Examen de Programación
Septiembre 2019
(Grados en Física y Matemáticas)

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

1) Se desea escribir una función que retorne una tupla con el número de vocales y de consonantes de una rodaja de un string situada entre los índices i (incluido) y j (excluido). La función recibe como parámetros el string y los valores de i y j, que se suponen correctos (no hace falta comprobar errores). El string solo tiene letras mayúsculas o minúsculas y por tanto no tiene signos de puntuación, ni números ni espacios en blanco. Observar que con estas condiciones el número de consonantes es la longitud de la rodaja menos el número de vocales. Por simplicidad el alfabeto es el del inglés, para evitar las letras acentuadas.

2) La siguiente tabla muestra las tarifas de un parque de aventuras:

Tipo	Participante	Número mínimo de personas	Precio (euros)
Individual	niño mayor de 5 años	1	18.0
	adulto	1	23.0
Grupo	niño de 5 a 11 años	10	14.5
		15	13.0
	niño de 12 años o más	10	16.0
		15	15.0

Escribir una función a la que se le pasen como parámetros el número de adultos, el número de niños, y la edad de los niños (supuestamente son todos de la misma edad). La función retornará el precio total de las entradas, usando la tarifa más ventajosa para su caso. Si la edad de los niños es menor que 5 se retornará `math.nan`. Se valorará la eficiencia y tamaño compacto del código.

3) Escribir una función que calcule y retorne el valor de la función e^x dado por la siguiente expansión en serie:

$$e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

La función recibe como parámetros el valor de x y el número de términos del desarrollo en serie, n, numerándose los términos desde 0 hasta n. Por eficiencia se debe evitar usar el operador ** o las función `pow()` o `factorial()`. Para ello, el numerador del término i-ésimo del desarrollo en serie se debe obtener del numerador anterior multiplicándolo por x y el denominador será el anterior por i.

4) En un computador con sistema operativo Linux se desea escribir un *script* para recuperar ficheros de varios directorios llamados `2017`, `2018` y `2019` situados en un disco externo cuya ruta absoluta es `/mnt/disco1`. Los ficheros recuperados se meterán en el directorio `recupera` que deberá crearse en el directorio del usuario.

   El *script* debe realizar los siguientes pasos. Se deben utilizar rutas absolutas para acceder a los directorios del disco externo, y rutas relativas para los contenidos del directorio del usuario.

   - situarse en el directorio del usuario
   - crear el directorio `recupera`
   - mover a `recupera` los ficheros terminados en `.csv` que estén en `2017`
   - copiar en `recupera` los ficheros terminados en `.py` que estén en `2018`
   - borrar de `2019` los ficheros terminados en `.txt`
   - copiar en `recupera` los directorios llamados `practica1` y `practica2` (con todos sus contenidos) que están en `2018` y `2019`, respectivamente
   - borrar de `2017` el directorio `practica3`, con todos sus contenidos

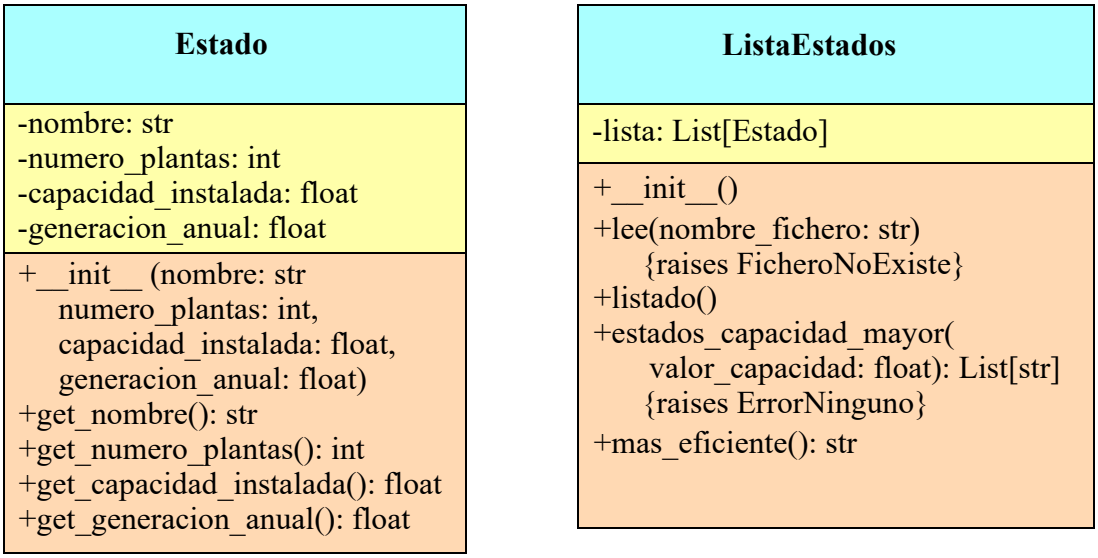
   Fases a seguir:

   - En primer lugar hacer un esquema de la distribución inicial de directorios.
   - En segundo lugar escribir el *script*.
   - En tercer lugar hacer un esquema de la distribución final de directorios y ficheros.

Examen de Programación
Septiembre 2019
(Grados en Física y Matemáticas)

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

Se dispone de un fichero con datos de las plantas solares instaladas en diversos estados de EEUU y se desea hacer parte del software que sirva para su análisis. Se dispone de la clase `Estado` ya realizada, que guarda los datos de las plantas solares de un estado concreto y responde al diagrama de clases que se muestra.



Los atributos de la clase se indican a continuación. La clase dispone también de un constructor al que se pasan los valores de los atributos y de un método observador para cada atributo:

- `nombre`: el nombre del estado
- `numero_plantas`: el número de plantas solares del estado
- `capacidad_instalada`: la potencia instalada del estado, en MW
- `generacion_anual`: valor de la energía total generada en el estado, en GWh

Se pide hacer la clase `ListaEstados` que responde al diagrama de clases que se muestra arriba. La clase dispondrá como atributo de una lista de objetos de la clase `Estado` y también tendrá diversos métodos para analizar las plantas solares:

- *constructor*: crea la lista vacía
- `lee()`: lee los datos de las plantas solares del fichero cuyo nombre se indica en el parámetro y los mete en la lista. El fichero contiene una línea de encabezamiento y luego los datos de cada estado, uno por cada línea, con sus campos (nombre, número de plantas, capacidad instalada en MW, potencia media por planta en MW y generación anual en GWh) separados por comas, como en el ejemplo que se muestra a continuación

```
State,Num Solar Plants,Inst. Capacity (MW),Avg MW Per Plant,Generation (GWh)
California,289,4395,15.3,10826
Arizona,48,1078,22.5,2550
Nevada,11,238,21.6,557
New Mexico,33,261,7.9,590
...
```

Los números reales están en formato inglés. Observar que hay un dato que habrá que ignorar, que es la potencia media por planta (4º dato de cada línea), ya que nuestra clase `Estado` no lo usa.

Con los datos de cada línea que no sea de encabezamiento se creará un objeto de la clase `Estado` y se añadirá a la lista. Se puede suponer que los datos del fichero son correctos.

Se deberá tratar la excepción que ocurre si el fichero no existe. En este tratamiento se pondrá un mensaje de error en pantalla y seguidamente se lanzará la excepción `FicheroNoExiste`, ya definida en el mismo módulo.

- `listado()`: muestra en pantalla un listado de todos los datos de la lista, en columnas, con la capacidad y la generación redondeadas a un decimal. El listado comenzará por un encabezamiento de una o dos líneas que explique los datos que vendrán a continuación. El encabezamiento incluirá el número de elementos de la lista.
- `estados_capacidad_mayor()`: calcula y retorna la lista de los nombres de los estados de la lista cuya capacidad instalada sea mayor que el parámetro `valor_capacidad`, que está en MW. Si no hubiese ningún estado que cumpla esta condición se lanzará la excepción `ErrorNinguno`, ya definida en el mismo módulo.
- `mas_eficiente()`: retorna el nombre del estado cuyas plantas solares son más eficientes en promedio. La eficiencia de las plantas de un estado es

$$eficiencia = \frac{g \cdot 1000}{c \cdot 12 \cdot 365}$$

siendo g su generación anual y c su capacidad instalada. Si la lista estuviese vacía se retorna `None`.

Finalmente, se pide hacer un programa principal en el mismo módulo, que haga lo siguiente:

a. Crea un objeto de la clase `ListaEstados`

b. Invoca al método `lee()` para leer los datos del fichero `"solar_plants.csv"`

c. Hace un listado invocando al método `listado()`

d. Muestra en pantalla los estados de capacidad instalada mayor que 500 MW.

e. Muestra el nombre del estado cuyas plantas solares son más eficientes.

Tratamiento de errores a realizar en el programa principal:

- Si en el paso b) se lanzase `FicheroNoExiste` se abandonan los restantes pasos y se escribe el mensaje `"El programa no se puede ejecutar"`
- Si en el paso d) se lanzase `ErrorNinguno` se pone un mensaje de error y luego se continúa normalmente con el paso e)

Valoración:

- encabezamiento de la clase y constructor: 0.5 puntos
- métodos `lee()` y `estados_capacidad_mayor()`: 1 punto cada uno
- método `listado()` y `mas_eficiente()`: 0.75 puntos cada uno
- programa principal: 1 punto