

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC1103 – Introducción a la Programación
2018 - 1

Interrogación 1

Instrucciones generales:

- **Para resolver la interrogación solo necesitas hasta la sección 8 del Recordatorio de contenidos adjunto.**
- La interrogación consta de 3 preguntas con igual ponderación.
- Solo se reciben consultas sobre el enunciado en los primeros 30 minutos.
- No está permitido utilizar material de apoyo adicional al recordatorio adjunto.
- Todas las preguntas están pensadas para ser resueltas en forma sencilla con lo visto en clases. No es necesario utilizar otros mecanismos más avanzados.
- Si el enunciado de la pregunta no lo indica explícitamente, no es necesario validar los ingresos del usuario. Puedes suponer que no hay errores por ingreso de datos.
- Responde cada pregunta en una hoja separada, marcando tu Número de alumno y el número de pregunta en todas las hojas. Si usas más de una hoja en una pregunta, enumera las hojas, la numeración es por pregunta.
- Puedes utilizar cualquier cosa que te hayan pedido programar en un ítem anterior dentro de las preguntas (incluso si no respondiste ese ítem).
- Tienes dos horas para resolver la prueba desde que se entregan los enunciados.

Pregunta 1

Cansado de trabajar, tu compañero te ofrece jugar ***Pig***. El juego consiste en dos jugadores que lanzan un dado por turnos, intentando llegar a los 100 puntos. Si al terminar un turno, el jugador tiene 100 puntos o más, gana el juego. Cada vez que un jugador recibe el dado puede decidir si lo lanzará o no. En cada turno se siguen las siguientes reglas:

- Si el resultado **no** es 1, entonces el jugador acumula los puntos del resultado y puede volver a lanzar el dado para seguir acumulando puntos, o terminar su turno.
- Si el resultado es 1, entonces pierde todos los puntos que había acumulado en el turno, y se termina su turno.

Como no tienen un dado, le ofreces programar el juego de forma que se vea así:

```
*** Turno jugador 1
¿Lanzar? (1) sí (2) no
>1
Tienes un 6
¿Lanzar? (1) sí (2) no
>1
Tienes un 4
¿Lanzar? (1) sí (2) no
>2
Jugador 1: 10 Jugador 2: 0

*** Turno jugador 2
¿Lanzar? (1) sí (2) no
>1
Tienes un 1
Jugador 1: 10 Jugador 2: 0

*** Turno jugador 1
¿Lanzar? (1) sí (2) no
>1
Tienes un 6
¿Lanzar? (1) sí (2) no
>1
Tienes un 6
¿Lanzar? (1) sí (2) no
>1
Tienes un 1
Jugador 1: 10 Jugador 2: 0

(... continúan ...)
Jugador 1: 97 Jugador 2: 57

*** Turno jugador 1
¿Lanzar? (1) sí (2) no
>1
Tienes un 5
¿Lanzar? (1) sí (2) no
>2
Jugador 1: 102 Jugador 2: 57
Felicitaciones, Jugador 1
```

Puedes usar un diálogo similar al anterior, siempre y cuando se le pregunte al usuario si desea lanzar nuevamente, se muestre el dado que sale en cada lanzamiento, se muestre los puntajes de ambos jugadores después de cada turno, y al finalizar el juego se muestre quién fue el ganador.

Pregunta 2

¡Llega la caza (fotográfica) de osos! Implementa la función `contar_osos(x,y,z)` que cuenta la cantidad de veces que aparece la palabra "OSO" en tres strings. La función recibe 3 strings como parámetros (x,y,z), y retorna un entero con el número de OSOs encontrados. Los 3 strings son del mismo largo entre ellos, y están compuestos por 'O' y 'S'. La Figura 1 muestra como contar los OSOs. Se consideran los strings x, y, z uno encima del otro como muestra la figura, los OSOs a considerar son solo los que se pueden leer horizontalmente de izquierda a derecha, verticalmente de arriba a abajo, y diagonalmente de izquierda a derecha. En el ejemplo de la Figura 1, con los strings "OOSOS" (x), "SSOSO" (y) y "OOOSO" (z), la función `contar_osos(x,y,z)` retornaría el entero `6`. Ojo que tu función debería funcionar para strings de cualquier largo (pero todos del mismo largo).

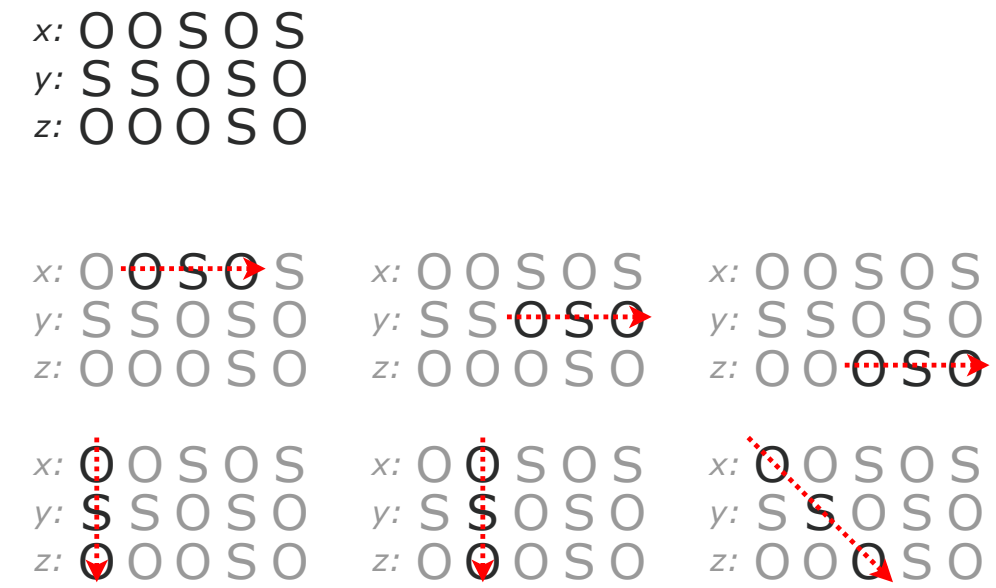


Figura 1: Ejemplo con los strings "OOSOS" (x), "SSOSO" (y) y "OOOSO" (z). En este caso, la función `contar_osos(x,y,z)` retornaría el entero `6`.

Pregunta 3

HackerRank es una plataforma online de competencias de programación. Cada competencia o *challenge* está compuesta por un conjunto de problemas de distinto puntaje. Los problemas a su vez, poseen casos de prueba. Los casos de prueba sirven para evaluar si el código enviado como solución, resuelve efectivamente el problema. El puntaje obtenido en cada problema depende de la proporción de casos de prueba con los que cumple el código enviado.

Por un motivo inexplicable la página de *HackerRank* dejó de funcionar y tú todavía tenías que hacer el *challenge* del laboratorio de esta semana. Por suerte, puedes importar un módulo de `Python` con las funcionalidades necesarias para que puedas probar tu código. Considera el módulo `hackerrank` que contiene las siguientes funciones:

- `obtener_cantidad_de_problemas()`: retorna un `int` con el número de problemas que hay en el *challenge* del laboratorio. Cada problema tiene un identificador único (ID) que es un número que va desde 1 hasta la cantidad total de problemas.
- `obtener_puntaje(id_problema)`: recibe el ID correspondiente a un problema del *challenge* y retorna un `int` con el puntaje máximo que se puede obtener en él.
- `obtener_enunciado(id_problema)`: recibe el ID correspondiente a un problema y retorna un `string` con el enunciado.
- `obtener_cantidad_casos_de_prueba(id_problema)`: recibe el ID correspondiente a un problema y retorna la cantidad de casos de prueba de éste. Cada caso está identificado con un ID único que va desde 1 hasta la cantidad total de casos.
- `probar_codigo(codigo, id_problema, id_caso_de_prueba)`: recibe un `string` con el código, el ID del problema y el ID del caso de prueba. Retorna `True` si el código entregado resuelve el caso de prueba para ese problema; `False`, en otro caso.

En primer lugar, debes crear las siguientes funciones:

a) `mostrar_lista_problemas()`: imprime el ID y el puntaje total de cada problema. La función imprime y no retorna nada.

b) `calcular_puntaje_obtenido(codigo, id_problema)`: recibe el `string` con el código a probar y el ID del problema. Retorna el puntaje obtenido, el cual es calculado de la siguiente forma:

$$\frac{casos_de_prueba_aprobados \times puntaje_maximo_del_problema}{casos_totales_de_prueba_del_problema}$$

Finalmente, debes programar el siguiente flujo apoyado por las funciones definidas anteriormente:

1. Mostrar la lista de problemas con sus respectivos ID y el puntaje total de cada uno.
2. Preguntar al usuario cuál problema desea resolver y mostrar su enunciado.
3. Para el problema seleccionado, ofrecer las siguientes opciones hasta que el usuario obtenga el 100 % del puntaje del problema o elija salir del programa :
   a) Recibir un código. (puedes asumir que la función `input()` es capaz de recibir todas las líneas de código que sean necesarias)
   b) Probar el código, mostrar el puntaje obtenido y el total.
   c) Salir del programa.

Puedes usar el siguiente ejemplo como guía:

```
Problemas del laboratorio:
Problema 1 - puntaje total: 100
Problema 2 - puntaje total: 200
Problema 3 - puntaje total: 300

Ingresa el ID del problema que deseas resolver:
```

```
> 1

El enunciado del problema 1 es: "(...)"

¿Qué deseas hacer?
(1) Ingresar código
(2) Probar código ingresado
(3) Salir
Ingresa una opción:
> 1

Ingresa el código:
> for _ in range(...)

¿Qué deseas hacer?
(1) Ingresar código
(2) Probar código ingresado
(3) Salir
Ingresa una opción:
> 2

Puntaje obtenido: 100 de 100 pts
```
