MODALIDAD B. INDIVIDUAL (PYTHON)

IDENTIFICADOR DEL ALUMNO:

HORA EXACTA COMIENZO PRUEBA:

HORA EXACTA FINALIZACIÓN PRUEBA:

INSTRUCCIONES DE LA PRUEBA

En el escritorio puede encontrar un directorio llamado Olimpiada2022. Dentro de éste, habrá dos directorios.

El primero, Software Olimpiada, contiene los entornos de ejecución necesarios. Entre ellos, Python.

El segundo llamado Participante Olimpiada, contiene directorios C++, Java y Python, en función del lenguaje deseado. Dentro del directorio **Python** podrá encontrar los ficheros de validación que se comentan en el enunciado y es en este directorio (Python) donde debe escribir sus soluciones.

FORMA DE TRABAJO

- Arranque el entorno que desee, pulsando dos veces en el enlace Python que puede encontrar en el directorio Software Olimpiada.
- En el editor que se arranca, deben escribir en los ficheros solución con el nombre indicado. No intente pulsar dos veces en el icono del fichero para abrirlo. Hágalo siempre desde el editor.
- Para probarlo, utilice los ficheros **`testEjercicio`**.
- Edítelos, puede ampliarlos como desee, escribiendo más casos de prueba.
- Para probar su solución, debe ejecutar los ficheros de test. Para ello, simplemente pulse el **botón de Ejecución.**
- Los ficheros de solución deben guardarse en el directorio Participante Olimpiada/Python, junto con los ficheros de validación.
- El último ejercicio debe solucionarse en la hoja de instrucciones que **deberá ser entregada antes de abandonar el aula.**

CUANDO TERMINE LA PRUEBA

- Renombre el directorio Participante Olimpiada.
- Póngale como nombre el de su identificador de participante.
- Devuelva estas instrucciones al profesor, que anotará la hora en que ha terminado la prueba.

DESCRIPCIÓN DE LA PRUEBA

Esta prueba consta de cinco (5) ejercicios, de dificultad variada.

Cada uno de los ejercicios del 1 al 4 debe implementarse en un fichero con nombre *`ejercicio1.py ejericio2.py … .`*

Dentro debe implementar una función. El nombre de esta función se especifica en cada ejercicio, así como los parámetros y el tipo de retorno. Por supuesto, puede implementar cualesquiera otra funciones o métodos que necesite. Pero es obligatorio implementar la que se solicita.

Cada estudiante dispone de **90 minutos** para solucionar, de forma individual, estos retos, excepto los alumnos de ESO, que disponen de 90+15 minutos.

En los días siguientes, los profesores valorarán las soluciones de las distintas sedes y otorgarán distinta puntuación a los alumnos que la hayan superado, ateniéndose a los siguientes criterios:

a) **La corrección de la solución**. Aquellas soluciones que funcionen correctamente o estén muy cerca de hacerlo, serán mejor puntuadas.

b) **Los ejercicios solucionados**. Los ejercicios tienen distinta dificultad (indicada en el enunciado). Aquellos ejercicios más difíciles, lógicamente, lograrán mayor puntuación si los resuelve.

c) **La claridad de la solución**. La utilización de nombres representativos, comentarios adecuados, procedimientos auxiliares, etc. mejora la calidad de la solución.

d) **La eficiencia de la solución**. Un algoritmo demasiado complejo para el problema o escribir código para el que existe una función, método, procedimiento, etc., estándar en el lenguaje elegido, restará calidad a la solución.

En caso de empate a puntos, se utilizará el tiempo para desempatar.

EJERCICIO 1: SUMA LOS SOLITARIOS (MÁXIMO 20 PUNTOS)

Dados tres números enteros, a, b y c, escribe una función o método que los sume y retorne el valor de la suma. Sin embargo, ten en cuenta que debes omitir en la suma aquellos que estén repetidos.

Por ejemplo,

`sumaSolitarios(a, b, c) → suma los no repetidos`

`sumaSolitarios(1, 2, 3) → 6`

`sumaSolitarios(3, 2, 3) → 2`

`sumaSolitarios(3, 3, 3) → 0`

Escribe, en un fichero llamado `ejercicio1.py`, una función llamada `sumaSolitarios` tal como la descrita arriba.

Se proporciona un fichero llamado `testEjercicio1.py` que contiene varios casos de prueba. Al ejecutarlo, se ejecuta los casos de prueba indicando en cada uno la salida esperada. A continuación, ejecuta tu método e imprime la salida generada para que puedas compararlas. No es un indicativo fiable de que tu solución esté correcta al 100%, pero sí puede servirte como aproximación. Puedes modificar el fichero `testEjercicio1` añadiendo los casos de prueba que desees.

EJERCICIO 2: EMPAQUETAR PEDIDOS (MÁXIMO 45 PUNTOS)

Una empresa conservera dispone de latas de bonito de 100 gramos y de 500 gramos y recibe pedidos en los que le indican el peso en gramos que se desea. No se considera el peso de la lata.

Al procesar estos pedidos, la empresa intenta hacer paquetes con las latas de las que dispone hasta obtener el peso exacto requerido, si es posible. Siempre que es posible intenta utilizar una lata de 500 gramos en lugar de 5 de 100 gramos.

Escribe una función o método `empaquetar` que recibe como parámetros el número de latas pequeñas disponibles, el número de latas grandes disponibles, y el peso total (en gramos) que se desea y que retorne el número de latas pequeñas que se deberían meter en el paquete. En caso de que no sea posible empaquetar el peso exacto, retorna -1.

Por ejemplo,

`empaquetar(pequeñas, grandes, peso) → total pequeñas`

`empaquetar(4, 1, 900) → 4`

`empaquetar(4, 1, 1000) → -1`

`empaquetar(4, 1, 700) → 2`

`empaquetar(4, 4, 1000) → 0`

`empaquetar(4, 1, 200) → 2`

Escribe, en un fichero llamado `ejercicio2.py`, una función llamada **`empaquetar`** tal como la descrita arriba.

Se proporciona un fichero llamado `testEjercicio2.py` que contiene varios casos de prueba. Al ejecutarlo, se ejecuta los casos de prueba indicando en cada uno la salida esperada. A continuación, ejecuta tu función e imprime la salida generada para que puedas compararlas. No es un indicativo fiable de que tu solución esté correcta al 100%, pero sí puede servirte como aproximación. Puedes modificar el fichero `testEjercicio2` añadiendo los casos de prueba que desees.

EJERCICIO 3: CONSIGUIENDO YOGURES EXTRA (MÁXIMO 50 PUNTOS)

Una tienda gourmet tiene, como producto estrella, los yogures artesanos. Además de estar buenísimos, te dan uno gratis cuando devuelves un cierto número de tarros de yogur vacíos.

Escribe una función llamada **`extra`** que averigüe el número total de yogures que podrás tomarte con el dinero del que dispones. La función debe tener la cabecera:

`extra(dinero, precio, vaciosParaGratis) → total yogures obtenidos`

`dinero`: Dinero del que dispones.

`precio`: Precio de cada yogur.

`vaciosParaGratis`: Número de tarros vacíos que debes devolver para conseguir un yogur extra.

Por ejemplo,

`extra(16, 2, 2) → 15`

Con 16 euros a 2 euros por yogur me tomo 8 yogures.

Devuelvo esos 8 tarros, y por cada 2 tarros me dan uno gratis, 4 yogures gratis adicionales que me tomo.

Devuelvo esos 4 tarros, y me dan otros 2 gratis que me tomo.

Devuelvo esos 2 tarros, y me dan 1 más gratis que me tomo.

8 + 4 + 2 + 1 = 15 yogures en total que me como.

`extra(15, 1, 3) → 21`

Se proporciona un fichero llamado `testEjercicio3.py` que contiene varios casos de prueba. Al ejecutarlo, se ejecuta los casos de prueba indicando en cada uno la salida esperada. A continuación, ejecuta tu función e imprime la salida generada para que puedas compararlas. No es un indicativo fiable de que tu solución esté correcta al 100%, pero sí puede servirte como aproximación. Puedes modificar el fichero `testEjercicio3` añadiendo los casos de prueba que desees.

EJERCICIO 4: SUBIR LA ESCALERA (MÁXIMO 80 PUNTOS)

Un niño está subiendo una escalera. El niño es un poco travieso y le gusta subir saltando 1 o 2 escalones a la vez. ¿De cuántas maneras distintas puede subir hasta arriba del todo de la escalera?

Escribe una función `escalera(escalones)` que reciba como entrada el número de escalones y que retorne el número de formas posibles en las que el niño puede llegar a la cima de la escalera.

`escalera(escalones) -> formas de subirlos de uno en uno o en saltos de dos`

Por ejemplo:

`escalera(4) → 5`

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2

Total: 5 maneras

```
escalera(5) → 8
escalera(6) → 13
```

Escribe, en un fichero llamado `ejercicio4.py`, una función llamada **`escalera`** tal como la descrita arriba.

Se proporciona un fichero llamado `testEjercicio4.py` que contiene varios casos de prueba. Al ejecutarlo, se ejecuta los casos de prueba indicando en cada uno la salida esperada. A continuación, ejecuta tu función e imprime la salida generada para que puedas compararlas. No es un indicativo fiable de que tu solución esté correcta al 100%, pero sí puede servirte como aproximación. Puedes modificar el fichero `testEjercicio4` añadiendo los casos de prueba que desees.

EJERCICIO 5: LA MEMORIA VIRTUAL (MÁXIMO 100 PUNTOS)

Para que un procesador ejecute una instrucción utilizando unos datos (por ejemplo, sumar dos números), es necesario que, tanto la instrucción (sumar) como los datos (ambos números) se encuentren en alguna posición de la memoria principal del ordenador. El procesador no accede a la memoria secundaria (disco duro, memorias externas, etc) en su busca.

Sin embargo, a medida que la memoria principal de la que dispone un ordenador ha ido creciendo, también lo ha hecho el tamaño de los procesos (o programas) que queremos ejecutar.

Por tanto, ¿cómo podemos ejecutar procesos que, a veces, ocupan incluso más tamaño que toda la memoria principal de la que dispone el ordenador?

La respuesta es la **memoria virtual**.

La memoria virtual es una técnica que permite que, para que un procesador ejecute un proceso, no sea preciso tener cargadas en memoria principal todas sus instrucciones y todos los datos que sean necesarios para su ejecución. Si no existiese, sería prácticamente imposible ejecutar muchos de los programas que utilizamos.

Su funcionamiento es muy sencillo. Se basa en los siguientes conceptos.

DIRECCIONES LÓGICAS VERSUS DIRECCIONES FÍSICAS

1. **Direcciones lógicas**: Por un lado, durante la ejecución de un programa, el procesador genera direcciones lógicas (aka direcciones virtuales) que son relativas a una dirección 0.

   Por otro lado, cada procesador es capaz de manejar a la vez una cantidad máxima de bits (supongamos 8 bits, un byte).

   Por tanto, si un proceso ocupa 100 bytes, el procesador puede generar la dirección lógica 0, la dirección lógica 1, la dirección lógica 2, hasta la dirección lógica 99. Cada una de estas direcciones hace referencia a un byte del proceso. La dirección 0, se refiere al primer byte, la dirección 1 al segundo byte, y así sucesivamente.
2. Pero los datos se guardan en **posiciones físicas** de memoria y lo más probable es que no coincida con la dirección lógica. Es decir, sería casi un milagro que la dirección lógica L de un proceso estuviese en la posición de memoria física L.
3. Por tanto, hace falta **traducir de direcciones lógicas a físicas**.

PAGINACIÓN

Habitualmente, el proceso se divide en fragmentos y sólo aquellos que son necesarios en un momento determinado se cargan en memoria. Cuando la división se realiza en fragmentos de igual tamaño se denominan **páginas**, y el sistema así diseñado se denomina de **memoria virtual con paginación**.

La memoria principal se divide en trozos de este mismo tamaño llamados **marcos de página**.

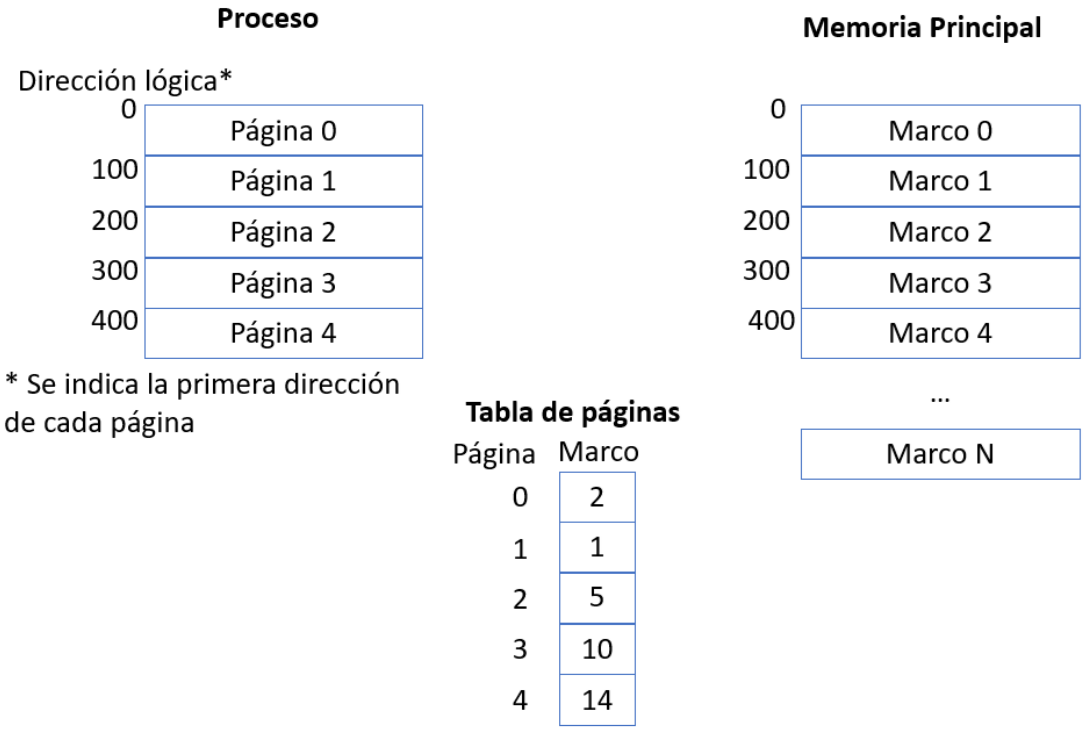
Y la técnica de memoria virtual con paginación consiste en cargar en marcos de página las páginas de los distintos procesos en ejecución a medida que van haciendo falta, y de liberar los marcos de página cuando las páginas que contienen ya no son necesarias para la ejecución de su proceso.

LA TABLA DE PÁGINAS

Para gestionar todo esto, se utiliza una **tabla de páginas** por proceso, que, entre otras cosas, almacena en qué marco de página está cargada cada página del proceso. Es la estructura clave para poder traducir las direcciones lógicas (las que usa el proceso) a físicas (las direcciones reales donde se almacenan los datos).

Supongamos que dividimos la memoria principal en marcos de 100 bytes y los procesos en páginas del mismo tamaño, 100 bytes. Supongamos también que el procesador maneja direcciones de 1 byte.

En la siguiente figura se muestra un proceso que está divido en 4 páginas, cargadas en los marcos que indica la tabla de páginas.



1. ¿En qué página del proceso estaría la dirección lógica 231?
2. ¿En qué marco de página está la página 3 del proceso?
3. ¿Qué dirección de memoria física (en memoria principal) se corresponde con la dirección lógica 231?
4. ¿Y con la 427?