

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

**IIC1103 – Introducción a la Programación**
**2 - 2017**

# Interrogación 2

**Instrucciones generales:**

- **Para resolver la interrogación solo necesitas hasta la sección 11 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.
- 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.
- Puedes utilizar cualquier cosa que te hayan pedido programar en un ítem anterior dentro de las preguntas (incluso si no respondiste ese ítem).
- Tienen dos horas para resolver la prueba desde que se entregan los enunciados.

# Pregunta 1

Marquitos tiene listo su horario para el primer semestre de 2018. Entre ellos no está Introducción a la Programación porque lo aprobará este semestre.

El horario de cada día está representado como una lista de valores `bool`, donde cada elemento de la lista es `True` si Marquitos tiene clases en ese módulo y `False` si no tiene.

A Marquitos le gusta aprovechar su tiempo: Siempre aprovecha las "ventanas" de tiempo en su horario para estudiar y **NUNCA** se queja por ellas.

Una ventana de tiempo está definida como una secuencia de módulos consecutivos en que Marquitos no tiene clases. Ésta debe ser posterior a su primera clase del día, y anterior a su última clase del día. Además, pueden haber múltiples ventanas en un mismo día. El siguiente *horario* = `[False, False, True, False, True, False]` tiene una sola ventana de largo 1, entre el tercer y quinto módulo.

A continuación se muestran otros ejemplos de horarios, junto con el largo de sus ventanas más grandes:

| **Horario** | **Ventana más grande** |
|---|---|
| $h_0$ =`[True, False, False, False, True, False, False, True]` | 3 |
| $h_1$ =`[False, False, True, True, False, False]` | 0 |
| $h_2$ =`[True, True, True, True, True, True]` | 0 |
| $h_3$ =`[True, False, False, False, False, False, False, True, False]` | 6 |
| $h_4$ =`[True, False, False, False, True, False, False, False, False]` | 3 |
| $h_5$ =`[False, False, False, False, False, False]` | 0 |

(a) Escribe una función de nombre *ventana_mas_grande(horario)*. Ésta debe recibir como argumento una lista *horario*, que representa el horario de un día, y retorna el largo de la ventana más grande (en cantidad de módulos) que hay en ese horario. Si no hay ninguna ventana, debe retornar 0. Ten en cuenta que el largo de los horarios no es fijo. **(4 puntos)**

Por ejemplo, si tomamos *horario* = $h_0$, la función debería responder 3.

(b) Escribe una función de nombre *dia_con_mayor_ventana(horario_semanal)* que reciba un *horario_semanal*, que es una lista de listas con los horarios de la semana, donde cada lista dentro de *horario_semanal* representa el horario de un día. La función debe retornar el **índice** del día donde se encuentra la ventana más grande. Si hay 2 o más días con la ventana más grande puede retornar cualquiera de ellos. Ten en cuenta que no se sabe a priori cuántos días hay en *horario_semanal* ni cuántos módulos tienen los horarios. **(2 puntos)**

Por ejemplo, si tomamos *horario_semanal* = $[h_0, h_1, h_2, h_3, h_4, h_5]$ entonces la función debería responder 3.

# Pregunta 2

La empresa *Smooth Operator SpA* está teniendo problemas con los operadores telefónicos de su *call center*. Muchos de los operadores están entregando un mal servicio y hasta ahora no tienen cómo identificarlos. Por esto, decidieron registrar los datos de todas llamadas que se estaban recibiendo. Además, implementaron un sistema de evaluación telefónica, en donde los clientes califican a los operadores luego de terminar su llamada. Luego de acumular meses de datos, se acercaron a los talentosos alumnos de Introducción a la Programación con el objetivo de poder interpretar los datos que han obtenido.

Para esto les entregaron el archivo `registro.csv`. Este archivo contiene información de cada llamada atendida por los operadores en los últimos 6 meses. La primera línea contiene un encabezado (los nombres de cada columna), y las siguientes líneas están escritas de la forma:

```
nombre_cliente;nombre_operador;duracion_llamada;rating_llamada
```

En este archivo:

- `nombre_cliente` es el nombre de quien llamó al call center.
- `nombre_operador` es el nombre del operador que atendió la llamada.
- `duracion_llamada` es la duración total de la llamada en minutos.
- `rating_llamada` es la calificación que dejó el cliente por la atención del operador.

Las primeras 10 líneas de este archivo se ven de la siguiente forma:

```
nombre;operador;tiempo_llamada;rating_llamada
pedro;juan;15;3
valentin;maria;2;5
diego;juan;18;2
jose;maria;5;4
anita;camilo;8;5
sofia;manuel;7;1
esteban;maria;10;3
cristina;maria;3;4
julio;marcos;1;2
```

Escribe un programa que calcule el desempeño de los operadores de este *call center*. Tu programa debe crear un archivo con nombre `reporte.csv`, donde en cada línea tenga los siguientes datos:

```
nombre_operador;duracion_maxima_llamada;rating_promedio_llamadas
```

En donde:

- `duracion_maxima_llamada` es la duración de la llamada más larga que atendió el operador.
- `rating_promedio_llamadas` es el promedio de las calificaciones que dejaron los clientes a las atenciones del operador.

Por ejemplo, el archivo que debería resultar si tomamos como entrada las primeras 10 líneas del archivo `registro.csv`:

```
operador;duracion_maxima_llamada;rating_promedio_llamada
juan;18;2.5
maria;10;4.0
camilo;8;5.0
manuel;7;1.0
marcos;1;2.0
```

*Observación: El archivo de salida no necesita estar ordenado.*

# Pregunta 3

El Aeropuerto está teniendo muchos problemas ya que los clientes reportan que faltan cosas en sus maletas. La principal sospecha está sobre el departamento Transportador del Aeropuerto, que mueve las maletas de un punto a otro. Tu misión es desarrollar un sistema que alerte cuando las maletas transportadas pesen menos de lo que deberían. Para ello, debes definir las siguientes clases y sus métodos, que tienen que ser compatibles con el código de ejemplo escrito más abajo.

## Articulo

- *__init__*: Inicializa el artículo. Recibe como entrada el nombre, peso en kilogramos y volumen en litros del artículo.

## Maleta

- *__init__*: Inicializa la maleta. Recibe de entrada a *volumen_maximo* y *clave*. *volumen_maximo* es el máximo volumen en litros que cabe en la maleta. *clave* es un entero entre 0 y 999 que impide que cualquiera abra la maleta. Inicialmente la maleta está abierta.
- *obtener_peso_total*: No recibe parámetros. Retorna la suma de los pesos de los artículos dentro de la maleta.
- *abrir*: Recibe como entrada la *clave*. Si *clave* es igual a la clave con la que se creó la maleta, la maleta se abre. Retorna `True` si fue posible abrir la maleta, `False` en otro caso.
- *cerrar*: No recibe parámetros. Cierra la maleta.
- *agregar_articulo*: Recibe de entrada una instancia de la clase *Articulo*. Si la maleta está abierta y el volumen del artículo no hace que se supere el límite de la maleta, agrega el artículo a la maleta. Retorna `True` si fue posible agregar el artículo a la maleta, `False` en otro caso.
- *sacar_ultimo_articulo*: No recibe parámetros. Si la maleta está abierta, quita de la maleta el último artículo agregado. Retorna el artículo si fue posible sacarlo de la maleta, en caso contrario retorna `None`.

## Aeropuerto

- *__init__*: Inicializa el aeropuerto. Recibe una instancia de la clase *Transportador* (Supón que está implementada), que cuenta con el método *transportar(maletas)* que recibe una lista de maletas y las transporta. Se sospecha que si hay maletas abiertas, entonces el transportador saca de las maletas el artículo que está más a la mano (el último artículo agregado).
- *ingresar_maleta*: Recibe una instancia de la clase *Maleta*. Ingresa una maleta al aeropuerto. Le asigna a la maleta ingresada un identificador único según el orden en que fue ingresada y lo retorna.
- *inspeccionar*: No recibe parámetros. Este es el método que alertará si algo malo está pasando con el transportador. Debe calcular el peso de cada maleta ingresada al aeropuerto, llamar al método transportar del Transportador del aeropuerto y volver a calcular el peso de las maletas. Para cada maleta, si luego de ser transportada su peso disminuyó, se debe mostrar el siguiente mensaje: `Alerta, la maleta {identificador_maleta} debería pesar {peso_supuesto}, pero pesa {peso_actual} kilos`.

A continuación se muestra un programa de ejemplo, junto con su ejecución:

---

```
notebook = Articulo('notebook', 10, 2)
cargador = Articulo('cargador', 1, 0.5)
ropa = Articulo('ropa', 2, 10)

maleta1 = Maleta(20, 123)
maleta2 = Maleta(20, 432)

maleta1.agregar_articulo(notebook)
maleta1.agregar_articulo(cargador)
```

```
maleta2.agregar_articulo(ropa)
maleta2.cerrar()
transportador = Transportador()
aeropuerto = Aeropuerto(transportador)

id_maleta1 = aeropuerto.ingresar_maleta(maleta1)
print('El ID de la primera maleta es el', id_maleta1)

id_maleta2 = aeropuerto.ingresar_maleta(maleta2)
print('El ID de la segunda maleta es el', id_maleta2)

aeropuerto.inspeccionar()
```

---

---

```
El ID de la primera maleta es el 0
El ID de la segunda maleta es el 1
Alerta, la maleta 0 debería pesar 11 kilos, pero pesa 10 kilos
```

---