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

IIC1103 – Introducción a la Programación
1 - 2016

Examen

Instrucciones generales:

- El examen consta de cuatro preguntas con igual ponderación (1/4 del total cada una).
- El examen dura 3 horas, y solo se reciben consultas sobre el enunciado en los primeros 30 minutos.
- Las últimas páginas son 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

Supón que existen dos funciones ya implementadas: `numeros_a_caracter` y `caracter_a_numeros`.

La función `caracter_a_numeros` recibe como input un caracter `c` y retorna un número entero entre 100 y 999, tal que el retorno es distinto para cada caracter. La función `numeros_a_caracter` es el inverso: recibe un número `n` entre 1 y 1000 y retorna el carácter `c` tal que `caracter_a_numeros(c) = n`, o `False` si `n` no corresponde a ningún caracter.

Podemos usar la funcion `caracter_a_numeros` como una forma de codificar un string. La codificación consiste en separar los números asociados a cada caracter por un `#`.

Parte a)

Implementa la función `codificar(string)`, que reciba un string y lo codifique según lo explicado anteriormente usando las funciones que ya están implementadas. Por ejemplo, si `caracter_a_numeros(a) = 129` y `caracter_a_numeros(l) = 340`, entonces `codificar('al')` debe retornar `129#340` y `codificar('lala')` debe retornar `340#129#340#129`.

Parte b)

Implementa la función `verificar(codigo)`, que reciba un string `codigo` y verifique si corresponde a una codificación válida de otro string. Recuerda que una codificación válida corresponde a aquella que contiene grupos de números entre 100 y 999 separados por un `#`. Así, `340#129` y `123` son codificaciones válidas, y `676#` o `98#987` no lo son.

Parte c)

Implementa la función `codigo_a_string(codigo)`, que reciba un string `codigo` y devuelva el string decodificado, o `False` si `codigo` no corresponde a la codificación de un string. Por ejemplo, `codigo_a_string('340#129#340#129')` debe retornar 'lala' y `codigo_a_string('123')` debe retornar `False`.

Pregunta 2

En el último proceso para la inscripción de cursos se produjo una caída del sistema, y no se alcanzaron a generar las listas con los alumnos inscritos en las distintas secciones de IIC1103. Lo único que se pudo rescatar fue un archivo de texto con todos los estudiantes a quienes se les asignó una vacante en alguna sección del curso.

Parte a)

Crea una clase `Estudiante` con los métodos necesarios para poder crear objetos con los siguientes atributos: número de alumno, apellido, nombres y promedio ponderado acumulado (PPA).

Parte b)

Crea una clase `Seccion` con los métodos necesarios para poder crear objetos con los siguientes atributos: número de la sección, PPA de la sección y una lista de objetos de la clase `Estudiante`. Agrega un método `calcular_ppa` que calcule el promedio de la sección a partir del PPA de cada uno de sus alumnos.

Parte c)

Te facilitan el archivo de texto con los alumnos inscritos en alguna sección. En este archivo cada línea contiene los datos de un alumno separados por ';' como sigue: `sección; nro_alumno; apellidos; nombres; PPA`. Un ejemplo de este archivo se muestra más abajo:

```
1; 2344567; Lazo Soto; Elba Rosa; 5.6
5; 7892346; Vidal Rey; Arturo Armando; 4.4
4; 5672345; Diaz Pérez; Marcelo Felipe; 5.0
1; 8900034; Herrera Tec; Johnny Elvis; 4.1
3; 4566677; Aranguiz Soto; John Charles; 4.7
1; 5899900; Alvarez Delpiano; Emeterio; 5.9
```

Escribe un programa principal que pueda leer el archivo de texto `inscripciones.txt` recién descrito, y crea los objetos `Seccion` y `Estudiante` correspondientes. Posteriormente, genera un archivo de salida para cada sección con la información de sus estudiantes. Considera que son 5 secciones, y por lo tanto serán 5 archivos de salida.

Cada uno de estos archivos debe tener como nombre de archivo el número de sección, e incluir los datos de los alumnos (un alumno por línea) ordenados alfabéticamente por apellido. La línea final debe tener el promedio de los PPA de todos los alumnos de la sección. Usa los objetos creados anteriormente para obtener la información que necesites.

Para el ejemplo anterior, el archivo de salida para la sección 1 debiera llamarse `1.txt` y tener los siguientes datos:

```
1; 5899900; Alvarez Delpiano; Emeterio
1; 8900034; Herrera Tec; Johnny Elvis
1; 2344567; Lazo Soto; Elba Rosa
5.2
```

No es necesario que los archivos de salida generados tengan exactamente el formato del ejemplo, pero sí deben contener la información solicitada en orden alfabético por apellido.

Pregunta 3

Para bajar el estrés luego de los exámenes, una buena idea es invitar a tus compañeros de curso a jugar naipes. Piensan en juntarse luego de esta prueba, y decides empezar a ejercitar tus habilidades con las cartas al mismo tiempo que practicas programación.

Parte a)

Una compañera encontró una caja con cartas en el centro de alumnos, pero sospechas que pueden haber varias cartas perdidas en esa caja. Escribe una función `cartas_perdidas` que reciba una lista de cartas y retorne una nueva lista con las cartas que faltan ordenadas según número y pinta.

La lista de cartas está en el formato $[[n_1, p_1], [n_2, p_2], ...]$

donde n_i es el número de la carta i y p_i es su pinta.

La pinta puede ser C (corazones), P (picas), D (diamantes) o T (tréboles). Considera que el orden de las pintas, de izquierda a derecha es corazones, picas, diamantes y tréboles. No hay cartas repetidas.

Un ejemplo de cartas ordenadas se muestra en la Figura 1. Los números de las cartas van del 1 al 13. (El As se representa por 1 y las cartas literales (J,Q,R) por 11, 12 y 13.)

Parte b)

Otro compañero encontró una gran pila de cartas que creen que podría tener más de un mazo, es decir, que podría tener cartas repetidas (por ejemplo, más de un As de corazones). Crea una función que reciba una lista de cartas, e imprima la cantidad de mazos completos y de mazos incompletos que tiene esa lista de cartas.

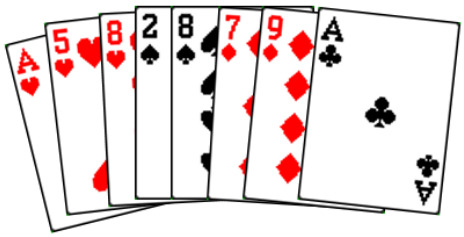

Figura 1: Conjunto de cartas ordenado según número y pinta, de izquierda a derecha

Pregunta 4

Estás preparando la mudanza de tu primo que viene a estudiar a Santiago y quieres ayudarlo a ahorrar dinero en el traslado de sus cosas. Para eso, le prometes que harás un programa que maximizará la cantidad de pertenencias que puede trasladar en un solo viaje.

Hay disponible un camión de mudanzas con capacidad `C` m^3, y tu primo te entrega dos listas: una con el contenido de cada caja (como "ropa" y "libros") y otra con el volumen en m^3 de cada una de esas cajas, respectivamente.

Escribe un programa en Python que en forma **RECURSIVA** entregue una lista con el nombre de las cajas cuyo volumen en conjunto sea el máximo que se puede trasladar en el camión, sin sobrepasar su capacidad.

Pueden existir soluciones alternativas donde cambia el orden de las cajas, por ej. `[Ropa, Zapatos]` y `[Zapatos, Ropa]`. En tu solución puedes considerar todas estas combinaciones o puedes indicar solo una de ellas. Hazlo como te resulte más fácil. Por otro lado, ten presente que podría haber alguna situación en que no se pueda transportar ninguna caja porque la capacidad del camión es muy pequeña.

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

Recordatorio contenidos examen - 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)`: 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>
```

12. 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 dependiendo del valor del elemento buscado: 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.

13. 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 o caminos de buscar una solución. Se puede formar un aŕbol 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 función en forma recursiva,
        # y aumento la suma con el
        # elemento que saqué
        for i in range(len(lista)):
            suma_conm_lista( lista[:i]+lista[i+1:],\
                sublista+[lista[i]], suma+lista[i])
```