EXAMEN FINAL DE PROGRAMACIÓN II
MAYO 2024

Lee detenidamente las normas antes de comenzar:

- *El fichero entregado debe ser un fichero válido de Python (NO una sesión interactiva de Jupyter), es decir, debe tener la extensión .py.*
- *Debéis añadir todas las funciones de los ejercicios al fichero plantilla.py.*
- *Las funciones implementadas deben respetar la signatura definida en el ejercicio (nombre, parámetros y valores de retorno correctos).*
- *La entrega se encuentra abierta en https://pracdlsi.dlsi.ua.es/*
- *Recordad que la red puede ir lenta, y que el servidor se cierra exactamente a la hora de finalización del examen:* ***reservad al menos 5 minutos*** *al final del examen para la entrega.*
- *No olvidéis indicar al inicio del fichero vuestro nombre mediante un comentario Python.* ***No uses ningún acento ni ñ.***
- *En el cuerpo de las funciones* ***no debe haber ninguna sentencia de tipo input ni print.***
    - *Si queréis que el ejercicio incluya vuestro código de prueba, lo debéis poner al final del archivo .py, dentro del bloque, incluido en la plantilla del examen:*

```
if __name__ == '__main__':
    # Aqui tu codigo de prueba
```

    - *Notad cómo tanto name como main van precedidos y sucedidos de DOS guiones bajos.*
- *Podéis entregar el fichero .py tantas veces como queráis hasta que se cierre el servidor de prácticas.* ***Se contará como válida exclusivamente la última entrega realizada****.*
- *No entreguéis código que contenga errores sintácticos: abortará el corrector y eso supondrá un 0 en el ejercicio.*
- *Podéis ir* ***guardando*** *vuestro código en la* ***unidad D: o E:*** *(cambia según el ordenador usado). De este modo, si se os reiniciase por cualquier motivo, no se perderá vuestro trabajo.* ***Borra todo*** *el contenido de esa unidad al acabar el examen.*
- *Si se detecta que el código entregado ha sido* ***plagiado*** *se procederá a actuar según el artículo 14 del Reglamento para la Evaluación de los Aprendizajes de la Universidad de Alicante (BOUA 9/12/2015)*
- *Tienes las transparencias de clase de teoría disponibles en https://www.dlsi.ua.es/~materuel/prog2.zip. Contraseña:* `Prog2-IA`

Ejercicio 1 (5 puntos)

Programa un gestor de contenedores de transporte Orientado a Objetos[1]. Para ello:

1. **(0.5 puntos)** Crea la clase `Paquete`, la cual tendrá los atributos de instancia `contenido` (el contenido del paquete, en formato `str`) y `peso` (el peso del paquete, en formato `int` o `float`, que se guardará en `float`). Si se proporciona un tipo de datos incorrecto a estos atributos, se elevará una excepción `TypeError`.
2. **(0.25 puntos)** Estos atributos se almacenarán como atributos privados en la clase, usando la convención de nomenclatura adecuada.
3. **(0.25 puntos)** Al imprimir por pantalla un objeto de la clase `Paquete`, se mostrará en formato `contenido (peso)`. Por ejemplo:

```
Zapatillas (0.5)
```

4. **(0.5 puntos)** La clase Paquete tendrá la propiedad `peso`, la cual tendrá asociados un método *setter* y un método *getter*.
5. **(0.5 puntos)** Crea la clase `Contenedor`, la cual tendrá los atributos de instancia `origen` (de donde viene el contenedor, en formato `str`), `destino` (donde se dirige el contenedor, en formato `str`) y `paquetes` (atributo de instancia opcional consistente en una lista de objetos de tipo `Paquete` que por defecto estará inicializada a `None`). Si se proporciona un tipo de datos incorrecto a estos atributos, se elevará una excepción `TypeError`.
6. **(0.5 puntos)** Al imprimir por pantalla un objeto de la clase `Contenedor`, se mostrará su origen, destino y contenido en el formato de ejemplo siguiente:

```
Alicante -> Albacete
Zapatillas (0.5)
Pantalones (0.4)
Camiseta (0.3)
```

7. **(0.5 puntos)** La clase `Contenedor` tendrá la propiedad peso, la cual tendrá un método *getter* asociado que proporcionará el peso total del `Paquete` en formato `float`, redondeado a dos cifras decimales.
8. **(0.5 puntos)** La clase `Contenedor` implementará el protocolo de secuencia de Python.
9. **(0.5 puntos)** Podremos añadir objetos de la clase `Paquete` a la clase `Contenedor` usando el operador +=
10. **(0.5 puntos)** Podremos añadir objetos de la clase `Paquete` a la clase `Contenedor` usando el operador +
11. **(0.5 puntos)** El operador + para añadir objetos de la clase `Paquete` al `Contenedor` será matemáticamente correcto (propiedad conmutativa de la suma)

---

[1] Recordad que debéis hacer copia de los `Paquetes` en los métodos que creen y devuelvan un nuevo `Contenedor`

Código de prueba:

```
if __name__ == '__main__':
    c1 = Contenedor('Alicante', 'Albacete', paquetes=[Paquete('Gorra', 0.2)])
    p1 = Paquete('Zapatillas', 0.5)
    p2 = Paquete('Pantalones', 0.4)
    c1 += p1
    c1 += p2
    p3 = Paquete('Camiseta', 0.3)
    c2 = p3 + c1
    p1.peso = 0.6
    print(c1)
    print(c2)
    print(c2.peso)
```

Salida:

```
Alicante -> Albacete
Gorra (0.2)
Zapatillas (0.6)
Pantalones (0.4)

Alicante -> Albacete
Gorra (0.2)
Zapatillas (0.5)
Pantalones (0.4)
Camiseta (0.3)

1.4
```

Ejercicio 2 (2 puntos)

Crea un decorador `@log`. Cuando decoremos una función con él, añadirá a un fichero `log.txt` la salida de cada una de las llamadas a dicha función (en formato `str`) en una nueva línea cada vez.

Código de prueba:

```
if __name__ == '__main__':
    @log
    def test(cadena):
        return (cadena)

    test('prueba1')
    test('prueba2')
    with open('log.txt') as f:
        print(f.read())
```

Salida:

```
prueba1
prueba2
```

Ejercicio 3 (1.5 puntos)

Crea una función `filtro(a, b, c)` que devuelva una función lambda con un parámetro. Esta función lambda, a su vez, devolverá

- El parámetro multiplicado por `c` si el parámetro es menor que `a`
- El parámetro sin alterar si el parámetro está comprendido entre `a` y `b` (incluidos)
- El parámetro dividido por `c` si el parámetro es mayor que `b`

Código de prueba:

```
if __name__ == '__main__':

    def procesar(func):
        return list(map(func, list(range(1, 10))))

    print(procesar(filtro(3, 6, 2)))
```

Salida:

```
[2, 4, 3, 4, 5, 6, 3.5, 4, 4.5]
```

Ejercicio 4 (1.5 puntos)

Crea una función `validar_contrasena(contrasena)` que valide contraseñas seguras, basándose en los siguientes criterios:

1. La contraseña debe tener al menos 8 caracteres de longitud.
2. Debe contener al menos una letra mayúscula.
3. Debe contener al menos una letra minúscula.
4. Debe incluir al menos un número.
5. Debe tener al menos un carácter especial de los siguientes: `!@#$%^&*()`
6. No puede haber 3 letras mayúsculas seguidas, 3 letras minúsculas seguidas ni 3 dígitos seguidos.

La función debe recibir un `str` como argumento y devolver un `int`, siendo este `0` si la contraseña es segura según los criterios mencionados, o el número correspondiente al primer criterio no cumplido.

Por ejemplo, `validar_contrasena("ASDFGqwerty")` devolvería `4`, ya que cumple con los criterios 1, 2 y 3, pero no con el 4 (los criterios 5 y 6 no sería necesario analizarlos por incumplir uno de los anteriores).

Código de prueba:

```
if __name__ == '__main__':
    print(validar_contrasena("PaSsWoRd12!"))
    print(validar_contrasena("pass123"))
    print(validar_contrasena("password"))
    print(validar_contrasena("PASSWORD"))
    print(validar_contrasena("PASSword"))
    print(validar_contrasena("PASSword1234"))
    print(validar_contrasena("PASSword1234!"))
```

Salida:

```
0
1
2
3
4
5
6
```