# Examen de Programación
# Junio 2023
# (Grados en Física y Matemáticas)

**Primera parte (1.25 puntos por cuestión, 50% nota del examen)**

1) Escribir una función que permita obtener el pago a realizar para adquirir las entradas a un parque temático, según las tarifas indicadas a la derecha, y tomando como parámetros el número de entradas infantiles, juveniles y adultas. La función también recibe como parámetro la temporada en la que nos encontramos, expresada mediante uno de los strings `'Baja'`, `'Media'` o `'Alta'`. Se puede suponer que los valores de los parámetros siempre son correctos.

| | Temporada | | |
|---|---|---|---|
| | Baja | Media | Alta |
| Infantil | 6€ | 6€ | 6€ |
| Juvenil | 10€ | 14€ | 21,5€ |
| Adulto | 20€ | 24€ | 39€ |

   Se valora que la tabla de tarifas se exprese en una tabla bidimensional.

2) Escribir una función a la que se le pasa una tupla de números enteros con las edades de varias personas y que retorne otra tupla de tres elementos indicando cuántas personas son niños (entre 0 y 10 años de edad), jóvenes (entre 11 y 17 años) y adultos (a partir de 18 años). Se puede suponer que las edades de la tupla nunca son negativas.

3) Escribir una función que evalúe de forma aproximada la función integral senoidal usando el siguiente desarrollo en serie, válido para valores de $x$ pequeños:

$$\int \frac{\sin x}{x} dx \cong \frac{x}{1} - \frac{x^3}{18} + \frac{x^5}{600} - \frac{x^7}{35280} + \frac{x^9}{3265920} - \frac{x^{11}}{439084800}$$

   La función recibe como parámetro el valor de $x$. La función se escribirá mediante un bucle que calcule el sumatorio. Se creará una tupla que contenga los valores de los denominadores.

4) Escribir una función que calcule y retorne el nombre de la ciudad con el coste mínimo de vivienda dadas dos listas paralelas similares a las que se muestran, que se pasan como parámetros. La primera lista tiene nombres de ciudades y la segunda el coste de un metro cuadrado de vivienda, en €, en cada una de las ciudades de la primera lista.

```
['San Sebastián',   [4212,
 'Barcelona',        3579,
 'Madrid',           3576,
 'Bilbao']           2743]
```

   Si las listas son de tamaño distinto o están vacías se lanzará la excepción predefinida `ValueError`.

*Nota 1*: Se valora la eficiencia del código.

*Nota 2*: Se recuerda que es necesario documentar todas las funciones.

# Examen de Programación
# Junio 2023
# (Grados en Física y Matemáticas)

**Segunda parte (5 puntos, 50% nota del examen)**

Se dispone de un fichero que contiene datos para calcular el coste de operación de un coche a los 10 años de uso. Se desea hacer software para establecer comparativas entre los costes de operación de diversos vehículos y buscar el vehículo que mejor se adapta a nuestras necesidades.

Se dispone de un módulo llamado `coches.py` con la clase `DatosCoche` ya hecha. Los objetos de esta clase actúan como contenedores de los datos de un coche. Más adelante se muestra su diagrama de clases. El constructor recibe los datos del coche que son:

- modelo: El modelo del coche
- tipo: El tipo del motor `"E"`=Eléctrico, `"T"`=Térmico de gasolina
- precio: El precio de venta antes de impuestos, en €
- ayudas: Las ayudas del estado, en €
- reventa: El valor de reventa a los 10 años, en % sobre el precio
- impuestos: El % de impuestos a aplicar al precio
- consumo: El consumo por cada 100 km. Para coches eléctricos son kWh/100km y para coches térmicos de gasolina son litros/100km
- seguro: Coste anual del seguro
- mantenimiento: Coste anual del mantenimiento programado

También dispone de métodos observadores de los dos primeros atributos y de un método llamado `coste_operacion()` que calcula y retorna el coste total de operación del vehículo a los 10 años, en €, usando los datos del coche y los parámetros del método que son los kilómetros recorridos al año, el coste del combustible en €/litro, el coste de la electricidad en €/kWh y el coste de la financiación en % sobre el precio.

Se desea crear en el mismo módulo `coches.py` la clase `CosteCoches` con el siguiente diagrama de clases. Esta clase contiene como único atributo una lista de objetos de la clase `DatosCoche` y operaciones para analizar el coste de operación de los diversos coches. Sus métodos son:

- *constructor*: crea la lista de datos de los coches vacía.
- `lee()`: lee del fichero cuyo nombre se indica los datos del coste de varios coches, crea con ellos objetos de la clase `DatosCoche` y los añade al atributo `__lista`. El fichero contiene tres líneas de encabezamiento que se ignorarán, y una línea por coche con el nombre del modelo ocupando los primeros 16 caracteres de la línea (puede contener espacios en blanco) y el resto de los datos separados por uno o más espacios en blanco. Ejemplo con las primeras líneas (observar que los números reales están en formato español):

```
Coste de operación de diversos coches Eléctricos (E) o Térmicos (T)
Modelo         Tipo precio ayudas reventa impuestos consumo   seguro mantenim.
                E/T    €      €      %       %    l|kWh/100km  €/año   €/año
Tesla Model 3    E   48000  4500    30      21        15,5      800     500
Volkswagen ID.3  E   35000  4500    30      21        17,7      700     500
Renault Zoe      E   32000  4500    25      21        17,7      600     400
```

| DatosCoche |
| --- |
| -modelo: str<br>-tipo: str<br>-precio: float<br>-ayudas: float<br>-reventa: float<br>-impuestos: float<br>-consumo: float<br>-seguro: float<br>-mantenimiento: float |
| +__init__ (modelo: str,<br>tipo: str, precio: float,<br>ayudas: float, reventa: float,<br>impuestos: float, consumo: float,<br>seguro: float, mantenimiento: float)<br>+get_modelo(): str<br>+get_tipo(): str<br>+coste_operacion(km_anuales: int,<br>precio_combustible: float,<br>precio_electricidad: float,<br>coste_financiacion: float): float |

| CosteCoches |
| --- |
| -lista: list[DatosCoche] |
| +__init__()<br>+lee(nombre_fichero: str)<br>{exception NoEncontrado, ValueError}<br>+comparativa(tipo_de_motor: str,<br>combustible: float, electricidad: float,<br>financiacion: float)<br>{exception ValueError}<br>+modelo_coste_menor_que(<br>coste_maximo: float, km_anuales: int,<br>combustible: float, electricidad: float,<br>financiacion: float): Optional[str] |

Si el fichero no existe se lanzará automáticamente `IOError`. Si el fichero tiene errores se genera automáticamente `ValueError`. No trataremos `ValueError` pues supondremos que el fichero es correcto. Trataremos `IOError` lanzando la excepción `NoEncontrado`, ya definida en el mismo módulo.

*Pista*: para separar un string leído del fichero en sus datos usando uno o varios espacios en blanco, puede usarse el método `split()` de los strings sin parámetros.

- `comparativa()`: Si el parámetro con el tipo de motor no es "E" o "T" lanza `ValueError`. En caso contrario hace un listado del coste de operación de todos los modelos de coche de la lista cuyo tipo coincida con el indicado en el parámetro `tipo_de_motor`, para comparar los costes según varios recorridos anuales. El método recibe como parámetros el tipo de motor ("E" o "T"), el precio de la gasolina en €/litro, el precio de la electricidad en €/kWh y el coste de financiación en % sobre el precio del coche. Se usarán los parámetros para calcular el coste. Para este listado, primero se ponen dos líneas de encabezamiento como se muestra en el ejemplo. A continuación se pone un coche por línea, con su modelo y el coste de operación para 10000, 20000 o 40000 kms al año, en columnas. Ejemplo de las primeras líneas esperadas:

```
Coste de operación a los 10 años
Modelo                  10000 km/año    20000 km/año    40000 km/año
Tesla Model 3              60575           63210           68480
Volkswagen ID.3            46559           49568           55586
```

- `modelo_coste_menor_que`: Retorna el primer modelo de la lista cuyo coste de operación a los 10 años es menor o igual que el `coste_maximo` indicado. El método recibe como parámetros adicionales los km que se recorren anualmente, el precio de la gasolina en €/litro, el precio de la electricidad en €/kWh y el coste de financiación en % sobre el precio del coche. Si no existe ningún modelo que cumpla la condición de coste máximo se retorna `None`.

Se pide, además, escribir un programa principal que pruebe todos los métodos de la clase `CosteCoches` a partir de los datos del fichero llamado `"coste-coches.txt"`. Si se lanza `NoEncontrado` se deben abandonar todos los pasos restantes del programa y poner un mensaje de error. No es preciso tratar la excepción `ValueError`. Con una sola llamada a cada método es suficiente.

Valoración:

1) Constructor: (0,5 puntos)

2) `lee`: (1.5 puntos)

3) `comparativa`: (1 punto)

4) `modelo_coste_menor_que`: (1 punto)

5) `main`: (1 punto)