Informática 2

- Pruebas de evaluación –

Grado en Ingeniería en Sistemas Audiovisuales y Multimedia

Curso 2023-2024

Universidad Rey Juan Carlos

Índice de contenidos

Enunciados de exámenes

Este documento contiene los enunciados de todos los exámenes realizados en evaluación continua en el curso 2023-2024, y que corresponden con las actividades de evaluación descritas en la guía de estudio disponible en acceso abierto en la BURJC online:

- Examen 1 – Examen parcial 1 de la guía de estudio
- Examen 2 – Examen parcial 2 de la guía de estudio
- Examen 3 – Examen parcial 3 de la guía de estudio

Soluciones de exámenes

Las soluciones a los tres exámenes se pueden descargar del archivo .zip que acompaña a este documento en el *handle* alojado en la Biblioteca URJC Digital.

Examen 1

La entrega de este examen se realizará a través de la tarea abierta para tal efecto en Aula Virtual, donde se subirán dos archivos .py (*ejercicio1.py*, *ejercicio2.py*) que deberán implementar las funcionalidades descritas en cada uno de los ejercicios.

Se recuerda que no se puede utilizar ningún tipo de material de ayuda, ni de este curso ni de otros. Su uso implicará la no corrección del examen, además de las medidas disciplinarias correspondientes.

Ejercicio 1 (5 puntos)

Se quiere desarrollar un programa para gestionar los datos de los trabajadores de una empresa. Para almacenar la información, se utilizará una lista llamada "lista_trabajadores". Como requisito, cada trabajador en la lista se representará como un diccionario con las claves "nombre" y "nómina", cuyos valores serán el nombre del trabajador, y un entero representando su nómina mensual (en euros), respectivamente.

Para ello, se pide implementar funciones que realicen las siguientes acciones:

1. Agregar trabajador: Esta función tomará como entrada, al menos, el nombre de un nuevo trabajador (proporcionado como una cadena de caracteres) y la cantidad en euros de su nómina, y añadirá el nuevo trabajador a "lista_trabajadores". Después de agregar el trabajador, devolverá el número total de trabajadores presentes en la lista.
2. Visualizar ejecutivos: Deberá mostrar aquellos trabajadores con salario mayor a 3000€ al mes. La visualización se hará de forma que cada trabajador se muestre en una línea, y cada línea muestre el identificador del trabajador (que se corresponderá con la posición que éste ocupa en la lista) y su nombre.

Una vez implementadas estas funciones, se pide usarlas en el programa para realizar las siguientes acciones. En primer lugar, añadir cuatro trabajadores predefinidos:

Nombre	Nomina
Alejandra Martínez	3250
Víctor Pérez	1680
William Adams	2400
Lucy Brown	4750

Los trabajadores deben añadirse en el orden que se muestra. No deben pedirse al usuario por teclado. Además, después de añadir el último de ellos, debe mostrarse por pantalla el número de trabajadores presentes en la lista, utilizando para ello el valor devuelto por la función encargada de agregarlos.

Finalmente, se visualizarán los trabajadores ejecutivos usando la función desarrollada.

(Ejercicio 2 en la otra cara)

Ejercicio 2 (5 puntos)

Se pide crear un programa que pida al usuario que introduzca seis números enteros por teclado, y los almacene en una lista.

Posteriormente, pedirá al usuario un número N entre 2 y 6 y, tras comprobar que es válido, llamará a una función que calculará la media y el máximo de **únicamente los N primeros valores** en la lista. Esta función debe ser implementada como parte del ejercicio, y debe devolver una tupla con ambos valores (media y máximo). Dentro de esta función no se podrán utilizar bucles (for / while); en su lugar, se utilizarán las funciones predefinidas en Python sum() y max(), que toman como argumento un iterable (tupla, lista, etc.) y devuelven la suma y el máximo de sus elementos, respectivamente.

Finalmente, se mostrará por pantalla el valor máximo de la sublista, y se escribirá a un fichero su valor medio. El fichero tendrá como nombre *información.txt* y deberá crearse en la carpeta del proyecto, junto al fichero .py del propio programa.

NOTA: Conviene recordar que existen funciones para transformar variables de un tipo a otro:

- str(): Para transformar un entero, flotante,… a cadena de caracteres (string)
- int(). Para transformar a entero una cadena de caracteres (string), truncar un flotante…

Examen 2

La entrega de este examen se realizará a través de la tarea abierta para tal efecto en Aula Virtual, donde se subirán los archivos .py que deberán implementar las funcionalidades descritas en cada uno de los ejercicios.

Para resolver el ejercicio 2 se puede utilizar el documento "info_sockets.pdf", disponible en Aula Virtual, que incluye una breve descripción de las funciones de la librería socket. El uso de cualquier otro tipo de material de ayuda, de este curso o de otros, implicará la no corrección del examen, además de las medidas disciplinarias correspondientes.

Ejercicio 1 (5 puntos)

Se pide desarrollar un sistema básico para que un zoológico pueda gestionar diferentes tipos de animales.

a) En primer lugar, crea clases en Python para diferentes tipos de animales. Crea una clase base **Animal** con atributos **nombre** y **especie**. Luego, crea dos subclases: **Mamífero** y **Ave**. La clase **Mamífero** deberá tener un atributo adicional **peso** para representar el peso del animal, y la clase **Ave** deberá tener un atributo adicional **envergadura** para representar su tamaño.
b) Después, implementa un método **calcular_comida** en las clases **Mamífero** y **Ave**, que devuelva la cantidad de comida consumida por cada animal en un mes, teniendo en cuenta que los mamíferos comen una cantidad que viene dada por 0,1 veces su peso, y las aves 0,05 veces su envergadura (Nota: no tener en cuenta las unidades).
c) A continuación, crea una clase **Zoo** que cuente con un atributo **animales**, el cual contendrá una lista de instancias tanto de **Mamífero** como de **Ave**.
d) Para rellenar este atributo, añade un método **añadir_animal** dentro de la clase **Zoo**, que tome como argumento una instancia de **Mamífero** o **Ave** y lo añada a la lista.
e) Crea un método **calcular_consumo_comida** dentro de la clase **Zoo** que muestre por pantalla la suma de la comida consumida por todos los animales del zoo.

Finalmente, usa las clases implementadas para incorporar la información de los siguientes animales (**Mamífero** y **Ave)** a un objeto de la clase **Zoo** y llama al método **calcular_consumo_comida** para mostrar el peso total de comida consumida en un mes en el zoo:

Nombre	**Especie**	**Peso**
Dumbo	Elefante	5000
Simba	León	250
Atila	Halcón	2,5
Charlie	Loro	0,15

(Ejercicio 2 en la otra cara)

Ejercicio 2 (5 puntos)

Se proporcionan dos scripts de Python: **udp_server.py** y **udp_client.py**, que actualmente utilizan UDP para la comunicación. Se pide:

- Convertir el servidor y el cliente de UDP a TCP. Modifica el código para crear sockets TCP y establecer una conexión entre ellos.
- Modificar el servidor para manejar conexiones de diferentes clientes de manera secuencial. Cuando un cliente se desconecte, el servidor debe estar listo para aceptar y gestionar una nueva conexión de otro cliente.
- Modificar el programa para que el servidor tome el puerto como argumento de línea de comandos. De la misma forma, el cliente deberá recibir la IP y el puerto al que conectarse por terminal.

Examen 3

La entrega de este examen se realizará a través de la tarea abierta para tal efecto en Aula Virtual, donde se subirá un .zip que contenga una carpeta por cada ejercicio resuelto, incluyendo los archivos .py que implementen las funcionalidades descritas en el enunciado correspondiente.

Para resolver el Ej. 2 se puede utilizar el documento "info_sockets.pdf" (en Aula Virtual), que incluye una breve descripción de las funciones de la librería socket. Usar cualquier otro material de ayuda implicará la no corrección del examen, además de las medidas disciplinarias correspondientes.

Ejercicio 1 (2 puntos)

Desarrolla clases en Python para modelar una biblioteca. La biblioteca almacena dos tipos de recursos: libros y revistas. Cada tipo de recurso tiene sus propios atributos: los libros tienen un título, un autor y un número de páginas, mientras que las revistas tienen un título, un autor y una edición. Ambos tipos de recursos deben tener un método para mostrar sus metadatos. Es obligatorio utilizar herencia para minimizar la repetición de código

La biblioteca debe tener una estructura de datos que almacene todos los recursos. Esta estructura de datos debe ser un diccionario, donde las claves son los rangos de iniciales de los autores ('A-I', 'J-R', 'S-Z') y los valores son listas de instancias de esos recursos. Se asume que todos los autores tienen nombres que comienzan con una letra del alfabeto inglés.

El programa debe incluir un método para agregar recursos a la biblioteca. Este método debe aceptar un recurso como argumento y agregarlo a la lista correspondiente en el diccionario, en función de la inicial del nombre de su autor.

Finalmente, el programa debe tener un método para mostrar todos los recursos en la biblioteca. Este método debe iterar sobre todas las entradas en el diccionario de recursos y, para cada recurso, imprimir los metadatos del recurso.

El siguiente ejemplo del funcionamiento esperado del programa debe usarse para probar las clases implementadas:

```
libro1 = Libro('Don Quijote de la Mancha', 'Miguel de Cervantes', 200)
revista1 = Revista('National Geographic', 'John Doe', 'Edición de Enero de 2023')

biblioteca = Biblioteca()

biblioteca.agregar_recurso(libro1)
biblioteca.agregar_recurso(revista1)

biblioteca.mostrar_recursos()
```

La salida del ejemplo anterior se muestra a continuación:

```
--- A-I ---
--- J-R ---
Don Quijote de la Mancha, de Miguel de Cervantes (200 pág.)
National Geographic, de John Doe (Edición de Enero de 2023)
--- S-Z ---
```

(Ejercicio 2 en la otra cara)

Ejercicio 2 (4 puntos)
Usando sockets, crear una aplicación formada por un cliente y un servidor TCP para jugar al juego *Wordle* donde el cliente debe adivinar una palabra de cinco letras en cinco intentos.

Al arrancar, el servidor quedará a la espera de que un cliente se conecte. Cuando esto suceda, lanzará un hilo que manejará las interacciones del juego.

El cliente, una vez establecida la conexión, dispondrá de cinco intentos para adivinar la palabra. En cada intento, deberá asegurarse de que el usuario introduce una *string* de longitud 5. En caso contrario, preguntará de nuevo hasta que se introduzca una palabra correcta. Después, enviará dicha palabra al servidor para su validación.

El servidor, al recibir la palabra del cliente, deberá responder con un *string* de longitud 5, cuyos caracteres indicarán el grado de acierto de la letra en su misma posición (no envía la solución):

- Si la palabra a adivinar contiene la letra, y además está en su sitio, se indicará con una 'X'
- Si la palabra a adivinar contiene la letra pero en otra posición, se indicará con una '?'
- Si la letra no está contenida en la palabra, se indicará con un '-'

Ejemplo (clave: JAMON):

Intento: JUDIA -> Respuesta: X---?

Intento: limon -> Respuesta: --XXX

Intento: Monos -> Respuesta ???X-

Como se puede ver en el intento MONOS, no se debe tener en cuenta si una letra se repite (como sí se hace en el juego original) y marcar, en este caso, la primera O como fallo (-). Además, el servidor no diferenciará mayúsculas de minúsculas (el método *upper()* puede ser de utilidad).

Al recibir la respuesta, el cliente la mostrará por pantalla para que el usuario pueda continuar adivinando la palabra si le restan todavía intentos. Si la ha adivinado, le dará la enhorabuena. Por el contrario, si se le han terminado los intentos y no ha adivinado la palabra, le despedirá.

El cliente termina su conexión cuando se le acaban los intentos o el cliente acierta. El servidor seguirá funcionando a la espera de nuevos clientes con los que jugar.

Ambos programas deben recibir por argumentos de línea de comandos la IP y el puerto en el que escuchará conexiones el servidor. Adicionalmente, el script del servidor se lanzará con un tercer parámetro que será la palabra a adivinar (la clave) por el cliente. Ejemplo de ejecución:

En una terminal: python3 wordle_server.py 127.0.0.1 5556 jamon

En otra terminal: python3 wordle_client.py 127.0.0.1 5556

A continuación, se muestran las terminales correspondientes a dos posibles partidas:

Partida 1:

```
Intento 1. Introduzca una palabra de 5 letras: mangos
Palabra invalida. Introduce una palabra de 5 letras: monja
Respuesta: ?????
Intento 2. Introduzca una palabra de 5 letras: jamon
Respuesta correcta! Enhorabuena
```

Partida 2:

```
Intento 1. Introduzca una palabra de 5 letras: toser
Respuesta: -?---
Intento 2. Introduzca una palabra de 5 letras: clavo
Respuesta: --?-?
Intento 3. Introduzca una palabra de 5 letras: argon
Respuesta: ?--XX
Intento 4. Introduzca una palabra de 5 letras: tazon
Respuesta: -X-XX
Intento 5. Introduzca una palabra de 5 letras: vagon
Respuesta: -X-XX
Casi... la palabra era JAMON
```

(Ejercicio 3 en la otra cara)

Ejercicio 3 (4 puntos)

Un palíndromo es una palabra que se lee igual hacia adelante que hacia atrás. Son palíndromos, por ejemplo, *radar*, *arra* y *reconocer*.

Escribe un programa que solicite al usuario ingresar una palabra por teclado y le indique por pantalla si la palabra introducida es un palíndromo o no.

Para ello, se deberá implementar la función *es_palindromo(palabra)*, que debe devolver *True* si la palabra recibida por parámetro es un palíndromo y *False* en caso contrario. Dicha comprobación se hará utilizando una estructura de datos Pila, que también deberá ser implementada. La clase Pila deberá contener, al menos, los métodos *push* y *pop* para agregar y quitar elementos de la misma.

A continuación, se muestra la salida por terminal de dos posibles ejecuciones del programa, donde la línea que empieza por “>” indica que el usuario introduce la palabra por teclado:

```
Introduzca una palabra:
> examen
"examen" no es un palíndromo.
```

```
Introduzca una palabra:
> radar
"radar" es un palíndromo.
```