75.40 Algoritmos y Programación I

Guía de Ejercicios

1.er cuatrimestre 2017

Recomendaciones al realizar las guías.

Generales:

- Sea claro y prolijo. Es muy importante que el código sea lo más claro y legible posible.
- Es muy importante que los identificadores de funciones y variables sean coherentes. El identificador debe ser suficientemente descriptivo.
- Ponga una línea en blanco entre las definiciones de función para simplificar la lectura del programa.
- Las expresiones matemáticas complejas pueden representarse en varios pasos.
- Los ejercicios marcados con el símbolo ★ son más difíciles y no son de resolución obligatoria.

Documentación:

- Documente correctamente las funciones y módulos que desarrolle.
- Documente partes del código cuyo significado pudiera no quedar del todo claro.
- No documente en exceso, pero tampoco ahorre documentación necesaria. La documentación debe ser breve y concisa.

1 Algunos conceptos básicos

Ejercicio 1.1. Escribir un programa que pregunte al usuario:

a) su nombre, y luego lo salude.

b) dos números, y luego muestre el producto.

Ejercicio 1.2. Implementar algoritmos que permitan:

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

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

c) Calcular el área de un rectángulo (alineado con los ejes x e y) dadas sus coordenadas $x1$, $x2$, $y1$, $y2$.

d) Calcular el perímetro de un círculo dado su radio.

e) Calcular el área de un círculo dado su radio.

f) Calcular el volumen de una esfera dado su radio.

g) Dados los catetos de un triángulo rectángulo, calcular su hipotenusa.

Ejercicio 1.3. Mostrar el resultado de ejecutar estos bloques de código en el intérprete de python:

a)
```
>>> for i in range(5):
...     print(i * i)
```

b)
```
>>> for i in range(2, 6):
...     print(i, 2 ** i)
```

Ejercicio 1.4. Implementar un algoritmo que, dado un número entero n, permita calcular su factorial.

Ejercicio 1.5. Implementar algoritmos que resuelvan los siguientes problemas:

a) Dados dos números, imprimir la suma, resta, división y multiplicación de ambos.

b) Dado un número entero n, imprimir su tabla de multiplicar.

Ejercicio 1.6. Escribir un programa que le pida una palabra al usuario, para luego imprimirla 1000 veces, en una única línea, con espacios intermedios.

Ayuda: Investigar acerca del parámetro `end` de la función `print`.

2 Programas sencillos

En los ejercicios a continuación, utilizar los conceptos de análisis, especificación y diseño antes de realizar la implementación.

Ejercicio 2.1. Escribir un programa que le pregunte al usuario una cantidad de pesos, una tasa de interés y un número de años y muestre como resultado el monto final a obtener. La fórmula a utilizar es:

$$C_n = C \times (1 + \frac{x}{100})^n$$

Donde C es el capital inicial, x es la tasa de interés y n es el número de años a calcular.

Ejercicio 2.2. Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius. Recordar que la fórmula para la conversión es: $F = \frac{9}{5}C + 32$

Ejercicio 2.3. Utilice el programa anterior para generar una tabla de conversión de temperaturas, desde 0 °F hasta 120 °F, de 10 en 10.

Ejercicio 2.4. Escribir un programa que imprima todos los números pares entre dos números que se le pidan al usuario.

Ejercicio 2.5. Escribir un programa que reciba un número n por parámetro e imprima los primeros n números triangulares, junto con su índice. Los números triangulares se obtienen mediante la suma de los números naturales desde 1 hasta n. Es decir, si se piden los primeros 5 números triangulares, el programa debe imprimir:

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

Nota: hacerlo usando y sin usar la ecuación $\sum_{i=1}^{n} i = n\,(n+1)/2$. ¿Cuál realiza más operaciones?

Ejercicio 2.6. Escribir un programa que tome una cantidad m de valores ingresados por el usuario, a cada uno le calcule el factorial e imprima el resultado junto con el número de orden correspondiente.

Ejercicio 2.7. Escribir un programa que imprima por pantalla todas las fichas de dominó, de una por línea y sin repetir.

Ejercicio 2.8. Modificar el programa anterior para que pueda generar fichas de un juego que puede tener números de 0 a n.

3 Funciones

Ejercicio 3.1. Escribir dos funciones que permitan calcular:

a) La duración en segundos de un intervalo dado en horas, minutos y segundos.

b) La duración en horas, minutos y segundos de un intervalo dado en segundos.

Ejercicio 3.2. Usando las funciones del ejercicio anterior, escribir un programa que pida al usuario dos intervalos expresados en horas, minutos y segundos, sume sus duraciones, y muestre por pantalla la duración total en horas, minutos y segundos.

Ejercicio 3.3. Escribir una función que, dados cuatro números, devuelva el mayor producto de dos de ellos. Por ejemplo, si recibe los números 1, 5, -2, -4 debe devolver 8, que es el producto más grande que se puede obtener entre ellos ($8 = -2 \times -4$).

Ejercicio 3.4. Área de un triángulo en base a sus puntos

a) Escribir una función que reciba las coordenadas de un vector `x,y` y devuelva la norma del vector, dada por $\|(\vec{x,y})\| = \sqrt{x^2 + y^2}$.
Ejemplo: `norma(3, 4)` → `5`

b) Escribir una función que reciba las coordenadas de dos vectores (`x1,y1,x2,y2`) y devuelva las coordenadas del vector diferencia (debe devolver un par de valores).
Ejemplo: `diferencia(8, 7, 5, 3)` → `(3, 4)`

c) Utilizando las funciones anteriores, escribir una función que reciba las coordenadas de dos vectores (`x1,y1,x2,y2`) y devuelva la distancia entre ambos.
Ejemplo: `distancia(8, 7, 5, 3)` → `5`

d) Escribir una función que reciba las coordenadas de un vector `x,y` y devuelva las coordenadas del vector normalizado correspondiente (debe devolver un par de valores).
Ejemplo: `normalizar(3, 4)` → `(0.6, 0.8)`

e) Utilizando las funciones anteriores (b y d), escribir una función que reciba las coordenadas de dos puntos (`x1,y1,x2,y2`) y devuelva el vector dirección unitario correspondiente a la recta que los une.
Ejemplo: `direccion(8, 7, 5, 3)` → `(0.6, 0.8)`

f) Escribir una función que reciba las coordenadas de un vector `x,y`, una dirección unitaria de una recta `dx,dy` y las coordenadas de un punto perteneciente a esa recta `cx,cy` y devuelva las coordenadas de la proyección del punto sobre la recta.
Ejemplo: `proyectar(2, 5, 0.6, 0.8, 3, 9)` → `(0.72, 5.96)`
Diseño del algoritmo:

1. Al punto a proyectar (`x,y`) restarle el punto de la recta (`cx,cy`)
2. Obtener las componentes de la matriz de proyección `P`, dada por:

$$p_{11} = d_x^2$$
$$p_{12} = p_{21} = d_x \cdot d_y$$
$$p_{22} = d_y^2$$

3. Multiplicar la matriz `P` por el punto obtenido en el paso 1:

$$r_x = p_{11} \cdot x + p_{12} \cdot y$$
$$r_y = p_{21} \cdot x + p_{22} \cdot y$$

4. Al resultado obtenido sumar el punto `cx,cy`, y devolverlo.

g) Escribir una función que calcule el área de un triángulo a partir de su base y su altura.
Ejemplo: `area_triangulo_base_altura(3, 4)` $\rightarrow$ `6`

h) Utilizando las funciones anteriores escribir una función que reciba las coordenadas de tres puntos en el plano (`x1,y1,x2,y2,x3,y3`) y devuelva el área del triángulo correspondiente.
Ejemplo: `area_triangulo_coordenadas(5, 5, 15, 30, 30, 0)` $\rightarrow$ `337.5`

4 Decisiones

Ejercicio 4.1. Escribir dos funciones que resuelvan los siguientes problemas:

a) Dado un número entero n, indicar si es par o no.

b) Dado un número entero n, indicar si es primo o no.

Ejercicio 4.2. Escribir una implementación propia de la función `abs`, que devuelva el valor absoluto de cualquier valor que reciba.

Ejercicio 4.3. Escribir una función que reciba por parámetro una dimensión n, e imprima la matriz identidad correspondiente a esa dimensión.

Ejercicio 4.4. Escribir funciones que permitan encontrar:

a) El máximo o mínimo de un polinomio de segundo grado (dados los coeficientes `a`, `b` y `c`), indicando si es un máximo o un mínimo.

b) Las raíces (reales o complejas) de un polinomio de segundo grado.
Nota: validar que las operaciones puedan efectuarse antes de realizarlas (no dividir por cero, ni calcular la raíz de un número negativo).

c) La intersección de dos rectas (dadas las pendientes y ordenada al origen de cada recta).
Nota: validar que no sean dos rectas con la misma pendiente, antes de efectuar la operación.

Ejercicio 4.5. Escribir funciones que resuelvan los siguientes problemas:

a) Dado un año indicar si es bisiesto.
Nota: un año es bisiesto si es un número divisible por 4, pero no si es divisible por 100, excepto que también sea divisible por 400.

b) Dado un mes, devolver la cantidad de días correspondientes.

c) Dada una fecha (día, mes, año), indicar si es válida o no.

d) Dada una fecha, indicar los días que faltan hasta fin de mes.

e) Dada una fecha, indicar los días que faltan hasta fin de año.

f) Dada una fecha, indicar la cantidad de días transcurridos en ese año hasta esa fecha.

g) Dadas dos fechas (día1, mes1, año1, día2, mes2, año2), indicar el tiempo transcurrido entre ambas, en años, meses y días.

Nota: en todos los casos, invocar las funciones escritas previamente cuando sea posible.

Ejercicio 4.6. Suponiendo que el primer día del año fue lunes, escribir una función que reciba un número con el día del año (de 1 a 366) y devuelva el día de la semana que le toca. Por ejemplo: si recibe `'3'` debe devolver `'miércoles'`, si recibe `'9'` debe devolver `'martes'`.

Ejercicio 4.7. ★ Escribir un programa que reciba como entrada un entero representando un año (por ejemplo 751, 1999, o 2158), y muestre por pantalla el mismo año escrito en números romanos.

Ejercicio 4.8. Programa de astrología: el usuario debe ingresar el día y mes de su cumpleaños y el programa le debe decir a qué signo corresponde.

Aries: 21 de marzo al 20 de abril.	Tauro: 21 de abril al 20 de mayo.
Geminis: 21 de mayo al 21 de junio.	Cancer: 22 de junio al 23 de julio.
Leo: 24 de julio al 23 de agosto.	Virgo: 24 de agosto al 23 de septiembre.
Libra: 24 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 Más sobre ciclos

Ejercicio 5.1. Escribir un programa que permita al usuario ingresar un conjunto de notas, preguntando a cada paso si desea ingresar más notas, e imprimiendo el promedio correspondiente.

Ejercicio 5.2. Escribir una función que reciba un número entero `k` e imprima su descomposición en factores primos.

Ejercicio 5.3. Manejo de contraseñas

a) Escribir un programa que contenga una contraseña 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.

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

d) Modificar el programa anterior para que sea una función que devuelva si el usuario ingresó o no la contraseña correctamente, mediante un valor booleano (`True` o `False`).

Ejercicio 5.4. Utilizando la función `randrange` del módulo `random`, escribir un programa que obtenga un número aleatorio secreto, y luego permita al usuario ingresar números y le indique si son menores o mayores que el número a adivinar, hasta que el usuario ingrese el número correcto.

Ejercicio 5.5. Algoritmo de Euclides

a) Implementar el algoritmo de Euclides para calcular el máximo común divisor de dos números n y m, dado por los siguientes pasos.

1. Teniendo n y m, se obtiene r, el resto de la división entera de m/n.
2. Si r es cero, n es el mcd de los valores iniciales.
3. Se reemplaza $m \leftarrow n$, $n \leftarrow r$, y se vuelve al primer paso.

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

Ejercicio 5.6. Potencias de dos.

a) Escribir una función `es_potencia_de_dos` que reciba como parámetro un número natural, y devuelva `True` si el número es una potencia de 2, y `False` en caso contrario.

b) Escribir una función que, dados dos números naturales pasados como parámetros, devuelva la suma de todas las potencias de 2 que hay en el rango formado por esos números (0 si no hay ninguna potencia de 2 entre los dos). Utilizar la función `es_potencia_de_dos`, descripta en el punto anterior.

Ejercicio 5.7. Números perfectos y números amigos

a) Escribir una función que devuelva la suma de todos los divisores de un número n, sin incluirlo.

b) Usando la función anterior, escribir una función que imprima los primeros m números tales que la suma de sus divisores sea igual a sí mismo (es decir los primeros m números *perfectos*).

c) Usando la primera función, escribir una función que imprima las primeras m parejas de números (a, b), tales que la suma de los divisores de a es igual a b y la suma de los divisores de b es igual a a (es decir las primeras m parejas de números *amigos*).

d) Proponer optimizaciones a las funciones anteriores para disminuir el tiempo de ejecución.

Ejercicio 5.8. Escribir un programa que le pida al usuario que ingrese una sucesión de números naturales (primero uno, luego otro, y así hasta que el usuario ingrese '-1' como condición de salida). Al final, el programa debe imprimir cuántos números fueron ingresados, la suma total de los valores y el promedio.

Ejercicio 5.9. Escribir una función que reciba dos números como parámetros, y devuelva cuántos múltiplos del primero hay, que sean menores que el segundo.

a) Implementarla utilizando un ciclo `for`, desde el primer número hasta el segundo.

b) Implementarla utilizando un ciclo `while`, que multiplique el primer número hasta que sea mayor que el segundo.

c) Comparar ambas implementaciones: ¿Cuál es más clara? ¿Cuál realiza menos operaciones?

Ejercicio 5.10. Escribir una función que reciba un número natural e imprima todos los números primos que hay hasta ese número.

Ejercicio 5.11. Escribir una función que reciba un dígito y un número natural, y decida numéricamente si el dígito se encuentra en la notación decimal del segundo.

Ejercicio 5.12. Escribir una función que dada la cantidad de ejercicios de un examen, y el porcentaje necesario de ejercicios bien resueltos necesario para aprobar dicho examen, revise un grupo de examenes. Para ello, en cada paso debe preguntar la cantidad de ejercicios resueltos por el alumno, indicando con un valor centinela que no hay más examenes a revisar. Debe mostrar por pantalla el porcentaje correspondiente a la cantidad de ejercicios resueltos respecto a la cantidad de ejercicios del examen y una leyenda que indique si aprobó o no.

6 Cadenas de caracteres

Ejercicio 6.1. Escribir funciones que dada una cadena de caracteres:

a) Imprima los dos primeros caracteres.

b) Imprima los tres últimos caracteres.

c) Imprima dicha cadena cada dos caracteres. Ej.: `'recta'` debería imprimir `'rca'`

d) Dicha cadena en sentido inverso. Ej.: `'hola mundo!'` debe imprimir `'!odnum aloh'`

e) Imprima la cadena en un sentido y en sentido inverso. Ej: `'reflejo'` imprime `'reflejoojelfer'`.

Ejercicio 6.2. Escribir funciones que dada una cadena y un caracter:

a) Inserte el caracter entre cada letra de la cadena. Ej: `'separar'` y `','` debería devolver `'s,e,p,a,r,a,r'`

b) Reemplace todos los espacios por el caracter. Ej: `'mi archivo de texto.txt'` y `'_'` debería devolver `'mi_archivo_de_texto.txt'`

c) Reemplace todos los dígitos en la cadena por el caracter. Ej: `'su clave es: 1540'` y `'X'` debería devolver `'su clave es: XXXX'`

d) Inserte el caracter cada 3 dígitos en la cadena. Ej. `'2552552550'` y `'.'` debería devolver `'255.255.255.0'`

Ejercicio 6.3. Modificar las funciones anteriores, para que reciban un parámetro que indique la cantidad máxima de reemplazos o inserciones a realizar.

Ejercicio 6.4. Escribir una función que reciba una cadena que contiene un largo número entero y devuelva una cadena con el número y las separaciones de miles. Por ejemplo, si recibe `'1234567890'`, debe devolver `'1.234.567.890'`.

Ejercicio 6.5. Escribir una función que dada una cadena de caracteres, devuelva:

a) La primera letra de cada palabra. Por ejemplo, si recibe `'Universal Serial Bus'` debe devolver `'USB'`.

b) Dicha cadena con la primera letra de cada palabra en mayúsculas. Por ejemplo, si recibe `'república argentina'` debe devolver `'República Argentina'`.

c) Las palabras que comiencen con la letra 'A'. Por ejemplo, si recibe `'Antes de ayer'` debe devolver `'Antes ayer'`

Ejercicio 6.6. Escribir funciones que dada una cadena de caracteres:

a) Devuelva solamente las letras consonantes. Por ejemplo, si recibe `'algoritmos'` o `'logaritmos'` debe devolver `'lgrtms'`.

b) Devuelva solamente las letras vocales. Por ejemplo, si recibe `'sin consonantes'` debe devolver `'i ooae'`.

c) Reemplace cada vocal por su siguiente vocal. Por ejemplo, si recibe `'vestuario'` debe devolver `'vistaerou'`.

d) Indique si se trata de un palíndromo. Por ejemplo, `'anita lava la tina'` es un palíndromo (se lee igual de izquierda a derecha que de derecha a izquierda).

Ejercicio 6.7. Escribir funciones que dadas dos cadenas de caracteres:

a) Indique si la segunda cadena es una subcadena de la primera. Por ejemplo, `'cadena'` es una subcadena de `'subcadena'`.

b) Devuelva la que sea anterior en orden alfábetico. Por ejemplo, si recibe `'kde'` y `'gnome'` debe devolver `'gnome'`.

Ejercicio 6.8. Escribir una función que reciba una cadena de unos y ceros (es decir, un número en representación binaria) y devuelva el valor decimal correspondiente.

Ejercicio 6.9. Implementar la función `pedir_entero(mensaje, min, max)`, que debe imprimir el `mensaje` y luego esperar a que el usuario ingrese un valor. Si el valor ingresado no es un número entero, o no es un número entre `min` y `max` (inclusive), se le debe avisar al usuario y pedir el ingreso de otro valor. Una vez que el usuario ingresa un valor válido, la función lo debe devolver.

7 Tuplas y listas

Ejercicio 7.1. Escribir una función que reciba una tupla de elementos e indique si se encuentran ordenados de menor a mayor o no.

Ejercicio 7.2. Dominó.

a) Escribir una función que indique si dos fichas de dominó *encajan* o no. Las fichas son recibidas en dos tuplas, por ejemplo: `(3,4)` y `(5,4)`

b) Escribir una función que indique si dos fichas de dominó *encajan* o no. Las fichas son recibidas en una cadena, por ejemplo: `3-4 2-5`. **Nota:** utilizar la función `split` de las cadenas.

Ejercicio 7.3. Campaña electoral

a) Escribir una función que reciba una tupla con nombres, y para cada nombre imprima el mensaje *Estimado <nombre>, vote por mí.*

b) Escribir una función que reciba una tupla con nombres, una posición de origen `p` y una cantidad `n`, e imprima el mensaje anterior para los `n` nombres que se encuentran a partir de la posición `p`.

c) Modificar las funciones anteriores para que tengan en cuenta el género del destinatario, para ello, deberán recibir una tupla de tuplas, conteniendo el nombre y el género.

Ejercicio 7.4. Vectores

a) Escribir una función que reciba dos vectores y devuelva su producto escalar.

b) Escribir una función que reciba dos vectores y devuelva si son o no ortogonales.

c) Escribir una función que reciba dos vectores y devuelva si son paralelos o no.

d) Escribir una función que reciba un vector y devuelva su norma.

Ejercicio 7.5. Dada una lista de números enteros, escribir una función que:

a) Devuelva una lista con todos los que sean primos.

b) Devuelva la sumatoria y el promedio de los valores.

c) Devuelva una lista con el factorial de cada uno de esos números.

Ejercicio 7.6. Dada una lista de números enteros y un entero k, escribir una función que:

a) Devuelva tres listas, una con los menores, otra con los mayores y otra con los iguales a k.

b) Devuelva una lista con aquellos que son múltiplos de k.

Ejercicio 7.7. Escribir una función que reciba una lista de tuplas (Apellido, Nombre, Inicial_segundo_nombre) y devuelva una lista de cadenas donde cada una contenga primero el nombre, luego la inicial con un punto, y luego el apellido.

Ejercicio 7.8. Inversión de listas

a) Realizar una función que, dada una lista, devuelva una nueva lista cuyo contenido sea igual a la original pero invertida. Así, dada la lista `['Di', 'buen', 'día', 'a', 'papa']`, deberá devolver `['papa', 'a', 'día', 'buen', 'Di']`.

b) Realizar otra función que invierta la lista, pero en lugar de devolver una nueva, modifique la lista dada para invertirla, **sin** usar listas auxiliares.

Ejercicio 7.9. Escribir una función `empaquetar` para una lista, donde epaquetar significa indicar la repetición de valores consecutivos mediante una tupla (valor, cantidad de repeticiones). Por ejemplo, `empaquetar([1, 1, 1, 3, 5, 1, 1, 3, 3])` debe devolver `[(1, 3), (3, 1), (5, 1), (1, 2), (3, 2)]`.

Ejercicio 7.10. Matrices.

a) Escribir una función que reciba dos matrices y devuelva la suma.

b) Escribir una función que reciba dos matrices y devuelva el producto.

c) ★ Escribir una función que opere sobre una matriz y mediante *eliminación gaussiana* devuelva una matriz triangular superior.

d) ★ Escribir una función que indique si un grupo de vectores, recibidos mediante una lista, son linealmente independientes o no.

Ejercicio 7.11. Plegado de un texto. Escribir una función que reciba un texto y una longitud y devuelva una lista de cadenas de como máximo esa longitud. Las líneas deben ser cortadas correctamente en los espacios (sin cortar las palabras).

Ejercicio 7.12. Funciones que reciben funciones.

a) Escribir una funcion llamada **map**, que reciba una función y una lista y devuelva la lista que resulta de aplicar la función recibida a cada uno de los elementos de la lista recibida.

b) Escribir una función llamada **filter**, que reciba una función y una lista y devuelva una lista con los elementos de la lista recibida para los cuales la función recibida devuelve un valor verdadero.

c) ¿En qué ejercicios de esta guía podría haber utilizado estas funciones?

8 Algoritmos de búsqueda

Ejercicio 8.1. Escribir una función que reciba una lista desordenada y un elemento, que:

a) Busque todos los elementos coincidan con el pasado por parámetro y devuelva la cantidad de coincidencias encontradas.

b) Busque la primera coincidencia del elemento en la lista y devuelva su posición.

c) Utilizando la función anterior, busque todos los elementos que coincidan con el pasado por parámetro y devuelva una lista con las posiciones.

Ejercicio 8.2. Escribir una función que reciba una lista de números no ordenada, que:

a) Devuelva el valor máximo.

b) Devuelva una tupla que incluya el valor máximo y su posición.

c) ¿Qué sucede si los elementos son cadenas de caracteres?

Nota: no utilizar `lista.sort()`

Ejercicio 8.3. Agenda simplificada

Escribir una función que reciba una cadena a buscar y una lista de tuplas (nombre_completo, telefono), y busque dentro de la lista, todas las entradas que contengan en el nombre completo la cadena recibida (puede ser el nombre, el apellido o sólo una parte de cualquiera de ellos). Debe devolver una lista con todas las tuplas encontradas.

Ejercicio 8.4. Sistema de facturación simplificado
Se cuenta con una lista ordenada de productos, en la que uno consiste en una tupla de (identificador, descripción, precio), y una lista de los productos a facturar, en la que cada uno consiste en una tupla de (identificador, cantidad).
Se desea generar una factura que incluya la cantidad, la descripción, el precio unitario y el precio total de cada producto comprado, y al final imprima el total general.
Escribir una función que reciba ambas listas e imprima por pantalla la factura solicitada.

Ejercicio 8.5. Escribir una función que reciba una lista ordenada y un elemento. Si el elemento se encuentra en la lista, debe encontrar su posición mediante búsqueda binaria y devolverlo. Si no se encuentra, debe agregarlo a la lista en la posición correcta y devolver esa nueva posición. (No utilizar `lista.sort()`.)

9 Diccionarios

Ejercicio 9.1. Escribir una función que reciba una lista de tuplas, y que devuelva un diccionario en donde las claves sean los primeros elementos de las tuplas, y los valores una lista con los segundos.

Por ejemplo:

```
>>> l = [ ('Hola', 'don Pepito'), ('Hola', 'don Jose'),
          ('Buenos', 'días') ]
>>> print(tuplas_a_diccionario(l))
{ 'Hola': ['don Pepito', 'don Jose'], 'Buenos': ['días'] }
```

Ejercicio 9.2. Diccionarios usados para contar.

a) Escribir una función que reciba una cadena y devuelva un diccionario con la cantidad de apariciones de cada palabra en la cadena. Por ejemplo, si recibe ”Qué lindo día que hace hoy” debe devolver: `{ 'que': 2, 'lindo': 1, 'día': 1, 'hace': 1, 'hoy': 1}`.

b) Escribir una función que cuente la cantidad de apariciones de cada caracter en una cadena de texto, y los devuelva en un diccionario.

c) Escribir una función que reciba una cantidad de iteraciones de una tirada de 2 dados a realizar y devuelva la cantidad de veces que se observa cada valor de la suma de los dos dados.

Nota: utilizar el módulo `random` para obtener tiradas aleatorias.

Ejercicio 9.3. Continuación de la agenda.
Escribir un programa que vaya solicitando al usuario que ingrese nombres.

a) Si el nombre se encuentra en la agenda (*implementada con un diccionario*), debe mostrar el teléfono y, opcionalmente, permitir modificarlo si no es correcto.

b) Si el nombre no se encuentra, debe permitir ingresar el teléfono correspondiente.

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

Ejercicio 9.4. Escribir una función que reciba un texto y para cada caracter presente en el texto devuelva la cadena más larga en la que se encuentra ese caracter.

10 Manejo de archivos

Ejercicio 10.1. Escribir un programa, llamado **head** que reciba un archivo y un número `N` e imprima las primeras `N` líneas del archivo.

Ejercicio 10.2. Escribir un programa, llamado **cp.py**, que copie todo el contenido de un archivo (sea de texto o binario) a otro, de modo que quede exactamente igual.
Nota: utilizar `archivo.read(bytes)` para leer como máximo una cantidad de bytes.

Ejercicio 10.3. Escribir un programa, llamado **cut.py**, que dado un archivo de texto, un delimitador, y una lista de campos, imprima solamente esos campos, separados por ese delimitador.

Ejercicio 10.4. Escribir un programa, llamado **wc.py** que reciba un archivo, lo procese e imprima por pantalla cuántas líneas, cuantas palabras y cuántos caracteres contiene el archivo.

Ejercicio 10.5. Escribir un programa, llamado **grep.py** que reciba una expresión y un archivo e imprima las líneas del archivo que contienen la expresión recibida.

Ejercicio 10.6. Escribir un programa, llamado **rot13.py** que reciba un archivo de texto de origen y uno de destino, de modo que para cada línea del archivo origen, se guarde una línea *cifrada* en el archivo destino. El algoritmo de cifrado a utilizar será muy sencillo: a cada caracter comprendido entre la a y la z, se le suma 13 y luego se aplica el módulo 26, para obtener un nuevo caracter.

Ejercicio 10.7. Persistencia de un diccionario

a) Escribir una función `cargar_datos` que reciba un nombre de archivo, cuyo contenido tiene el formato `clave, valor` y devuelva un diccionario con el primer campo como clave y el segundo como diccionario.

b) Escribir una función `guardar_datos` que reciba un diccionario y un nombre de archivo, y guarde el contenido del diccionario en el archivo, con el formato `clave, valor`.

11 Objetos

Ejercicio 11.1. Fracciones

a) Crear una clase `Fraccion`, que cuente con dos atributos: `dividendo` y `divisor`, que se asignan en el constructor, y se imprimen como `X/Y` en el método `__str__`.

b) Crear un método `sumar` que recibe otra fracción y devuelve una nueva fracción con la suma de ambas.

c) Crear un método `multiplicar` que recibe otra fracción y devuelve una nueva fracción con el producto de ambas.

d) Crear un método `simplificar` que modifica la fracción actual de forma que los valores del `dividendo` y `divisor` sean los menores posibles.

Ejercicio 11.2. Vectores

a) Crear una clase `Vector`, que en su constructor reciba una lista de elementos que serán sus coordenadas. En el método `__str__` se imprime su contenido con el formato `[x,y,z]`

b) Crear un método `escalar` que reciba un número y devuelva un nuevo vector, con los elementos multiplicados por ese número.

c) Crear un método `sumar` que recibe otro vector, verifica si tienen la misma cantidad de elementos y devuelve un nuevo vector con la suma de ambos. Si no tienen la misma cantidad de elementos debe levantar una excepción.

Ejercicio 11.3. Botella y Sacacorchos

a) Escribir una clase *Corcho*, que contenga un atributo *bodega* (cadena con el nombre de la bodega).

b) Escribir una clase *Botella* que contenga un atributo *corcho* con una referencia al corcho que la tapa, o `None` si está destapada.

c) Escribir una clase *Sacacorchos* que tenga un método *destapar* que le reciba una botella, le saque el corcho y se guarde una referencia al corcho sacado. Debe lanzar una excepción en el caso en que la botella ya esté destapada, o si el sacacorchos ya contiene un corcho.

d) Agregar un método *limpiar*, que saque el corcho del sacacorchos, o lance una excepción en el caso en el que no haya un corcho.

Ejercicio 11.4. Modelar una clase *Mate* que describa el funcionamiento de la conocida bebida tradicional local. La clase debe contener como miembros:

a) Un atributo para la cantidad de cebadas restantes hasta que se lava el mate (representada por un número).

b) Un atributo para el estado (lleno o vacío).

c) El constructor debe recibir como parámetro `n`, la cantidad máxima de cebadas en base a la cantidad de yerba vertida en el recipiente.

d) Un método `cebar`, que llena el mate con agua. Si se intenta cebar con el mate lleno, se debe lanzar una excepción que imprima el mensaje `'Cuidado! Te quemaste!'`

e) Un método `beber`, que vacía el mate y le resta una cebada disponible. Si se intenta beber un mate vacío, se debe lanzar una excepción que imprima el mensaje `'El mate está vacío !'`

f) Es posible seguir cebando y bebiendo el mate aunque no haya cebadas disponibes. En ese caso la cantidad de cebadas restantes se mantendrá en 0, y cada vez que se intente beber se debe imprimir un mensaje de aviso: `'Advertencia: el mate está lavado.'`, pero no se debe lanzar una excepción.

12 Polimorfismo, Herencia y Delegación

Ejercicio 12.1. Papel, Birome, Marcador

a) Escribir una clase `Papel` que contenga un texto, un método `escribir`, que reciba una cadena para agregar al texto, y el método `__str__` que imprima el contenido del texto.

b) Escribir una clase `Birome` que contenga una cantidad de tinta, y un método `escribir`, que reciba un texto y un papel sobre el cual escribir. Cada letra escrita debe reducir la cantidad de tinta contenida. Cuando la tinta se acabe, debe lanzar una excepción.

c) Escribir una clase `Marcador` que herede de `Birome`, y agregue el método `recargar`, que reciba la cantidad de tinta a agregar.

Ejercicio 12.2. Juego de Rol

a) Escribir una clase `Personaje` que contenga los atributos `vida`, `posicion` y `velocidad`, y los métodos `recibir_ataque`, que reduzca la vida según una cantidad recibida y lance una excepción si la vida pasa a ser menor o igual que cero, y `mover` que reciba una dirección y se mueva en esa dirección la cantidad indicada por velocidad.

b) Escribir una clase `Soldado` que herede de `Personaje`, y agregue el atributo `ataque` y el método `atacar`, que reciba otro personaje, al que le debe hacer el daño indicado por el atributo ataque.

c) Escribir una clase `Campesino` que herede de `Personaje`, y agregue el atributo `cosecha` y el método `cosechar`, que devuelva la cantidad cosechada.

13 Listas enlazadas

Ejercicio 13.1. Agregar a la clase *ListaEnlazada* un método `next` que vaya devolviendo uno a uno cada elemento de la lista, desde el primero hasta el último. Al llegar al final de la lista debe levantar una excepción de la clase *StopIteration*. Para el correcto funcionamiento de este método, ¿es necesario agregar un atributo adicional a la clase?

Ejercicio 13.2. Utilizando el método `next` del ejercicio anterior, redefinir el método `__str__` de *ListaEnlazada*, para que se genere una salida legible de lo que contiene la lista, similar a las listas de python.
Nota: este método debe devolver una cadena, no imprimirla por pantalla.

Ejercicio 13.3. Agregar a *ListaEnlazada* un método `extend` que reciba una *ListaEnlazada* y agregue a la lista actual los elementos que se encuentran en la lista recibida.

Ejercicio 13.4. Una **lista circular** es una lista cuyo último nodo está ligado al primero, de modo que es posible recorrerla infinitamente.
Escribir la clase *ListaCircular*, incluyendo los métodos `insert`, `append`, `remove` y `pop`.

Ejercicio 13.5. Una **lista doblemente enlazada** es una lista en la cual cada nodo tiene una referencia al anterior además de al próximo de modo que es posible recorrerla en ambas direcciones.
Escribir la clase *ListaDobleEnlazada*, incluyendo los métodos `insert`, `append`, `remove` y `pop`.

Ejercicio 13.6. Escribir un método de la clase *ListaEnlazada* que invierta el orden de la lista (es decir, el primer elemento queda como último y viceversa, y se invierte la dirección de todos los enlaces).
Nota: operar directamente sobre los elementos de la lista.

14 Pilas y colas

Ejercicio 14.1. Escribir una clase *TorreDeControl* que modele el trabajo de una torre de control de un aeropuerto, con una pista de aterrizaje. La torre trabaja en dos etapas: *reconocimiento* y *acción*.

a) Escribir un método `reconocimiento`, que verifique si hay algún nuevo avión esperando para aterrizar y/o despegar, y de ser así los encole en la cola correspondiente. Para ello, utilizar `random.randrange(2)`.

b) Escribir un método `acción`, que haga aterrizar o bien despegar, al primero de los aviones que esté esperando (los que esperan para aterrizar tienen prioridad). Debe desencolar el avión de su cola y devolver la información correspondiente.

c) Escribir un método `__str__` que imprima el estado actual de ambas colas.

d) Escribir un programa que inicialice la torre de control, y luego llame continuamente a los métodos `reconocimiento` y `acción`, imprimiendo la acción tomada y el estado de la torre de control cada vez.

Ejercicio 14.2. Atención a los pacientes de un consultorio médico, con varios doctores.

a) Escribir una clase `ColaDePacientes`, con los métodos `nuevo_paciente`, que reciba el nombre del paciente y lo encole, y un método `proximo_paciente` que devuelva el primer paciente en la cola y lo desencole.

b) Escribir una clase `Recepcion`, que contenga un diccionario con las colas correspondientes a cada doctor o doctora, y los métodos `nuevo_paciente` que reciba el nombre del paciente y del especialista, y `proximo_paciente` que reciba el nombre de la persona liberada y devuelva el próximo paciente en espera.

c) Escribir un programa que permita al usuario ingresar nuevos pacientes o indicar que un consultorio se ha liberado y en ese caso imprima el próximo paciente en espera.

Ejercicio 14.3. Juego de Cartas

a) Crear una clase `Carta` que contenga un palo y un valor.

b) Crear una clase `PilaDeCartas` que vaya apilando las cartas una debajo de otra, pero sólo permita apilarlas si son de un número inmediatamente inferior y de distinto palo. Si se intenta apilar una carta incorrecta, debe lanzar una excepción.

c) Agregar el método `__str__` a la clase PilaDeCartas, para imprimir las cartas que se hayan apilado hasta el momento.

15 Modelo de ejecución de funciones y recursión

Ejercicio 15.1. Escribir una función que reciba un número positivo n y devuelva la cantidad de dígitos que tiene.

Ejercicio 15.2. Escribir una función que simule el siguiente experimento: Se tiene una rata en una jaula con 3 caminos, entre los cuales elige al azar (cada uno tiene la misma probabilidad), si elige el *1* luego de 3 minutos vuelve a la jaula, si elige el *2* luego de 5 minutos vuelve a la jaula, en el caso de elegir el *3* luego de 7 minutos sale de la jaula. La rata no aprende, siempre elige entre los 3 caminos con la misma probabilidad, pero quiere su libertad, por lo que recorrerá los caminos hasta salir de la jaula.

La función debe devolver el tiempo que tarda la rata en salir de la jaula.

Ejercicio 15.3. Escribir una función que reciba 2 enteros n y b y devuelva `True` si n es potencia de b.

Ejemplos:

```
>>> es_potencia(8, 2)
True
>>> es_potencia(64, 4)
True
>>> es_potencia(70, 10)
False
```

Ejercicio 15.4. Escribir una funcion recursiva que reciba como parámetros dos strings a y b, y devuelva una lista con las posiciones en donde se encuentra b dentro de a.

Ejemplo:

```
>>> posiciones_de("Un tete a tete con Tete", "te")
[3, 5, 10, 12, 21]
```

Ejercicio 15.5. Escribir dos funciones mutualmente recursivas par(n) e impar(n) que determinen la paridad del numero natural dado, conociendo solo que:

- 1 es impar.
- Si un número es impar, su antecesor es par; y viceversa.

Ejercicio 15.6. Escribir una función que calcule recursivamente el n-ésimo número triangular (el número $1 + 2 + 3 + \cdots + n$).

Ejercicio 15.7. Escribir una función que calcule recursivamente cuántos elementos hay en una pila, suponiendo que la pila sólo tiene los métodos apilar y desapilar, y no altere el contenido de la pila.
¿Implementarías esta función para un programa real? ¿Por qué?

Ejercicio 15.8. Escribir una funcion recursiva que encuentre el mayor elemento de una lista.

Ejercicio 15.9. Escribir una función recursiva para replicar los elementos de una lista una cantidad n de veces. Por ejemplo, `replicar ([1, 3, 3, 7], 2) = ([1, 1, 3, 3, 3, 3, 7, 7])`

16 Ordenar listas

Ejercicio 16.1. Implementar una función que reciba una lista y devuelva otra lista con los mismos elementos que la primera, ordenados de mayor a menor mediante el método de inserción.

17 Algunos ordenamientos recursivos

Ejercicio 17.1. Escribir una función `merge_sort_3` que funcione igual que el merge sort pero en lugar de dividir los valores en dos partes iguales, los divida en tres (asumir que se cuenta con la función `merge(lista_1, lista_2, lista_3)`). ¿Cómo te parece que se va a comportar este método con respecto al merge sort original?

Ejercicio 17.2. Mostrar los pasos del ordenamiento de la lista: `6 0 3 2 5 7 4 1` con los métodos de inserción y con merge sort. ¿Cuáles son las principales diferencias entre los métodos? ¿Cuál usarías en qué casos?

Ejercicio 17.3. Ordenar la lista `6 0 3 2 5 7 4 1` usando el método quicksort. Mostrar el árbol de recursividad explicando las llamadas que se hacen en cada paso, y el orden en el que se realizan.

18 Lenguaje C

Ejercicio 18.1. Escribir una función que permita calcular el área de un rectángulo dada su base y altura.

Ejercicio 18.2. Escribir una función que reciba un número entero `n` y calcule el factorial de `n`.

a) En forma iterativa.

b) En forma recursiva.

Ejercicio 18.3. Escribir una función que reciba un arreglo de números y la cantidad de elementos, y devuelva el promedio.

Ejercicio 18.4. Implementar la función `unsigned int strlen(const char *s)` que devuelve la longitud de la cadena `s` (sin contar el último caracter `'\0'`).

a) En forma iterativa.

b) En forma recursiva.

Ejercicio 18.5. Implementar la función `void invertir(char *s)` que invierte la cadena `s` (*in-place*).

Ejercicio 18.6. Implementar la función `void strcpy(const char *origen, char *destino)` que copia la cadena `origen` en la dirección de memoria apuntada por `destino`. Asumir que `destino` apunta a un arreglo de caracteres con espacio suficiente (`strlen(origen) + 1`).

Ejercicio 18.7. Implementar una función que recibe un arreglo de números y su longitud y lo ordena mediante el algoritmo de ordenamiento por selección.

Ejercicio 18.8. Implementar una función que reciba un `mensaje` y dos números enteros `min` y `max`. La función debe pedir al usuario que ingrese un número entero entre `min` y `max` (inclusive) y devolverlo. Si el usuario ingresa un valor inválido se le debe informar y pedir que ingrese un nuevo valor, repitiendo hasta que ingrese un número válido.

Ejercicio 18.9. Implementar una función que reciba una cadena de texto y luego imprima la cadena enmarcada entre asteriscos (*). Asumir que la cadena no contiene ningún caracter `\n`. Por ejemplo, si se recibe la cadena `"Lenguaje C"`, debe imprimir:

```
**************
* Lenguaje C *
**************
```

Ejercicio 18.10. ★ Implementar una función que permita ejecutar cálculos matemáticos expresados en notación polaca inversa (`https://es.wikipedia.org/wiki/Notaci%C3%B3n_polaca_inversa`).

Ejemplo: `calcular_rpn("5 1 2 + 4 * + 3 -")` devuelve 14, ya que la expresión ingresada es equivalente a $5 + ((1 + 2) * 4) - 3$.

Ayuda: El algoritmo de notación polaca inversa hace uso de una pila, que se puede implementar en C mediante un arreglo con un tamaño fijo que determinará la cantidad máxima permitida de valores apilados.