**Introducción a los sistemas**

**¿Qué es un sistema?**

Iniciamos el recorrido introduciendo los conceptos básicos relacionados con sistemas, sistemas informáticos y software. Esta unidad nos servirá para unificar las definiciones que debemos conocer antes de comenzar a programar.

Seguramente el término “**sistema**” lo habrán conocido cuando cursaban su escuela secundaria, en la asignatura Biología.

También, en la vida cotidiana utilizamos este concepto para hablar del sistema ferroviario, sistema de salud, sistema inmunológico, por nombrar algunos.

Entonces... ¿Cómo definimos un sistema?

**Un sistema es un conjunto de elementos relacionados entre sí que tienen un cierto orden u organización y que cumplen una función.**

Te contamos que el concepto de sistema surge del denominado enfoque sistémico, basado en la Teoría General de los Sistemas formulada a mediados del siglo XX.

Inicialmente, esta teoría estaba basada en el estudio de los organismos como sistemas biológicos, pero luego se generalizó su alcance a todo tipo de sistemas. De tal manera que hoy se utiliza el término sistema en todas las áreas del conocimiento humano.

En este curso nos interesan los **sistemas de información**, que son aquellos *sistemas que procesan datos con la finalidad de generar, transformar y distribuir la información.*

 Los sistemas de información están formados por **hardware**(elementos físicos) y **software**(elementos lógicos que se llaman programas). En este curso nos ocuparemos del desarrollo del software, estudiando distintas técnicas de programación para la construcción de los programas.

**¿Qué es un software?**

El software en un sistema corresponde a todos los elementos lógicos, intangibles.

**Ahora bien, las computadoras, los cables, los celulares, los dispositivos en donde guardamos información, ¿son Software?**

Seguramente estarás pensando que no, todos estos elementos son tangibles.

El software de un sistema informático es el **conjunto de programas** necesarios para que una computadora funcione. Un **programa** es un conjunto de instrucciones que indican a una computadora las tareas que tiene que realizar.

Para la construcción de **software** se debe seguir un determinado procedimiento que garantice que el resultado sea un producto de calidad. Hay varios modelos o procedimientos a seguir, lo que se traducirá en distintas metodologías de desarrollo, algunas de las cuales veremos más adelante.

### Ciclo de vida de software

Todo programa está formado por un conjunto de **módulos o subprogramas**que interactúan entre sí. En muchos casos, durante todo el proceso de desarrollo, intervienen distintas personas con diferentes responsabilidades. Es por eso que los programas atraviesan diferentes etapas durante su proceso de construcción, y es lo que se conoce como **"ciclo de vida**".

El desarrollo de un sistema se realiza durante todo el ciclo de vida, que es el período de tiempo que se extiende desde la idea original del problema a resolver hasta el mantenimiento y desarrollo de las mejoras.

Independientemente de las metodologías de desarrollo que se utilicen, podemos identificar las siguientes **etapas o momentos en el desarrollo del software:**

##### **Análisis del problema**

En esta etapa se debe determinar cuál es el problema a resolver y los límites y alcances que tendrá el software que lo resolverá. Es el momento de reunirse con quien nos solicita el programa para saber cuáles son los requerimientos.

##### **Especificación del software**

En este momento los profesionales de sistemas se encargan de definir las entradas y las salidas del software, y qué restricciones tendrán los datos. También se describen los componentes que se deberán desarrollar, qué características y comportamiento tendrán y cómo estarán relacionados.

##### **Desarrollo del software**

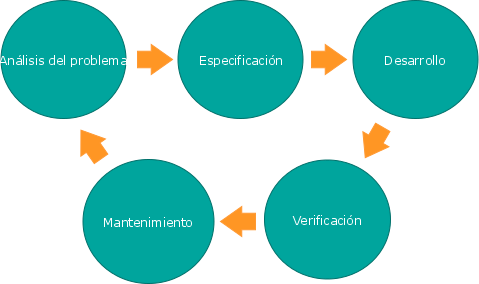
Corresponde al proceso de construcción de software propiamente dicho. Es en donde los programadores escriben el código fuente utilizando algún lenguaje de programación.

##### **Verificación del software**

Una vez que el software está desarrollado se debe probar para verificar que responde a las definiciones y no tiene errores.

##### **Mantenimiento del software**

Todo sistema debería tener mantenimiento ya que siempre habrá que realizar alguna modificación, agregando nueva funcionalidad o bien cambiando alguna característica porque se ha modificado alguna especificación.

  
El siguiente esquema nos presenta una visualización del ciclo de vida del software:

### Metodologías de desarrollo de sistemas

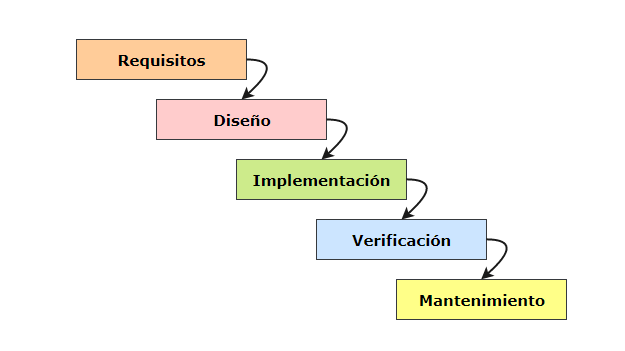
Como definimos anteriormente, las **etapas del ciclo de vida del software**son los procesos generales que suele tener todo desarrollo. Ahora bien, a lo largo de las últimas décadas se han estudiado estos procesos para buscar cuál es el método óptimo. Como resultados de esos trabajos se han desarrollado diferentes metodologías de desarrollo de sistemas. Cada una de ellas presenta ventajas y desventajas, y no hay una que sea mejor que las otras en todos los casos.

Dependiendo del proyecto, la cantidad de personas que intervengan y hasta los plazos de desarrollo y entrega del producto, algunas metodologías serán mucho más eficientes que otras. Por eso es importante conocer cuáles son las principales metodologías, para poder tomar tus propias decisiones.

**Una metodología de desarrollo de software es un entorno de trabajo que es usado para estructurar, planear y controlar el proceso de desarrollo de sistemas.**

La **metodología**define una representación que facilita la manipulación de modelos, junto con la comunicación e intercambio de información entre todas las personas involucradas en el desarrollo del sistema.

Les presentaremos a continuación los principales **modelos de desarrollo**:

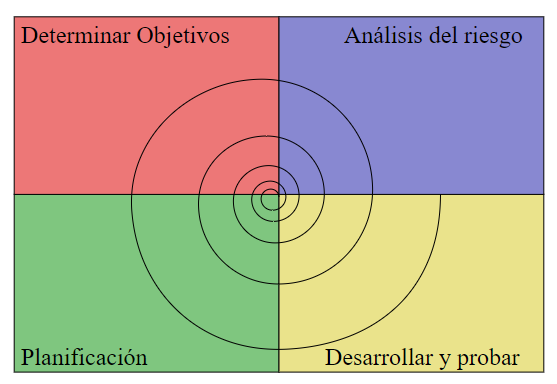
1. Modelo de cascada,
2. Modelo espiral,
3. Modelos de desarrollo iterativo e incremental,
4. Modelo de desarrollo ágil.
5. **El modelo cascada** define las siguientes etapas que deben cumplirse de forma sucesiva:

Este modelo es el denominado **modelo clásico de software**, ya que está relacionado con las etapas del **ciclo de vida**.

El nombre de **cascada**se debe a que las etapas se realizan de forma individual y en secuencia, es decir, hasta que no finaliza una fase no comienza la otra. Es por eso que se va avanzando en las etapas de una en una.  
Este es el modelo de construcción de los primeros sistemas informáticos. Hoy en día se sigue utilizando, pero tiene como desventaja que debe pasar mucho tiempo antes de que el cliente pueda llegar a ver el sistema.

### 2. Modelo espiral

En el **modelo espiral**, durante cada ciclo se realizarán cuatro actividades:



Este modelo se basa en el desarrollo en cascada, pero a diferencia de ser una serie de pasos de manera sucesiva, se desarrolla en ciclos que se repiten en forma de espiral.  
Comenzando desde el centro, se suele interpretar como que dentro de cada ciclo de la espiral se sigue un Modelo Cascada, pero no necesariamente debe ser así.

### 3. Modelos de desarrollo iterativo e incremental

En estos modelos también se realiza un **desarrollo por ciclos.**

Si estamos realizando un **desarrollo iterativo,**en cada ciclo o iteración, se revisa y mejora el producto desarrollado. Es importante señalar que este ciclo no implica añadir funcionalidades en el producto, pero si la revisión y la mejora.

Énfasis en validación y verificación con usuarios.

Un ejemplo de **desarrollo iterativo** es aquel basado en refactorizaciones, en el que cada ciclo mejora más la calidad del producto.

Si estamos realizando un **desarrollo incremental**, en cada ciclo iremos liberando partes del software (prototipos) periódicamente, en cada iteración, y cada nueva versión, normalmente, aumenta la funcionalidad y mejora en calidad respecto a la anterior.

### 4. Modelo de desarrollo ágil

El **modelo de desarrollo ágil** se basa en los modelos iterativos e incrementales, en donde los ciclos son muy cortos (en general una o dos semanas) y con la distinción que dentro de ellos no se encuentran etapas en cascada.

El modelo ágil está centrado en las personas, ya que, en lugar de haber una planificación bien específica de todo el desarrollo al principio, se basa en la presentación en muy corto plazo de módulos y en la retroalimentación de esas validaciones.   
De esta manera el cliente se encuentra involucrado desde el principio del desarrollo y participa durante todo el proceso de construcción.

### Lenguajes de programación

Para que una computadora pueda realizar un proceso es necesario que se le brinde una lista de instrucciones que sea capaz de comprender y de ejecutar. Como vimos anteriormente, un programa o software es ese conjunto de instrucciones. Esos programas se escriben utilizando un lenguaje de programación.

¿Qué es un lenguaje de programación?

**Un lenguaje de programación es un lenguaje utilizado para escribir programas que sean entendidos y procesados por una computadora.**

Ahora te presentaremos los principales **tipos de lenguajes de programación**utilizados en la actualidad:

#### **Lenguaje de máquina**

Este lenguaje es el que entiende directamente una computadora. Está formado por instrucciones binarias (dígitos 0 y 1) que especifican operaciones y las direcciones de memoria implicadas en dichas operaciones.

El código máquina es el denominado código binario. Estas instrucciones dependen del hardware de la computadora, y por lo tanto, varían de una en otra.

Este lenguaje al ser entendible por computadoras es muy poco legible y claro para los programadores, es por eso que se han desarrollado otros lenguajes de programación más cercanos a los idiomas para que sea mucho más fácil programar. Estos lenguajes se dividen en lenguajes de bajo y de alto nivel.

#### **Lenguaje de bajo nivel**

Es más simple de utilizar que el lenguaje máquina, pero también depende del hardware en donde se esté ejecutando.   
**El principal lenguaje de bajo nivel es el ensamblador**.

#### **Lenguajes de alto nivel**

Estos lenguajes son los más utilizados por los programadores, ya que están diseñados para que las personas lo pueden entender de manera mucho más fácil que los lenguajes de bajo nivel.   
Otra ventaja que tiene es que un programa escrito en este tipo de lenguaje es independiente del hardware en donde se va a ejecutar.

Los principales lenguajes de alto nivel son C, C++, Pascal, Java, VisualBasic. 

Los programas fuente escritos en estos lenguajes tiene que ser traducidos al lenguaje máquina. Para eso se utiliza un **programa traductor** que es un intérprete o un compilador.

**Lenguajes interpretados y compilados**

Como hemos visto, un programa escrito en un lenguaje de alto nivel es entendible solo por una persona, es decir, el programador. Por lo tanto, para que pueda ser entendible y ejecutado por una computadora debe ser traducido al lenguaje máquina, por medio de un intérprete o un compilador. De esta manera tenemos**lenguajes interpretados o compilados**.

**¿Qué es un lenguaje interpretado?**

**Los lenguajes interpretados son aquellos que se ejecutan por medio de un intérprete.   
Un intérprete es un traductor que toma el código fuente, lo traduce y luego lo ejecuta.**

Un lenguaje interpretado clásico es el "Basic" que hoy, prácticamente no se utiliza.

Por otro lado, los **lenguajes compilados** son aquellos que necesitan un compilador para poder ejecutarse.

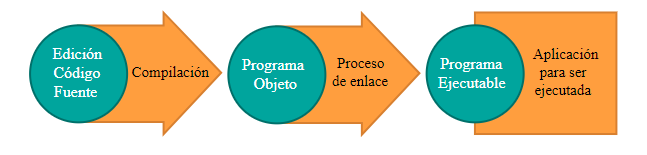
**¿Qué es un compilador?**

**Un compilador es un programa especial que toma el código fuente de alto nivel y lo convierte en instrucciones de código entendible por una computadora, denominado código objeto.**

En general, el **programa objeto** es la traducción al código máquina, y que depende de cada modelo de computadora. Por lo tanto, el compilador debe ser específico para el modelo de computadora que va a ejecutar el programa.

**El proceso de compilación abarca las siguientes etapas:**

1. Escritura del programa fuente.
2. Compilación del programa fuente.
3. Verificación de errores de compilación.
4. Obtención del programa objeto.
5. Enlace del programa objeto con todos los programas necesarios del sistema operativo.
6. Ejecución del programa.



Síntesis de unidad 1 (introducción a los sistemas): un tipo de sistema es el sistema de información, conformado por hardware (tangible) y software (intangible/lógico), el software pasa diferentes etapas de construcción (ciclos de vida de software). Se utilizan diferentes metodologías de desarrollo de software (procesos de construcción de sistemas c/u con sus ventajas y desventajas), dependiendo de las características del software, cantidad de recursos y tiempo de entrega, algunas metodologías serán mejores que otras (modelo cascada – desarrollo incremental – modelo espiral – desarrollo iterativo – desarrollo ágil). Los diferentes tipos de lenguajes son de bajo nivel (ensamblador) o alto nivel (que requieren ser interpretados por un intérprete ó compilados por un compilador).

**Resolución de problemas**

En esta unidad vamos a presentarles una metodología de **Análisis de Problemas**que les permitirá identificar los componentes que forman un problema y comprender qué es lo que hay que resolver, qué datos se necesitan, con qué valores y cuáles serán las salidas y los resultados del problema.

**¿Qué es un problema?**

Podemos decir que un problema **es el planteamiento de una situación a resolver mediante la aplicación de algún método**.

En la vida cotidiana resolvemos muchos problemas sin darse cuenta y de forma automatizada pero también lo hacemos de forma innovadora o creativa.

**En esta materia nos ocuparemos de problemas que se resuelven programando una computadora y que se describen por medio de enunciados.**

Debemos entonces hacer una diferencia entre problema y enunciado.

**Los enunciados** son la descripción de un problema que se puede resolver mediante una computadora. Es decir, **los problemas se plantean mediante enunciados**.   
En los enunciados debería incluirse toda la información necesaria para poder resolver el problema.

Volviendo a los problemas generales, ante un problema, lo que debemos hacer es resolverlo, utilizando un método propio o planeamiento para encontrar una solución.

### ¿Cómo empezar a resolver un problema?

Para poder resolver un problema, la primera cuestión a plantearse parece bastante evidente. Pero esta "evidencia" es lo que hace que en la mayoría de los casos se pase por alto, o bien se realice muy rápido, con la consecuencia de traernos inconvenientes futuros a lo largo de toda la resolución del problema.

Entonces, debemos enfocarnos en:

**Comprender el problema**  
  
Comprender un problema implica "saber" cuál es el problema. Así, si no sabemos cuál es el problema ni la información para resolverlo, difícilmente podamos llegar a una solución adecuada.

Esto que resulta tan simple decirlo, no siempre se realiza de la forma apropiada. Por ello, es que nos planteamos una**metodología**para poder determinar un orden y así realizar el análisis del problema.   
  
¿Cómo podemos realizar un adecuado análisis?

Para comprender un problema debemos realizar lo que se denomina **Análisis del Problema** (también llamado **Análisis de Enunciados**).

Pasamos a explicar cómo podemos realizar un adecuado análisis de un problema...

### Análisis del problema

Consideremos un problema que se nos presenta descripto en un enunciado.

Entonces, el primer paso será **leer el enunciado del problema**. Ahora bien, ¿cuántas veces debe leerse un problema? La cantidad de veces que sea necesario.   
Seguramente, no alcanzará con leer el enunciado una sola vez, ni otra más, recién en una tercera lectura podremos empezar a comprender el problema con todos sus datos.

La lectura tampoco finalizará acá, ya que será necesario volver a releer el enunciado ante cuestiones, dudas o situaciones que no se puedan definir en las siguientes etapas.

Este primer paso de una lectura reiterada del problema, los programadores muchas veces la suelen pasar por alto, o bien, leen el enunciado “por arriba” comenzando rápidamente a escribir el programa.  
Nuestra experiencia nos hizo ver que hay que tomarse todo el tiempo necesario en el momento del **análisis del problema**, porque los errores arrastrados serán luego más costosos de resolver.

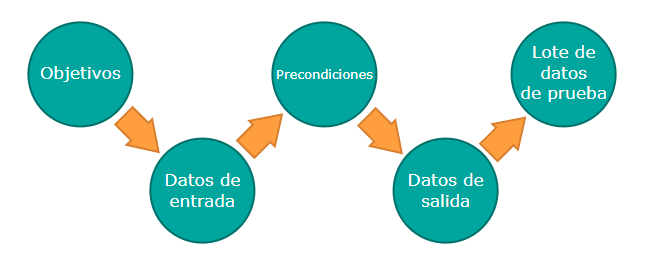
Proponemos como ejemplo de un **problema**: un viaje de vacaciones.  
  
Una de las primeras cuestiones que nos planteamos es el propósito del viaje. Si bien un viaje de vacaciones, es de esperar que sean para descansar, podemos tener varios propósitos como ser: conocer un lugar distinto, ir a una playa, disfrutar de sierras o montañas, practicar un deporte como esquiar, compartir unos días con familiares o amigos, etc.

Es decir, que el **problema tiene un propósito, un OBJETIVO.** Este objetivo del problema será el primero de los componentes que debemos identificar.

Veamos a continuación cuáles son todas las componentes que debemos identificar de un problema...

**Componentes de un problema**

Como hemos dicho, el objetivo del problema es el primero de los componentes que debemos identificar en un problema para luego continuar con los demás componentes, que se muestran en el siguiente esquema:



### 1. Objetivos

Como hemos mencionado, el objetivo es el primero de los componentes que debemos identificar en un problema.

El **objetivo** explica el problema a resolver, lo que hay que realizar. Es importante destacar que el objetivo indica el **QUÉ**y no el **CÓMO**. En el objetivo se plantea lo que se tiene que realizar, pero no de qué manera se va a resolver.

¿Cómo definimos un objetivo?

* El objetivo, es un párrafo en el que se tiene que explicar de manera coloquial desde dónde se parte y hacia dónde se debe llegar.
* El objetivo explica el problema a resolver, lo que se tiene que realizar.

Retomando nuestro ejemplo del viaje de vacaciones...  
  
Una vez planteado el objetivo del viaje, llegarán otras varias preguntas como, por ejemplo, qué presupuesto disponemos, cuántos días durará el viaje, si viajaremos en algún medio de transporte en particular, quiénes van a viajar, etc.  
  
Todos estos datos representan la información con la que contamos para llegar a una solución, es decir que, de acuerdo a estas respuestas, a estos **DATOS**, serán las vacaciones.

Entonces, luego de tener bien en claro qué es lo que se tiene que hacer, el paso siguiente es identificar la información con la que se cuenta para poder lograr el objetivo. Esto se realizará especificando los **Datos de Entrada.**

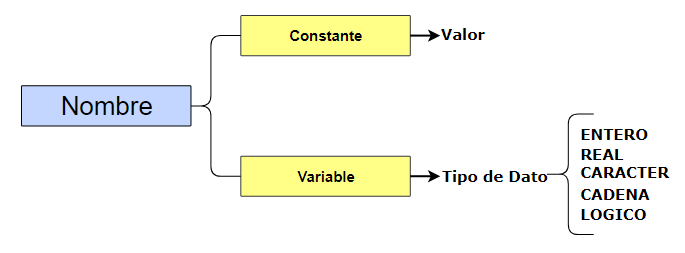
**2. Datos de entrada**

Los Datos de Entrada especifican la información que se tiene antes de comenzar a resolver el problema.   
Son los datos con los que se cuenta para poder llegar a una solución.

Las **características** que se deben especificar para cada dato de entrada son:

* Nombre.
* Constantes y variables.
* Tipos de datos.

En el siguiente esquema presentamos cómo es la relación de los elementos de cada Dato de Entrada.



### 2.1. Nombre

A cada Dato de Entrada le vamos a definir un NOMBRE. El nombre debe ser representativo del dato para no tener confusiones sobre su valor. Por ejemplo, si el dato va a representar la **Edad de una persona**, lo ideal es que el dato se llame EDAD.

Una de las buenas prácticas de programación establece que el nombre del dato debe representar su valor.   
Es importante evitar los nombres genéricos, que muchas veces son muy tentadores, como N1, N2, Dato, NÚMERO,  X, etc. , ya que dificultan la lectura y comprensión del problema.

Volviendo a nuestro ejemplo del viaje de vacaciones...  
  
Los **datos de entrada** podrían ser: PRESUPUESTO, DÍAS DE DURACIÓN, MEDIO DE TRANSPORTE.   
  
Suponiendo que te gusta manejar y disfrutar de un viaje en auto, ¿Vas a destinar el mismo presupuesto y la misma cantidad de días para todas las vacaciones?   
Seguramente que no, ya que más allá de la variación de precios, el presupuesto seguramente será distinto. Lo mismo sucederá con la cantidad de días, en algún viaje le podrás destinar un fin de semana largo, 2 días, o en otro viaje podrás destinar 14 días. En cambio, siempre viajarás en auto, ya que te gusta y disfrutás del manejo.   
  
Llegamos a la conclusión que el medio de transporte, uno de los datos de entrada identificados, permanece siempre igual (es su auto), por lo tanto, será un dato CONSTANTE, ya que para todos los viajes será el mismo.   
Por otro lado, los otros dos datos de entrada identificados, *presupuesto* y *días de duración*, serán datos VARIABLES ya que serán distintos para cada viaje.   
  
Resumiendo:

| **DATO** | **TIPO** |
| --- | --- |
| **MEDIO DE TRANSPORTE** | **CONSTANTE** |
| **PRESUPUESTO** | **VARIABLE** |
| **DURACIÓN** | **VARIABLE** |

Esto que acabamos de realizar es una clasificación de los datos de entrada en VARIABLES y CONSTANTES, y que desarrollaremos en la siguiente sección.

### 2.2. Constantes y variables

Además de indicar el nombre, debemos realizar una clasificación adicional por cada Dato de entrada, y será de acuerdo a si el valor del dato va a ser el mismo o si puede cambiar durante la ejecución del programa.

Si el dato tiene siempre el mismo valor para todos los problemas, entonces el dato de entrada es **CONSTANTE** si en cambio, su valor puede ser distinto para cada problema o bien puede cambiar dentro del mismo problema, el dato de entrada es **VARIABLE.**

Las constantes matemáticas son ejemplos de **datos Constantes**, como, por ejemplo, el número PI.  
Mientras que **datos Variables** pueden ser los datos de una factura (fecha, productos, importe), ya que lo usual es que los datos de cada factura sean distintos.

Para tener en cuenta:   
Los Datos de Entrada Constantes deben indicarse con su valor.   
Por ejemplo, **PI = 3,14** o **IVA = 0,21.**Siempre debe haber, por lo menos, un dato de entrada **Variable**.

Volvamos a nuestro ejemplo de las vacaciones y los datos de entrada que definimos…  
  
Sabemos que los **DÍAS DE DURACIÓN**, es un dato VARIABLE. Analizando un poco más en profundidad, qué valores posibles podrá tener, como se trata de días, el valor será un número. Si queremos ser más específicos, debería ser un NÚMERO NATURAL.   
Del mismo modo, el otro dato variable, el **PRESUPUESTO**, también será un número, pero en este caso, el número podrá ser un [NÚMERO REAL](https://es.wikipedia.org/wiki/Número_real). (decimales, por ejemplo)

Esta especificación que acabamos de mencionar con los datos corresponde a lo que llamaremos TIPOS DE DATOS y se lo desarrollaremos en el siguiente apartado.

### 2.3. Tipos de datos

Ya sabemos que **el Tipo de Dato indica cuál es el conjunto de valores posibles del dato**. Además, permite definir cuáles serán las operaciones que se podrán aplicar a este dato.

**El tipo de dato se debe especificar para los Datos de Entrada Variables, ya que en los Constantes se deducen de acuerdo a su valor.**

Vimos también que los tipos de datos pueden ser números (enteros, reales, por ejemplo). Pero esos no son los únicos. Se estarán imaginando que algunos problemas necesitan trabajar con nombres de personas, letras o símbolos para indicar algún código, y hasta con valores de tipo SI-NO.

Es por eso que en la etapa del Análisis del problema vamos a trabajar con los siguientes tipos de datos estándar:

| **Tipo de Dato** | **Valor** |
| --- | --- |
| **Entero (Integer)** | Números enteros |
| **Real (Float)** | Números reales |
| **Caracter (Character)** | Un solo símbolo (letra, dígito, signo, etc.) |
| **Cadena (String)** | Un conjunto de caracteres (palabra, frase, etc.) |
| **Lógico (Boolean)** | Verdadero o Falso |

Las operaciones sobre cada tipo de datos las veremos más adelante, ya que por ahora solo interesa saber los tipos de datos y sus posibles valores.

Retomando nuevamente el ejemplo del viaje de vacaciones...   
  
Sabemos que DÍAS DE DURACIÓN es un dato VARIABLE de tipo ENTERO, pero...¿cualquier valor dentro de los números enteros es correcto?   
  
Sabemos que por lo menos podemos tomarnos 1 día de vacaciones. Esa será la cantidad mínima de días que se podrá viajar, no tiene sentido irse 0 días o -5 días de viaje. Por lo tanto, solo serán válidos algunos números enteros.  En este caso podemos pensar en un subconjunto que está formado por los números enteros mayores o iguales a 1.   
  
Por otro lado, si bien tener muchos días de vacaciones sería algo que deseamos todos, nuestras obligaciones (y nuestro jefe) nos limitarán la cantidad máxima de días ¿Cuál será? ¿30, 60, 28? Será un aspecto a definir con mayor profundidad en las próximas unidades, pero sabemos que existe un límite máximo en los datos.

Estos límites o subconjuntos que acabamos de indicar sobre los TIPOS de DATOS es lo que llamamos **PRECONDICIONES** o, también, RESTRICCIONES, y lo veremos en la próxima sección.

### 3. Precondiciones

Con las **Precondiciones**se indican los valores posibles que deberán tener los Datos de Entrada Variables.

Acá tenemos que prestar mucha atención para poder diferenciar el Tipo de dato de las precondiciones, ya que son conceptos similares, aunque indican cosas distintas.

El **tipo de dato** indica el conjunto posible de valores, por ejemplo, los números reales, mientras que las **precondiciones**establecen cuáles de esos números reales serán válidos para el problema.

Por ejemplo, ante un dato de entrada que indica la longitud del lado de un cuadrado, el tipo de dato será REAL. Ahora bien, ¿cualquier número REAL es válido para la longitud de un lado?   
Por supuesto que no, ya que para que sea lado, deberá ser un número mayor a 0.  **Esa es la precondición.**   
Las precondiciones las podemos especificar como condiciones matemáticas o expresarlas en palabras, siempre y cuando no haya ambigüedades y quede bien claro cuáles son los valores posibles.

Por ejemplo, la **precondición de la longitud del lado de un cuadrado**, se puede expresar de la siguiente manera:

**LADO > 0**O también:   
  
**LADO tiene que ser mayor a 0.**

Las precondiciones son importantes porque indican cuándo un dato es un dato válido para el problema. Así, tenemos que tener en cuenta que: **un dato de entrada será válido cuando cumpla con todas las Precondiciones**.

### 4. Datos de salida

Hasta el momento hemos especificado todo lo relacionado con los datos que necesitamos para resolver el problema, es decir la ENTRADA del problema. Así como se indica la entrada, se debe especificar los datos que se generarán al resolver el problema, que se llaman **Datos de Salida.**

Los **Datos de salida** se utilizan para especificar la información que se tiene después de resolver el problema.

Para este tipo de datos no será necesario realizar ninguna otra clasificación porque es de esperar que todos los datos de salida sean siempre VARIABLES, pues dependerán de los Datos de Entrada.

Es decir, que solo alcanza con definir la lista de nombres de los Datos de Salida.

### 5. Lote de Datos de Prueba

Hemos analizado y especificado la Entrada y la Salida del problema, para luego plantear un *conjunto de datos de prueba*. Este lote de datos servirá para más adelante, luego de escribir el programa, para poder realizar lo que se denomina *Prueba de Escritorio*.

Una forma de verificar que el programa no tenga errores lógicos es mediante la realización de la Prueba de Escritorio, tomando como datos de entrada los especificados en este **Lote de Datos de Prueba.**

Una vez desarrollado el programa, nos preguntamos: ¿Cómo sabemos si está bien?

**Tendremos que usar el Lote de Datos de Prueba. Si el programa produce las salidas esperadas en este lote, entonces podremos afirmar que el programa funciona.**

**El lote de datos de prueba** se compone de *una lista de valores para los datos de entrada y las salidas correspondientes* a esos datos. Podemos representarlo en forma de tabla de valores, sin necesidad de especificar cómo se llegó a ese resultado, ya que de eso se ocupará el programa.

| **ENTRADAS** | | | | | **SALIDAS** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dato Entrada 1 | Dato Entrada 2 | ... | Dato Entrada N | Dato Salida 1 | | ... | Dato Salida N |
| - | - | - | - | - | | - | - |
| - | - | - | - | - | | - | - |

Armar el lote de Datos no es una tarea para tomar a la ligera, ya que serán los datos con los que se probará el programa. Si el lote es erróneo o hay situaciones que no se contemplan, entonces la prueba será incompleta y no se sabrá si el programa funcionará correctamente para todas las situaciones.

Veamos un ejemplo:

**Tenemos un programa que realiza la suma de dos números si ambos son positivos o realiza la resta si ambos son negativos.**  
  
El lote de datos de prueba deberá abarcar las dos situaciones, ambos números negativos o ambos números positivos. No tiene sentido realizar una tabla de cientos de valores, ambos positivos, ya que todos contemplan una misma situación y no se está evaluando cuando ambos números son negativos. Para este caso, un lote de datos ideal estaría formado por un par de números positivos y otro par de números negativos.   
Por ejemplo, podemos probar con los números  -2;-3;-5; 7; 8; 15; 1000.

**Estructuras de datos**

### Datos de información

En la unidad 2 aprendimos a reconocer un problema y a especificar todos los elementos del mismo. Seguramente estarás recordando que, dentro de esos elementos que identificamos, se encontraban **los Datos de Entrada y los Datos de Salida**.

En esta unidad, veremos cómo se representan esos datos y para eso introduciremos las ideas de tipos de datos, variables y constantes. También presentaremos los tipos de datos a utilizar en los algoritmos y las operaciones que podremos realizar con cada uno de ellos.

Cuando hablamos de datos, tanto de entrada como de salida y decimos que tienen que tener un nombre para poder identificarlos, pero...

¿Dónde se identifican los datos?

**Para poder utilizar y repetir los datos, debemos guardarlos en la memoria. La computadora tiene también una memoria para almacenar los datos, la información que necesita y que genera todo algoritmo.**

No entraremos en detalle sobre cuestiones físicas o características de la memoria, ya que es un tema de otra asignatura.  Lo que nos interesa en esta materia, son los datos y sus posibles valores. Entonces:

Volvamos al ejemplo que planteamos sobre nuestro viaje de vacaciones....

recordemos que definimos varios datos, entre ellos DÍAS DE DURACIÓN y PRESUPUESTO. Ya habíamos determinado que ambos datos tendrán valores numéricos, pero no nos quedamos ahí, ya que establecimos que los DÍAS son números **enteros** y el PRESUPUESTO es un número **real.**

Ahora bien, ¿qué significa esto? Evidentemente un número real tiene características distintas que un número entero, por ejemplo, la parte decimal. Del mismo modo que si nosotros escribimos un número entero o real en un papel, utilizamos distintos espacios, estos números ocuparán en memoria una cantidad de espacio diferente.

### Estructura de datos primitivas

Además del espacio que ocupan estos números de distinto formato en la memoria, hay otras características distintivas.

Pensemos en un **número entero**, por ejemplo, **la edad de una persona**. Con esta edad habrá operaciones o acciones que podremos hacer, como determinar si la edad supera los 18 años para determinar si es una persona mayor de edad. También, si tenemos muchas edades, podremos calcular el promedio de esas edades.

Es decir, con la edad, al ser un dato numérico, podremos hacer operaciones que se aplican a números, como por ejemplo sumar, restar, calcular promedios, obtener un máximo, entre otras operaciones.  
Ahora bien, con el nombre de la persona, no podremos calcular el promedio, ya que no tiene sentido. Seguramente, usted ya se dio cuenta que con un nombre no se pueden hacer operaciones matemáticas, debido a que el dato no es un número.

En definitiva, seguramente pudieron identificar que hay **dos características** que dependen del valor de los datos:

* **El espacio que ocupa en memoria.**
* **Las operaciones disponibles.**

Por lo tanto, es necesario poder identificar de alguna manera el conjunto de datos posibles para poder definir el **espacio** y las **operaciones**, y es debido a ello que surge la necesidad de tener un Tipo de Dato.

De este modo podemos definir al **Tipo de Dato** como la característica que tiene un valor que determina las operaciones que se pueden realizar y cómo es almacenado en la memoria.

En este módulo, en el momento de desarrollar un algoritmo, trabajaremos con los Tipos de Datos simples, también llamados primitivos o estándares, que se encuentran definidos en la mayoría de los lenguajes de programación. En las siguientes unidades veremos la posibilidad de crear nuevos Tipos de Datos.

### Tipos de datos

Los tipos de datos estándares son los mismos que definimos en el Análisis previo, y son los siguientes:

| **Tipo de Dato** | **Valor** |
| --- | --- |
| **Entero (Integer)** | Números enteros |
| **Real (Float)** | Números reales |
| **Caracter (Character)** | Un solo símbolo (letra, dígito, signo, etc.) |
| **Cadena (String)** | Un conjunto de caracteres (palabra, frase, etc.) |
| **Lógico (Boolean)** | Verdadero o Falso |

##### **[Tipo de dato: Entero](http://aulavirtual.codoacodo.com.ar/mod/lesson/view.php?id=43&pageid=47" \l "collapse1)**

El dato de tipo Entero será un **valor numérico dentro** del conjunto de los números enteros. Algunos ejemplos de valores enteros son:**5 ; 257 ; 0 ; -734 ; -1 ; 1**  
Recordemos que no nos interesan las cuestiones físicas de los lenguajes o del límite de memoria, por lo tanto, no pondremos límites a los valores, teniendo disponible el conjunto de números enteros desde -∞ hasta ∞.   
En un lenguaje de programación los tipos de datos están acotados ya que ocupan una cantidad finita de espacio en la memoria, pero de este tema nos ocuparemos cuando veamos ejemplos de lenguajes.

##### **[Tipo de dato: Real](http://aulavirtual.codoacodo.com.ar/mod/lesson/view.php?id=43&pageid=47" \l "collapse2)**

El dato de tipo Real será de un **valor numérico**que forma parte del conjunto de los números reales. Algunos ejemplos de valores reales son: **2,25 ; 1429,1456 ; 0,0 ; -12,0 ; -27,34.**

##### **[Tipo de dato: Caracter](http://aulavirtual.codoacodo.com.ar/mod/lesson/view.php?id=43&pageid=47" \l "collapse3)**

El valor de tipo caracter está formado por **un único elemento** que puede ser un dígito, una letra o un símbolo de puntuación o de los llamados símbolos especiales, como por ejemplo, los de pregunta, exclamación, etc. Es decir, que un valor de tipo caracter está limitado por el conjunto de símbolos que se pueden escribir en una computadora. Algunos ejemplos de valores de tipo caracter son:**‘a’ , ‘B’ , ‘1’ , ‘:’ , ‘^’ , ‘ ‘ , ‘%’ , ‘F’ , ‘!’**Tené en cuenta que los valores están encerrados entre comillas simples. Esta notación la utilizamos para poder diferenciar los números de los caracteres. Por ejemplo, el **número entero 1**es un valor distinto que el caracter que representa al número 1.   
De esta forma indicamos que 1 es distinto del **caracter ‘1’.** Este tipo de dato es muy útil cuando trabajamos con valores o códigos de una sola letra o dígito.

##### **[Tipo de dato: Cadena](http://aulavirtual.codoacodo.com.ar/mod/lesson/view.php?id=43&pageid=47" \l "collapse4)**

Este valor de tipo cadena está formado por un**conjunto de caracteres**. También se suele llamar cadena de caracteres, pero para simplificar, nosotros lo llamamos **cadena**. En nuestros programas no será necesario limitar a una cantidad máxima esta cadena de caracteres, ya que forma parte de una característica física de los lenguajes de programación que, como les contamos previamente, no nos preocupa.   
Ejemplos de datos de cadena pueden ser nombres, direcciones y cualquier otro valor que esté representado por palabras o textos, como, por ejemplo: **“Juan Pérez” , “Av. Rivadavia 950” , “AUX-130” , “”.** Este último dato es lo que conocemos como una cadena vacía (sin caracteres) y la representamos con dos comillas seguidas.

##### **[Tipo de dato: Lógico](http://aulavirtual.codoacodo.com.ar/mod/lesson/view.php?id=43&pageid=47" \l "collapse5)**

Un dato lógico está formado por alguno de los siguientes valores: **VERDADERO o FALSO**. Estos datos los utilizaremos para realizar operaciones lógicas o para trabajar con condiciones. Por ahora solo lo mencionamos, ya que le dedicaremos bastante tiempo y realizaremos muchos ejercicios para afianzar este tema más adelante.   
El tipo de dato lógico también se conoce como **Boolean**, en inglés.

### Variables y constantes

Ahora es el momento de desarrollar los conceptos de programación para trabajar con los datos.

Para eso, recordemos que los datos se almacenan en la memoria, por lo tanto, habrá un espacio dedicado a guardar cada dato.

Seguramente estarás pensando que, como todo algoritmo tendrá varios datos, habrá en la memoria varios espacios destinados al almacenamiento de todos los datos. Evidentemente deberá haber alguna forma de no confundir estos espacios y poder trabajar con los datos correctamente. Para evitar esas confusiones utilizaremos un nombre único para ese espacio de memoria a lo largo de todo el algoritmo.

De esta manera podemos definir lo que es una **variable:**

Una **variable**es un espacio en memoria, asociado a un nombre lógico y a un valor, con un tipo de dato particular.

### 1. Variables

Suele ser común relacionar una **variable** con una caja en donde guardar un valor de un tipo de dato. La caja debe ser lo suficientemente espaciosa para poder contener el valor. Y la **memoria**, sería un gran fichero en donde guardar elementos con nombre para poder identificarlos.

Gráficamente, podemos imaginarlo así:

Una variable tendrá un ***nombre*** y un ***tipo de dato***asociado, es decir, que tendrá un nombre y un valor. Así, tendrás que prestar mucha atención porque es muy fácil confundir el nombre de la variable con su valor. Pero...

¿Cómo sabemos el espacio que ocupará el valor?

Para saber el espacio que ocupará el valor, contamos con el Tipo de Dato.

**Las variables tienen este nombre porque el valor puede modificarse (variar) durante la ejecución del algoritmo.**

##### **Variables en seudocódigo**

Seguramente te estarás preguntando cómo definir una variable en un algoritmo.

Una variable se define declarando su nombre y el tipo de dato. De forma genérica, lo definimos como NOMBRE:TIPO DE DATO.

Por ejemplo, si queremos guardar la edad y el nombre de una persona, podemos definir dos variables de la siguiente manera:

*SEUDOCÓDIGO*

EDAD: ENTERO  
NOMBRE: CADENA

### 2. Concepto de Constante

Una Constante, al igual que una Variable, tiene un nombre asociado a un valor de un tipo de dato.

Ahora bien, ¿cuál es la diferencia?

Como su nombre lo indica, una **Constante** tiene el mismo valor durante toda la ejecución del algoritmo, no se puede cambiar.

Ejemplos de constantes son las constantes matemáticas, por ejemplo, PI, e, etc.  Además, cada programa podrá tener sus propias constantes para ese problema.

##### **Constantes en seudocódigo**

Para definir una Constante tendremos que especificar su nombre y su valor. El tipo de dato no es necesario indicarlo ya que será deducido de acuerdo al valor de la constante.

De forma genérica lo definimos como **NOMBRE=VALOR**

Veamos algunos ejemplos de definiciones de Constantes:

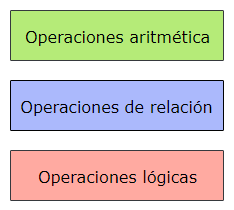
*SEUDOCÓDIGO*

PI = 3,14  
IVA = 0,21  
VELOCIDAD = 200  
SALIDA = 'X'

### Operaciones

Hasta el momento, hemos podido especificar un problema algorítmico, y además conocer los tipos de datos con los que es posible definir los Datos de Entrada y los Datos de Salida.

Continuando con nuestros temas presentamos las operaciones que podemos realizar con cada tipo de dato. Así, agruparemos las operaciones en **tres categorías**de acuerdo al tipo de operaciones. De esta manera tenemos:



### 1. Operaciones aritméticas

Las operaciones aritméticas son las que se aplican a los datos numéricos.

| **Operando** | **Operación** | **Se aplica a.....** |
| --- | --- | --- |
| **+** | Suma | ENTERO Y REAL |
| **-** | Resta | ENTERO Y REAL |
| **\*** | Multiplicación | ENTERO Y REAL |
| **/** | División | ENTERO Y REAL |
| **DIV** | División entera | ENTERO |
| **RESTO** | Resto de la División entera | ENTERO |

Las operaciones **+ - \***son las operaciones aritméticas que ya conocés.

La **División**, cuyo operador es el signo**/**se conoce como División Real, ya que devuelve el cociente real entre dos números, ya sean Enteros o Reales.    
  
En cambio, la **División Entera**, cuyo operador es **DIV**devuelve el cociente entero entre dos números también enteros.   
  
La operación **RESTO**es el módulo o resto de la división entre dos números enteros.

Veamos un ejemplo para aclarar estas operaciones:

7 / 2 = 3,5

7 DIV 2 = 3

7 RESTO 2 = 1

Las operaciones DIV y RESTO se aplican solamente a valores enteros.

### 2. Operaciones de relación

Estos operadores se utilizan para establecer relaciones de orden entre los datos. Son muy prácticas cuando queremos saber, por ejemplo, si dos variables tienen el mismo valor, o si alguna es mayor que otra.

Pensemos que le estamos pidiendo datos a un usuario y le decimos que ingrese un **\***(asterisco) cuando quiera finalizar. Vamos a necesitar el **operador =** para comparar cada valor que ingresa el usuario con el **carácter \*** para poder determinar que el usuario ya no tiene datos para ingresar, de la siguiente manera:

DATO = '\*'

Del mismo modo, podemos verificar que una nota tenga un valor mayor o igual a 4 para saber si un alumno está aprobado. La operación la escribimos como:

NOTA >= 4

El conjunto de operaciones de relación se presentan en la siguiente tabla: 

| **Operando** | **Operación** | **Se aplica a...** |
| --- | --- | --- |
| **<** | Menor | ENTERO, REAL, CARACTE0 |
| **<=** | Menor o igual | ENTERO, REAL, CARACTER |
| **>** | Mayor | ENTERO, REAL, CARACTER |
| **>=** | Mayor o igual | ENTERO, REAL, CARACTER |
| **=** | Igual | ENTERO, REAL, CARACTER, CADENA, LÓGICO |
| **≠** | Distinto | ENTERO, REAL, CARACTER, CADENA, LÓGICO |

Veamos algunos ejemplos para aclarar estas operaciones:

*SEUDOCÓDIGO*

4 < 90  
'C' = 'C'  
VERDADERO = VERDADERO  
'120' ≠ '1'

### 3. Operaciones lógicas

Son aquellas que se aplican a los valores booleanos y representan las operaciones de la lógica proposicional. Estas operaciones ya las conoce porque se utilizan en la vida diaria.

Veamos algunos ejemplos:

Supongamos que tenemos ganas de salir un sábado a la noche, entonces podremos decir: *"Vamos a ir al cine****o****a una pizzería."* Esta afirmación indica que vamos a ir a ver una película o iremos a comer a una pizzería, es decir, que realizaremos alguna de las dos cosas.

En cambio, si decimos: *"Vamos a ir al cine****y****a una pizzería"*. Queremos indicar que vamos a hacer las dos cosas, veremos una película y también comeremos en una pizzería.

Con este ejemplo habrán notado la diferencia entre el **“o”** y el **“y”.**Estas son las operaciones lógicas de disyunción y conjunción, y que representamos en la siguiente tabla:

| **Operando** | **Operación** | **Se aplica a...** |
| --- | --- | --- |
| **Y** | Conjunción | LÓGICO |
| **O** | Disyunción | LÓGICO |
| **NO** | Negación | LÓGICO |

##### **Presentamos a continuación las operaciones y cuáles son las tablas de verdad correspondientes:**

| **p** | **q** | **p Y q** |
| --- | --- | --- |
| VERDADERO | VERDADERO | **VERDADERO** |
| VERDADERO | FALSO | **FALSO** |
| FALSO | VERDADERO | **FALSO** |
| FALSE | FALSO | **FALSO** |

| **p** | **q** | **p O q** |
| --- | --- | --- |
| VERDADERO | VERDADERO | **VERDADERO** |
| VERDADERO | FALSO | **VERDADERO** |
| FALSO | VERDADERO | **VERDADERO** |
| FALSO | FALSO | **FALSO** |

| **p** | **NO p** |
| --- | --- |
| VERDADERO | **FALSO** |
| FALSO | **VERDADERO** |
|  |  |