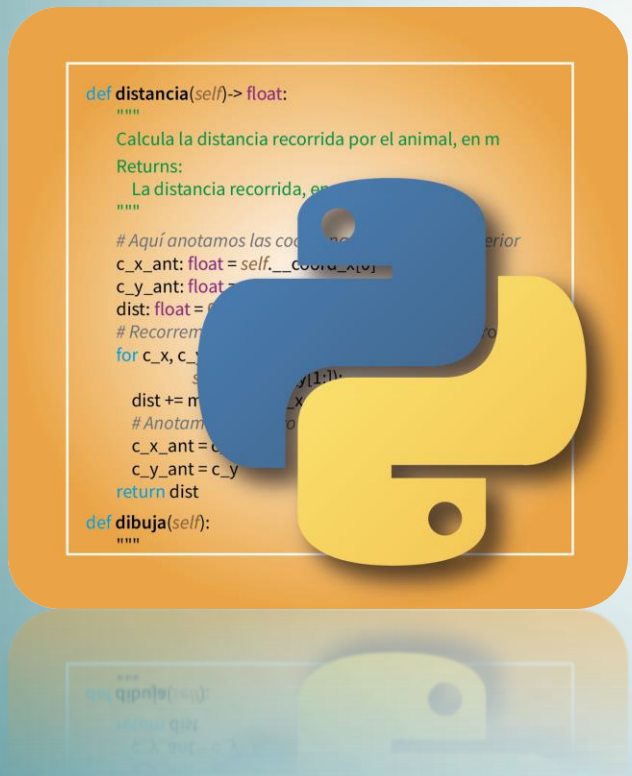
Programación

Práctica 9. Recorridos y búsquedas



Michael González Harbour

José Javier Gutiérrez García

José Carlos Palencia Gutiérrez

José Ignacio Espeso Martínez

Adolfo Garandal Martín

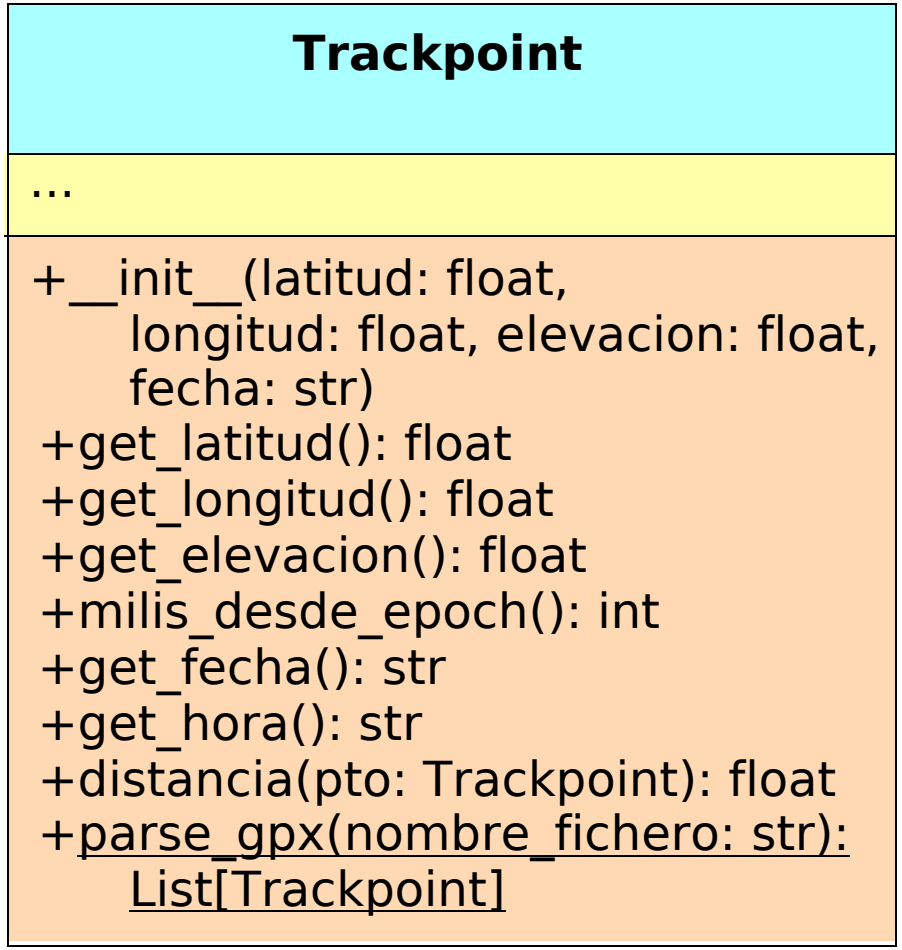
Departamento de Ingeniería Informática y Electrónica

Práctica 9

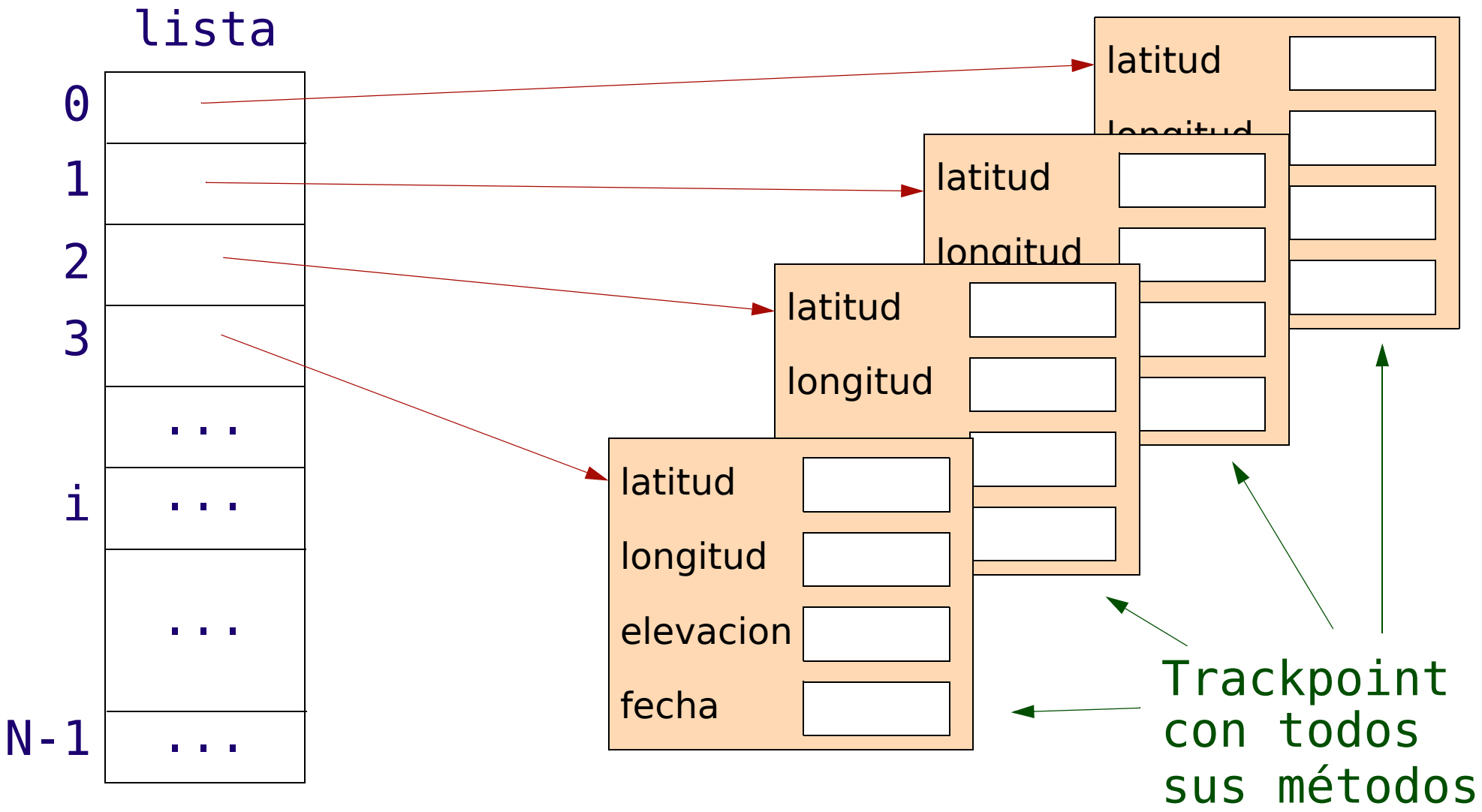
Objetivos: Practicar recorridos y búsquedas en una lista de objetos

Descripción: Se dispone de una clase que representa un punto de una ruta registrada mediante un GPS

La ruta completa es una lista de objetos de la clase `Trackpoint` y se puede leer de un fichero en formato GPX con `parse_gpx()`



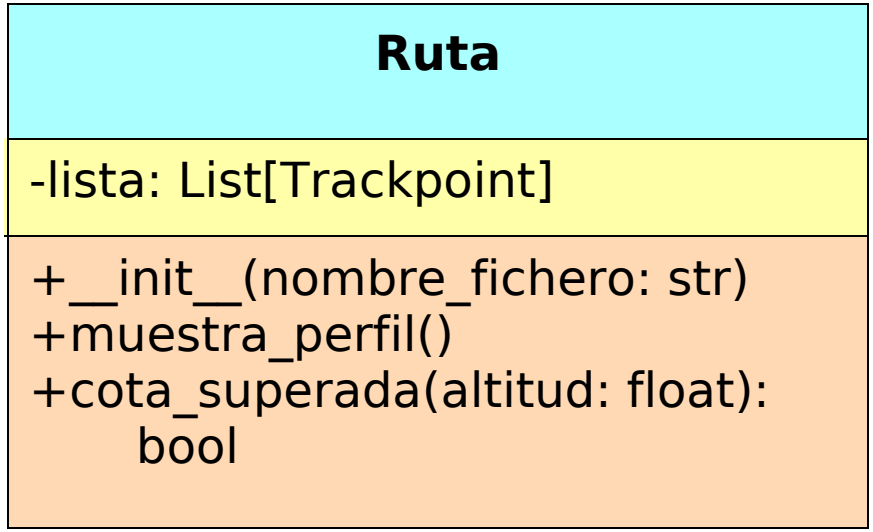
Estructura de la lista de Trackpoints



Clase **Ruta**

Se desea escribir el módulo `ruta_gps.py` que contiene la clase `Ruta`

Esta clase tiene como atributo una lista de objetos de la clase `Trackpoint`



Métodos:

- *Constructor*: introduce en el atributo los `Trackpoints` guardados en un fichero cuyo nombre se pasa como argumento
  - Usa para ello el método estático `parse_gpx()`
  - Se dispone de varios ficheros con rutas GPX para probar este constructor
  - Este constructor se da ya implementado

Clase **Ruta** (cont.)

`muestra_perfil`(): representa, utilizando `matplotlib.pyplot`, el perfil de la ruta, es decir, la elevación de cada punto en *m* frente a la distancia recorrida en *km*

- Para obtener la distancia recorrida, habrá que crear una variable acumuladora donde se vayan sumando las distancias entre el punto actual y el anterior a medida que se vaya recorriendo la lista
- Para calcular la distancia entre dos puntos se dispone del método `distancia`()

`cota_superada`(): retorna un valor lógico que indica si la ruta asciende por encima de un valor que se pasa como parámetro, en *m*

- hay que utilizar, obligatoriamente, un algoritmo de búsqueda

Programa principal

Se pide escribir un programa principal, contenido en otra clase, que haga lo siguiente:

- Crear un objeto de la clase `Ruta` a partir del fichero `Sierra de Hijar.gpx`
- Indicar si la ruta ha superado las cotas de 1000, 1500 y 2500 m
- Mostrar la gráfica del perfil de la ruta

Parte avanzada

Se pide añadir a la clase `Ruta` los dos métodos siguientes, y probarlos añadiendo al `main` instrucciones que los invoquen:

- `hay_punto_cercano`(): retorna un valor lógico que indica si algún punto de la ruta se encuentra a una distancia menor a la que la que se pasa como parámetro, en *km*, respecto al punto cuya latitud y longitud también se pasan como parámetros, en grados

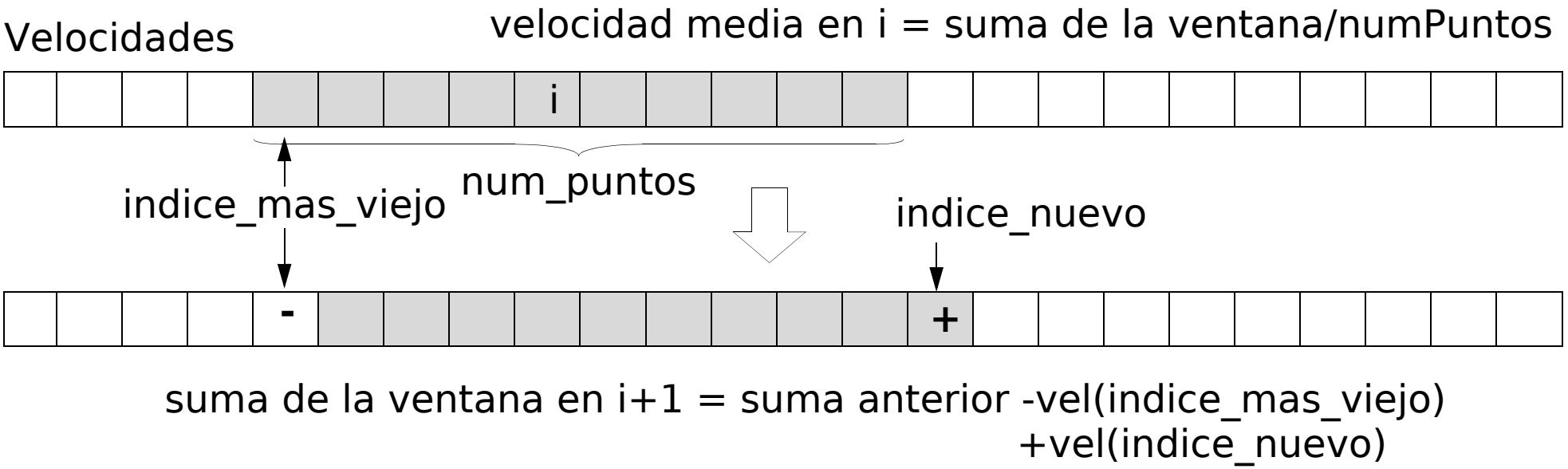
Ruta
-lista: List[Trackpoint]
+__init__(nombre_fichero: str) +muestra_perfil() +cota_superada(altitud: float): bool +hay_punto_cercano(latitud: float, longitud: float, dist_min: float): bool +velocidad_vs_tiempo(num_puntos: int)

Parte avanzada

- `velocidad_vs_tiempo`(): representa una gráfica de la velocidad, en *km/h*, en funcion del tiempo transcurrido, en *horas*
  - Recibe como argumento el número de medidas de velocidad que promedia para evitar fluctuaciones en la velocidad (típicamente, 10 o más)
  - La velocidad en cada punto es la distancia recorrida desde el punto anterior dividida entre la diferencia de tiempo entre ambos puntos
  - Si la ruta tiene un número de `Trackpoints` menor o igual que `num_puntos+2` se pone en pantalla un mensaje de error y no se pinta la gráfica
  - Para este método usar el pseudocódigo que se indica a continuación

Diseño del método velocidadVsTiempo

Utilizaremos la técnica de la "*ventana*" móvil para calcular de forma eficiente la media de las velocidades en el intervalo `num_puntos`



Para poder calcular bien la media se hace el cálculo desde `primer_indice=(num_puntos+1)//2` hasta `ultimo_indice=len(lista)-num_puntos//2 - 1`

Pseudocódigo

Definimos dos métodos privados:

```
# retorna el intervalo desde el punto i-1 al i en horas
método intervalo (i: int) retorna real
    retorna (milis_desde_epoch() del punto i de lista -
        milis_desde_epoch del punto i-1 de lista)/1000.0/3600.0
fin método

# retorna la velocidad entre los puntos i-1 e i en km/h
método velocidad (i: int) retorna real
    retorna (distancia entre los puntos i e i-1 de lista)/
        1000.0/intervalo(i)
fin método
```

Pseudocódigo (cont.)

```
método velocidad_vs_tiempo (num_puntos: int)
    # Comprobación de errores
    si tamaño de lista < num_puntos+2 entonces
        Mostrar mensaje de error
    si no
        tiempo_total: real = 0
        suma_velocidades: real = 0
        primer_indice: entero = (num_puntos+1)//2
        ultimo_indice: entero= tamaño de lista-num_puntos//2-1
        lista_tiempos: Lista[real] = lista vacía
        lista_vel : Lista[real] = lista vacía

        # Calcular el tiempo hasta antes del primer índice
        #  y sumar las velocidades de esos puntos
        para i desde 1 hasta primer_indice-1
            #añadimos el tiempo desde el pto anterior en horas
            tiempo_total = tiempo_total + intervalo(i)
            # sumamos la velocidad en Km/hora
            suma_velocidades = suma_velocidades + velocidad(i)
        fin para
```

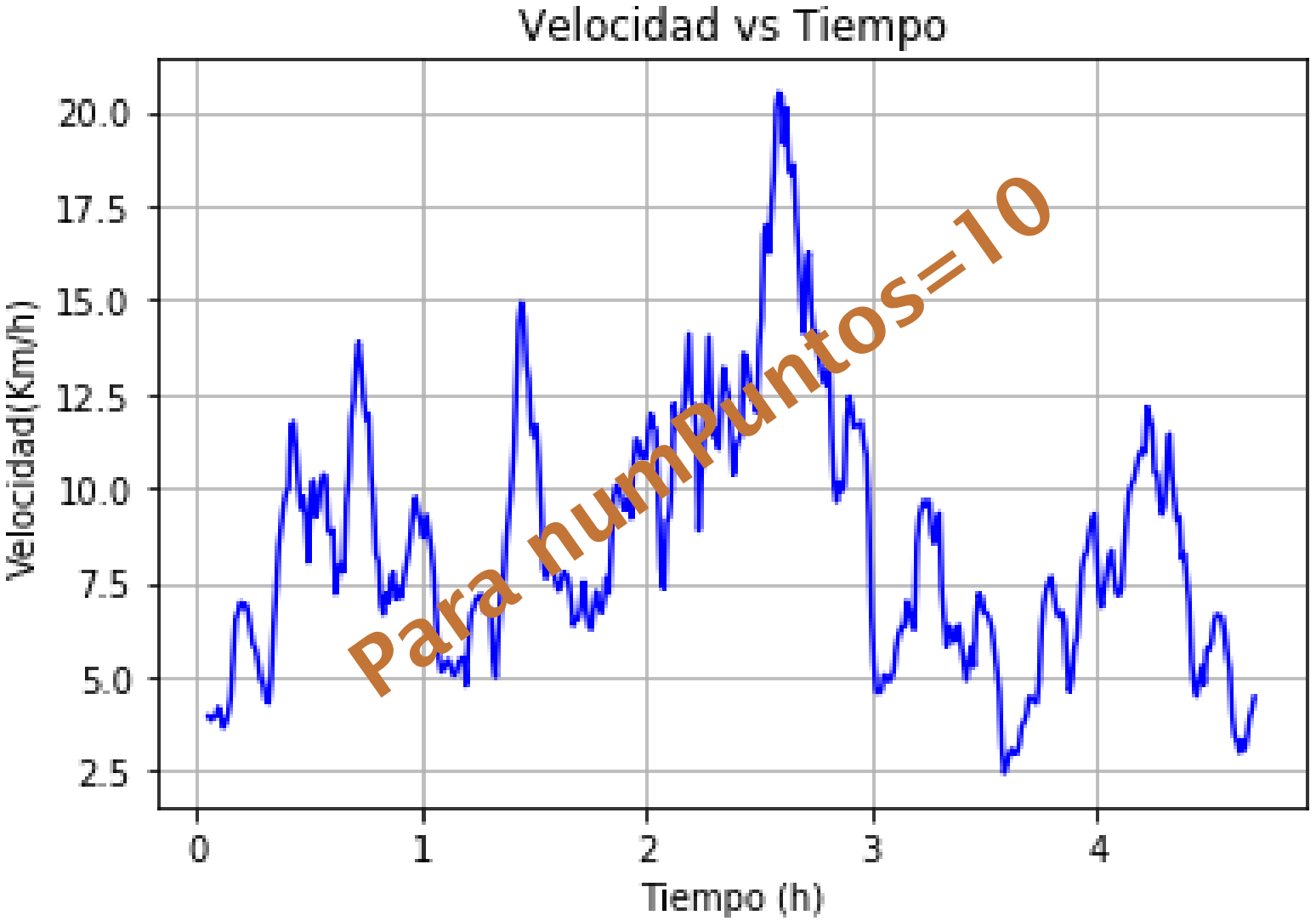
Pseudocódigo (cont.)

```
    # Añadir velocidades hasta completar num_puntos-1
    para i desde primer_indice hasta num_puntos-1
        suma_velocidades = suma_velocidades + velocidad(i)
    fin para

  # recorre los puntos entre primer_indice y ultimo_indice
    para i desde primer_indice hasta ultimo_indice
        # actualizar tiempo_total y la suma de velocidades
        tiempo_total = tiempo_total + intervalo(i)
        indice_nuevo: int = i+num_puntos//2
        suma_velocidades = suma_velocidades +
            velocidad (indice_nuevo)
        # calcular la velocidad media en este punto
        velo: real = suma_velocidades/num_puntos

        # meter el tiempo y velocidad en las listas
        añade tiempo_total a lista_tiempos
        añade velo a lista_vel
```

Pseudocódigo (cont.)

```
                # quitar a suma de velocidades la del pto mas viejo
                indice_mas_viejo: int = i-(num_puntos-1)//2
                suma_velocidades = suma_velocidades -
                    velocidad(indice_mas_viejo)
            fin para
            hacer un gráfico con lista_tiempos y lista_vel
        fin si
fin método
```

Parte avanzada (ejemplo de la gráfica)



Entregar 2 archivos

1. Código fuente del módulo
2. ***Informe*** con:
   - Una captura de pantalla de la gráfica del perfil de elevación
   - captura de los resultados del `main` indicando si la ruta ha superado las cotas de 1000, 1500 y 2500 m
   - *parte avanzada*: los resultados del método `hay_punto_cercano()` aplicado a los datos indicados en la plantilla del informe
   - *parte avanzada*: una captura de pantalla de la gráfica obtenida con el nuevo método `velocidad_vs_tiempo()`