Datos y variables

Sitio: Agencia de Aprendizaje a lo largo de la Vida
Curso: Tecnicas de Programación 1° G
Libro: Datos y variables

Imprimido por: Sebastian Puche
Día: martes, 13 de agosto de 2024, 22:20

Tabla de contenidos

1. Datos y más datos

Como vimos, los programas utilizan algoritmos para ejecutar acciones sobre los datos. Pero…. ¿Cuáles son los datos que vamos a usar en programación? Y más importante aún, ¿qué es un dato?

Un dato es algo que uno necesita para resolver el problema.

Por ejemplo:

> **Hay 2 amigos que quieren 3 caramelos cada uno. ¿Cuántos caramelos necesito?**

En este caso, por enunciado nos dicen que son 2 amigos y quieren 3 caramelos cada uno.

Eso son datos que nos brindan para poder emplear un algoritmo y así obtener una solución a nuestro problema.

1.1. ¿Qué tipos de datos hay?

No es una respuesta simple y definitiva… depende. Depende del programa que uno está usando o del lenguaje de programación con el que estamos trabajando. Pero en general podemos distinguir 2 (dos) grandes grupos:

- Datos simples: atómicos. Por ahora trabajaremos con estos.
- Datos compuestos: un dato que está compuesto por un conjunto de datos simples o de otros datos compuestos (tema que se verá más adelante en la carrera).

Tipos de datos básicos (atómicos)

En general, en todos los lenguajes de programación son los mismos.

Nosotros utilizaremos:

- Enteros: admite números enteros, sin decimales, positivos y negativos (2, 3453, 1).
- Real: admite cualquier número real, con o sin decimales, positivos y negativos (0, 084, -4, 45,6).
- Cadena: admite cualquier cadena de caracteres (edad, altura).
- Lógico: es un tipo de dato booleano que admite solo verdadero (true) y falso (false).

Los tipos de datos básicos los podemos distinguir fácilmente: los usamos constantemente en nuestra vida diaria.

Por ejemplo:

“La Sra. Andrea Susana de 40 años paga $ 2252.14 de luz”.

Como podemos observar esta oración, tenemos varios datos:

- nombre y apellido
- edad
- monto a abonar
- concepto de la factura

¿Qué es un dato, entonces?

Es la representación simbólica de una recopilación de información. Es importante tener en cuenta que el dato no tiene sentido en sí mismo, sino que se utiliza en la toma de decisiones a partir de un procesamiento adecuado y teniendo en cuenta su contexto.

¿Y a qué llamamos información?

Está constituida por un grupo de datos ya supervisados, procesados y ordenados. Permite resolver problemas y tomar decisiones, ya que su aprovechamiento racional es la base del conocimiento.

La información es vital para todas las actividades. El ser humano tiene la capacidad de generar códigos, símbolos y lenguajes que enriquecen la información, la modifican, la reproducen y la recrean constantemente, otorgándole nuevos sentidos.

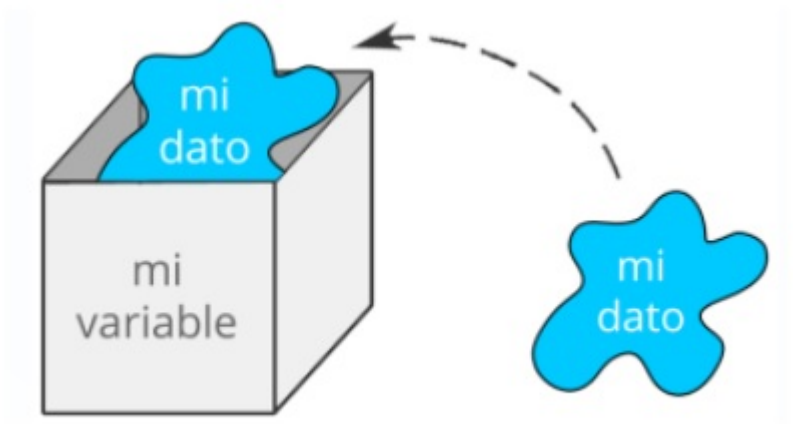
Tanto los datos como la información, pueden ser guardadas en variables de acuerdo a nuestra resolución. Pero….¿qué son las variables? Ya lo veremos…

¿Cuáles son los datos de entrada y de salida? Los datos de entrada son aquellos datos que necesita nuestro algoritmo para poder procesarlos y obtener los resultados esperados como datos de salida o información.

2. Las variables

¿Un dato es una variable? No, pero son dos conceptos relacionados. Podemos considerar que una variable es un contenedor identificado con un nombre. Puede albergar 1 sólo valor por vez aunque este, como lo indica su nombre, puede ir variando a lo largo de la ejecución del programa.

Cuando se hace referencia al nombre de la variable, indirectamente se hace referencia también a su contenido, que es el valor que la variable alberga en ese momento.



Toda variable conlleva un tipo de dato asociado y es el tipo de dato de la variable lo que nos indica el conjunto de valores que puede tomar y las operaciones que pueden realizarse con ella.

¿Recordás cuáles son los tipos de datos simples?

Un tipo de dato simple es aquel cuyo contenido se trata como una unidad y no puede reducirse en unidades más pequeñas (entero, real, cadena, lógico).

¡Importante!

Una variable es el contenedor de un tipo específico de dato.

2.1. ¿Cómo declarar una variable?

Una variable se declara para indicarle al programa a partir de qué lugar empieza a existir, qué nombre tendrá y qué tipo de datos almacenará.

Al momento de la declaración de una variable pondremos la palabra Definir, luego el nombre de la variable que querramos ponerle y a continuación la palabra “como” y seguido el tipo de dato que le corresponda.

Por ejemplo:

Definir cantidadDeAmigos como Entero

Con esta sentencia lo que hacemos es que a partir de este momento, la variable cantidadDeAmigos sea un espacio de memoria reservado para nuestro algoritmo y podamos almacenar en ella un valor numérico del tipo entero.

2.2. ¿Cómo elegir el nombre de una variable?

El nombre de una variable tiene que ser lo suficientemente descriptivo para que rápidamente podamos entender cuál será su propósito a lo largo de nuestro algoritmo.

En nuestro caso utilizaremos la nomenclatura camelcase (lleva este nombre debido a la joroba del camello).

Comienza

- con la primera palabra del nombre en minúsculas y luego
- sólo se pone en mayúscula la primera letra de cada palabra subsiguiente.
- En nuestro ejemplo queda: cantidadDeAmigos.

Las convenciones de codificación tienen los objetivos siguientes:

- Crean una apariencia coherente en el código, para que los/as lectores/as puedan centrarse en el contenido, no en el diseño.
- Permiten a los/as lectores/as comprender el código más rápidamente al hacer suposiciones basadas en la experiencia anterior.
- Facilitan la copia, el cambio y el mantenimiento del código.

Los objetivos principales son la coherencia y la legibilidad dentro del proyecto, el equipo, la organización o el código fuente de la empresa.

2.3. Características y restricciones de las variables

Veamos algunas características y restricciones que se deben tener en cuenta al momento de elegir el nombre de una variable. Algunas son por normas de buena programación y otras porque directamente los lenguajes no lo admiten:

- No pueden comenzar con un número.
- Sí pueden contener números en su nombre.
- Siempre comienzan en minúscula.
- No utilizar verbos para su nomenclatura.
- No pueden contener caracteres especiales como ser letras acentuadas, ñ, o cualquier otro símbolo (+ - * / ? ! $ % # etc) excepto el underscore (guión bajo _). Es decir, se utilizan los caracteres a-z y 0-9.
- Nombres de variables en singular.

¿Cómo guardamos valor en una variable?

La sentencia de asignación es la manera de setear o almacenar el valor a una variable.

Ejemplo de asignación del valor 2 a una variable llamada cantidadDeAmigos.

```
cantidadDeAmigos =  2
```

2.4. ¿Qué es una constante?

Constante: es un valor que se define de manera similar a la variable, pero que a diferencia de ésta, debe setearse o asignársele un valor antes de la ejecución del programa y no puede ser modificada una vez comenzada su ejecución.

Ejemplo de asignación del valor 'Unidad Uno' a la constante TITULO.

TITULO = 'Unidad Uno'

3. Resolución de problemas mediante la formulación de algoritmos

En el proceso de resolución de problemas se comienza por un enunciado el cual debe ser completo y preciso. Es lógico que no tiene sentido tratar de resolver un problema que no se comprende claramente.

Muchas veces, luego de haber probado la eficacia de un algoritmo, se llega a la necesidad de replantear su estrategia o método para hacerlo más eficaz o eficiente, a esto lo llamamos reingeniería del algoritmo.

¿Qué es lo necesario a tener en cuenta para poder resolver un problema usando un algoritmo?

4. Metodologías Top Down y Bottom Up

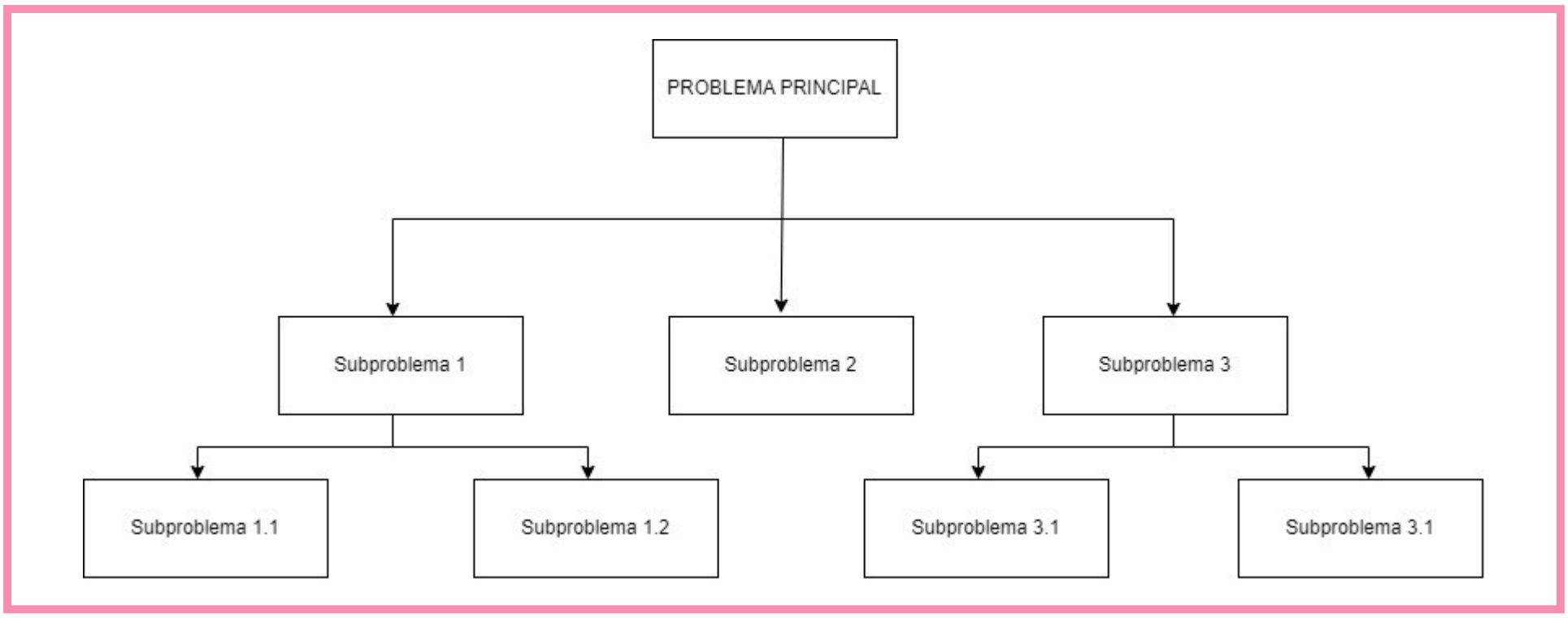
La programación consiste en el uso de algoritmos para resolver problemas, pero ¿de qué manera lo hacemos? Existen varias maneras posibles de realizarlo pero nos detendremos en dos metodologías, dos maneras diferentes de hacerlo.

Te invitamos a conocerlas en los siguiente capítulos.

4.1. Metodología Top Down

Se basa en una frase muy conocida: “Divide y vencerás“.

Consiste en dividir un problema en un conjunto de subproblemas menores, los cuales a su vez pueden ser divididos aún más, y se continúa con este proceso hasta obtener subproblemas que puedan ser manejados y sencillos de resolver. Esta metodología busca solucionar los problemas menores, para luego conectarlos y de esta forma lograr solucionar el problema principal.



Metodología Top Down: ventajas

- Mayor legibilidad: al dividir el problema en varios problemas menores, es fácil de entender qué es lo que quiso hacer aquella persona que diseñó la solución.
- Mayor productividad: al dividir el problema principal es posible que se le asignen los subproblemas a diferentes personas, con lo que se podría llegar a la solución final de una forma más rápida.

Metodología Top Down: programación modular

Todas estas consideraciones se acercan a la idea de programación modular.

- Cada problema debe descomponerse en una serie de problemas más pequeños hasta llegar a un nivel en que cada uno de ellos no pueda reducirse más.
- En ese momento se ha llegado al nivel más bajo del análisis.
- Es entonces cuando realmente se puede resolver el problema planteado al principio.
- Cada uno de estos problemas de orden superior puede usar, para su resolución, problemas mínimos, comunes a varios niveles.
- Una vez demostrada la necesidad de descomponer un problema general en problemas mínimos, resulta obvio que estos no son sino los módulos de que consta el problema.

De esa forma se realiza una programación modular y programación estructurada: el software obtenido es modular.

4.2. Metodología Bottom-up

La metodología Bottom-up hace énfasis en la programación y pruebas tempranas, que pueden comenzar tan pronto se ha especificado el primer módulo. Este enfoque tiene el riesgo de programar cosas sin saber, cómo se van a conectar al resto del sistema, y esta conexión puede no ser tan fácil como se creyó al comienzo. La reutilización del código es uno de los mayores beneficios del enfoque bottom-up.

El desarrollo de software moderno usualmente combina tanto top-down como bottom-up. Aunque un conocimiento completo del sistema se considera usualmente necesario para un buen diseño, haciendo que teóricamente sea un enfoque top-down, la mayoría de proyectos de desarrollo de software tratan de usar código existente en algún grado. El uso de módulos existentes le dan al diseño un sabor bottom-up.

4.3. Pseudocódigo

¿Querés comenzar a programar?

Para eso, vamos a comenzar aprendiendo que es el pseudocódigo.

El pseudocódigo es una representación de instrucciones expresadas en lenguaje natural. Tienen un orden específico que nos marca el rumbo para resolver nuestro problema.

La realización de un pseudocódigo, permite varias cosas:

- Utilizar un lenguaje común a todos los programadores.
- Lograr un nivel de abstracción cuando se realizan programas.
- Facilitar la traducción de las instrucciones a un lenguaje de programación.

Para comenzar a tratar los conceptos de resolución de problemas, utilizaremos una situación en el siguiente capítulo que se puede dar en lo cotidiano.

5. Enunciado del problema

Debo tomar el colectivo para viajar hasta el trabajo en la avenida Corrientes 244.

Lo primero que me debo preguntar es si “alcanza con el enunciado para vislumbrar la solución”.

Para plantear la solución de un problema primero debo analizar el enunciado del problema, y en caso de hacer falta, puedo mencionar ciertos puntos que aclaren los datos que creo pueden faltar. De esta manera “completo” el enunciado para poder encarar la propuesta de una solución.

Entonces, se definen hipótesis cuando los datos del enunciado no son suficientes para la resolución del problema.

Importante: nunca las hipótesis deben contradecir el enunciado.

Para este problema definimos:

- Hipótesis:
- Tengo suficiente saldo en la tarjeta SUBE.
- Conozco la dirección del trabajo.
- Funciona correctamente el servicio de colectivo.

También me debo preguntar ¿Qué datos tengo?

De la lectura del enunciado surge que:

Mi trabajo está ubicado en Corrientes 244.

5.1. Pasos de la solución

Estos son los pasos para la solución:

1. Salgo de mi casa.
2. Camino hasta la parada del colectivo.
3. Aguardo su llegada.
4. Subo al colectivo.
5. Abono mi pasaje.
6. Cuando estoy cerca toco el timbre.
7. Desciendo del colectivo.
8. Camino hasta mi trabajo.

Se trata de una enumeración detallada y ordenada de los pasos a seguir para resolver un problema.

Hemos expresado las líneas de cada paso con un número pues las mismas expresan una secuencia. Esto quiere decir que el orden de la ejecución es línea a línea de acuerdo a los números.

Con este primer ejemplo, aprendimos a detallar paso a paso un problema (como en el caso de la receta de cocina) y que al finalizar, este conjunto de instrucciones, he solucionado mi problema.

La ejecución del problema desarrollado me va a permitir viajar hasta mi trabajo.

Es importante también que la información proporcionada en el enunciado del problema cumpla con la regla de las “Tres Ce” (Claro, Conciso y Completo).

En este primer problema aprendimos:

En este primer problema aprendimos:

- Cómo transformar un enunciado de un problema en un conjunto de pasos para resolverlo.
- Que la solución de un problema está compuesta por una secuencia de pasos.