Carrera: Ingeniería en Informática

Materia: Informática

Enunciados de ejercicios de práctica obligatoria y opcional

J. Daniel SABELLA ROSA

El presente trabajo responde al ordenamiento temático de los primeros capítulos del libro "Algoritmos y Programación I. Aprendiendo a programar usando Python como herramienta". Los contenidos se nutrieron inicialmente del aporte de Rosita Wachenchauzer y, adicionalmente, de Andrés Otaduy, Silvina Busto, J. Daniel Sabella Rosa, Juan Pablo Peña, Ricardo Tomsic, Juan Roberto Alunni, Pedro Martin Salaberry, Hugo Pirón y Juan Lagostena, bajo la supervisión de Adriana Galli (ciclos lectivos 2013 a 2015). Desde la actualización de 2018 se incluyó en cada capítulo una muy breve introducción dirigida a estudiantes y docentes. A partir de 2020 se modificaron algunos ejercicios para hacerlos compatibles con la versión 3 de Python.

Recopilación y reelaboración de enunciados realizada sobre la base de ejercicios propuestos (ciclos lectivos 2013-2014) y revisados (ciclos lectivos 2014 a 2020) por docentes de la materia Informática, de la carrera Ingeniería en Informática, de la Universidad Nacional de Avellaneda (UNDAV) - **Marzo de 2021**

1. Conceptos básicos: *asignación, secuencia, iteración, estado de variables*

El presente capítulo tiene por intención facilitar la comprensión del uso secuencial de instrucciones de asignación (cambio de estado de variables), de invocación de las funciones de impresión print() y de ingreso de datos input(), y de iteración del ciclo for.

Ejercicio 1.1. Mostrar el resultado de ejecutar en el intérprete de Python 3, la siguiente secuencia de instrucciones; sacar conclusiones sobre qué sucede y escribirlas:

a) >>> alfa = 10

b) >>> beta = 15

c) >>> gama = alfa + beta

d) >>> print (“el valor de la variable alfa es: “, alfa)

e) >>> print (“el valor de la variable beta es: “, beta)

f) >>> print (“el valor de la variable gama es: “, gama)

g) >>> print ("el valor de la expresión 'alfa + beta' es: ", alfa + beta)

h) >>> alfa = alfa + 10

i) >>> print (“el valor de la variable alfa es: “, alfa)

j) >>> beta = beta * 2

k) >>> print (“el valor de la variable beta es: “, beta)

l) >>> print (“el valor de la variable gama es: “, gama)

m) >>> print (‘el valor de la expresión “alfa + beta” es: ‘, alfa + beta)

n) >>> gama = alfa + beta

o) >>> print (“el valor de la variable gama es: “, gama)

p) >>> gama = gama // 3

q) >>> print (“el valor de la variable gama es: “, gama)

r) >>> print ("el valor de la expresión 'alfa + beta' es: ", alfa + beta)

s) >>> gama = gama / 3

t) >>> print (“el valor de la variable gama es: “, gama)

Ejercicio 1.2. Ídem, ejercicio anterior:

a) >>> nombre = input ('Ingrese su nombre: ')

b) >>> edad = int (input('Ingrese su edad en cantidad de años: '))

c) >>> print ('Hola', nombre, ', tu edad es', edad, 'años')

Ejercicio 1.3. Mostrar el resultado de ejecutar en el intérprete de Python 3, la siguiente secuencia de instrucciones; elaborar conclusiones sobre qué sucede y compartirlas:

```
>>> def main():
        tanque = 1000    # Esta variable representa la cantidad de agua que hay en el tanque
        pileta = 3000    # Esta variable representa la cantidad de agua que hay en la pileta
        balde = 10       # Esta variable representa la capacidad de agua del balde
        print  ( 'El tanque contiene:', tanque, 'litros de agua y la pileta:', pileta, 'litros' )
        cant_baldes = 1  # Esta variable representa la cantidad de veces que cargo el balde
        print  ( 'Transvasa', cant_baldes, 'baldes de agua de:', balde, 'litros' )
        tanque = tanque - cant_baldes * balde
        pileta = pileta + cant_baldes * balde
        print  ( 'El tanque contiene:', tanque, 'litros de agua y la pileta:', pileta, 'litros' )
        cant_baldes = cant_baldes + 1
        print  ( 'Transvasa', cant_baldes, 'baldes de agua de:', balde, 'litros' )
        tanque = tanque - cant_baldes * balde
        pileta = pileta + cant_baldes * balde
        print  ( 'El tanque contiene:', tanque, 'litros de agua y la pileta:', pileta, 'litros' )
        cant_baldes = cant_baldes + 1
        print  ( 'Transvasa', cant_baldes, 'baldes de agua de:', balde, 'litros' )
        tanque = tanque - cant_baldes * balde
        pileta = pileta + cant_baldes * balde
        print  ( 'El tanque contiene:', tanque, 'litros de agua y la pileta:', pileta, 'litros' )

>>> main()              # Invocación a la función main para que se ejecute
```

Ejercicio 1.4. Expresar la situación que se describe a continuación, a través de *asignación* sobre *variables* (Python 3) cuyos identificadores son ***dinero*** y ***paraguas***:

Bartolomé vende paraguas. Tiene 10 paraguas y $10000.

a) Escriba instrucciones que permitan ver en pantalla los valores iniciales de las variables ***dinero*** y ***paraguas***.

b) Agregue instrucciones que describan las respectivas variaciones que resultan de la venta de 5 paraguas, a $400 cada uno (utilice variables para representar *la cantidad vendida* y *el precio de venta*), y muestre en pantalla las consecuencias de la venta.

c) Agregue instrucciones para representar y mostrar la venta de la mitad (entera) de paraguas restantes, con la respectiva variación de las variables ***dinero*** y ***paraguas***.

Ejercicio 1.5. Implementar algoritmos que permitan:

a) Obtener el perímetro de un rectángulo, dada su base y su altura.

b) Obtener el área de un rectángulo, dada su base y su altura.

c) Dados dos números *n1* y *n2*, indicar la suma, resta, multiplicación, división y división entera de ambos. Analizar el resultado, con los números: 5 y 2 ; 2 y 5 ; 5 y 0.

Ejercicio 1.6. Mostrar el resultado de ejecutar en el intérprete de Python 3, los siguientes bloques de código; comparar y obtener conclusiones sobre qué sucede en cada caso:

a)
```
>>> for  i  in  range ( 6 ):
        print  ( “el valor del cuadrado de”, i , “es ==>“, i * i )
```

b)
```
>>> for  j  in  range ( 1, 6 ):
        print  ( j ,"dividido 3" )
        print  ( "  división:", j / 3, " división entera:" , j // 3 , " resto:", j % 3 )
```

c)
```
>>> desde = 0
>>> hasta = 6
>>> salto = 2
>>> for  i  in  range ( desde, hasta, salto ):
        print  ( “la potencia”, i , “de dos es:“, 2 ** i )
```

d)
```
>>> for  elem  in [ 1, 2, 4, 3, 5, 6 ]:
        print  ( elem )
```

Ejercicio 1.7. Copiar y ejecutar en Python Tutor (Python 3.6) los casos b), c) y d) de **1.6**

http://www.pythontutor.com/visualize.html#mode=edit

Ejercicio 1.8. Implementar algoritmos que resuelvan los siguientes problemas:

a) Dado un número natural ***n***, imprimir su tabla de multiplicar desde **0** hasta **n**.

b) Dado un número natural ***n***, imprimir la suma total de los naturales de **1** a **n**.

Ejercicio 1.9. Reescribir el programa del ejercicio 1.3 para que *pida al usuario que ingrese* la cantidad de agua inicial del ***tanque*** y la ***pileta***, y que muestre los cambios, al transvasar *un balde*, luego otros *dos*, luego otros *tres*, luego otros *cuatro* y, finalmente, otros *cinco*.

Ejercicio 1.10. Para la siguiente secuencia de código Python 3.6, complete respectivamente el estado de las variables y la salida que se despliega en pantalla (sólo cuando ésta ocurre):

Instrucción Python	Estado de variables	Salida en pantalla
>>> n1 = 3	n1 → … 3	
>>> n2 = 5	n1 → … 3 n2 → … 5	
>>> for x in range (n1, n2): print ("Valor ==> ", x * x)	n1 → … 3 n2 → … 5 x → … 9	
	n1 → … 3 n2 → … 5 x → … 16	
>>> print ("Valor final de x: ", x)	n1 → … n2 → … x → …	
>>> n2 = n2 // n1	n1 → … n2 → … x → …	
>>> print ("Valor de n2: ", n2)	n1 → … n2 → … x → …	

Ejercicio 1.11. Copiar y ejecutar en Python Tutor (Python 3.6) la secuencia de código del ejercicio **1.10**

http://www.pythontutor.com/visualize.html#mode=edit

2. Programas sencillos (metodología de resolución de problemas)

El presente capítulo tiene por intención facilitar la aplicación de los primeros tres pasos de la metodología propuesta en la bibliografía principal.

Ejercicio 2.1. Desarrolle los primeros 3 pasos de la metodología de construcción de programas, para el caso que se enuncia a continuación: Escribir un programa que le solicite al usuario una cantidad de pesos (capital), una tasa de interés anual (tasa) y un número entero de años (tiempo), para mostrar, como resultado, el monto final a obtener, de acuerdo a la fórmula:

$$C_n = C \times (1 + t / 100)^n$$

Donde **C** es el capital inicial, **t** es la tasa de interés y **n** es el tiempo en años. Identifique las variables con nombres adecuados a su significado.

Ejercicio 2.2. Ídem anterior: Escribir un programa Python que le solicite al usuario que ingrese el costo (en pesos) de cada uno de los siguientes dispositivos de almacenamiento:

- ✓ Memoria RAM de 4GB para PC
- ✓ Pendrive 16 GB
- ✓ Disco rígido interno 2 TB

Considerando que en el Sistema Internacional de Unidades (Decimal) un Terabyte (TB) equivale a 10^{12} bytes y un Gigabyte (GB) equivale a 10^9 bytes, el programa deberá informar en la pantalla:

a) ¿Cuánto costaría almacenar 1 GB en cada uno de esos dispositivos?

b) ¿Cuánto más cara (en forma relativa) es la memoria RAM que el pendrive?

c) ¿Cuánto más cara (en forma relativa) es la memoria RAM que el disco rígido?

d) ¿Cuánto más caro (en forma relativa) es el pendrive que el disco rígido?

Ejercicio 2.3. Ídem anterior: Escribir un programa que solicite diez valores expresados en grados Fahrenheit y los muestre convertidos a grados Celsius. Considere que la fórmula para conversión de grados Celsius a grados Fahrenheit es:

$$\mathbf{F} = 9 / 5 \times \mathbf{C} + 32$$

3. Funciones

El presente capítulo tiene por intención facilitar la comprensión del uso de funciones (su definición y documentación, con o sin parámetros, su invocación con o sin argumentos y la correspondiente utilización según sea que devuelva resultado o no) y de módulos Python.

Ejercicio 3.1. Mostrar el resultado de ejecutar en Python 3 la función `main()` y elaborar conclusiones (señalar definiciones e invocaciones de función, distinguir formas de invocación, señalar argumentos y cambios de estado en parámetros, finalmente ejecutar la instrucción comentada con #):

```
def triplica(param):
  """ Recibe valor en param y devuelve el valor de resulta """
  resulta = param * 3
  return resulta

def prueba_triplica(entrada):
  """ Recibe valor en entrada y muestra resultados de invocaciones """
  print ( '---[ Debug:] valor del parametro', entrada )
  salida = triplica(entrada)
  print ( 'La salida de', entrada, 'es ==>', salida )
  entrada = 'alfa'
  print ( '\n---[ Debug:] valor del argumento es', entrada )
  salida = triplica(entrada)
  print ( 'La salida de', entrada, 'es ==>', salida )
  print ( '---[ Debug:] valor del parametro', entrada )
  print ( '\nError de invocacion sin parentesis ==>', triplica )

def main():
  prueba_triplica(8)
  print ( '\n-[ Debug:]: valor del argumento es 200' )
  print ( 'La salida de', 200, 'es ==>', triplica(200) )
  #print ('\nError: invoca sin argumentos ==>', triplica())
```

Ejercicio 3.2. Definir (y documentar) con un nombre apropiado, una función:

a) Que calcule e imprima el valor de cualquier número ***n*** elevado al cubo.

b) Modificar la solución anterior, para que la función calcule cualquier potencia dada, de cualquier número dado y devuelva el resultado al programa principal. Pruebe la función con valores: 2^{0}, 2^{1}, 2^{2}, 2^{3} y $36^{0.5}$. Consultar la documentación mediante la función *help( )* provista por Python y verificar: ¿todavía describe adecuadamente lo que hace la función? ¿el nombre de la función sigue siendo apropiado?

Ejercicio 3.3. Definir (y documentar) dos funciones con nombre apropiado. Una función que**…:**

a) Calcule la *cantidad de segundos* de un tiempo dado (por parámetros) en *horas*, *minutos* y *segundos*, y devuelva el resultado al programa principal.

b) Calcule la *cantidad de horas*, *minutos* y *segundos* de un tiempo dado (por parámetro) en *segundos*, y devuelva el resultado al programa principal.

Ejercicio 3.4. Escribir un programa que lea de teclado dos tiempos expresados en *horas*, *minutos* y *segundos*, los sume y muestre por pantalla el resultado en *horas*, *minutos* y *segundos*. El programa deberá utilizar las funciones del ejercicio anterior.

Ejercicio 3.5. Definir (con nombre apropiado) una función que reciba por parámetros la *cantidad de bytes* descargados y el tiempo expresado en *minutos* y *segundos*, y devuelva la *velocidad de descarga* expresada en kB/seg (kilobytes por segundo). [Considere el Sistema Internacional de Unidades (Decimal) donde un kilobyte (kB) equivale a 10^3 bytes].

Ejercicio 3.6. Definir (y documentar) con un nombre apropiado, una función que reciba por parámetros dos números, cuyos valores representan la base y la altura de un triángulo, y devuelva como resultado el área respectiva. Pruebe la función con los pares (2, 4) y (3, 5).

Ejercicio 3.7. Definir (y documentar) dos funciones. Una función **…**

a) denominada “numero_pi”, que devuelva como resultado el valor redondeado del número PI: 3.14159. [Utilice el dato *math.pi* del módulo *math* y la función *round(n, d)*]

b) con nombre apropiado, que reciba el radio de un círculo por parámetro y devuelva como resultado el valor del área respectiva. Utilice la función del ítem a).

Ejercicio 3.8. Definir (y documentar) una función que reciba un número ***n*** por parámetro y muestre en pantalla los primeros ***n*** números triangulares, junto con su índice (considerar que el número triangular de orden *x* se obtiene mediante la suma de los números naturales desde 1 hasta *x*). Es decir, si se piden los primeros 5 números triangulares, la función debería imprimir:

1 - 1
2 - 3
3 - 6
4 - 10
5 - 15

[Importante: No usar la ecuación $\sum_{i=1}^{n} i = n * (n + 1) / 2$]

Ejercicio 3.9. Definir (y documentar) una función denominada "factorial", que reciba un número natural por parámetro y devuelva como resultado su factorial (cuya definición matemática es: n ! = 1 x 2 x 3 x … x (n-1) x n).

Ejercicio 3.10. Para la siguiente secuencia de código Python 3.6, complete respectivamente el estado de las variables y la salida que se despliega en pantalla (sólo si ésta ocurre):

Instrucción Python	Estado de variables	Salida en pantalla
`def f_calc (n):` `  resulta = n * 2 // 3` `  return resulta`		
`acum = 1.0`	acum → …..	
`for ix in range (2,5,2):` `  acum = acum * f_calc(ix)` `  print (ix, "==>", acum)`	ix → ….. acum → …..	
	ix → ….. acum → …..	
`print ("Final:",ix,"y",acum)`	ix → ….. acum → …..	

Opcionales:

Ejercicio 3.11. Definir una función denominada "pinta_cuadro", que reciba como parámetro un número natural ***n*** y dibuje con el carácter **'x'** un cuadrado relleno, de lado igual al parámetro. Ejemplo: `pinta_cuadro (3)`

```
xxx
xxx
xxx
```

Ejercicio 3.12. Definir una función denominada "domino_n", que reciba como parámetro un número natural ***n*** y que imprima por pantalla, de una por línea y sin repetir, todas las fichas de un juego similar al dominó, incluyendo los números de ***0*** a ***n***.

Ejercicio 3.13. Definir una función denominada "suma_n", que reciba como parámetro un número natural ***n*** y devuelva como resultado la suma de naturales desde ***1*** hasta ***n***.

4. Decisiones

El presente capítulo tiene por intención facilitar la comprensión del uso de expresiones booleanas y del condicional if en sus distintas formas (if…, if… else, if… elif, if… elif… else).

Ejercicio 4.1. Definir y documentar : …

a) una función denominada “es_par” que, dado un número entero por parámetro, devuelva un valor booleano que indique si es par, o no. Dé un ejemplo de invocación.

b) Modificar la solución anterior, para que devuelva como resultado “S” si es par, “N” si es impar o “0” si es cero. Consulte la documentación mediante la función *help ()*.

Ejercicio 4.2. Definir y documentar una función denominada “val_abs”, que reciba un número por parámetro y devuelva como resultado su valor absoluto. [No utilizar la función *abs ()* provista por Python]

Ejercicio 4.3. Definir una función denominada “mayor_de_3” que devuelva como resultado el mayor de tres números dados por parámetro. Pruebe la función con 6 ternas de valores.

Ejercicio 4.4. Escribir un programa que pida al usuario que ingrese dos números naturales, *n1* y *n2*, e imprima en pantalla la secuencia de enteros comprendida entre *n1* y *n2* (incluidos) con la siguiente particularidad: si el número es múltiplo de 3, en lugar del número debe imprimir “TRES”, si es múltiplo de 5, en vez del número debe imprimir “CINCO” y si es múltiplo de 3 y de 5, en lugar del número debe imprimir “TRES Y CINCO”.

Ejercicio 4.5. Definir una función denominada “es_bisiesto” que reciba por parámetro un número, que representa un año, y devuelva un resultado booleano que indique si es, o no es, bisiesto. Pruebe la función para múltiplos de 4, de 100 y de 400. [Nota: un año es bisiesto si es un número divisible por 4, pero no es divisible por 100, excepto que también sea divisible por 400. Ejemplos: 1984 y 2000 son bisiestos, 1800 no es bisiesto].

Ejercicio 4.6. Definir una función “cant_dias_mes” que, dados por parámetro dos números, que representan el mes y el año, devuelva como resultado la cantidad de días correspondientes al mes. En la definición se debe invocar a la función del enunciado 4.5.

Ejercicio 4.7. Definir una función “fecha_valida” que, dada por parámetros una fecha en números (día, mes, año), devuelva un resultado booleano que indique si es válida o no. ¿Qué función debería invocar?

Ejercicio 4.8. Definir una función “fecha_siguiente” que, dada por parámetros una fecha en números (día, mes, año), devuelva como resultado la fecha (día, mes, año) del día siguiente.

Ejercicio 4.9. Para la siguiente secuencia de código Python 3.6, complete respectivamente el estado de las variables y la salida que se despliega en pantalla (sólo si ésta ocurre):

Instrucción Python	Estado de variables	Salida en pantalla
>>> for boole in (True,False): b1 = boole b2 = b1 for cant in range(2): print (b1, 'y', b2, '=', b1 and b2) b2 = not b2	boole → ….. b1 → ….. b2 → ….. cant → ….. b2 → …..	
	boole → ….. b1 → ….. b2 → ….. cant → ….. b2 → …..	
	boole → ….. b1 → ….. b2 → ….. cant → ….. b2 → …..	
	boole → ….. b1 → ….. b2 → ….. cant → ….. b2 → …..	
>>> print (b1, 'o', b2, 'es', b1 or b2)	b1 → ….. b2 → …..	

Opcionales:

Ejercicio 4.10. Definir una función denominada “es_primo”, que reciba un número entero por parámetro y devuelva un resultado booleano que indique si es primo, o no. [Un número natural es primo, si solamente es divisible por sí mismo y por 1].

Ejercicio 4.11. Definir una función denominada “rango_etario”, que reciba por parámetro un número natural (que representa una edad) y devuelva como resultado la denominación de su respectivo grupo etario. Considere la siguiente clasificación: “primera infancia” (0 a 5 años), “infancia” (6 a 11 años), “adolescencia” (12 a 18 años), “juventud” (19 a 29 años), “adultez” (30 - 64 años) y “vejez” (65 años o más).

Ejercicio 4.12. Definir una función “nombre_dia” que, dado por parámetro un número de día de la semana, devuelva como resultado su nombre. Considere 2do día a “lunes”.

Ejercicio 4.13. Escribir un programa de astrología que pida al usuario que ingrese el número de día y el número de mes correspondientes a su fecha de cumpleaños, e imprima en pantalla el signo del zodíaco al que pertenece. Considere las siguientes fechas:

Aries: 21 de marzo al 20 de abril.
Tauro: 21 de abril al 20 de mayo.
Geminis: 21 de mayo al 20 de junio.
Cáncer: 21 de junio al 21 de julio.
Leo: 22 de julio al 22 de agosto.
Virgo: 23 de agosto al 22 de septiembre.
Libra: 23 de septiembre al 22 de octubre.
Escorpio: 23 de octubre al 22 de noviembre.
Sagitario: 23 de noviembre al 21 de diciembre.
Capricornio: 22 de diciembre al 20 de enero.
Acuario: 21 de enero al 19 de febrero.
Piscis: 20 de febrero al 20 de marzo.

5. Ciclos

El presente capítulo tiene por intención facilitar la comprensión del uso de ciclos indefinidos (while), particularmente en sus dos formas: ciclo interactivo y ciclo con centinela.

Ejercicio 5.1. Considerando la siguiente función para obtener la multiplicación de dos números, por sumas sucesivas:

```
def prod_en_sumas_for (num, cant):
    """Calcula el producto de dos números por sumas sucesivas"""
    prod = 0
    for i in range(cant):
        prod = prod + num
    return prod
```

Definir (y documentar) una función denominada “prod_en_sumas” que reciba por parámetros dos números enteros y devuelva como resultado su multiplicación, obtenida por sumas sucesivas mediante un *ciclo indefinido*. Incluya instrucciones de depuración (debugging) que muestren todas las sumas que se realizan hasta obtener el resultado final. Pruebe el comportamiento de la función cuando el primero de los dos números es negativo. ¿Qué sucede si el número negativo es el segundo?

Ejercicio 5.2. Escribir un programa:

a) que contenga una contraseña de 4 caracteres inventada, que le pregunte al usuario la contraseña y no le permita continuar hasta que la haya ingresado correctamente.

b) Modificar el programa anterior para que solamente permita una cantidad fija de intentos. Al finalizar, deberá imprimir en pantalla “Eureka” si acertó la clave o, en caso contrario, “Clave incorrecta” y la cantidad de intentos fallidos.

c) Modificar el programa anterior para que después de cada intento agregue una pausa cada vez mayor, utilizando la función *time.sleep(…)* del módulo *time*.

Ejercicio 5.3. Escribir un programa que elija un número al azar, entre 1 y 99, y lo mantenga en secreto (utilice la función *random.randrange (1,100)* del módulo *random*). El programa debe solicitar al usuario que adivine el número. Si el número que ingresa el usuario es mayor, el programa debe responder "Incorrecto, mi número es menor"; si el número ingresado es menor, el programa debe responder "Incorrecto, mi número es mayor". En ambos casos deberá solicitar otro número hasta que el usuario acierte el correcto. Mostrar la cantidad de intentos realizados para adivinar.

Ejercicio 5.4. Definir una función "max_com_div" que, dados por parámetro dos números ***n*** y ***m***, devuelva como resultado el Máximo Común Divisor entre ambos, implementando el algoritmo de Euclides. Se describen a continuación los pasos de ese algoritmo matemático:

1. Teniendo **n** y **m**, se obtiene **r**, el resto de la división entera de **m** / **n**.
2. Si **r** es cero, entonces **n** es el MCD de los valores iniciales.
3. Se reemplaza **m** ← ***n***, **n** ← **r**, y se vuelve al primer paso.

Hacer un seguimiento del algoritmo implementado, para los pares de números: (15,9); (9,15); (10,8) y (12,6).

Ejercicio 5.5. Escribir un programa que reciba, una a una, las calificaciones del usuario, preguntando a cada paso si desea ingresar más notas; finalmente, el programa debe imprimir el promedio correspondiente y el valor de la calificación más baja.

Ejercicio 5.6. Definir y documentar una función "es_potencia_de_dos" que reciba como parámetro un número natural, y devuelva el valor booleano *True* si el número es una potencia de 2, y *False* en caso contrario.

Ejercicio 5.7. Definir y documentar una función que, dados dos números naturales pasados como argumentos (parámetros de la función), devuelva la suma de todas las potencias de 2 que hay en el rango formado por esos números (0, si no hay potencia de 2 entre ellos). Utilice la función "es_potencia_de_dos" del ejercicio anterior.

Ejercicio 5.8. Para la siguiente secuencia de código Python 3.6, complete respectivamente el estado de las variables y la salida que se despliega en pantalla (sólo si ésta ocurre):

Instrucción Python	Estado de variables	Salida en pantalla
`>>> n = 12`	n → ….	
`>>> while n > 0:` `  print ('[DEBUG]', n)` `  if n % 4 == 0:` `  print ('Múltiplo')` `  n = n // 3`	n → ….	
	n → ….	
	n → ….	
`>>> print ('valor de n:', n)`		

Ejercicio 5.9. Escribir un programa que le pida al usuario que ingrese una sucesión de calificaciones de estudiantes (primero el número de legajo, luego la calificación, y así para cada estudiante, hasta que el usuario ingrese el legajo **-1** como condición de salida). Al final, el programa debe imprimir cuántas calificaciones fueron ingresadas, su valor máximo ingresado, la suma total de las calificaciones y el promedio.

Opcionales:

Ejercicio 5.10. Definir una función que reciba como parámetro un número natural e imprima todos los números primos que hay hasta ese número. Utilice la función del ejercicio 4.10.

Ejercicio 5.11. Modificar el programa del enunciado 5.3, para que el usuario pueda dar por terminado el juego, ingresando el valor 0.

Ejercicio 5.12. Definir una función "reloj" que, dados por parámetro tres números naturales que representan la hora, los minutos y los segundos iniciales, imprima en pantalla, a cada segundo, la información horaria actualizada. Considere válidas las horas de 0:00:00 hasta 23:59:59. Utilice la función *time.sleep(1)* del módulo *time*.

Ejercicio 5.13. Para la siguiente porción de código Python 3.6, complete respectivamente el estado de la variable y la salida que se despliega en pantalla (sólo si ésta ocurre):

Instrucción Python	Estado de variables	Salida en pantalla
`>>> c = 1`	c → **...**	
`>>> while c < 20:` `  c = c + 5` `  print ('//:',c // 4)` `  if c % 4 > 0:` `  print ('%:', c % 4)` `  print ('x' * (c // 4))`	c → **...** **...**	

6. Cadenas de caracteres

El presente capítulo tiene por intención facilitar la comprensión del tratamiento de cadenas de caracteres y del uso de operaciones con secuencias.

Ejercicio 6.1. Suponer qué hace el siguiente código y luego comprobar su ejecución:

```
def pba_cadenas():
    """ Prueba operaciones con cadenas """
    palabra = "murcielago"
    pausa = input('dar enter para seguir >>> ')
    print ( 'palabra:', palabra )
    print ( 'palabra [0]:', palabra [0] )
    print ( 'palabra [-1]:', palabra [-1] )
    seg1 = palabra [:3]
    seg2 = palabra [3:6]
    seg3 = palabra [6:]
    pausa = input('\ndar enter para seguir >>> ')  # '\n' salta linea
    print ( 'seg1 + seg2 + seg3:', seg1 + seg2 + seg3 )
    print ( 'seg1 , seg2 , seg3:', seg1, seg2, seg3 )
    print ( 'seg3 [0:2] * 3:', seg3 [0:2] * 3 )
    pausa = input('\ndar enter para seguir >>> ')
    print ( 'seg3 [0:-1]:', seg3 [0:-1] )
    print ( 'seg3 [-4:-1]:', seg3 [-4:-1] )
    largo = len(seg3)
    pausa = input('\ndar enter para seguir >>> ')
    print ( 'largo:', largo )
    print ( 'seg3[-largo]:', seg3[-largo] )
    pausa = input('\ndar enter para seguir >>> ')
    #print 'seg3[largo]:', seg3[largo]
    for ind in range(largo):
        print ('indice:', ind, 'caracter:', seg3[ind])
    pausa = input('\ndar enter para seguir >>> ')
    for letra in seg3:
        print ( 'caracter:', letra )
    pausa = input('\ndar enter para seguir >>> ')
    #palabra[1] = 'o'
    copia = ""
    print ( 'copia:', copia, ' ==> len():', len(copia) )
    for letra in seg3:
        copia = copia + letra
        print ('copia:', copia, ' ==> len():', len(copia))
```

Ejercicio 6.2. Definir una función “segm_3_txt” que, dados como parámetros una cadena de caracteres y un carácter (que denominaremos selector),

a) imprima los tres primeros caracteres de la cadena, si el valor del selector es la letra ‘P’, o los tres últimos caracteres si el valor del selector es ‘U’, o el mensaje ‘Error en el selector’ si el valor del selector no es ‘P’ ni ‘U’.

b) modificar la solución anterior, para que sólo imprima el primero o el último carácter, respectivamente, cuando la longitud de la cadena sea menor que 3.

Ejercicio 6.3. Definir una función que reciba una cadena de caracteres como parámetro y devuelva como resultado la cadena invertida. (Ej: para el argumento ‘Hola Undav!’ debería devolver al programa principal ‘!vadnU aloH’. No utilizar segmento de cadena [**: :** -1])

Ejercicio 6.4. Definir una función que reciba una cadena de caracteres como parámetro e imprima la cadena original yuxtapuesta a la cadena invertida. [Ej: para el argumento ‘espejo’ debería imprimir ‘espejoojepse’]. ¿Qué función debería invocar?

Ejercicio 6.5. Definir una función “intercala_chr” que, dados como parámetros una cadena de caracteres y un carácter, devuelva como resultado la cadena con el carácter insertado entre sus caracteres originales. [Ej: para los argumentos ‘veamos’ y ‘-‘, debería devolver al programa principal ‘v-e-a-m-o-s’]

Ejercicio 6.6. Definir una función “sustituye_chr” que, dados como parámetros una cadena de caracteres ***txt*** y dos caracteres ***c1*** y ***c2***, devuelva como resultado una cadena con la sustitución, en ***txt***, de todos los caracteres iguales a ***c1***, por el carácter ***c2*****.** [Ej: pasados como argumentos el texto ‘mi primer modulo.py’, el carácter ‘ ‘ y el carácter ‘_‘, debería devolver al programa principal ‘mi_primer_modulo.py’]

Ejercicio 6.7. Definir una función ‘oculta_digitos’ que, dada una cadena de caracteres como parámetro, devuelva como resultado la cadena con todos sus dígitos reemplazados por el carácter ‘*’. [Ej: para el argumento ‘su clave es: 1540’, debería devolver al programa principal ‘su clave es: ****’]

Ejercicio 6.8. Definir una función ‘separa_miles’ que, dada por parámetro una cadena de caracteres que contiene un largo número entero, devuelva como resultado la cadena con las separaciones de miles incluidas en el número. [Ej: para el argumento ‘1234567890’ debería devolver al programa principal ‘1.234.567.890’]

Ejercicio 6.9. Definir una función que, dadas dos cadenas de caracteres como parámetros, devuelva como resultado la cadena que sea anterior en orden alfabético. [Ej: para los argumentos 'kde' y 'gnome' debería devolver al programa principal 'gnome']

Ejercicio 6.10. Para las siguientes secuencias de código Python 3, complete el estado de las variables y la salida que se despliega en pantalla (sólo si ésta ocurre), respectivamente:

a)

Instrucción Python	Estado de variables	Salida en pantalla
`tope = 8` `mensaje = 'triangulo'` `while tope > 0:` `  print (mensaje[:tope])` `  tope = tope - 3` `print (mensaje[tope])`	`tope` → … `mensaje` → … … … …	

b)

Instrucción Python	Estado de variables	Salida en pantalla
`n = 3` `argum = "canibales"` `largo = len (argum)` `while largo > 0:` `  st = argum[-largo:]` `  resu = ""` `  for i in range (0,len(st),n):` `  resu = resu + st[i]` `  print (st, '>', resu)` `  largo = largo - n` `print ("Final:", st )`	`n` → … `argum` → … `largo` → … `st` → … `resu` → … `i` → … … … …	

Opcionales:

Ejercicio 6.11. Definir una función 'letras_iniciales' que, dada por parámetro una cadena de caracteres, devuelva como resultado una cadena con las primeras letras de cada palabra. [Ej: para el argumento 'Universal Serial Bus' debería devolver al programa principal 'USB']

Ejercicio 6.12. Definir (y documentar) una función 'inicial_mayuscula' que, dada por parámetro una cadena de caracteres, devuelva como resultado dicha cadena con la primera letra de cada palabra en mayúsculas. [Ej: para el argumento 'república argentina' debería devolver al programa principal 'República Argentina'].
Utilice la función *ord(…)*, que devuelve el número ordinal de un carácter, y la función *chr(…)*, que devuelve el carácter correspondiente a un ordinal del rango [0,256]. Ejemplos: *ord ('a')* es 97 y *chr (65)* es 'A'.

Ejercicio 6.13. Definir una función 'sin_vocales' que, dada por parámetro una cadena de caracteres, devuelva la cadena sin las letras vocales. [Ej: para el argumento 'República Argentina' debería devolver al programa principal 'Rpblc rgntn'].

Ejercicio 6.14. Definir una función 'es_palindromo' que, dada por parámetro una cadena de caracteres, devuelva un resultado booleano que indique si se trata de un palíndromo. [Ej: 'arroz a la zorra' es palíndromo, porque se lee igual de izquierda a derecha que de derecha a izquierda]. ¿Qué función debería invocar?

Ejercicio 6.15. Definir una función 'es_subcadena' que, dadas por parámetro dos cadenas de caracteres, devuelva un resultado booleano que indique si la primera cadena está contenida en la segunda. [Ej: la cadena 'bandera' es subcadena de 'abanderado'.]

Ejercicio 6.16. Definir una función 'binario_a_decimal' que reciba por parámetro una cadena de unos y ceros (es decir, un número en representación binaria) y devuelva como resultado el valor decimal correspondiente. [Ej: para el argumento '0111' debería devolver al programa principal el valor 7, para '1000' el valor 8, etc.]

Ejercicio 6.17. Definir una función 'cuenta_letra' que, dados como parámetros un carácter y una cadena de caracteres, devuelva como resultado la cantidad de veces que se encuentra ese carácter en la cadena. [Ej: en 'república argentina' hay 3 'a'.]

Ejercicio 6.18. Definir una función "matriz_identidad" que reciba como parámetro una dimensión ***n***, e imprima la matriz identidad correspondiente a esa dimensión. [Nota: La matriz identidad de dimensión ***n*** (de ***n*** filas, por ***n*** columnas), es la matriz diagonal que tiene valor 1 en cada elemento de su diagonal principal y 0 en el resto de los elementos]. Ejemplo para dimensión 3:

```
1 0 0
0 1 0
0 0 1
```

7. Tuplas y Listas

El presente capítulo tiene por intención facilitar la comprensión del tratamiento de tuplas y listas y de la diferenciación entre secuencias inmutables y mutables.

Ejercicio 7.1. Suponer qué hace el siguiente código y luego comprobar su ejecución:

```
def pba_tuplas():
    """ Prueba operaciones con tuplas """
    tup1 = (1, "manzana", "uva", "pera")
    tup4 = (4, "roja", "azul", "amarilla")
    tuplete = tup1 + tup4
    tuplon = tup1, tup4
    tuplita = tup1 [1:3] + tup4 [1:3]
    incogn = tup1 [0] + tup4 [0]
    pausa = input('dar enter para seguir >>> ')
    print ( "tuplete:", tuplete )
    print ( "tuplon:", tuplon )
    print ( "tuplita:", tuplita )
    print ( "incogn:", incogn )
    no1, no2, no3, no4 = tuplita
    nome = tuplon[0][1]
    pausa = input('dar enter para seguir >>> ')
    print ("no1:", no1," no2:", no2, " no3:", no3, " no4:", no4)
    print ("nome:", nome)
    pausa = input('dar enter para seguir >>> ')
    for palabra in tup1:
        print ( "Hola", palabra )
    if len(tup1) == len(tup4):
        for pos in range (1,len(tup1)):
            print ('pos:', pos, "elementos:", tup1[pos], '-', tup4[pos])
    pausa = input('dar enter para seguir >>> ')
    for pos in range(len(tuplon)):
        print ("es", tuplon[pos][1])
    tup_f = ("nadie")
    tup_v = ()
    tup_v = tup_v + ("nadie",)
    pausa = input('dar enter para seguir >>> ')
    if tup_f == tup_v:
        print ( "tup:", tup_f )
    else:
        print ( "tup_f:", tup_f, "  tup_v:", tup_v )
```

Tuplas

Ejercicio 7.2. Definir una función …

a) que reciba como parámetro una tupla ***tup*** con nombres, y para cada nombre imprima el mensaje “Estimado <nombre>, vote por mí “.
b) modificar la solución anterior, para que el mensaje distinga el género del destinatario, considerando que ***tup*** es una tupla integrada por tuplas de la forma (nombre, género). [Valores de género: ‘M’ → masculino; ‘F’ → femenino; ‘O’ → otres]

Ejercicio 7.3. Definir una función ‘encaja_domino’, que reciba por parámetro dos tuplas que representan fichas de dominó y devuelva un resultado booleano que indique si encajan o no. [Ej: las fichas (3, 4) y (4, 1) encajan, porque coinciden en el número 4. Ídem (4, 4) y (5, 4)]

Ejercicio 7.4. Definir una función ‘suma_vectores’ que, dadas por parámetro dos tuplas que representan vectores de igual dimensión, devuelva como resultado la tupla que representa su suma. [La suma de vectores (u1, u2, u3) y (v1, v2, v3), se puede calcular como el vector que resulta de sumar sus componentes homólogas: (u1 + v1 **,** u2 + v2 **,** u3 + v3)].

Ejercicio 7.5. Definir una función ‘producto_escalar’, que reciba como parámetros dos tuplas, que representan vectores de igual dimensión, y devuelva como resultado el valor de su producto escalar. [El producto escalar de los vectores (u1, u2, u3) y (v1, v2, v3), se puede calcular como el valor numérico que resulta de sumar los productos de las componentes homólogas: u1 * v1 + u2 * v2 + u3 * v3]

Opcionales:

Ejercicio 7.6. Definir una función ‘es_tupla_ordenada’, que reciba como parámetro una tupla de elementos y devuelva un resultado booleano que indique si se encuentran ordenados de menor a mayor.

Ejercicio 7.7. Definir una función ‘tanto_envido’ que, dadas por parámetro dos tuplas que representan naipes españoles de la forma (palo, numero), devuelva como resultado el valor de los puntos del envido en el juego del truco. [Si ambos palos son iguales, el envido se calcula con la suma del “valor para envido” de los números numero1 y numero2, a lo cual se suman 20 puntos. Si ambos palos difieren, el envido se calcula con el mayor “valor para envido” entre numero1 y numero2. El valor para envido de las figuras (sota, caballo o rey) es cero, mientras que para el resto de los naipes españoles, es el propio número.]

Ejercicio 7.8. Suponer qué hace el siguiente código y luego comprobar su ejecución:

```
def pba_listas():
    """ Prueba operaciones con listas """
    palabra = "murcielago"
    tup1 = (1, "manzana", "uva", "pera")
    tup4 = (4, "roja", "azul", "amarilla")
    listup = [tup1]
    listup.append (tup4)
    listup.append (palabra[3:-3])
    longi = len(listup)
    pausa = input('dar enter para seguir >>> ')
    print ("listup:", listup)
    print ("longi:", longi)
    print ("listup [1:2]", listup [1:2])
    nom1, nom2, nom3 = listup
    nome = listup [0][1]
    pausa = input('dar enter para seguir >>> ')
    print ("nom1:", nom1, "  nome:", nome)
    pos = listup.index('ciel')
    listup.insert (pos,'mur')
    pausa = input('dar enter para seguir >>> ')
    print ("pos:", pos)
    print ("listup:", listup)
    listup.remove('mur')
    pausa = input('dar enter para seguir >>> ')
    print ("listup:", listup)
    lisa = []
    for digi in range(97,102):
        lisa.append(digi)
    pausa = input('dar enter para seguir >>> ')
    print ('lisa:', lisa)
    for num in lisa:
        if num < 100:
            letra = chr(num)
            pos = lisa.index(num)
            lisa.remove(num)
            lisa.insert(pos,letra)
    lisa[4] = 'd'
    pausa = input('dar enter para seguir >>> ')
    print ('lisa:', lisa)
```

Listas

Ejercicio 7.9. Definir dos funciones. Una función denominada…

a) ‘interseccion’ que reciba por parámetro dos listas, que representan conjuntos, y devuelva como resultado otra lista que incluya, sin repeticiones, los elementos comunes a ambas listas pasadas como argumentos.

b) ‘union’ que reciba por parámetro dos listas, que representan conjuntos, y devuelva como resultado otra lista que incluya, sin repeticiones, los elementos que pertenezcan a una u otra lista pasadas como argumentos.

Ejercicio 7.10. Definir dos funciones. Una función que reciba por parámetro una lista de números enteros y…

a) devuelva como resultado otra lista con los factoriales de esos números. Utilice la función del ejercicio 3.9.

b) devuelva como resultado otra lista que incluya solamente los que sean primos. Utilice la función del ejercicio 4.10.

Ejercicio 7.11. Definir dos funciones. Una función con dos parámetros, una lista de enteros y un entero ***k***,…

a) que devuelva como resultado otra lista que incluya sólo los que sean múltiplos de ***k***.

b) que devuelva como resultado otra lista compuesta por tres listas: la primera con los mayores a ***k***, la segunda con los iguales a ***k*** y la tercera con los menores.

Ejercicio 7.12. Definir una función ‘lista_de_minimos’ que, dadas por parámetro dos listas de igual longitud, compuestas por números, devuelva como resultado una nueva lista que contenga en cada posición el menor entre los elementos correspondientes de ambas listas.

Ejercicio 7.13. Definir una función que reciba como parámetro una lista de tuplas, cada una de la forma (apellido, nombre, inicial_segundo_nombre), y devuelva como resultado una lista de cadenas de caracteres, cada una de las cuales contenga primero el nombre, un espacio, luego la inicial con un punto, un espacio y luego el apellido.

Ejercicio 7.14. Definir dos funciones. Una función que reciba como parámetro una lista,…

a) y devuelva como resultado una nueva lista con los mismos elementos, pero en orden invertido. (Ej: para el argumento [‘Di’, ‘buen’, ‘dia’, ‘a’, ‘papi’], debería devolver al programa principal [‘papi’, ‘a’, ‘dia’, ‘buen’, ‘Di’], sin modificar la lista original.)

b) que la invierta (sin usar listas auxiliares) y la devuelva modificada al programa principal. ¿Es necesaria la instrucción *return* en la función?

Ejercicio 7.15. Para las siguientes secuencias de código Python 3, dados los valores de los identificadores que se enuncian en cada caso, complete respectivamente (en hoja aparte) el estado de las variables y la salida que se despliega en pantalla (sólo si ésta ocurre):

a) Sean los valores de los identificadores:

```
lis1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
lis2 = [[10, 20, 30], [40, 50, 60], [70, 80, 90]]
lis3 = []
```

Instrucción Python	Estado de variables
`>>> if len(lis1) == len(lis2) :` `  tope = len(lis1)` `else:` `  tope = 0`	tope → …
`>>> for  f  in range (tope) :` `  fila = []` `  for c in range(tope) :` `  fila.append(lis1[f][c] + lis2[f][c])` `  lis3.append (fila)`	f → … fila → … c → … fila → … lis3 → …

b) Sean los valores de los identificadores:

```
luno = []
ldos = luno
```

Instrucción Python	Estado de variables	Salida en pantalla
`>>> for i in range(3):` `  luno.append(i)` `  print (luno)` `  ldos.insert(0,i)` `  print (ldos)`	luno → … ldos → …	

Opcionales:

Ejercicio 7.16. Definir una función 'encera_lista' que, dadas por parámetro una lista compuesta por números y una tupla de igual longitud, compuesta por unos y ceros, modifique la lista, de modo que contenga valor cero en cada posición correspondiente a un cero de la tupla. (Ej: sean los argumentos [11, 23, 3, 45, 56] y (1, 0, 0, 1, 0), la lista modificada [11, 0, 0, 45, 0] debería ser devuelta al programa principal)

Ejercicio 7.17. Definir una función que reciba por parámetro una lista de números enteros y devuelva como resultado una tupla de dos elementos, compuesta, respectivamente, por la sumatoria y el promedio de los números de la lista.

8. Diccionarios

El presente capítulo tiene por intención facilitar la comprensión del tratamiento de diccionarios en Python (acceso a los "valores" a través de sus "claves") y de integrar conocimientos adquiridos en capítulos precedentes.

Ejercicio 8.1. Suponer qué hace el siguiente código y luego comprobar su ejecución:

```
def pba_dic():
    """ Prueba operaciones con diccionario """
    vocales = ("a", "e", "i", "o", "u")
    txt = "ese texto"
    dic = {}
    for letra in vocales:
        dic[letra] = ord(letra)
    pausa = input('\nDar enter para seguir >>> \n') #'\n' es salto de linea
    for letra in dic:
        print ( letra, "==> dic[letra]:", dic[letra] )
    pausa = input('\nDar enter para seguir >>> \n')
    for letra in dic:
        dic[letra] = [ dic[letra] ]
        print ( "[Debug:]\n", dic )
    pausa = input('\nDar enter para seguir >>> \n')
    for ind in range(len(txt)):
        letra = txt[ind]
        if letra in dic:
            dic[letra].append(ind)
            print ( '[Debug ind:]', ind, ' en txt:', letra,\
                    ' dic[', letra, ']:', dic[letra] )
    print ( "\n[Debug:]\n", dic )
```

Ejercicio 8.2. Definir una función que reciba como parámetro una cadena y que devuelva un diccionario, en donde las claves sean los caracteres de la cadena y los valores, la respectiva cantidad de apariciones del carácter en la cadena.

Por ejemplo, si recibe "el mejor ejemplo"

debiera resultar el diccionario: { 'e' : 4, 'l' : 2, ' ' : 2, 'm' : 2, 'j' : 2, 'o' : 2, 'r' : 1, 'p' : 1 }

Ejercicio 8.3. Definir una función que reciba como parámetro una cadena (de palabras y espacios) y que devuelva un diccionario, en donde las claves sean las palabras de la cadena y los valores, la respectiva cantidad de apariciones de la palabra en la cadena.

Por ejemplo, si recibe "el mejor ejemplo es el segundo"

debiera resultar: { 'el' : 2 , 'mejor' : 1 , 'ejemplo' : 1 , 'es' : 1 , 'segundo' : 1 }

Ejercicio 8.4. Dado un diccionario, cuyos ítems asocian a cada nombre el respectivo número de teléfono: agenda = { "Undav Es" : 42292400 , "Undav Pi" : 54367500 }

a) Escribir un programa que vaya solicitando al usuario que ingrese nombres a la agenda,
- si el nombre se encuentra en la agenda, debe mostrar el número de teléfono.
- si el nombre no se encuentra, debe permitir ingresar el teléfono correspondiente.

El usuario puede utilizar la cadena " * " para salir del programa.

b) Modificar el programa para que en el caso en que el nombre ya se encuentre en la agenda, además de mostrar el número de teléfono, opcionalmente, permita modificarlo si no es correcto.

Ejercicio 8.5. Definir una función "costo_total", que reciba como parámetros dos diccionarios, a saber:

1. Listado de precios: con artículos y sus respectivos precios.
2. Inventario: con artículos y sus respectivas existencias (cantidad disponible)

y que devuelva como resultado, el monto total que se recaudaría por vender todos los artículos del inventario.

Ejercicio 8.6. Considerando los diccionarios del enunciado 8.5, definir una función "factura_venta" que reciba como parámetros dos diccionarios y una lista de tuplas (que representa un Pedido de Compras) de la forma artículo y cantidad a comprar, y que devuelva como resultado el costo total de la factura (que resulta de recorrer la lista de compras, sumando los precios de cada artículo multiplicado por la cantidad). Además, la función deberá imprimir los pedidos de los artículos que no se encuentren disponibles (sin stock suficiente) o que no existan en el inventario.

Ejercicio 8.7. Dado el siguiente diccionario, cuyos elementos asocian a cada "provincia" argentina las respectivas cantidades de habitantes y de km^2 de superficie (en ese orden):

dic_censo = { 'Buenos Aries' : (15625084, 307571), 'Mendoza' : (1738929, 148827), 'Tucuman' : (1448188, 22524), …, 'Santa Crux' : (273964, 243943), … }

escribir instrucciones en Python 3, para:

a. Eliminar del diccionario los datos correspondientes a la provincia "Santa Crux", si existe, y asociarlos a una nueva provincia "Santa Cruz".
b. Recorrer el diccionario mostrando, en cada renglón de pantalla, el nombre de la provincia, la superficie y la densidad poblacional (cantidad de habitantes por km2).
c. ¿Cómo ordenaría alfabéticamente los resultados por provincia?

Opcionales:

Ejercicio 8.8. Definir una función que reciba como parámetro una lista de tuplas (de dos componentes) y que devuelva un diccionario en donde las claves sean los primeros elementos de las tuplas, y los valores, una lista con los segundos elementos respectivos.

Por ejemplo, dada la lista: [("Martin", 8) , ("Hugo", 9) , ("Martin", 7) , ("Martin", 9)]
Debiera resultar (en cualquier orden): { 'Martin' **:** [8, 7, 9] **,** 'Hugo' **:** [9] }

Ejercicio 8.9. Definir una función que reciba como parámetro un texto y que devuelva como resultado un diccionario, donde a cada carácter presente en el texto se le asocie como valor la cadena más larga en la que se encuentra ese carácter.

Ejercicio 8.10 Definir una función "reparte_letras" que reciba como parámetros una tupla compuesta por letras distintas (sin repeticiones) y una cadena de caracteres compuesta por palabras y espacios, y que devuelva como resultado una lista integrada por una lista de letras y un diccionario. La lista de letras debe contener las letras de la tupla que no pertenecen a palabra alguna de la cadena de caracteres. Las letras de la tupla que pertenecen a palabras de la cadena, deben ser clave en el diccionario, teniendo como valor la lista de palabras que la contienen.

Por ejemplo, si recibe los siguientes argumentos:
tuplita = ('a', 'j', 'k', 's') cadenita = "este texto tiene muchas letras"
debiera resultar la lista:
[['j' , 'k'] , **{** 'a' **:** ['muchas', 'letras'] , 's' **:** ['este', 'muchas', 'letras'] **}**]

Ejercicio 8.11. Para la siguiente secuencia de código Python 3, dados los valores de los identificadores que se detalla, complete en hoja aparte el estado de las variables y la salida que se despliega en pantalla (sólo cuando ésta ocurre):

```
cadena = "arroz a la zorra"
maxi = 0
d_let = {}
d_estad = {}
```

Código Python	*Estado de variables*	*Salida en pantalla*
`lisa = cadena.split()` `palabra = ''.join(lisa)` `lisa = []` `for letra in palabra:` `  lisa.append(letra)` `  if letra in d_let:` `  d_let[letra] = d_let[letra] + 1` `  else:` `  d_let[letra] = 1` `print ('[Debug]:', d_let,'\n')` `cond = (lisa[0] == lisa[-1])` `while cond and len(lisa) > 1:` `  lisa.pop(0)` `  lisa.pop(-1)` `  print ('[Debug]:\n', lisa)` `  cond = (lisa[0] == lisa[-1])` `if cond:` `  print ('\n',cadena,'es palindromo')` `for letra,cant in d_let.items():` `  if  cant in d_estad:` `  d_estad[cant].append(letra)` `  else:` `  d_estad[cant] = [letra]` `  if maxi < cant:` `  maxi = cant` `  print (letra,'==>', cant, 'veces')` `print ('\nLetras mas repetidas:')` `print (d_estad[maxi])`	lisa → . . . palabra → . . . lisa → . . . letra →	