Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2014

IIC 1103 - Introducción a la Programación

Examen ⋆ Tiempo: 180 minutos ⋆ Sin apuntes ⋆ Sin consultas

1. Para cada una de las siguientes funciones, explique con sus palabras cuál será el retorno en función de sus parámetros.

a)

```
# Considere que d es un entero mayor que cero
def misterio1(d):
  a = 0
  while(d > 0):
    if(d%2 == 0):
      a += 1
    d //= 10
  return a
```

b)

```
# Considere que a y b son enteros mayores que cero
def misterio2(a,b):
  a1 = a
  b1 = b
  while(a1 != b1):
    if(a1 < b1):
      a1 += a
    else:
      b1 += b
  return a1
```

2. El Gimnasio "Full Sport" define sus precios en base a 4 categorías de servicios: básico, premium, super premium y complementario (gratis para todos los clientes). Cada cliente tiene una tarjeta donde carga un monto a gastar y desde donde se le descuentan los servicios que utiliza de acuerdo al precio del servicio y ciertos descuentos, que dependen del tipo de categoría de cliente al que pertenece (descuento porcentual) y su edad (descuento bruto, que se aplica luego del descuento porcentual). Si un cliente no tiene suficiente carga en su tarjeta, no se le realiza el cobro, ya que no se le permite usar el servicio.

   Se le pide que escriba el código correspondiente a 6 funciones que facilitarán realizar los cobros.

Cuadro 1: Valores de servicios

Servicio	Valor
básico	$ 10.000
premiun	$ 25.000
super premium	$ 45.000
complementario	$ 0

Cuadro 2: Descuentos porcentuales

Categoría	Descuento
400	50 %
300	25 %
200	10 %
otros	0 %

Cuadro 3: Descuentos por edad

Edad	Descuento
Menor a 18 o 60 o más	$ 5.000
Entre 18 y 30	$ 1.000
otros	$ 0

Código a completar:

```
# Retorna el valor base de un servicio
def valor_servicio(tipoServicio):

# Retorna el descuento proporcional por categoría
```

```python
def descuento_proporcional_por_categoria(categoria):

# Retorna el descuento bruto por edad
def descuento_bruto_por_edad(edad):

# Retorna el valor de cobro por servicio incluyendo descuentos
def valor_cobro(tipoServicio, edad, categoria):

# Realiza cobro según tipo de servicio, y retorna la carga restante
# Aplica los descuentos correspondientes y despliega mensaje indicando
# cuánto cobro a la persona identificada por su nombre
def realizar_cobro(tipoServicio, nombre, carga, edad, categoria):

# Despliega mensaje indicando cuántos consumos puede obtener una persona,
# identificada por su nombre, de cada uno de los servicios no gratuitos
def consumos_posibles(nombre, carga, edad, categoria):
```

Código prueba:

```python
# Despliega estado cuenta
def mostrar_estado_cuenta(nombre, carga):
    print("Cliente:", nombre, "- carga:", carga)

# Cada cliente se representa por una lista de 4 datos:
# nombre, edad, categoría, carga
clientes = [["Hugo", 46, 400, 80000],
            ["Oma",  70, 300, 50000],
            ["Vale", 10, 200, 40000]]

for c in clientes:
    mostrar_estado_cuenta(c[0], c[3])
    consumos_posibles(c[0], c[3], c[1], c[2])

for tS in ["básico", "premium", "super premium", "complementario"]:
    print("Clientes consumen servicio:", tS)
    for c in clientes:
        c[3] = realizar_cobro(tS, c[0], c[3], c[1], c[2])

for c in clientes:
    mostrar_estado_cuenta(c[0], c[3])
```

3. En esta pregunta describiremos un tablero usando una lista que contiene la concatenación de todas las filas del tablero, comenzando de arriba hacia abajo. Por ejemplo, la lista $[1, 2, 3, 4, 3, 4, 1, 2, 2, 3, 4, 1, 4, 1, 2, 3]$ representa al tablero que se muestra a continuación:

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3

Dado $n \in \mathbb{N}$, diremos que un tablero de tamaño $n^2 \times n^2$ es un *sudoku* si se cumplen las siguientes condiciones:

- En cada fila se encuentran los números del 1 al n^2.
- En cada columna se encuentran los números del 1 al n^2.
- En cada sub-tablero de tamaño $n \times n$ que tiene como esquina superior izquierda una posición (i, j) tal que i y j son múltiplos de n, se encuentran los números del 1 al n^2 (considerando que los índices comienzan desde cero).

Por ejemplo, el tablero que se muestra arriba es un *sudoku* de tamaño $2^2 \times 2^2$.

En esta pregunta debes escribir una función que reciba una lista y retorne `True` si y sólo si dicha lista representa un sudoku de tamaño $n^2 \times n^2$ para algún $n \in \mathbb{N}$. Para esto, puedes suponer que existen las siguientes funciones:

- `raiz_cuarta(n)` retorna $\sqrt[4]{\texttt{n}}$ en caso de que este número sea un entero, y retorna -1 de lo contrario.
- `contiene_numeros(lista, n)` retorna `True` si y sólo si la `lista` contiene todos los números entre 1 y `n` (incluyéndolos).

4. Estás programando un cajero automático, el cual tiene billetes de 1000, 2000 y 5000. Para implementar los giros de dinero, escribe la función recursiva giro que debe imprimir todas las formas posibles en que el cajero le puede dar el monto solicitado a la persona usando combinaciones de los billetes disponibles. Asume que el cajero siempre tiene suficientes billetes de cada tipo para entregar el monto que la persona quiere con cualquier combinación de billetes, que el monto a girar es mayor o igual a 1000 y que no importa que tu programa repita combinaciones de billetes.

```
def giro(monto, billetes):
  #tu código va aquí
```

Ejemplo: para girar 5000, se llamaría a la función así: `giro(5000, [])` Tu función puede imprimir todas las combinaciones que aparecen más abajo, o solamente aquellas que son únicas, no importando el orden en que las muestre:

```
[1000, 1000, 1000, 1000, 1000]
[1000, 1000, 1000, 2000]
[1000, 1000, 2000, 1000]
[1000, 2000, 1000, 1000]
[1000, 2000, 2000]
[2000, 1000, 1000, 1000]
[2000, 1000, 2000]
[2000, 2000, 1000]
[5000]
```

5. La empresa Metro se ha visto en la necesidad de estudiar los tiempos de espera promedio de sus pasajeros para así ver cuál es la mejor combinación de tamaño y rapidez de sus trenes. Para ello te han pedido que hagas una simulación que logre determinar el tiempo de espera promedio de cada uno de los pasajeros que llegan a una estación. Un estudio muestra que la probabilidad de que llegue un pasajero en cada segundo es de un 25 %. Además el estudio muestra que los pasajeros saben donde están las entradas a los carros, por lo que al llegar a la estación se ponen en una de las filas para entrar a algún carro. La elección de la fila la hacen según cual sea más corta. Los carros no vienen vacíos, sino que llegan con un cierto número de pasajeros. Cuando un tren llega a la estación las personas en las filas comienzan a subir al carro al cual pertenece su fila hasta que el carro se llene o hasta que no queden más personas en la fila. Si el carro se llena esperan el próximo tren (no suben a otro carro). También se sabe que si un pasajero encuentra que todas las filas de espera son de tamaño igual o superior a 20 personas, se retira de la estación y se va en micro.

   a) Implementa dos clases: la clase Metro y la clase Estación junto a sus constructores y atributos.
   - El constructor de la clase Metro recibe dos parámetros el número de carros del metro y la capacidad máxima de cada uno. La clase Metro es la encargada de guardar en sus atributos el número de carros y una lista de capacidades máximas de cada uno de los carros.
   - El constructor de la clase Estación recibe sólo un parámetro que indica la cantidad de puertas que tienen los metros (se asume que esta cantidad es la misma para todos los metros). Esta clase es la encargada de guardar como atributo todas las puertas de los metros; cada puerta representa la cola de pasajeros esperando por subir al metro.

   b) Para la clase Metro deberás implementar dos métodos. Un método que retorna el número de pasajeros en un determinado carro y otro método que represente el ingreso de una persona a un carro determinado del metro. Tenga en cuenta que no es necesario conocer la persona de cada carro sino simplemente el número de personas que hay en cada carro.

   c) Para la clase Estación deberás implementar dos métodos. Un método que permita agregar la hora de llegada de un pasajero a alguna cola. Y un método que permita simular el comportamiento de esa estación, el cual recibe el tiempo de simulación junto al el tiempo mínimo y el tiempo máximo que hay entre cada tren.

   A continuación se entrega la clase Cola que representa las colas de personas esperando por subir a un Metro (no puede agregar más métodos ni atributos).

```
class Cola:

   #Constructor de la clase
   #atributos: cantPersona indica el numero maximo de personas en la cola,
   #personas guarda la hora de llegada de todas las personas a la cola
   def __init__(self):
      self.cantPers = 21
      self.personas = []

```

```
    #Agrega una la hora de llegada de una persona en la cola
    def agregar(self, tiempoLlegada):
        if(len(self.personas)<self.cantPers):
            self.personas.append(tiempoLlegada)
            return True
        return False

    #Retorna hora de llegada de una persona, sacandola de esta
    def proximo(self):
        retorno = self.personas[0]
        i=0
        while (i<len(self.personas)):
            if(i+1<len(self.personas)):
                self.personas[i]=self.personas[i+1]
            i += 1
        return retorno
```