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

IIC1103 – Introducción a la Programación
1 - 2016

Interrogación 2

Instrucciones generales:

- La interrogación consta de tres preguntas con igual ponderación (1/3 del total cada una).
- La interrogación dura 2 horas, y solo se reciben consultas sobre el enunciado en los primeros 30 minutos.
- La última página es un recordatorio; no está permitido utilizar material de apoyo adicional.
- Si el enunciado de la pregunta no lo indica explícitamente, no es necesario validar los ingresos del usuario. Puedes asumir que no hay errores por ingreso de datos.
- Puedes definir e implementar todas las funciones o métodos auxiliares que necesites.
- Responde cada pregunta en una hoja separada, marcando tu RUT y el número de pregunta en todas las hojas prestando atención a la dirección del trazo, tal como se muestra en el siguiente ejemplo:

Pregunta 1

Para los próximos trabajos de invierno, ProyectaCAi quiere formar cuadrillas de trabajo (de cuatro personas) que tengan estudiantes de distintas áreas del saber. Ellos cuentan con un archivo "postulantes.txt" donde en cada línea aparece el nombre del postulante a los trabajos, una coma, y una letra mayúscula que representa a los humanistas (H), científicos (C), matemáticos (M) y artistas (A). El siguiente es un ejemplo del archivo "postulantes.txt":

```
Francisca Valenzuela,A
Héctor Noguera,A
Manuel Pellegrini,C
Alberto Hurtado,H
Ricardo Aravena,M
Fernando Chomali,C
```

Escribe un programa en Python que lea el archivo "postulantes.txt" y genere un archivo de salida con cuadrillas de cuatro personas, una de cada área del saber. Cada persona del archivo de postulantes puede estar en una única cuadrilla. Si ya no quedan humanistas (H), científicos (C), matemáticos (M) o artistas (A) al intentar formar una nueva cuadrilla, tu programa debe terminar y generar un archivo con las cuadrillas que pudo formar.

El archivo de salida debe mostrar una cuadrilla por línea, y cada línea debe tener el nombre de los integrantes de la cuadrilla separados por una coma. No debes hacer nada con los postulantes que no queden en una cuadrilla. El nombre del archivo de salida debe ser "cuadrillas.txt".

Para el ejemplo indicado anteriormente, un posible archivo de salida es:

```
Francisca Valenzuela,Manuel Pellegrini,Alberto Hurtado,Ricardo Aravena
```

Pregunta 2

Una eficiente medida para cuidar el medio ambiente es promover que el auto sea usado por más de una persona a la vez. Un centro de alumnos con escaso presupuesto te solicita ayuda en la creación de un software que promueva el uso compartido de vehículos para viajes dentro o fuera de Santiago entre los alumnos de la universidad.

Un viaje tiene un origen, un destino, la distancia en kilómetros del viaje, el costo del viaje fijado por el conductor para costear la bencina (en pesos), el número de asientos disponibles, y el nombre del conductor.

A los conductores les interesa calcular el valor de cada kilómetro por pasajero, en el caso de que logren llenar todos los asientos disponibles. Por ejemplo, un conductor piensa que para un viaje de 100 kms desde Santiago a Valparaiso necesita $20.000 para costear la bencina, en su auto con 4 asientos disponibles. Entonces, el costo de cada kilometro por pasajero es $20.000/100/4 = 50$ pesos/kilometro. Ningún viaje que se cargue en el sistema tendrá un costo mayor a $1000 por pasajero y por kilómetro (puedes asumir que ningún usuario intentará ingresar valores mayores).

La clase UCarpool debe permitir cargar viajes y encontrar viajes. Esta última funcionalidad debe entregar todos los viajes entre el origen y destino especificados en donde el costo por pasajero sea mínimo. Ten presente que podrían haber varios viajes que cumplan esta condición.

Implementa en Python las clases UCarpool y Viaje, con sus atributos y métodos respectivos. Como requisito, tu programa debe poder ejecutarse usando el siguiente código principal:

```
# Creacion del objeto de la clase UCarpool
pool = UCarpool()

# Carga de viajes a la plataforma
v1 = Viaje("Santiago","Valparaiso",100, 20000, 4, "Fernando Fernández")
pool.cargar_viaje(v1)
v2 = Viaje("Santiago","Valparaiso",100, 22500, 3, "Marcela Marquez")
pool.cargar_viaje(v2)
v3 = Viaje("Santiago","Valparaiso",100, 24000, 2, "Domingo Dominguez")
pool.cargar_viaje(v3)
v4 = Viaje("Santiago","Arica",2000, 100000, 2, "Fernando Fernandez")
pool.cargar_viaje(v4)
v5 = Viaje("Santiago","Valparaiso",100, 18000, 3, "Maria Marin")
pool.cargar_viaje(v5)
v6 = Viaje("Rancagua","Talca",170, 45000, 2, "Gonzalo Gonzalez")
pool.cargar_viaje(v6)
v7 = Viaje("Santiago","Valparaiso",100, 30000, 6, "Lucia Lucero")
pool.cargar_viaje(v7)
v8 = Viaje("Valdivia","Santiago",860, 89000, 2, "Benito Benitez")
pool.cargar_viaje(v8)

# Obtencion de los viajes que van de Santiago a Valparaiso con minimo
# costo por kilometro por pasajero
encontrados = pool.encontrar_viaje("Santiago","Valparaiso")

# Se imprimen todos los viajes encontrados
for v in encontrados:
        print(v)
```

El código anterior imprime:

```
Fernando Fernandez: 50.0 (4 cupos)
Lucia Lucero: 50.0 (6 cupos)
```

Pregunta 3

Únicos es un sencillo juego de números. Cada jugador, en secreto, escoge 1 o más números enteros *distintos* del rango 1 a 100 y luego los muestra a los demás. Los números que se repiten entre 2 o más jugadores se eliminan. El jugador que obtiene la suma más alta con los números que le quedan es el ganador.

a) Dos jugadores (40 puntos)

En esta versión de "Únicos" participan dos jugadores, el jugador `0` y el jugador `1`. Escribe la función `unicos2(a, b)` que recibe dos listas de enteros, `a` y `b`, con los números escogidos por los jugadores `0` y `1`, respectivamente. La función debe retornar una lista `[g, p]` de dos elementos, donde `g` es el índice del ganador (0 ó 1), y `p` es su puntaje. Si hay empate, el ganador es el jugador `0` (porque él inventó el juego).

Como ejemplo de uso, para el siguiente código:

```
p0 = [5,8,12,100,13,9,1,4]    #puntaje: 8+9+1=18
p1 = [100,12,15,3,2,13,4,5]   #puntaje: 15+3+2=20
r = unicos2(p0,p1)
print("El ganador es", r[0], "con", r[1], "puntos")

p0 = [4,2,3,1]                #puntaje: 6
p1 = [6,1,3]                  #puntaje: 6
r = unicos2(p0,p1)
print("El ganador es", r[0], "con", r[1], "puntos")

p0 = [1,2,3,4,5]              #puntaje: 1+4+5=10
p1 = [3,100,2]                #puntaje: 100
r = unicos2(p0,p1)
print("El ganador es", r[0], "con", r[1], "puntos")
```

la salida debe ser:

```
El ganador es 1 con 20 puntos
El ganador es 0 con 6 puntos
El ganador es 1 con 100 puntos
```

b) Más de dos jugadores (20 puntos)

Escribe la función `unicos(J)`. Esta función recibe una lista `J`, donde el elemento `i` de `J` es una lista que contiene los números escogidos por el jugador `i`. La función debe retornar una lista de dos elementos, `[g, p]`, donde `g` es el índice del ganador, y `p` es su puntaje. Si hay empate, el ganador puede ser cualquiera de los que obtienen el puntaje máximo.

Como ejemplo de uso, para el siguiente código:

```
p = [ [5,8,12,100,13],          #puntaje: 8+12+13=33
      [1,2,3],                  #puntaje: 0
      [100,14,22],              #puntaje: 14+22=36
      [1,2,4,5],                #puntaje: 4
      [3]                       #puntaje: 0
    ]
r = unicos(p)
print("El ganador es", r[0], "con", r[1], "puntos")
```

la salida debe ser:

```
El ganador es 2 con 36 puntos
```

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC1103- Introducción a la Programación

Recordatorio contenidos I2 - Primer semestre 2016

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 (de 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 (de 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. 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).

8. Programación orientada a objetos

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

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`. Si no está contenido, retorna `False`.
- `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. 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)` o bien `print(string, file=archivo)`: 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>
```