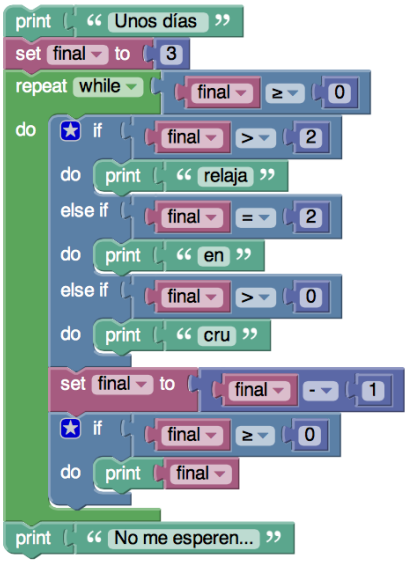
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2013

IIC 1103 ⋆ Introducción a la Programación

Evaluación de Medio Semestre - Solución

1. Un programador ha desaparecido. Lo último que se supo es que estaba frente a su computador utilizando una extraña página con figuras y textos ininteligibles. Se te pide que por favor ayudes a encontrarlo, o al menos des una pista respecto de su paradero. Lo último que se encontró en su pantalla fue lo siguiente:



Su familia está muy preocupada, pues lo único que logran distinguir son los mensajes *Unos días* y *No me esperen...* Para ayudarlos, debes explicarles paso a paso la ejecución del programa anterior. No dejes ningún detalle de lado, pues de lo contrario dudarán de tu conocimiento. Luego, muéstrales todo lo que imprime el programa anterior, a ver si logran descubrir en qué está metido este programador.

Solución

El programa se ejecuta como sigue:

1. El programa imprime en pantalla *Unos días*.
2. Le da el valor 3 a una variable llamada `final`.
3. Mientras `final` sea mayor a 0, se repiten las siguientes acciones:
   (a) Si final es mayor a 3, imprime *relaja*
   (b) En otro caso, si final es igual a 2, imprime *en*

(c) En otro caso, si final es mayor a 0, imprime *cru*
(d) Reduce el valor de final en 1
(e) Si final es mayor o igual a 0, entonces imprime el valor de `final`

4. Finalmente, imprime *No me esperen.*

Su ejecución es la siguiente:

1. Paso 1, se imprime *Unos días.*
2. Paso 2, $final = 3$.
3. Paso 3, $final = 3 > 0$, por lo tanto, entramos en el *while.*
4. Dado que $final = 3$, se ejecuta (a), el cual imprime *relaja.*
5. Paso (d), ahora $final = 2$
6. Como $final > 0$, se ejecuta el paso (e) el cual imprime *2.*
7. Volvemos al paso 3, y como $final = 2 > 0$, entramos en el *while.*
8. Dado que $final = 2$, se ejecuta (b), el cual imprime *en.*
9. Paso (d), ahora $final = 1$
10. Como $final > 0$, se ejecuta el paso (e) el cual imprime *1.*
11. Volvemos al paso 3, y como $final = 1 > 0$, entramos en el *while.*
12. Dado que $final = 1$, se ejecuta (c), el cual imprime *cru.*
13. Paso (d), ahora $final = 0$
14. Como $final = 0$, se ejecuta el paso (e) el cual imprime *0.*
15. Se evalúa el paso 3, pero como $final = 0$, no entra en el *while.*
16. Se ejecuta el paso 4, el cual imprime *No me esperen.*

Finalmente podemos ver el *output* del programa:

```
Unos días
relaja
2
en
1
cru
0
No me esperen
```

Por lo visto, la familia se puede quedar tranquila, pues el programador se está tomando unos días relajados en un crucero.

2. Una tropa de soldados intenta interceptar ondas de radio enviadas por sus enemigos. Para esto, sitúan tres antenas en distintas posiciones. Cada antena intercepta el mensaje secuencialmente, letra por letra, y lo guarda en un string. Las letras que no logra interceptar son representadas en el string por un guión bajo (_). Por ejemplo, si una antena detecta una `a`, luego no logra detectar nada, y luego detecta una `o`, entonces entregará el string `a_o`. Lamentablemente, los soldados no tienen la capacidad de procesar los strings que generan las tres antenas de forma automática, por lo que te piden generar un programa para ayudarlos. Tu misión será crear una función que reciba los tres strings generados por las antenas y retorne también un string con el texto consolidado. Este texto consolidado debe considerar lo siguiente:

- Si hay una letra que fue interceptada de la misma forma por al menos dos de las tres antenas, entonces se concluye que estas dos antenas están en lo correcto.
- Si hay una letra que no logró ser interceptada por al menos dos de las tres antenas, esa letra se deja con un guión bajo, lo que indica que dicha letra no logró ser interceptada.
- Si hay una letra que las tres antenas interceptaron de forma distinta, o que dos interceptaron de forma distinta y la tercera no pudo interceptar, la función retorna un mensaje indicando que hay una inconsistencia, e indica también una posición en la cual se encontró una inconsistencia.
- Si los strings no tienen todos el mismo largo, la función debe retornar un string que indica que hay una inconsistencia en los largos de los mensajes.

A continuación se muestran tres ejemplos de lo que debe retornar su función:

`intercep('h_l__ca_i_an','ho_a _a_it_n','__la c_p_ta_')` retorna `'h_la ca_itan'`
`intercep('_ _us _rd__es','a_s_s or_ene_','a_su_ o_d_n_s')` retorna `'a_sus ord_nes'`
`intercep('_a_bi_','co___o','ce_b_o')` retorna `'inconsistencia en la posición 1'`

Solución

```
def intercep(s1, s2, s3):
    if len(s1) == len(s2) == len(s3):
        texto = ""
        for i in range(len(s1)):
            if s1[i] == s2[i] or s1[i] == s3[i]:
                texto += s1[i]
            elif s2[i] == s3[i]:
                texto += s2[i]
            else:
                return "Inconsistencia en la posición "+str(i)
        return texto
    else:
        return "Inconsistencia en el largo de los mensajes"
```

3. Se te ha contratado para realizar un programa que se ejecutará en cajeros automáticos. Estos cajeros deben ofrecer las siguientes funcionalidades:

   - Obtener dinero
   - Cambiar contraseña
   - Consultar saldo.

   El funcionamiento del cajero es el siguiente:

   a) Se espera a que un usuario ingrese su tarjeta.
   b) Una vez que se ingresa una tarjeta, el cajero obtiene la identificación del usuario al cual pertenece la tarjeta ingresada.
   c) Se pregunta al usuario la contraseña y se verifica que la contraseña ingresada sea correcta. En caso de que la contraseña sea incorrecta, se vuelve a preguntar dos veces más.
   d) Si el usuario ingresó tres veces mal la contraseña, el cajero se queda con la tarjeta, indicando al usuario que debe contactarse con su ejecutivo.

e) Una vez que el usuario ingresa correctamente su contraseña, el cajero le ofrece: (1) obtener dinero, (2) cambiar contraseña, (3) consultar saldo, o (4) retirar tarjeta.

f) Si el usuario pide obtener dinero, entonces el cajero se asegura que haya suficiente dinero en la cuenta y, en tal caso, entrega el dinero, acutaliza el saldo del usuario y termina la operación. De lo contrario se informa que no hay suficiente dinero y termina la operación.

g) Si el usuario decide cambiar su contraseña entonces se le pide la contraseña antigua y, en caso de estar correcta, se pide la contraseña nueva. Luego se realiza el cambio y se termina la operación. En caso de ingresar la contraseña antigua de forma incorrecta, se vuelve a pedir dos veces más. Si la contraseña se ingresa mal tres veces, el cajero se traga la tarjeta tal como en el punto *d*).

h) Si el usuario decide consultar su saldo, se muestra el saldo en pantalla y luego se termina la operación.

i) Una vez terminada la operación, el cajero debe volver a ofrecer las opciones del punto *e*).

j) Si el usuario retira su tarjeta, el programa debe volver al punto *a*).

Para realizar el programa anterior, el cajero provee las siguientes funciones:

- `esperar_ingreso_de_tarjeta()`: Detiene el programa hasta que se ingrese una tarjeta. Mientras tanto, muestra el mensaje el mensaje *Bienvenido, inserte su tarjeta.*
- `obtener_id_desde_tarjeta()`: retorna el id del dueño de la tarjeta.
- `pedir_clave(mensaje)`: pide una clave al usuario mostrándole el mensaje `mensaje`. Retorna la clave ingresada por el usuario.
- `clave_correcta(clave,id_usuario)`: retorna `True` o `False` dependiendo de si `clave` es la clave correcta del usuario cuyo id es `id_usuario`.
- `tragar_tarjeta()`: Se traga la tarjeta y muestra por tres segundos en pantalla el mensaje *Su tarjeta ha sido retenida, por favor contacte a su ejecutivo.*
- `ofrecer_opciones()`: Ofrece las cuatro opciones mencionadas en el punto *e*). Retorna el código de la opción escogida por el usuario de acuerdo al punto *e*).
- `cambiar_clave(id_usuario,nueva_clave)`: Cambia la clave del usuario cuyo id es `id_usuario` por la clave `nueva_clave`.
- `preguntar_cantidad()`: Pregunta al usuario cuánto dinero quiere sacar y retorna el valor escogido por el usuario.
- `obtener_saldo(id_usuario)`: Retorna el saldo del usuario cuyo id es `id_usuario`.
- `mostrar_saldo(monto)`: Muestra tres segundos en pantalla el mensaje *Su saldo es* `monto`.
- `entregar_dinero(monto)`: Entrega `monto` pesos al usuario.
- `actualizar_saldo(nuevo_saldo, id_usuario)`: Cambia el saldo actual del usuario cuyo id es `id_usuario` por `nuevo_saldo`.
- `mostrar_falta_de_saldo()`: Muestra en pantalla por tres segundos el mensaje *Saldo Insuficiente.*
- `expulsar_tarjeta`: Expulsa la tarjeta y muestra por tres segundos en pantalla *Gracias por operar con nosotros.*

Importante: El programa no debe terminar nunca, pues el cajero tiene que funcionar 24/7.

Solución

```
# Función auxiliar para pedir contraseña al usuario
# Retorna True si se autentificó sin problemas
def autentificar_usuario(id_usuario):
    cantidad_ingresos = 1
    clave = pedir_clave("Ingrese su contraseña")

    while not clave_correcta(clave,id_usuario) and cantidad_ingresos < 3:
        clave = pedir_clave("Contraseña incorrecta, inténtelo nuevamente")
        cantidad_ingresos += 1

    return clave_correcta(clave,id_usuario)

# Ciclo principal del cajero
while True:

    esperar_ingreso_de_tarjeta()
    id_usuario = obtener_id_desde_tarjeta()
    tragar = False

    if not autentificar_usuario(id_usuario):
        tragar_tarjeta()
    else:
        opcion = 0
        # (1) obtener dinero, (2) cambiar contraseña, (3) consultar saldo, o (4)
            retirar tarjeta.
        while opcion != 4:
            opcion = ofrecer_opciones()
            if opcion == 1:
                monto = preguntar_cantidad()
                if monto <= obtener_saldo(id_usuario):
                    entregar_dinero(monto)
                else:
                    mostrar_falta_de_saldo()

            elif opcion == 2:
                if not autentificar_usuario(id_usuario):
                    tragar_tarjeta()
                    tragar = True
                else:
                    cambiar_clave(id_usuario,pedir_clave("Ingrese su nueva clave
                        "))

            elif opcion == 3:
                mostrar_saldo(obtener_saldo(id_usuario))
        if not tragar:
            expulsar_tarjeta()
```

4. Un cuadrado latino de tamaño $n \times n$ es una matriz cuadrada cuyas entradas son números naturales entre 1 y n (inclusive), en la cual cada fila y cada columna contiene todos los números entre 1 y n. Por ejemplo, el siguiente es un cuadrado latino de tamaño 5×5:

1	4	2	5	3
5	3	1	4	2
4	2	5	3	1
3	1	4	2	5
2	5	3	1	4

Para representar cuadrados latinos utilizaremos listas. Un cuadrado latino de tamaño $n \times n$ será representado por una lista de tamaño n^2. Los elementos en la primera fila se encontrarán en las entradas $[0]$, $[1]$, ..., $[n-1]$, los de la segunda fila en las entradas $[n]$, $[n+1]$, ..., $[2n-1]$ y así sucesivamente. Por ejemplo, el cuadrado latino mostrado más arriba es representado por la lista

$$[1, 4, 2, 5, 3, 5, 3, 1, 4, 2, 4, 2, 5, 3, 1, 3, 1, 4, 2, 5, 2, 5, 3, 1, 4].$$

En esta pregunta se te pide programar una función `es_cuadrado_latino` que reciba una lista y retorne `True` o `False` dependiendo de si dicha lista representa o no un cuadrado latino.

Puedes suponer que la lista involucrada tendrá como tamaño el cuadrado de un número natural. Además, si `a` es una lista y `n` un número, puedes usar la instrucción `n in a`, que se evalúa a `True` o `False` dependiendo de si `n` se encuentra o no en alguna entrada de `a`.

Para resolver este problema, está prohibido utilizar tanto conjuntos (`set`) como listas de listas (listas multidimensionales).

Solución

```
import math

def es_cuadrado_latino(lista):

    n = int(math.sqrt(len(lista)))

    #verificamos que los números estén entre 1 y n
    for i in range(1,n):
        if lista[i] < 1 or lista[i] > 0:
            return False

    #verificamos filas
    for i in range(n):
        encontrados = []

        #para cada elemento de la fila, si ya lo habíamos
        #encontrado, retornamos False.
        #de lo contrario lo marcamos como encontrado.
        for j in range(n):
            if lista[n * i + j] in encontrados:
                return False
            encontrados.append(lista[n * i + j])

    #verificamos columnas
    for i in range(n):
        encontrados = []

        #para cada elemento de la columna, si ya lo habíamos
        #encontrado, retornamos False.
        #de lo contrario lo marcamos como encontrado.
        for j in range(n):
            if lista[n * j + i] in encontrados:
                return False
            encontrados.append(lista[n * j + i])

    return True
```