver1-log.csv`
   - `n1-driver2-log.csv`

   En estos ficheros tenemos la distancia desde el origen (*km*) y velocidad del vehículo en ese momento (*km/h*) para una serie de puntos a lo largo de la ruta. La primera columna contiene las distancias y la segunda las velocidades, separados por comas:

   Ejemplo: n1-driver1-log.csv

   ```
   0,0
   0.10013,8.5168
   0.20027,16.901
   0.3004,24.103
   0.40053,30.048
   0.50066,35.717
   ...
   ```

2. Perfil (alturas) de las dos rutas:
   - `a1-height.csv`
   - `n1-height.csv`

   En estos ficheros tenemos la información de las rutas (latitud, longitud, elevación, distancia y gradiente (pendiente):

Ejemplo: n1-height.csv

```
Latitude,Longitude,Elevation(metres),Distance (KMs),Gradient
43.31839,-1.98127,5.7,0,
43.31821,-1.98169,6.2,0.039,
43.31876,-1.98207,6,0.108,
43.31887,-1.98214,6,0.121,
43.31936,-1.98249,6.1,0.183,
43.31939,-1.98252,6.1,0.187,
43.31945,-1.98256,6.1,0.195,0
43.31917,-1.9833,6.3,0.262,0
...
```

Tendremos que saltar la primera línea del fichero con los nombres de las columnas.

3. Límites de velocidad de cada ruta:
   - `a1-speed-limit.csv`
   - `n1-speed-limit.csv`

Ejemplo: a1-speed-limit.csv

```
0;90
5;120
95;90
```

En estos ficheros tenemos la información de los límites de velocidad. En la primera columna está la distancia desde el origen (en *km*) a partir de la que está en vigor el límite de velocidad.

En la segunda columna está la velocidad máxima permitida desde dicho punto (en *km/h*). En el ejemplo anterior, los conductores pueden conducir hasta $90\ km/h$ durante los primeros 5 km, luego hasta $120\ km/h$ desde el punto kilométrico (PK) 5 al 95, y hasta $90\ km/h$ desde el PK 95.

TAREAS

Tarea 1:

Escribe un programa (*script*) que analice los perfiles de elevación de ambas rutas:

1. Generará una gráfica con dos subgráficas (*subplots*):
   - alturas (m) de ambas rutas relativas a la distancia desde el origen (km).
   - las propias rutas (longitudes contra latitudes).

   Cada una de las subgráficas debe estar identificada con un título y los ejes debidamente etiquetados. Por ejemplo, una etiqueta adecuada en un eje en el que se muestre el tiempo sería *tiempo (s)*. La gráfica tota debe grabarse (guardarse) con el nombre *'route-elevations.png'* en la misma carpeta que la de los scripts.
2. El script debe calcular y mostrar como texto los siguientes valores estadísticos: media, desviación estándar, elevación mínima y máxima. La salida sería:

```
Estadísticas de la ruta n1:
Altura media: 318.37 (sd: 235.54)
Rango de alturas: [4.20, 656.10]

Estadísticas de la ruta a1:
Altura media: 189.71 (sd: 181.70)
Rango de alturas: [0.00, 825.80]
```

Tarea 2:

Escribe un programa (*script*) que analice los registros de ambas rutas para cada conductor:

1. Generará una gráfica con dos subgráficas (*subplots*), una por cada conductor, con las velocidades (km/h) de ambas rutas relativas a la distancia desde el origen (km).

   Cada una de las subgráficas debe estar identificada con un título, una leyenda para cada asignar los conductores a las gráficas, y los ejes debidamente etiquetados. Por ejemplo, una etiqueta adecuada en un eje en el que se muestre el tiempo sería *tiempo (s)*. La gráfica tota debe grabarse (guardarse) con el nombre *'route-elevations.png'* en la misma carpeta que la de los scripts.

2. El script debe calcular y mostrar como texto los siguientes valores estadísticos: media, desviación estándar, velocidad mínima y máxima. La salida sería:

```
Estadísticas del conductor1 en la ruta n1:
Velocidad media: 95.99 (sd: 10.14)
Rango de velocidades: [0.00, 118.44]

Estadísticas del conductor2 en la ruta n1:
Velocidad media: 114.10 (sd: 10.78)
Rango de velocidades: [0.00, 121.43]

Estadísticas del conductor1 en la ruta a1:
Velocidad media: 101.29 (std.dev: 14.82)
Rango de velocidades: [0.00, 119.00]

Estadísticas del conductor2 en la ruta a1:
Velocidad media: 104,79 (std.dev: 15.62)
Rango de velocidades: [0.00, 122.17]
```

Tarea 3:

Escribe el *script* `mainMenu.m` que muestre este menú:

```
#######  MENÚ  ########:
1. Muestra las gráficas y estadísticas de las rutas
2. Muestra las gráficas y estadísticas de los conductores
3. Cálculos de tiempo para cada conductor y ruta
4. Comprobar los límites de velocidad
5. Cálculo de consumo de combustible para cada conductor y
ruta
6. Salir
Elige una opción:
```

La persona usuaria del programa introducirá el número de la opción deseada y se ejecutará su script. Cada una de las entradas del menú se corresponde con una de las tareas del proyecto. Si el número es incorrecto, se mostrará un mensaje que diga *"Opción incorrecta: debe ser un número entre 1 y 6"*.

Antes de mostrar el menú se debe borrar la consola con `clc`. Este menú se mostrará después de ejecutar una opción, hasta que se seleccione la opción 6 (Salir), que hará terminar el script principal sin más mensajes.

Para visualizar correctamente la información, los scripts que generan resultados deben ser modificados para que muestren un mensaje *"Pulsa una tecla para continuar"* y no terminarán hasta que se escriba algo. De esta manera, la salida se mostrará hasta que se pulse una tecla y el menú aparezca de nuevo.

Tarea 4:

Escribe un script que estime el tiempo que le llevó a cada conductor cada ruta. Para ello haremos estos pasos:

1. Escribe una función que interpole linealmente a partir de un conjunto de valores x-y:

```
function [ interpolatedY ] = interpolateLinearly( xVector, yVector , x)

% This function gets a vector with x-values and a vector with y-values
% and returns the interpolated value of y for the given x.
% Esta función recibe un vector con los valores de x, otro
% con los valores de y, y devuelve el valor interpolado de y para un x
dado.
```

   Comprueba que la función es correcta con los siguientes valores:

```
interpolateLinearly([10 20 40],[120 150 130],10) => 120
interpolateLinearly([10 20 40],[120 150 130],15) => 135
interpolateLinearly([10 20 40],[120 150 130],25) => 145
interpolateLinearly([10 20 40],[120 150 130],40) => 130
```

2. Escribe una función que, dada una cantidad de kilómetros, devuelva el número equivalente de metros:

```
function [ m ] = toMeters( km )
```

   Comprueba que la función es correcta con los siguientes valores:

```
toMeters(2.3) => 2300
toMeters(0.3) => 300
```

3. Escribe una función que, dada una velocidades en *km/h*, devuelva la velocidad equivalente en *m/s*:

```
function [ msSpeed ] = toMetersPerSecond( speedKmH )
```

   Comprueba que la función es correcta con los siguientes valores:

```
toMetersPerSecond(120) => 33.3333
toMetersPerSecond(15) => 4.1667
```

4. Escribe una función que estime el tiempo en el que se realiza la ruta:

```
function [ estimatedTime ] = estimateTime( kms, speedKmH, numSlices)
```

```
%This function is given a vector with a vehicle's speed (speedKmH) at
%different points (kms) and the number of integration slices we want to
%use (numSlices)
% Esta función recibe un vector con la velocidad del vehículo (speedKmH)
% en diferentes puntos (kms) y el
% número de pasos de integración (numSlices)
```

Si tomamos los ficheros de registros por condutor y ruta, *kms* tendría los valores de la primera columna del fichero, y *speedKmH* sería la segunda columna.

Esta función interpola linealmente las velocidades del vector de velocidades para el número de intervalos de integración(*numSlices*) determinado. Para cada intervalo interpolaremos el valor de la velocidad usando la función que hicimos en la subtarea anterior, y estimaremos el tiempo necesario para cubrir la distancia que comprende cada intervalo. El resultado será la suma total de tiempos estimados para la ruta.

Comprueba que la función es correcta con los siguientes valores:

```
estimateTime([0 1 2 3],[40 50 40 30],1000) => 264.2797
estimateTime([0 1 2 3],[40 50 40 20],1000) => 285.5495
```

5. Escribe una función que, dada una cantidad de segundos, devuelva una cadena de caracteres con el formato HH:MM:SS (horas, minutos y segundos, cada una con 2 dígitos). No se tendrán en cuenta las fracciones de segundo.

```
function [ hms ] = toHMS( seconds )
```

Comprueba que la función es correcta con los siguientes valores:

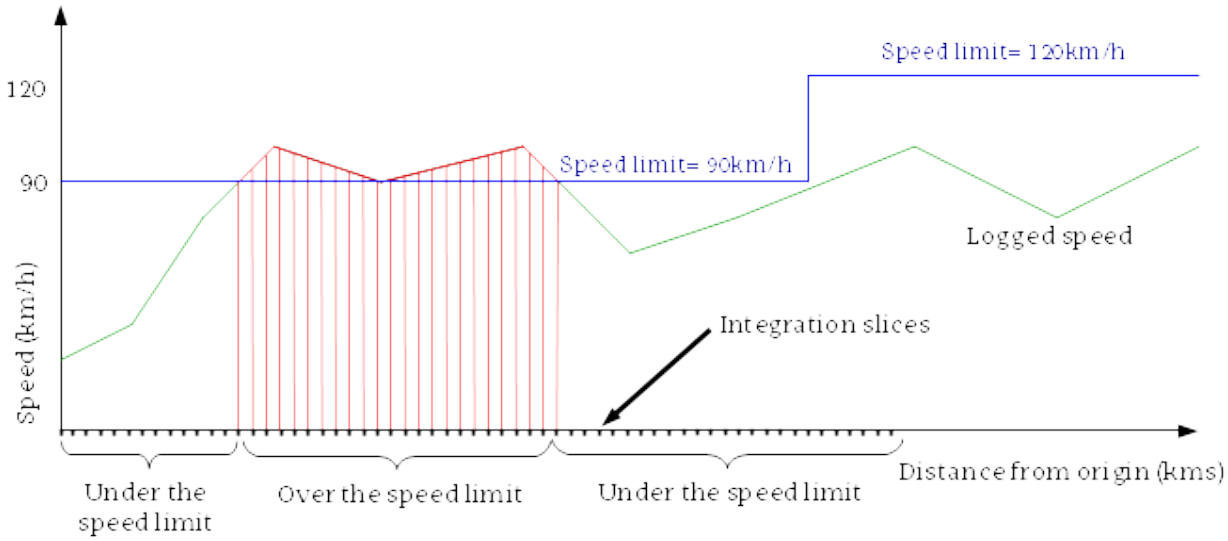
```
toHMS(234.4) => 00:03:54
toHMS(3601) => 01:00:01
```

6. Usando la función *estimateTime* fescribe un script que estime el tiempo que lleva cada combinación ruta-conductor en el formato *horas:minutos:segundos*:

```
Estimated time for driver1 in route n1: 1:58:54
Estimated time for driver2 in route n1: 1:56:13
Estimated time for driver1 in route a1: 1:54:09
Estimated time for driver2 in route a1: 1:51:12
```

Tarea 5:

Escribe un script que analiza cuánto tiempo condujo cada conductor sobre el límite de velocidad en cada ruta.



Necesitamos usar los 4 ficheros de registro y los 2 ficheros con los límites de velocidad. Cada uno de estos ficheros tiene una diferente resolución (intervalos de muestreo), por lo que necesitamos interpolar.

Idea básica: Dado un registro ruta-conductor, integraremos los trozos de ruta en los que el conductor ha sobrepasado el límite de velocidad. Para hacerlo, vamos a usar intervalos pequeños, y para cada uno calcularemos (interpolando) la velocidad del conductor al principio del intervalo.

Compararemos dicha velocidad con el límite en el mismo punto. Si sobrepasa, calcularemos el área del intervalo y lo sumaremos al número de km que ha conducido sobre el límite.

Te sugerimos hagas los siguientes pasos:

1. Los límites de velocidad deben ser interpolados de forma diferente que las velocidades. Por ejemplo, si los límites de velocidad son:

```
kms=          [0   10   40];
speedLimit=   [90  120  90];
```

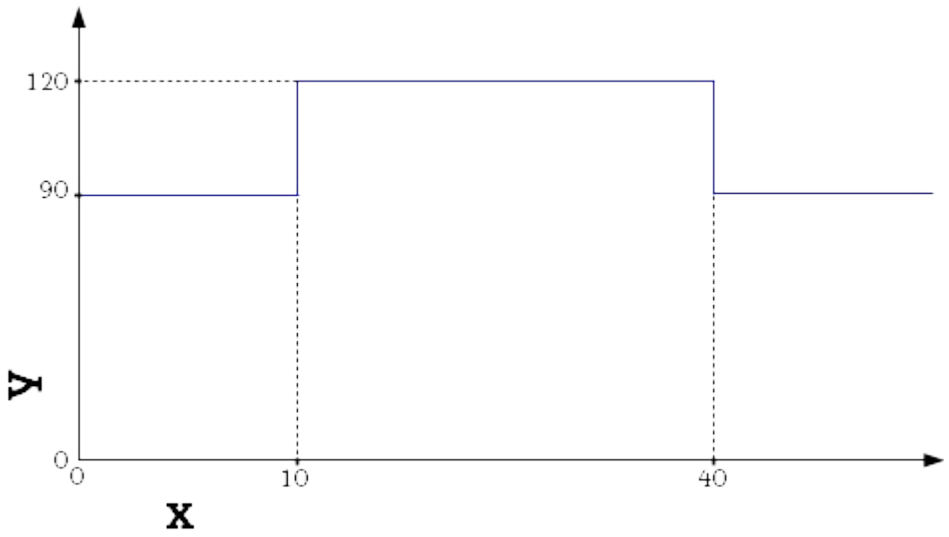
quiere decir que el límite en los primeros 10km es 90km/h, luego 120 km/h entre los puntos kilométricos 10 y 40, y 90km/h desde el PK 40 en adelante.

Escribe la siguiente función que devuelve el valor interpolado a la izquierda de y para cierto valor x dado:

```
function [ interpolatedY ] = interpolateToTheLeft( xVector, yVector , x)

% Esta función recibe un vector con los valores de x, otro
% con los valores de y, y devuelve el valor interpolado de y para un x
dado.
```

Con los valores del ejemplo anterior, esta función debe devolver el valor del vector *yVector* correspondiente al valor de x más cercano a la izquierda en xVector. Visualmente, la salida sería:



Comprueba que la función es correcta con los siguientes valores:

```
interpolateToTheLeft([0 10 40],[90 120 90],3) => 90
interpolateToTheLeft([0 10 40],[90 120 90],15) => 120
interpolateToTheLeft([0 10 40],[90 120 90],30) => 120
interpolateToTheLeft([0 10 40],[90 120 90],100) => 90
```

2. Escribe una función que, dado un conjunto de velocidades en diferentes puntos de una ruta (*driverLogKm* y *driverLogSpeed*), los límites de

velocidades en la misma ruta (*limitKms* y *limitSpeeds*) y el número de intervalos usado en los cálculos (*numSlices*), devuelva el número de km en los que el conductor sobrepasó los límites (*kmsAboveSpeedLimit*) y el porcentaje de km en la ruta sobre los límites de velocidad (*percentAboveSpeedLimit*).

```
function [ kmsAboveSpeedLimit,percentAboveSpeedLimit ] =

      checkSpeedLimits( driverLogKm, driverLogSpeed, limitKms, limitSpeeds, numSlices)
```

3. Escribe un script que use la función anterior para estimar el riesgo de recibir una multa por exceso de velocidad en cada registro ruta-conductor. Para cada par ruta-conductor imprimiremos un mensaje que dependa del porcentaje de la ruta en el que el conductor sobrepasó el límite de velocidad. Si es por encima del 10% consideraremos *riesgo alto*, y *riesgo leve* por debajo. Por supuesto, en el caso de 0% será *sin riesgo*.

   La salida sería:

```
Analyzing: Driver= driver1, Route= n1
Mild infraction risk: Kms abover the speed limit= 10.43 (5.64% of
the route)

Analyzing: Driver= driver2, Route= n1
HIGH INFRACTION RISK: Kms abover the speed limit= 149.39 (80.75%
of the route)

Analyzing: Driver= driver1, Route= a1
No risk of infraction

Analyzing: Driver= driver2, Route= a1
HIGH INFRACTION RISK: Kms abover the speed limit= 131.29 (70.97%
of the route)
```

Tarea 6:

Escribe un script que estime el consumo de combustible del vehículo en cada ruta-conductor. Usaremos la siguiente fórmula:

$$f = \frac{d * (0.00009v + (0.021a + 0.087\theta)^2)}{33} \quad ,$$

donde f es el combustible en litros consumido para conducir d metros con una pendiente de θ (rad) a una velocidad v (m/s) y aceleración a (m/s^2). Esta fórmula asume una velocidad, aceleración y pendiente constante, por lo que deberemos integrar sobre la distancia con los datos del registro ruta-conductor.

Sugerimos seguir estos pasos para realizar la tarea:

1. Escribe una función que calcule el gasto de combustible para distancias cortas, donde la velocidad, aceleración y pendiente son constantes:

   ```
   function [ fuelExpenditure ] = calculateFuelExpenditure(v,a,theta,d)
   ```

   Comprueba la corrección de la función con estos valores (*33.33 m/s = 120 km/h*):

   ```
   calculateFuelExpenditure(33.33,0,0,1000) => 0.0909
   calculateFuelExpenditure(33.33,1,0.3,1000) => 0.1581
   calculateFuelExpenditure(33.33,1,-0.3,1000) => 0.0917
   ```

2. Escribe una función que calcule el gasto de combustible en una ruta completa:

   ```
   function [ fuelExpenditure ] =
   calculateRouteFuelExpenditure(routeKms,routeHeights,logKms,logSpeeds,num
   Slices )
   ```

   donde tenemos el perfil de elevación de la ruta en los vectores *routeKms* y *routeHeights*, la velocidad en cada punto en los vectores *logKms* y *logSpeeds*, y el número de intervalos de integración en *numSlices*. Esta función devolverá el gasto de combustible en la ruta.

   Integra sobre la distancia en la ruta. Para calcular la velocidad, aceleración y pendiente al principio de cada intervalo de integración:

- Atención a las unidades usadas en cada variable o parámetro. No mezcles metros con kilómetros ni km/h con m/s.
- Puedes calcular la velocidad en cualquier punto en el intervalo de d_i calculando el valor interpolado de la velocidad en d_i .
- Para calcular la aceleración, sólo necesitas la velocidad en d_i y un valor cercano anterior al mismo, por ejemplo $d_{i-1}=d_i-0.01$.
- Para calcular el ángulo de la pendiente (α en *rad*) del segmento entre los puntos $p_0=\langle x_0 y_0 \rangle$ y $p_1=\langle x_1 y_1 \rangle$, usa la siguiente fórmula:

$$\alpha=\arctan\frac{(y_1-y_0)}{(x_1-x_0)} \quad .$$

3. Escribe un script que calcule e imprima el consumo estimado de combustible para cada ruta-conductor. La salida podría ser:

```
Analyzing: Driver= driver1, Route= n1
Estimated fuel consumption: 8.383010 liters of fuel

Analyzing: Driver= driver2, Route= n1
Estimated fuel consumption: 8.611214 liters of fuel

Analyzing: Driver= driver1, Route= a1
Estimated fuel consumption: 9.009569 liters of fuel

Analyzing: Driver= driver2, Route= a1
Estimated fuel consumption: 9.324132 liters of fuel
```