Examen de Programación
Julio 2021
(Grados en Física y Matemáticas)

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

1) Escribir una función que retorne el tipo marginal del impuesto del IRPF que corresponde a los ingresos de una persona que se pasan como parámetro real, de acuerdo a la tabla que se muestra. Si los ingresos son negativos, se retornará el 0%. El resultado se retorna como un número entero que representa un porcentaje (es decir, 19, 24, etc.). Además de los ingresos, la función recibe como parámetros dos listas: una de 5 elementos con los valores límite de los ingresos (columna 2 de la tabla) y otra de 6 elementos con los tipos marginales (columna 3 de la tabla).

Ingresos €		
Desde	Hasta	Tipo marginal
>=0	<=12450	19%
>12450	<=20200	24%
>20200	<=35200	30%
>35200	<=60000	37%
>60000	<=300000	45%
>300000		47%

2) Escribir una función Python que escriba en un fichero de texto cuyo nombre se le pasa como parámetro una tabla de conversión de temperaturas Farenheit a Celsius. Se escribirá un encabezamiento y luego en cada línea una temperatura Farenheit y su conversión a Celsius, con las temperaturas Farenheit en el intervalo de 0 a 100 grados, de diez en diez, y con un decimal, como en este ejemplo:

```
Farenheit Celsius
      0.0   -17.8
     10.0   -12.2
     20.0    -6.7
...
```

La conversión de temperaturas Farenheit (F) a Celsius (C) se hace con la expresión $C = (F - 32)/1.8$

3) El índice de Shannon H' es una medida de la diversidad de especies en un ecosistema. Se calcula como:

$$H' = -\sum_{i=1}^{R} p_i \ln(p_i)$$

siendo R el número de especies y p_i la proporción de individuos de la especie i (i entre 1 y R).

Se pide escribir una función que tome como parámetro una lista de números reales con los valores de p_i. La función debe retornar el índice de Shannon, H', aunque si la lista tiene menos de tres elementos se debe lanzar la excepción `DatosInsuficientesError`, ya importada.

4) Dentro del directorio del usuario se dispone de un directorio llamado `experimento` que a su vez contiene dos directorios, uno llamado `datos` y otro `medidas`. El directorio `datos`

contiene los datos de entrada de un experimento, consistentes en ficheros de texto cuyos nombres acaban en .txt. Similarmente, el directorio medidas contiene los resultados obtenidos del experimento, que son ficheros de texto (acabados en .txt). También puede contener otros ficheros.

Suponiendo que el directorio de trabajo inicial es el del usuario, y utilizando exclusivamente rutas relativas, escribir las órdenes linux/unix necesarias para:

a) crear en el directorio del usuario un nuevo directorio vacío llamado resultados

b) crear dentro de resultados un directorio vacío llamado experimento2

c) copiar todos los ficheros de texto de datos a experimento2

d) hacer lo mismo con los ficheros de texto contenidos en medidas

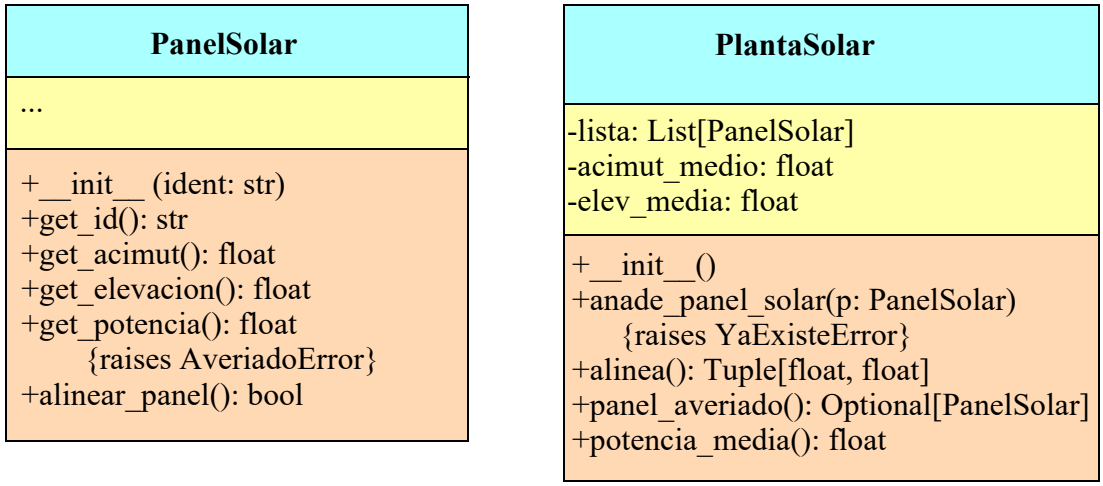
e) borrar todos los contenidos del directorio datos, pero mantener el propio directorio

f) borrar el directorio medidas con todos sus contenidos

Examen de Programación
Julio 2021
(Grados en Física y Matemáticas)

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

Se dispone en el módulo `panel.py` de una clase llamada `PanelSolar` que permite guardar la información y operaciones de un panel solar utilizado para producir electricidad. La clase responde al siguiente diagrama de clases, en el que se omiten los atributos por no ser relevantes para el problema. Sus métodos hacen lo siguiente:



- *constructor*: se le pasa el identificador del panel como parámetro.
- `get_id()`: Retorna el identificador del panel.
- `get_acimut()`: Retorna el acimut del panel, en grados, que nos indica hacia dónde apunta en la dirección horizontal. El Sur se representa con el valor 180 grados. Si el panel está averiado retorna `math.nan`.
- `get_elevacion()`: Retorna la elevación del panel, en grados, que nos indica hacia dónde apunta en la dirección vertical, sobre el horizonte. La vertical se representa con el valor 90 grados. Si el panel está averiado retorna `math.nan`.
- `get_potencia()`: Retorna la potencia eléctrica, en vatios, que el panel está produciendo en este instante. Lanza la excepción `AveriadoError` si el panel está averiado y no consigue generar electricidad.
- `alinear_panel()`: Alinea el panel con el sol. Retorna `True` si se ha conseguido alinear con el sol, y `False` en caso contrario, por ejemplo si fallan las comunicaciones o un motor.

En el mismo módulo `panel.py` se encuentra ya creada la excepción `AveriadoError`, que representa la condición de avería de un panel solar que le impide generar electricidad.

Lo que se pide es implementar el módulo `planta_solar.py` que contendrá tres elementos. En primer lugar, la clase `PlantaSolar`, que representa una planta compuesta por un conjunto de paneles solares, cada uno representado mediante un objeto de la clase `PanelSolar`. La clase `PlantaSolar` responde al diagrama de clases que se muestra arriba y debe contener los siguientes atributos privados:

- `lista`: un lista de objetos de la clase `PanelSolar`.
- `acimut_medio`: un número real que indica el acimut medio en grados.

- elev_media: un número real que indica la elevación media en grados.

Los métodos de la clase PlantaSolar deben hacer lo siguiente:

- *constructor*: crea la lista vacía y los atributos acimut_medio y elev_media con valor cero.
- anade_panel_solar(): Añade un nuevo panel solar, que se pasa como parámetro, al final de la lista, siguiendo los siguientes pasos. En primer lugar comprueba si ya existe un panel con el mismo identificador, y en ese caso lanza YaExisteError (para ello habrá que recorrer cada uno de los paneles que tenemos almacenados en la lista, comparando su identificador con el de p). Si no hubo errores (es decir, si no se lanza la excepción), añade el panel p al final de la lista.
- alinea: Alinea todos los paneles de la planta llamando a alinear_panel() para cada uno y, con los que resulten correctamente alineados y no estén averiados, calcula el acimut y elevación medios, almacenándolos en los respectivos atributos. Finalmente retorna en una tupla estos valores de acimut y elevación medios.

  Recordar que para saber si un panel se alinea correctamente se puede comprobar el valor retornado por alinear_panel(). Y para saber si está averiado, se puede mirar si el resultado de get_acimut() o get_elevacion() vale math.nan.
- panel_averiado(): Encuentra y retorna el primer panel averiado de la lista. Retorna None si no hay paneles que cumplan esa condición. Obligatoriamente se debe utilizar un algoritmo de búsqueda.

  Recordar que para saber si un panel está averiado, se puede mirar si el resultado de get_acimut() vale math.nan.
- potencia_media(): Retorna la potencia media de los paneles de la lista en vatios, o math.nan si algún panel está averiado. Debe tratarse la excepción AveriadoError lanzada por get_potencia(), retornando en ese caso math.nan.

Los otros dos elementos del módulo planta_solar.py que se piden, son la excepción YaExisteError, sin métodos, y un programa principal que pruebe todos los métodos de la clase PlantaSolar, comenzando por el constructor y la llamada a anade_panel_solar(). Si al añadir un panel solar se lanza YaExisteError se deben abandonar todos los pasos restantes del programa y poner en pantalla un mensaje de error. Con una sola llamada a cada método es suficiente.

Valoración:

1) Encabezamiento de la clase y constructor: 0,2 puntos

2) Excepción YaExisteError: 0,2 puntos

3) anade_panel_solar(): 0,8 puntos

4) alinea(): 1,2 puntos

5) panel_averiado(): 0,8 puntos

6) potencia_media(): 0.8 puntos

7) main: 1 punto