

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

IIC1103 – Introducción a la Programación
2018 - 2

Interrogación 1

Instrucciones generales:

- **Para resolver la interrogación solo necesitas hasta la sección 9 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, anotando 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 de esa 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 y media para resolver la prueba desde que se entregan los enunciados.

Pregunta 1

Estás realizando una investigación de pregrado en HumaLab UC, un grupo que investiga en computación centrada en las personas. El grupo ha desarrollado un dispositivo que monitoriza a los adultos mayores mientras duermen para detectar periodos en donde estos no respiran, llamados **apneas**.

Este dispositivo entrega un caracter por cada segundo que pasa. Si el caracter es una **"X"**, significa que el adulto mayor no está respirando. Si este es un **"."**, significa que éste sí está respirando. Un ejemplo de un periodo de medición es el siguiente:

`...XXX...X.X....XXXX......X.XXX.........XXX....XX..X.`

En el `string` anterior, vemos que el adulto estuvo respirando por 3 segundos, luego tuvo una apnea de 3 segundos, luego volvió a respirar por 3 segundos, luego tuvo una apnea de 1 segundo, y así hasta que finaliza el periodo de medición con una respiración de un segundo. En esta secuencia hubo 9 apneas. Como el grupo sabe que estás cursando el curso Introducción a la Programación, te han pedido que programes las siguientes funciones:

a) `(20 pts) apneas(secuencia):` recibe un `string` con una secuencia, y retorna un `int` que representa la cantidad de periodos de apnea que hubo durante el periodo de medición.

b) `(20 pts) max_apnea(secuencia):` recibe un `string` con una secuencia, y retorna un `int` que representa la duración de la apnea más larga que hubo durante el periodo de medición.

c) `(20 pts) kventana(secuencia, k):` recibe un `string` con la secuencia y un `int` `k`. Retorna `True` si la `secuencia` tiene algún período de `k` segundos consecutivos durante el cual el adulto mayor estuvo más tiempo sin respirar que respirando, o `False` en caso contrario.

Puedes suponer que la secuencia siempre partirá y terminará con el caracter **"."**. En la siguiente tabla se muestran distintos ejemplos de como debiese funcionar tu programa:

Secuencia	`apneas`	`max_apnea`	`kventana (k=5)`	`kventana (k=8)`
`...XXX...X.X....XXXX......X.XXX.........XXX....XX..X.`	9	4	`True`	`False`
`...XXX....X...XXXX...XXXX....XXX....XX..XXXXX.`	7	5	`True`	`True`
`.........................`	0	0	`False`	`False`
`.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.`	1	35	`True`	`True`
`.X.............XX.............XX............X.....`	4	2	`False`	`False`

Cuadro 1: Ejemplo funciones pregunta 1

Pregunta 2

Luego de las celebraciones de las fiestas patrias, notas que gastaste más dinero que el que tenías presupuestado gastar. Para que esto no te vuelva a ocurrir, decides partir registrando todos tus gastos con el fin de llevar una contabilidad y tener estadísticas de ellos. Por cada compra que hagas guardarás un registro de ella, el cual contiene una categoría (lo que compraste), el monto que gastaste, y el mes en que lo compraste. Como ya conoces las maravillas de `Python`, decides programar algunas funciones que te ayuden en esta tarea.

Para esto, puedes **importar** la librería `cuentas` que tiene las siguientes funciones **ya implementadas**:

- `obtener_cantidad(categoria)`: recibe un `string` con el nombre de la categoría y retorna un `int` con la cantidad de registros existentes para esa categoría. Puedes suponer que una categoría siempre tendrá al menos un registro. Por ejemplo, si `obtener_cantidad("bebida")` retorna 10, significa que hubo 10 registros de compras de bebidas.
- `obtener_mes(categoria, i)`: recibe un `string` con el nombre de la categoría y un `int` que representa el índice del registro (los cuales van desde 0 hasta la cantidad de registros existentes para esa categoría - 1). Retorna un `int` con el número del mes correspondiente al registro y categoría entregada. Por ejemplo, si `obtener_mes("bebida", 0)` entrega 9, significa que la primera bebida (índice `0`) fue comprada en el mes de septiembre (9).
- `obtener_monto(categoria, i)`: recibe un `string` con el nombre de la categoría y un `int` que representa el índice del registro (los cuales van desde 0 hasta la cantidad de registros existentes para esa categoría - 1). Retorna un `int` con el monto correspondiente al registro y categoría entregada. Por ejemplo, si `obtener_monto("bebida", 1)` entrega 15, significa que la segunda bebida (índice `1`) comprada costó 15.

Con el objetivo de generar estadísticas de tus gastos, **debes definir las siguientes funciones**:

a) `(20 pts) promedio_categoria(categoria)`: recibe un `string` con el nombre de la categoría y retorna un `float` con el promedio de gasto de los registros de esa categoría.

b) `(20 pts) total_mensual_categoria(mes, categoria)`: recibe un `int` que puede ir desde el 1 al 12 y representa el mes (1 equivale a enero y 12 a diciembre) y un `string` con el nombre de la categoría. Retorna un `int` con la cantidad total de dinero gastado en la categoría y mes dados.

c) `(20 pts) mensual_promedio_categoria(categoria)`: recibe un `string` con el nombre de la categoría y retorna un `float` con el gasto mensual promedio para la categoría dada.

Los gastos ya vienen cargados en la librería `cuentas`. Tu misión es programar las funciones mencionadas para que puedan ser utilizadas desde el siguiente código:

```
terminar = False
while not terminar:

    opcion = input('''¿Qué deseas hacer?
                    1) Obtener promedio de gastos de una categoría.
                    2) Obtener total de gastos de una categoría para un mes dado.
                    3) Obtener el gasto mensual promedio de una categoría.
                    4) Salir.''')

    if opcion == 1:
        categoria = input('Ingrese categoría: ')
        print(promedio_categoria(categoria))
    elif opcion == 2:
        categoria = input('Ingrese categoría: ')
        mes = int(input('Ingrese mes: '))
        print(total_mensual_categoria(mes, categoria))
    elif opcion == 3:
        categoria = input('Ingrese categoría: ')
        print(mensual_promedio_categoria(categoria))
    elif opcion == 4:
        terminar = True
```

Pregunta 3

"Ruleta DCC" es el juego más exitoso de la tradicional "Fonda de Don Yadran", que organiza el Departamento de Ciencia de la Computación (DCC). Por supuesto, funciona en Python. Desafortunadamente se perdió el código, así es que te piden que lo vuelvas a crear.

Para participar, hay que apostar 100, por lo que el juego parte con 100 puntos. La ruleta genera solamente números aleatorios entre 1 y 36. El juego termina si sale 7 (en ese caso, quien estaba jugando se lleva el puntaje acumulado), si sale 13 (en ese caso, pierde todo), o si se le terminan los puntos.

Cuando empiezan a salir los números:

- Si detectas una secuencia de 3 números pares, se suma al puntaje todos los números de ruleta (no importa si es 7 o 13, ni si hay secuencias de números pares o impares) que salgan hasta que aparece uno de los tres números pares (el que también se suma). En el siguiente turno, la secuencia de números se reinicia.
- Si detectas una secuencia de 3 números impares, se resta al puntaje la suma de los siguientes 5 números de ruleta que salgan (no importa si es 7 o 13, ni si hay secuencias de números pares o impares). En el siguiente turno, la secuencia de números se reinicia.

Lo que debes hacer es implementar un programa que simule el juego "Ruleta DCC". A continuación se muestran algunos ejemplos del funcionamiento del programa:

```
========= Ejemplo nº1 =========
***RuletaDCC***
Puntos: 100
4
11
3
6
8
2
Pares: 6 8 2
+ 3
+ 3
+ 13
+ 1
+ 3
+ 10
+ 32
+ 16
+ 2
Puntos: 183
7
Fin del juego!
Te llevas 183

========= Ejemplo nº2 =========
***RuletaDCC***
Puntos: 100
1
5
13
Fin del juego!
Te llevas 0

========= Ejemplo nº3 =========
***RuletaDCC***
Puntos: 100
1
11
5
Impares: 1 11 5
- 35
- 7
- 28
- 31
- 3
Puntos: -4
Fin del juego!
Te llevas 0

========= Ejemplo nº4 =========
***RuletaDCC***
Puntos: 100
23
26
36
22
Pares: 26 36 22
+ 26
Puntos: 126
10
22
16
Pares: 10 22 16
+ 3
+ 1
+ 10
Puntos: 140
1
16
5
21
5
Impares: 5 21 5
- 10
- 2
- 2
- 2
- 4
Puntos: 120
12
5
12
7
Fin del juego!
Te llevas 120
```

No es necesario que el output sea exactamente el mismo, pero se debe mostrar: **(1)** el número aleatorio en cada iteración; **(2)** si ocurre una secuencia de pares o impares, los tres números de la secuencia; **(3)** cada número aleatorio que surja luego de la ocurrencia de tres números impares o tres números pares, junto con indicar si se suman o restan al puntaje; **(4)** el puntaje luego de alguna de estas secuencias; **(5)** cuando termina el juego, y finalmente; **(6)** el puntaje una vez terminado el juego.

Recordatorio contenidos

En los ejemplos, lo marcado como `<texto>` se interpreta como lo que debes rellenar con tu código según cada caso.

1. Tipos de datos y operadores

Tipo de dato	Clase	Ejemplo
Números enteros	`int`	`2`
Números reales	`float`	`2.5`
Números complejos	`complex`	`2 + 3j`
Valores booleanos	`bool`	`True/False`
Cadenas de texto	`str`	`"hola"`

Operación	Descripción	Ejemplo
`+`	Suma	`2.3+5.4`
`-`	Resta	`45.45-10.02`
`-`	Negación	`-5.4`
`*`	Multiplicación	`(2.3+4.2j)*3`
`**`	Potenciación	`2**8`
`/`	División	`100/99`
`//`	División entera	`100//99`
`%`	Módulo	`10%3`

Prioridad (mayor a menor): `()`; `**`; `*`, `/`, `//` o `%`; `+` o `-` .

Operación	Descripción	Ejemplo
`== (!=)`	Igual (distinto) a	`2==2`
`<(<=)`	Menor (o igual)	`1<1.1`
`>(>=)`	Mayor (o igual)	`3>=1`
`and`	Ambos `True`	`2>1 and 2<3`
`or`	Algún `True`	`2!=2 or 2==2`
`not`	Negación	`not True`

Prioridad (mayor a menor): `()`; `or`; `and`; `not`; comparadores.

2. Funciones predefinidas

- `int(arg)` convierte `arg` a entero.
- `float(arg)` convierte `arg` a número real.
- `str(arg)` convierte `arg` a cadena de texto (string).
- `list(arg)` genera una lista con elementos según `arg`, que debe ser *iterable* (strings, listas, tuplas, `range`).

3. Función `print`

- Un argumento: `print(arg)`
- Dos o más argumentos:
  `print(arg1, arg2,arg3)`
- Uso de parámetros, por ejemplo, para eliminar salto de línea y separar con guión:
  `print(arg, sep='-', end='')`

4. Función `input`

- `ret = input(texto)` guarda en `ret` un `str` ingresado.
- `ret = int(input(texto))` guarda en `ret` un `int` ingresado.
- `ret = float(input(texto))` guarda en `ret` un `float` ingresado.

5. `if/elif/else`

```
if <cond 1> :
  <codigo si se cumple cond 1>
  if <cond 1.1> :
    <codigo si se cumple 1.1>
  else :
    <codigo si no se cumple 1.1>
elif <cond 2> :
  <codigo si se cumple cond 2 pero no cond 1>
else :
  <codigo si no se cumple cond 1 ni cond 2>
```

6. `while`

```
while <condicion> :
  <codigo que se ejecuta repetidas veces
mientras se cumpla condicion>
```

7. `for`

```
for x in <rango> :
  <codigo que se ejecuta repetidas veces
durante el rango dado>
```

8. Funciones propias

```
def funcion(<argumentos>):
  <codigo de funcion>
  return <valor de retorno>
```

Variables y parámetros definidos dentro de funciones no son visibles fuera de la función (*scope* local).

9. Strings, clase `str`

Acceso a caracteres particulares con operador `[]`, partiendo con índice cero. Porción de string con *slice*, por ejemplo si `string='Hola'`, `string[1:3]` es `'ol'`. Algunos métodos y funciones de strings:

- Operador `+`: une (concatena) dos strings.
- Operador `in`: cuando `a in b` retorna `True`, entonces el string `a` está contenido en el string `b`.
- `string.find(a)`: determina si `a` está contenido en `string`. Retorna la posición (índice) dentro de `string` donde comienza la primera aparición del sub-string `a`. Si no está, retorna `-1`.
- `string.upper()`, `string.lower()`: retorna `string` convertido a mayúsculas y minúsculas, respectivamente.
- `string.strip()`: retorna un nuevo string en que se eliminan los espacios en blanco iniciales y finales de `string`.
- `string.split(a)`: retorna una lista con los elementos del `string` que están separados por el string `a`. Si se omite `a`, asume que el separador es uno o más espacios en blanco o el salto de línea.
- `p.join(lista)`: suponiendo que `p` es un string, retorna un nuevo string conteniendo los elementos de la lista "unidos" por el string `p`.

- Función `len(string)`: entrega el número de caracteres de `string`.

Una forma de iterar sobre los caracteres de `string`:

```
for char in string:
  <operaciones con el caracter char>
```

10. Listas

Secuencias de elementos-objetos. Se definen como `lista = [<elem1>,<elem2>,...,<elemN>]`. Los elementos pueden o no ser del mismo tipo. Para acceder al elemento `i`, se usa `lista[i]`. La sublista `lista[i:j]` incluye los elementos desde la posición `i` hasta `j-1`. Algunos métodos y funciones de listas:

- Operador `+`. concatena dos listas.
- Operador `in`: `a in b` retorna `True` cuando el elemento `a` está contenido en la lista `b`. Y `False` en otro caso.
- `lista.append(a)`: agrega `a` al final de la lista.
- `lista.insert(i,a)`: inserta el elemento `a` en la posición `i`, desplazando los elementos después de `i`.
- `lista.pop(i)`: retorna el elemento de la lista en la posición `i`, y lo elimina de `lista`.
- `lista.remove(elem)`: elimina la primera aparición de `elem` en la lista.
- Función `len(lista)`: entrega el número de elementos de `lista`.

Para iterar sobre los elementos de `lista`:

```
for elem in lista:
  <lo que quieran hacer con elem>
```

11. Programación oo

```
class <NombreClase>:
  def __init__(self, <parametros>):
    self.<atributos> = <algun parametro>
  def __str__(self):
    <codigo sobrecarga de funcion str()>
    return <string que representa
    el objeto>
  def <metodo propio>(self, <parametros>):
    <codigo de modulo propio>
```

12. Archivos

Abrir un archivo:

`archivo = open(<nombre archivo>,<modo>)`,

p. ej. `archivo = open('archivo.txt','r')`. `<modo>` puede ser `'w'` para escribir un archivo nuevo, `'r'` para leer un archivo (predeterminado), y `'a'` para escribir en un archivo ya existente, agregando datos al final del archivo.

Algunos métodos del objeto que retorna la función `open`:

- `archivo.readline()`: retorna un string con la línea siguiente del archivo, comenzando al inicio del archivo.
- `archivo.write(string)`: escribe en el archivo el string `string`.
- `archivo.close()`: cierra el archivo.

Para leer un archivo entero puedes usar `for`, que iterará línea por línea del archivo:

```
archivo = open('archivo.txt','r')
for linea in archivo:
  <lo que quieran hacer con linea>
```

13. Búsqueda y ordenamiento

Búsqueda secuencial o lineal Busca secuencialmente un elemento dentro de una lista de tamaño n hasta encontrarlo. Peor caso: elemento no está en lista, n comparaciones. Uso: lista desordenada.

Búsqueda binaria Supone lista ordenada. Divide la lista en sublistas: si el elemento es mayor que el elemento medio de la lista, se sigue por la sublista derecha; si no, por la lista izquierda. Peor caso: para una lista de n elementos, se realizan $log_2(n)$ comparaciones.

Ordenamiento por selección Busca el índice del elemento más pequeño de la lista, e intercambia los valores entre el índice encontrado y el primer elemento; luego, encuentra el segundo elemento más pequeño, y lo intercambia con el elemento de la segunda posición, realizando la misma operación para el tercero, cuarto, etc..

Ordenamiento por inserción Itera por los elementos de la lista, partiendo por el primer elemento, y generando una lista ordenada con los elementos mientras itera. Es similar a cómo una persona ordena una lista de cartas.

Métodos de Python para ordenamiento

- `lista.sort()`: ordena `lista` en forma ascendente.

14. Recursión y backtracking

Elementos de una función recursiva:

- Caso base: para terminar la recursión.
- Llamada recursiva: llamada a la misma función dentrl de la función, con parámetros distintos que hacen disminuir el problema original.

Para programar una función recursiva, descomponer la función en elementos que puedan ser llamados con subconjuntos de datos. Ejemplo: suma de los primeros N números

```
def suma(N):
  if N == 1: # caso base
    return 1:
  else: #descomponer en N y restantes N-1
    return N + suma(N-1)
```

Backtracking: cuando hay muchas formas de buscar una solución. Se puede formar un arbol con las soluciones, y el algoritmo comienza en el nodo raíz. Cada camino dentro del árbol es un código recursivo, y dentro del código se exploran las ramas del árbol. Ejemplo: encontrar la suma de todas las formas de combinar una lista de números:

```
def suma_conm_lista(lista, sublista=[], suma=0):
  # Caso base: sublista tiene N elementos,
  # o la lista tiene 0 elementos
  if len(lista) == 0:
    print sublista, suma
    return
  else:
    # Quito un elemento de la lista y lo
    # agrego a la sublista para llamar a la
    # funcion en forma recursiva, y aumento
    # la suma con el elemento que saque
    for i in range(len(lista)):
      suma_conm_lista( lista[:i]+lista[i+1:],\
          sublista+[lista[i]], suma+lista[i])
```