# Abstracción de procesos

## Abstracción

La clase pasada vimos cómo es la estructura típica de un **programa imperativo**: entrada de datos, procesamiento/transformación de datos, salida de datos. Esto nos ayuda de alguna manera a **pensar en nuestra solución informática**. Hoy trataremos de formalizar estas ideas y extenderlas para poder enfrentar problemas más complejos.

**El diseño de un programa** ha de estar guiado por un **planteamiento, o modelo, del problema** que queremos resolver en el mundo real. Este modelo nos permite proponer de mejor forma un **modelo de la solución**, que a su vez nos guiará en las decisiones que tomemos con respecto a cómo utilizaremos las herramientas que provee el lenguaje de programación que usemos para **implementarla**. La figura 1 ilustra este proceso.

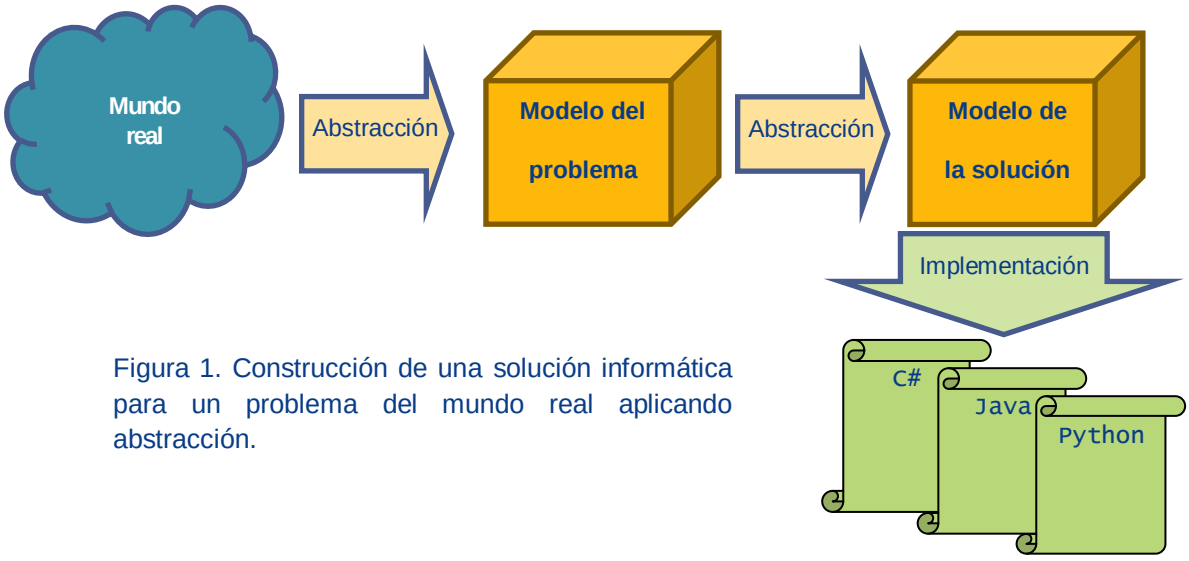


Figura 1. Construcción de una solución informática para un problema del mundo real aplicando abstracción.

La **abstracción** es la herramienta fundamental del diseño. La intención final de la abstracción es determinar las **características importantes** de algo y dejar de lado los detalles. La **implementación** es el **proceso inverso** a la abstracción, y tiene como intención **completar los detalles necesarios** para construir un programa real, en algún lenguaje de programación, a partir de un modelo de la solución.

En la figura 1, podemos distinguir dos tipos de abstracción. Primero está la abstracción de problemas, que podemos definir formalmente de la siguiente manera:

Importante

**Abstracción de problemas** (Definición): Técnica que tiene por objeto construir un mundo imaginario que contenga **sólo la estructura necesaria** para resolver un problema.

La abstracción de problemas no es un concepto nuevo para nosotros y la utilizamos constantemente cuando, por ejemplo, resolvemos problemas matemáticos o de física. En general, la abstracción de problemas puede verse como un proceso de tres pasos:

1. Identificar **entidades** y **aspectos** de la situación que son **importantes**.
2. **Dar nombre** a estos elementos importantes.
3. **Definir operaciones** que se pueden aplicar sobre las entidades y que son consistentes con las reglas del problema.

En informática, y en la ingeniería en general, la "situación" normalmente se conoce a través de un **enunciado**. Un enunciado generalmente es un **texto**, el cual puede incluir gráficos, tablas y ecuaciones, y puede llegar a ser bastante complejo, como las bases públicas de un proyecto. En este texto, se nos describe la problemática a resolver (objetivo), **información específica** necesaria para comprender el problema y **restricciones** que debemos cumplir. Todo esto puede estar de manera **explícita** (escrito en el enunciado) como **implícita** (se deduce del enunciado).

Sin embargo, es común que los enunciados **omitan información** que se considera de conocimiento público o de especialistas en la disciplina/contexto, y que **contengan información irrelevante**. Muchas veces simplemente no entregan toda la información necesaria para resolver el problema y el ingeniero debe interactuar con el cliente para llenar estos vacíos.

Hasta ahora hemos trabajado con enunciados directos que ya tienen el problema planteado. En el futuro, trataremos de enfrentar problemas con enunciados más realistas y será nuestra misión modelarlos adecuadamente. Por ejemplo, consideremos el Enunciado 1.

Enunciado 1

A Carlos le ha ido muy bien con su negocio de efectos lúdicos para DJs. En particular, su efecto "burbujas de la noche" está siendo muy cotizado por las principales discotecas del litoral central, y este verano prometen ser un hit.
El efecto consiste en la liberación de docenas de globos pesados (son más bien como pelotas de goma de 77 g cada una y de distintos colores pasteles) sobre los clientes que se encuentran en la pista de baile durante la canción que cierra la presentación del DJ estrella. La gracia, es que Carlos asegura que los globos caen sobre las cabezas de los danzantes (en realidad a 171,5 cm del suelo, según promedio nacional) exactamente con el *beat* clímax de la canción que se esté tocando.
Carlos logra esto instalando los globos en un intrincado sistema de amarre que los deja colgando a 30 cm del techo del local y que cuenta con un cronómetro de alta precisión

(¡hasta 0,0001 segundo!) que puede sincronizar, y aquí está su idea brillante, con el inicio de un silencio en la pista de percusión del tema que suele aparecer en el puente musical que antecede al clímax de la canción.
El gran desafío que Carlos es que le toma mucho tiempo poder encontrar el valor que debe ingresar al cronómetro para lograr la sincronización. Por ejemplo, en el último evento en la Estación Mapocho, el sistema se instaló a un techo falso a 24 m de altura. El tema final era "Praise You" de Fatboy Slim, que tiene un silencio en la percusión de 12.05 s en el puente. A Carlos le tomó ¡casi 3 días! determinar, con prueba y error, que el cronómetro debía fijarse a 9,9325 s para que los globos dieran en la cabeza de los felices asistentes junto con el *beat* inicial del clímax. Además del tiempo, Carlos tiene que gastar mucho material para realizar sus numerosas pruebas.
Este problema ha hecho que, hasta ahora, Carlos no pueda extender su negocio y, por ejemplo, animar más de un evento por fin de semana. Él necesita resolver esta dificultad antes de la temporada de verano y está en busca de una solución informática que lo apoye.

Siguiendo el conjunto de pasos enunciados anteriormente, podemos crear el modelo presentado en el Planteamiento 1. En este planteamiento hemos incluido la información que es **relevante para construir nuestro programa** (solución) y hemos dejado fuera aquello que es irrelevante.

## Planteamiento 1

- Los globos, al ser liberados, hacen una caída libre sobre las personas. Al ser redondos y de goma, podemos suponer que son objetos densos y que la caída libre que realizan está gobernada por la ecuación:
$$h(t) = h_0 - \frac{1}{2} g t^2.$$
- Sabemos que $\boldsymbol{g}$ corresponde a la aceleración gravitacional, igual a 9,80665 m/s$^2$.
- Necesitamos determinar el tiempo que tarda el globo en caer a la altura deseada. Sea $\boldsymbol{d}$ la distancia que debe caer cada globo, entonces la ecuación anterior puede escribirse como:
$$t = \sqrt{\frac{2d}{g}}.$$
- Podemos calcular $\boldsymbol{d}$ como la altura del techo menos la altura a la que cuelgan los globos (0,3 m) menos la altura objetivo (1,715 m). Es necesario entonces conocer la altura del techo del local (dato de entrada).
- Los globos, entonces, deben liberarse $\boldsymbol{t}$ segundos antes del primer beat del clímax del tema que se está tocando.
- Pero el objetivo es determinar el tiempo $\boldsymbol{t_c}$ que debe fijarse en el cronómetro para sincronizar los globos con ese beat (dato de salida).
- Esto puede calcularse como:
$$t_c = t_s - t,$$
en donde $\boldsymbol{t_s}$ sería la duración del silencio en el puente musical antes del clímax, otro dato que debemos preguntar al usuario (dato entrada).
- $\boldsymbol{t_c}$ debe ser un valor positivo (es decir, el tiempo que el globo debe caer libremente ha de ser menor que la duración del silencio en el puente). También, los globos deben inicialmente colgar sobre las cabezas de las personas, por lo que la altura del techo no debiera ser menor a 2,2 m + 0,3 m = 2,5 m. Si cualquiera de estas cosas no ocurre, el sistema no puede instalarse. Por último, debe considerarse que $\boldsymbol{t_c}$ puede tener cuatro decimales.

Como se puede apreciar, el problema **ha cambiado** a preguntar algunos valores que debemos utilizar en varias fórmulas para poder obtener el dato que se requiere, más unas cuantas consideraciones. Pero este problema es **equivalente** al del Enunciado 1, en el sentido que ambos requieren **la misma solución**. La diferencia está en que el problema planteado está escrito con un **lenguaje más formal**, llenado los vacíos que tenía el enunciado original (por ejemplo que los globos hacen una caída libre) y evitando incluir aspectos que no aportan (como que los globos pesan 77 g).

En este planteamiento hemos **identificado las entidades importantes**. Primero se han determinado los **datos de entrada**: la altura del techo y la duración del silencio musical. Luego se han descrito los **datos intermedios** que tenemos que establecer: la distancia que cae cada globo y el tiempo que se tardan en recorrer esta distancia, además de **cómo calcularlos** (fórmulas). Finalmente se ha descrito un **objetivo** concreto (calcular el tiempo a fijar en el cronómetro $t_c$) y la **respuesta** que el programa debe entregar ($t_c$). Las fórmulas identificadas y las condiciones especificadas en el último punto corresponden a **operaciones y restricciones** que la solución debe considerar.

**Pregunta 1**

Con tu grupo de trabajo, y utilizando lo visto anteriormente, contesten la primera pregunta que entregue su profesor.

El **modelo del problema** que hemos obtenido permite concentrarnos en los **aspectos importantes** que una solución debe considerar. En informática, las soluciones se construyen aplicando **abstracción de procesos**, que podemos definir de la siguiente manera:

**Importante**

**Abstracción de procesos** (Definición): Técnica que tiene por finalidad describir **qué procesos** deben realizarse y **en qué orden** para lograr un objetivo, **sin detallar el "cómo" deben realizarse**.

La abstracción de procesos tampoco es nueva para nosotros y la utilizamos en nuestro diario vivir. Por ejemplo, hace unos pocos años atrás cuando nuestra mamá o papá nos despertaba en la mañana y nos decía "*prepárate para ir al liceo*". Esta simple orden definía un **objetivo** más o menos claro: "estar listo para ir al liceo", y para alcanzarlo, debíamos realizar una **secuencia de pasos**: quitarse el pijama, ducharse, vestirse, tomar desayuno, lavar la loza que usamos, lavarse los dientes, tender nuestra cama y finalmente revisar que llevamos todo lo necesario para el día en nuestra mochila. Notemos que existen **restricciones en el orden** en que podemos realizar estos pasos.

Por ejemplo, da lo mismo si tomamos desayuno y revisamos nuestra mochila antes de quitarnos el pijama, ducharnos y vestirnos, pero **no sería lógico** vestirnos y después ducharnos.

**Pregunta 2**

Con tu grupo de trabajo, respondan ahora la segunda pregunta de la actividad.

Lo que hemos descrito para este ejemplo tan cotidiano, aplica perfectamente a nuestra resolución de problemas del mundo real. Existe un **objetivo**, que corresponde al objetivo identificado en el planteamiento del problema a resolver, que requerirá la **ejecución de una secuencia de pasos** para lograrlo. Por ejemplo, la Solución 1 presenta un modelo de la solución para el Planteamiento 1.

**Solución 1**

*Objetivo*: Determinar el tiempo $t_c$ que se debe esperar desde el inicio del silencio en percusión que se incluye en el puente musical de una canción

1. Preguntar al usuario cuál es la altura del techo $h_t$ (en metros)
2. Preguntar al usuario cuál es la duración del silencio $t_s$ (en segundos)
3. Calcular la distancia $d$ (en metros) que un globo debe caer: $d = h_t - d_t - h_f$, donde $d_t = 0{,}3$ es la distancia a la que cuelgan los globos desde el techo y $h_f = 1{,}715$ es la altura objetivo.
4. Calcular el tiempo (en segundos) que tarda cada globo en caer los $d$ en metros: $t = \sqrt{2d/g}$, con $g = \mathbf{9{,}80665}$
5. Calcular en tiempo para el cronómetro: $t_c = t_s - t$
6. Mostrar el valor obtenido

*Requiere*: altura del techo ($h_t$) mayor o igual a 2,5 m; duración del silencio ($t_s$) mayor o igual al tiempo que se necesita un globo para caer $d$ metros ($t$)

**Pregunta 3**

Ahora responde la tercera pregunta de la actividad con tu grupo de trabajo.

# Refinamiento algorítmico

La Solución 1 resuelve **completamente** el problema planteado en el Enunciado 1, y su implementación en el lenguaje de programación Python es bastante directa. Sin embargo, esto normalmente no ocurre con **problemas más complejos**, y la secuencia de pasos que obtenemos aplicando abstracción de procesos nos entrega solamente una **estrategia general** para resolver el problema. Normalmente, una estrategia general tiene **demasiados detalles omitidos** como para intentar implementarla directamente en un lenguaje de programación.

Pero en el desarrollo de la pregunta 2, pudimos darnos cuenta que cada uno de los pasos necesarios para lograr el objetivo "estar listo para ir al liceo" define un **sub-objetivo**, y habrá entonces una **secuencia de pasos más específicos** que permita ir alcanzándolos uno a uno. Por ejemplo, el **subproceso** "ducharse" define el sub-objetivo "estar duchado", que puede lograrse con los siguientes pasos: dar el agua de la ducha, regular hasta que el agua tenga una temperatura adecuada, mojarse, enjabonarse, enjuagarse, cortar el agua de la ducha, secarse. Obviamente, cada uno de estos sub-subprocesos define un sub-sub-objetivo, como por ejemplo "estar enjabonado", que podremos alcanzar siguiendo un conjunto de pasos todavía más específicos.

Es decir, para lograr un **objetivo complejo** podemos usar **repetidamente** la abstracción de procesos para ir **refinando** nuestra solución. Primero obtendremos un modelo de solución que describe una solución general. Luego podemos construir separadamente modelos de solución para los subprocesos que consideremos todavía muy complejos, y luego modelos de solución para aquellos sub-subprocesos que sigan siendo muy complejos como para enfrentarlos directamente. Esta estrategia es conocida como **división en subproblemas** (o subprocesos, o subtareas) y consta de los siguientes pasos:

1. Existe un **objetivo** (inicialmente al **objetivo del problema planteado**).
2. Para alcanzar este objetivo, se identifican **subprocesos** que deben ejecutarse en un cierto orden.
3. Identificar los subprocesos que definen **sub-objetivos complejos** (se identifican subproblemas).
4. Para estos subprocesos, repetir los pasos 1 al 4 hasta alcanzar **objetivos manejables**.
5. Resolver cada **sub-objetivo simple**.
6. Una vez resueltos los sub-objetivos más simples, se **componen** las soluciones para conseguir un **sub-objetivo más complejo**.
7. Repetir los pasos 6 y 7 **hasta alcanzar el objetivo inicial**.

Existen distintas formas de dividir un problema en subproblemas, nosotros utilizaremos el **refinamiento algorítmico**, que nos indica que objetivos complejos debemos expresarlos

como **llamadas a subrutinas** (funciones o procedimientos) que definiremos en un siguiente nivel de refinamiento. Veamos cómo se aplican estas ideas con el siguiente ejemplo:

**Enunciado 2**

Un profesor tiene un listado con la puntuación obtenida por cada estudiante en las tres pruebas aplicadas a su curso. Cada prueba tenía 60 repartidos en cinco preguntas.
Ahora se enfrenta a la tarea de calcular las notas parciales y el promedio obtenido por cada estudiante durante el semestre, y el coordinador de ha dado ¡dos días de plazo! para entregarlas. Por si esto no fuera poco, el coordinador le ha solicitado entregar dos promedios: uno es el promedio simple de las tres notas y el otro es un promedio "con ayuda" en que la mejor nota se convierte en coeficiente 2. Seguramente el coordinador sospecha que los resultados no han sido tan buenos y quiere evaluar la posibilidad de otorgar esta ayuda a los estudiantes.
Para poder responder en el plazo, el profesor nos ha encargado un programa en Python que calcule estos promedios para cada estudiante.

Lo primero que debemos hacer es realizar la abstracción del problema, la que se presenta en el Planteamiento 2.

**Planteamiento 2**

- El profesor tiene la puntuación obtenida en 3 pruebas para cada estudiante: $punt1$, $punt2$ y $punt3$ (datos de entrada).
- Las 3 pruebas tienen la misma puntuación máxima, igual a 60 puntos.
- Luego la nota correspondiente a cada puntuación ($nota1$, $nota2$ y $nota3$ respectivamente) puede obtenerse como:

$$nota = 6 \cdot \frac{puntuación}{puntuación\ máxima} + 1$$

- El programa debe obtener dos promedios con las notas obtenidas por un estudiante en las pruebas.
- Se debe entregar el promedio simple de las 3 notas (dato de salida), que llamaremos $prom3Notas$.
- También se debe entregar el promedio simple de 4 notas (dato de salida), en que la cuarta nota es la mejor nota obtenida por el estudiante ($mejorNota$), que llamaremos $prom4Notas$.
- Las notas y los promedios deben estar en el intervalo [1,0, 7,0], redondeadas a un decimal de precisión. Esto significa que las puntuaciones deben estar en el rango 0-60.

En el Planteamiento 2, nuevamente hemos obtenido un modelo del problema en donde se identifican los datos de entrada ($punt1$, $punt2$ y $punt3$), datos intermedios que deben determinarse ($nota1$, $nota2$ y $nota3$) y los datos de salida que se requieren ($prom3Notas$ y $prom4Notas$). También identifica operaciones que nos ayudan (conversión puntuación

a nota) y algunas restricciones (intervalos y precisión). Este modelo permite construir fácilmente una modelo general de la solución:

**Solución 2: Estrategia general**

*Objetivo*: Calcular los promedios prom3Notas y prom4Notas para un estudiante (flotantes, un decimal)

1. Obtener las puntuaciones conseguidas en las pruebas: punt1, punt2 y punt3
2. Convertir punt1, punt2 y punt3 en las notas correspondientes: nota1, nota2 y nota3
3. Calcular prom3Notas como el promedio simple de nota1, nota2 y nota3
4. Calcular prom4Notas como el promedio simple de nota1, nota2, nota3 y $mejorNota$
5. Entregar los promedios obtenidos (prom3Notas y prom4Notas) al usuario

*Requiere*: punt1, punt2 y punt3 sean valores enteros entre 0 y 60

Podemos darnos cuenta que la estrategia general de la Solución 2 describe el **bloque principal** del un programa que resuelve el problema planteado. Como en este problema los datos de entrada corresponden a valores numéricos simples, que no involucran, por ejemplo, lectura de archivos o construcciones de datos complejos (que sí tendremos que hacer en el futuro), ni tampoco vamos a realizar (por ahora) una validación del valor entregado por el usuario, este subproblema (paso 1) puede llevarse directamente a un lenguaje de programación (en algunos lenguajes más fácilmente que en otros). Lo mismo ocurre con el dato de salida (paso 5).

El paso 2 parece más complejo y porque se trata de un proceso que se repite más de una vez, conviene **encapsularlo** en una función, que llamaremos `calculaNota()`. Necesitamos identificar los parámetros de esta función. En nuestro planteamiento del problema, queda claro que se trata de **transformar una puntuación a la nota correspondiente** en el intervalo [1, 7], redondeada a un decimal. Luego la puntuación a transformar es el parámetro que se necesita.

El paso 3 también podría considerarse simple, puesto que se trata de una expresión aritmética, más el redondeo a un decimal. Pero calcular promedios puede ser una subrutina útil en el futuro y es mejor encapsularla en una función: `calculaPromedio3Notas()`. Obviamente esta función necesita las tres notas para hacer el cálculo, que serían sus parámetros formales.

El paso 4 es similar al anterior, pero además se requiere determinar y considerar en el promedio la mejor nota una segunda vez. Esto puede considerarse complejo, por lo que lo encapsularemos en la función `calculaPromedio3NotasConAyuda()`, que tendrá los mismos parámetros que la función que calcula el promedio normal.

Con este análisis podemos definir el modelo de solución algorítmica mostrado en Solución 2.

**Solución 2: Bloque principal**

```
# Entrada de datos
1. punt1 = puntuación obtenida por un estudiante en la primera prueba
   (entregada por el usuario)
2. punt2 = puntuación obtenida por el estudiante en la segunda prueba
   (entregada por el usuario)
3. punt3 = puntuación obtenida por el estudiante en la tercera prueba
   (entregada por el usuario)

# Procesamiento de los datos
4. nota1 = calculaNota(punt1)
5. nota2 = calculaNota(punt2)
6. nota3 = calculaNota(punt3)

7. prom3Notas = calculaPromedio3Notas(nota1, nota2, nota3)
8. prom4Notas = calculaPromedio3NotasConAyuda(nota1, nota2, nota3)

# Salida de datos
9. Mostrar en pantalla los valores prom3Notas y prom4Notas
```

Las funciones `calculaNota()` y `calculaPromedio3Notas()` pueden **implementarse directamente** en un lenguaje de programación, ya que todos los detalles de cómo hacerlo están claros. Pero esta solución tiene que **refinarse** porque la función `calculaPromedio3NotasConAyuda()` no está completamente definida.

El sub-objetivo es calcular el promedio de cuatro notas, en que tres de ellas se reciben y la cuarta corresponde a la repetición de la mejor de las notas recibidas. Existen varias formas de conseguir este objetivo, pero la siguiente es una alternativa bastante directa.

**Solución 2: calculaPromedio3NotasConAyuda()**

```
Objetivo: Calcular el promedio de 3 notas considerando la mejor de ellas
          coeficiente 2.
Entrada: Las notas a promediar: nota1, nota2, nota3.
Asume: nota1, nota2 y nota3 son valores flotantes con un decimal entre 1,0 y
       7,0.-
Salida: prom, promedio de las notas recibidas considerando la mejor de ellas
        coeficiente 2.
Asegura: prom es valor flotante, redondeado a un decimal de precisión.

1. mejor = mejorNota(nota1, nota2, nota3)
2. suma = nota1 + nota2 + nota3 + mejor
3. prom = suma / 4
4. redondear prom a un decimal de precisión
```

Podemos darnos cuenta que los modelos de una **subrutina**, además de declarar el objetivo y los pasos a seguir, especifican los **datos de entrada** (parámetros formales) y los **dominios** que se asume para ellos. Si la subrutina es una **función**, además se ha de especificar los **valores devueltos** y las **restricciones** que éstos cumplen.

En el modelo de la función `calculaPromedio3NotasConAyuda()`, el paso 1 es el único que podría considerarse "poco claro", pero en realidad es muy simple de hacer en la mayoría de los lenguajes de programación. Por lo tanto, no se requiere mayor refinamiento y nuestro modelo de la solución está completo.

Lógicamente esto no es siempre así, y entre más complicado un problema, más ciclos de refinamiento tendremos que hacer. Después de un refinamiento, pueden seguir existiendo sub-objetivos complejos, que dejaremos expresados como llamadas a subrutinas. En un siguiente ciclo, seguramente aparecerán sub-sub-objetivos más simples, que podremos dejar al **ingenio del programador que las implemente**. Si quedan sub-sub-objetivos complejos, haremos otro ciclo de refinamiento hasta que sólo queden funciones o procedimientos simples.

Pregunta 4

Trabaja con tu grupo en la tercera pregunta de la actividad.

## Implementación

Como dijimos, la implementación es la operación **inversa** de la abstracción. Como analogía, podemos pensarlo como pasar del plano de una habitación a mezclar cemento y colocar ladrillos. Aquí decidimos **los detalles** necesarios para **llevar nuestro modelo de solución a un programa**. Esto invariablemente depende del **lenguaje de programación** que usemos. En algunos lenguajes será más simple **codificar** un paso de una forma y en otros lenguajes será mejor de otra manera, con otras estructuras de datos, con diferentes bibliotecas, etc. Nosotros estamos aprendiendo a **codificar** en Python, y nuestras decisiones entonces se inclinarán por las herramientas que estaremos aprendiendo a lo largo del semestre.

Una consideración general que podemos hacer es sobre la codificación de las condiciones requeridas por los datos de entrada a nuestras subrutinas. Cuando estas restricciones, normalmente de tipo y dominio, son simples, podemos **verificarlas al interior de la subrutina**. Aprenderemos a hacer esto en las próximas clases. Mientras tanto, simplemente las **documentaremos**, es decir indicaremos a los posibles usuarios de nuestras subrutinas qué condiciones deben cumplir los datos de entrada para que éstas

funcionen correctamente y los resultados sean válidos. Si el usuario no se preocupa de dar valores correctos, entonces la subrutina podría no funcionar (gatillando una excepción) o entregar valores inválidos. Nuevamente dependerá del lenguaje de programación que se utilice para decidir entre estas dos opciones.

**Ejercicio propuesto**

Realiza la implementación de la Solución 2 en lenguaje Python. Como ayuda, prueba en el intérprete de Python la siguiente invocación de la función `max()`:

```
>>> max(2, 4, 7, 1, 3)
```

## Bibliografía

Zelle, J. (2004). *Python Programming: an introduction to computer science.* Wilsonville, Oregon: Franklin, Beedle & Associates.