

### 3. Bases del discurso: Tipos de datos

La palabra **dato** procede de un antiguo participio pasivo del verbo latino *dare*, o en castellano, dar; por lo que como es lógico, *datum* se podría transcribir como lo dado o lo que se da, por lo que podremos definir dato como **la información que se da sobre un suceso concreto y que permite conocer el conocimiento exacto del suceso o al menos deducir sus consecuencias.**

En el ámbito de la Informática, tiene una acepción bastante más acertada, y es: **“información dispuesta de manera adecuada para su tratamiento por una computadora”**, y no sólo eso, sino que también esta palabra ha creado nuevas expresiones dentro de ese ámbito y que hoy en día se escuchan casi cada día en la sección Tecnología de nuestro telediario como “base de datos”, “gestión de datos”, “inteligencia de datos”, “minería de datos”, “protección de datos”, y un largo etcétera, destacando entre ellos Ciencia de Datos.

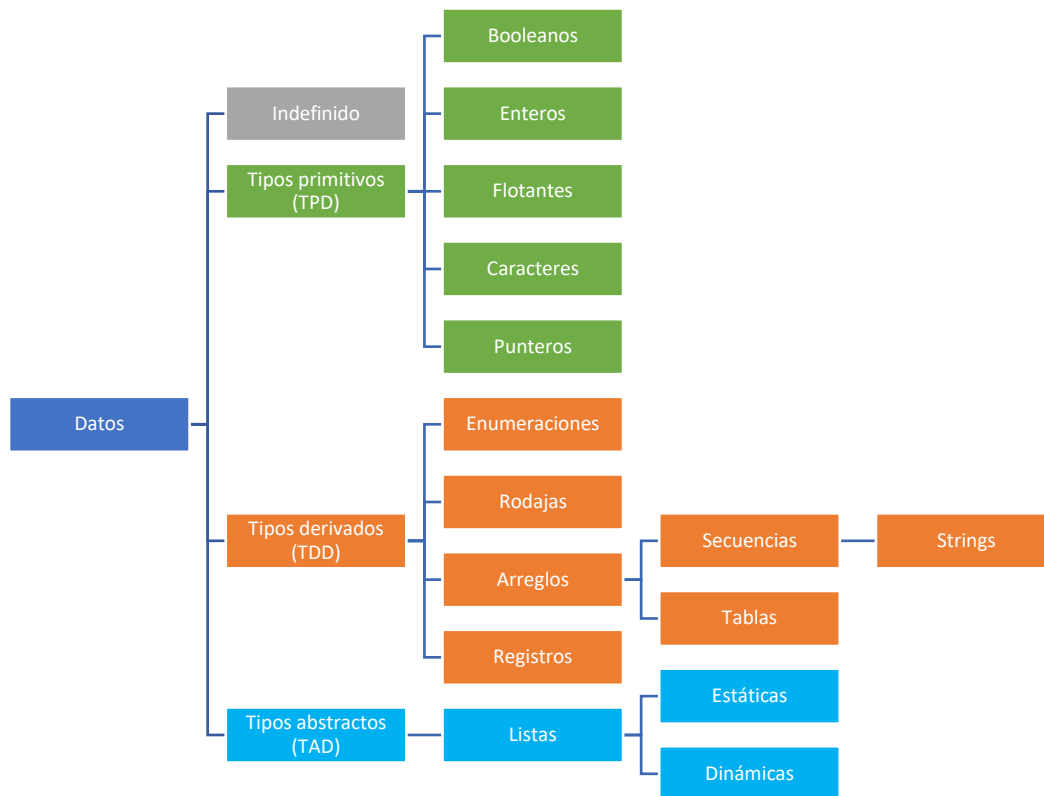
La **Ciencia de Datos** es una de las disciplinas del futuro, más bien es un **campo interdisciplinario** que utiliza diferentes disciplinas como **la estadística, la computación o la algorítmica** para tratar con los distintos datos que se nos pueden presentar en un escenario determinado. Para ello, es muy importante **analizar con detenimiento los datos y estructurarlos**, es decir, clasificarlos.

En la disciplina informática, **el tipo de dato se suele definir como el atributo que indica al programador sobre la clase de datos que se va a manejar**. Esto incluye imponer **restricciones** en los datos, como qué **valores** pueden tomar y qué **operaciones** se pueden realizar.

Conforme a la clasificación de los datos existen tres clases principales:

- **Tipos primitivos de datos (TPD)**: son datos muy básicos y que forman en sí una **unidad de información atómica**, una unidad mínima de almacenamiento. En esta clasificación pertenecen los **caracteres alfanuméricos**, los **booleanos** (valores de verdad), los **nº enteros**, los **nº reales** y los **punteros**.
- **Tipos derivados de datos (TDD)**: son datos **creados a partir de la composición de tipos primitivos y estructurados de una forma** determinada. Destacan las **secuencias** o vectores, las **tablas** o matrices, las **enumeraciones**, las **rodajas** y los **registros**. La disposición de los datos primitivos que forman parte del tipo derivado se denomina estructura de dato.
- **Tipos abstractos de datos (TAD)**: son **datos definidos mediante un modelo matemático y compuestos por un conjunto de datos** más simples **y una serie de operaciones** que se pueden realizar con ellos. **El comportamiento de estos datos no lo define el programador, sino el usuario**. Un buen ejemplo de ello son las **listas**. La implementación de este tipo de datos en un lenguaje de programación se realiza mediante una estructura de datos.

En el contexto de los lenguajes de programación **débilmente tipados**, se podría hablar del **tipo de dato indefinido**. Este tipo de dato surge puesto en este tipo de lenguajes no se declaran previamente las variables, y se declaran inmediatamente tras su primera aparición en el código del programa, pero se tipan una vez se les haya conferido un valor determinado. Si operamos con este tipo de datos, surgirá un error de compilación.



### 3.1 Tipos Primitivos de Datos

Los **tipos primitivos de datos** se pueden clasificar en:

1. **Booleanos:** nombrados así en honor a George Boole, primer matemático que materializó las variables lógicas (que toman sólo el valor de verdadero o falso) a los circuitos eléctricos, **asignando el valor 1 (TRUE) como encendido, y el valor 0 (FALSE) como apagado.** No existen bombillas encendidas y apagadas a la vez ni enunciados atómicos verdaderos y falsos a la vez. Es un **valor discreto y binario que indica la veracidad de una propiedad.** Es un sí o un no, un ying o un yang.
2. **Enteros:** se denominan así porque **representan cantidades exactas.** Son los números **utilizados para contar,** por ejemplo para indicar los pisos que tiene un edificio, incluso los negativos, indicándote que se trata de uno subterráneo. Sería un sinsentido tener un rebaño de 6.22 ovejas o un andén 9 y  $\frac{3}{4}$  salvo que vivas en el universo literario de J.K. Rowling. Aunque el concepto de número no existió hasta las primeras civilizaciones, se ha demostrado que varios animales saben contar, lo que indica que **el conteo es propiedad intrínseca de la naturaleza.** Y por si esto fuera poco, en el año 5.000 a.C. en Sumeria se creó la primera computadora del mundo, un ábaco capaz de realizar todo tipo de operaciones, incluso con números negativos, basado eso sí en el conteo de cálculos o piedras pequeñas. Se le considera de hecho el predecesor de la calculadora.

3. **Flotantes:** son aquellos números que **no representan cantidades exactas, sino fracciones, partes o porciones** de una unidad como una tarta, una pizza o un **porcentaje** de una muestra poblacional, o una **aproximación de un valor irracional**, el cual no puede representar como fracción. **Los números enteros también están incluidos entre los números reales**, ya que ocho porciones de una pizza formarían la unidad entera, o de manera más formal, cualquier entero se puede representar como el cociente de dos reales, P.ej.  $2 = 2/1 = 4/2 = -4/-2 = 1/0.5 = -1/-0.5$ , etc. Aunque cabe aclarar que **lo que se denomina número real en Matemáticas no corresponde exactamente con lo que se define en Informática como su tipo de dato correspondiente**, puesto la memoria tiene una capacidad finita y no se pueden guardar ni números muy muy grandes ni números muy muy pequeños ni todos los decimales del número  $\pi$ , que de él hasta ahora se han calculado 62 billones, necesitamos una aproximación decimal (lo que implica que se puede representar en formato racional), es lo que se le conoce en Informática como **coma flotante**.
4. **Caracteres:** todos aquellos **caracteres alfanuméricos** que no sirven ni para contar, representar porciones, aproximar valores reales o indicar la veracidad de una propiedad son los caracteres propiamente dichos, entre los que incluyen **las letras, los signos, los símbolos y los propios dígitos** del 0 al 9 usados para escribir los números. Conjuntos de caracteres forman la composición íntegra de esta obra.
5. **Punteros:** dependiendo del manual del lenguaje de programación con el que se trabaje, a veces se le considera como un tipo de dato primitivo y otras veces no. Realmente es un **“objeto” de referencia cuyo valor se refiere a otro valor almacenado en otra parte de la memoria del ordenador utilizando su dirección**. El puntero **apunta a una ubicación** en memoria, y a **la obtención del valor almacenado en esa ubicación se la conoce como desreferenciación** del puntero. Análogamente, el número de página en el índice de este libro podría considerarse un puntero a la página correspondiente; al ir a la página con el número de página especificada en el índice habremos desreferenciado el puntero. No entraremos en mayores detalles sobre este tipo de dato en concreto, se verá más adelante en el capítulo 10.

Por último, cabe aclarar que los distintos lenguajes de programación pueden considerar más o menos tipos primitivos de datos de los que se exponen según ellos consideren como lo básico.

Por ejemplo, en C no existe como tipo primitivo los booleanos por lo que en su lugar debe usarse un Integer que tome los valores 0 y 1 en sustitución de false y true. De hecho, los lenguajes débilmente tipados no tienen en su gramática el tipo de dato primitivo “puntero”. Sin embargo, C tiene otros tipos de dato primitivo como los “unsigned character” (variable que reserva 16 bits de memoria codificados en sistema hexadecimal)

**El tipo de dato String o cadena de caracteres se considera primitivo hoy día por la mayoría de lenguajes de programación.** Sin embargo, en este libro se considerará como los antiguos lenguajes, como tipo derivado.

Otros lenguajes como Java tienen algún tipo de dato duplicado como los Float, pero cuya diferencia es el espacio de memoria que ocupan, de esta forma “short” indica los reales de capacidad 16 bits, “float” los reales de capacidad 32 bits, y “double” los reales de capacidad 64 bits. Recordemos que la unidad mínima de información del ordenador es el bit, cuyo valor sólo comprende entre 0 y 1.

A continuación, se exponen las distintas **operaciones que podemos realizar con los distintos TPDs** y cómo los indicaremos en pseudocódigo y en los distintos lenguajes de programación que se desarrollarán a lo largo de este libro.

**Con los booleanos**, podemos realizar las operaciones lógicas de **negación** (not) (cambiar de true a false y viceversa), **disyunción** (or) (devuelve true si uno de los dos operandos es true) y **conjunción** (and) (devuelve true si y solo si ambos operandos son true) Aquí se muestran las tablas lógicas de las tres operaciones, que indica el valor de la operación según los valores de las variables booleanas a, b:

a	not a
true	false
false	true

a	b	a or b
true	true	true
true	false	true
false	true	true
false	false	false

a	b	a and b
true	true	true
true	false	false
false	true	false
false	false	false

Cabe destacar de que ADA tiene una operación extra denominada **disyunción exclusiva** (xor), la cual devuelve true si uno de los dos operandos es true, pero no ambos. Se podría utilizar también en pseudocódigo. Se muestra a la derecha su tabla de verdad.

a xor b equivale a la expresión (a or b) and not (a and b)

Por otra parte, los operadores en C son radicalmente distintos a los anteriormente citados (not, or, and), que son compatibles tanto para pseudocódigo como para Pascal, ADA y Python.

- not a  $\rightarrow$  ! a
- a or b  $\rightarrow$  a || b
- a and b  $\rightarrow$  a && b

a	b	a xor b
true	true	false
true	false	true
false	true	true
false	false	false

**El tipo de dato booleano sólo puede tomar dos valores: true y false**, que a veces, como en C, ya que en C no existe el tipo booleano, puede escribirse como 1 y 0, respectivamente.

**Con los enteros**, destacan dos tipos de operaciones. El primer tipo se corresponde con las **operaciones aritméticas**, y estas son **la suma (+), la resta (-), el producto (\*), la división entera (/)** (que devuelve el cociente de la división), **el módulo (%)** (que devuelve el resto de la división\*), **la exponenciación (^) y el valor absoluto (abs)**. Utilizaremos esta notación para expresar las siguientes operaciones en el pseudocódigo.

En Pascal y en C, la exponenciación y el valor absoluto no están incluidos en la gramática del lenguaje. A diferencia de C, Pascal utiliza como operadores para la división entera y el módulo (div, mod), respectivamente.

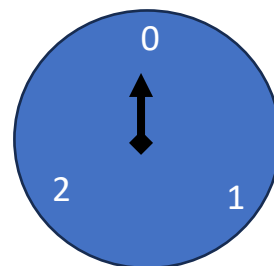
Python sí tiene exponenciación, y para ello utiliza el operador \*\*, al igual que en ADA, pero no operador de valor absoluto, cosa que ADA sí tiene. Por otra parte, la división entera la indica con una doble barra (/), que es la que devuelve sólo el cociente de la división. Si utilizáramos la barra simple en Python (/), se convertiría en una división común, lo que hará Python será obtener el resultado exacto de la división, decimales incluidos.

Por otra parte, **ADA realiza una distinción entre el resto de una división y el módulo de un número en una base**, relacionado esto último con la **aritmética modular**, bajo los operadores rem y mod, respectivamente. **Estas operaciones son idénticas si los operandos tienen el mismo signo\***, pero si estos tuvieran distinto signo, se distingue mejor la diferencia entre resto y módulo.

Por ejemplo, para  $x \text{ rem } 3$ , los restos de estas divisiones son los siguientes: Recordemos que una división entera  $a / b$  debe cumplir la propiedad  $a = b * q + r$ , siendo  $a$  el dividendo,  $b$  el divisor,  $q$  el cociente y  $r$  el resto,  $r < b$ .

x	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
x rem 3	-1	0	-2	-1	0	-2	-1	0	1	2	0	1	2	0	1

Y para  $x \text{ mod } 3$ , se podría asimilar dicha operación como si fuera un reloj de tres horas; 0, 1 y 2 y mover la aguja empezando desde el cero  $x$  veces hacia la derecha si el número es positivo y  $x$  veces hacia la izquierda si el número es negativo. La hora que marque la aguja será el resultado de la operación.



x	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
x mod 3	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1

Luego, **las operaciones “a rem b” y “a mod b” dan el mismo resultado cuando “a, b” tienen el mismo signo o “a” sea múltiplo de “b”**. Por ello, cuando nos pidan por ejemplo un número par, la condición que debe cumplir puede ser perfectamente  $a \text{ rem } 2 = 0$  o  $a \text{ mod } 2 = 0$ , o en otros lenguajes,  $a \% 2 = 0$

Cabe destacar que **no todas las operaciones aritméticas que se nos ocurran** (p.ej. raíz cuadrada, raíz quinta, logaritmo, seno e incluso exponenciación en algunos lenguajes etc.) **están disponibles en los lenguajes de programación**, por lo que para utilizar una operación no existente en dichos lenguajes **debemos o bien utilizar un algoritmo** que defina dicha

operación, **o bien instalar el paquete de recursos en el cual esa operación está incluida como una función** (p.ej. sqrt(), exp(), log(), sin(), etc.), suelen tener el nombre de Maths o Numerics. Se especificará siempre en el enunciado si hay que importar alguna librería y cómo se hace según el lenguaje de programación que se use.

El segundo tipo de operaciones corresponde con las **operaciones relacionales**. En pseudocódigo y en Pascal, estas son: **mayor que** (>), **mayor o igual que** (>=), **menor que** (<), **menor o igual que** (<=), **igual a** (=) y **distinto de** (<>)

En el resto de lenguajes, los cuatro primeros operadores son idénticos, para la operación “distinto de”, su operador en ADA es (/=), mientras que en Python y C es (!=); y para la operación “igual a”, su operador en Python y en C es (==)

Esto es debido a que la asignación en estos dos lenguajes utiliza como operador (=), a pesar de encontrarse con la oposición de varios programadores, ya que cuando se asigna un valor a una variable, no se iguala la variable a dicho valor, sino que guarda en una dirección de memoria con el nombre de esa variable ese valor.

Luego, hay que tener cuidado de no equivocarse con la operación “igual a”, la cual devuelve siempre un valor booleano, y la asignación. Si la comparación de igualdad se expresa mediante el operador (=), entonces el operador de asignación será (:=), es lo que sucede con Pascal y ADA. Si pasa al contrario como en Python y C, en el que el operador (=) es el de la asignación, entonces, la operación “igual a” tendrá como símbolo (==)

Matemáticamente, el conjunto de los números enteros es infinito, abarca desde el  $-\infty$  hasta el  $+\infty$ , pero en la Informática, **el tipo de dato entero está acotado entre dos números**: uno muy muy pequeño y otro muy muy grande. **Esto es debido a que la memoria tiene una capacidad finita** y no se pueden almacenar ni números muy muy muy pequeños ni números muy muy muy grandes.

**Estos extremos están por defecto etiquetados en los lenguajes de programación.** Por ejemplo, para Pascal, [MinInt, MaxInt]; para Ada, [Integer'First, Integer'Last]; para Python, [sys.minint, sys.maxint] (debe haberse instalado previamente la librería sys): y para C, [INT\_MIN, INT\_MAX] (debe haberse instalado previamente la librería limits.h)

Dependiendo de la **arquitectura del procesador** (es decir, el número máximo de bits que puedo almacenar en memoria para el valor de una variable), el valor de estos extremos es para MinInt =  $-2^{31}$  o  $-2^{63}$  y para MaxInt =  $2^{31} - 1$  o  $2^{63} - 1$ , según la arquitectura de 32 bits y la de 64 bits, respectivamente.

Como dato adicional, no viene mal saber que **se pueden utilizar barras bajas ( \_ ) como separador de dígitos**, realizando la misma función que el punto que separa millares en español, o la coma en inglés. En lugar de escribir 1,234,567 podemos escribir 1234567 o 1\_234\_567.

**Con los flotantes**, o vulgarmente dicho, los reales, **podemos usar todas las operaciones que se habían definido anteriormente para los enteros, a excepción de la división entera y el módulo, que se sustituyen por la división normal (/)**. Esta división da el resultado exacto con

precisión decimal de dividir a entre b, devolverá un número real k que multiplicado por b diera como resultado a.

Conforme a su representación, surge el mismo problema de memoria, pero esta vez **deben guardarse tanto los dígitos de parte entera como los dígitos de la parte decimal** de nuestro número real, y distinguirse entre sí. **Esta manera de representar reales por los valores de su parte entera y de su parte decimal se denomina como flotante**, puesto en realidad, el número se representa a través de una mantisa (el número decimal sin la coma), multiplicado por una potencia de base 10 elevada a un exponente determinado. Este exponente decide cuántas posiciones a la izquierda o al a derecha se desplaza la coma con respecto al último dígito donde termina la parte entera del número, de forma que quede sólo un dígito del 1 al 9 delante de la coma (cuidado, en inglés, nuestra coma decimal se representa mediante un punto), de **una forma parecida a la notación científica** en la calculadora.

P.ej.  $3234.5 \rightarrow 3.2345E+3$ ,  $0.00016 \rightarrow 1.6E-4$ , que sustituyen a las formas de representación matemáticas  $3.2345 * 10^3$  y  $1.6 * 10^{-4}$ , respectivamente.

Evidentemente, podemos escribir cualquier número decimal usándolo como es en su esencia, con su parte entera, su coma decimal y su parte decimal. Sin embargo, hay que tener en cuenta de que **si por alguna razón necesitáramos asignarle un valor entero a una variable flotante, es obligatorio escribir el número con la coma decimal** para indicar que su tipo de dato es el flotante y no confundirlo con un entero. Luego, en lugar de asignar a una variable el valor 5, se le asignará el valor 5.0.

Como máximo, a la hora de asignar valor a las variables flotantes, se pueden utilizar números de hasta 24 dígitos (de los cuales 7 son de la parte decimal) para sistemas de 32 bits; y números de hasta 53 dígitos (de los cuales 15 son parte decimal) para sistemas de 64 bits.

Finalmente, tenemos **los caracteres**, que abarcan desde espacios en blanco, puntos, comas, signos, símbolos, hasta las letras que comúnmente conocemos del abecedario latino, de la A a la Z tanto en minúsculas como en mayúsculas.

**Las operaciones** que se pueden realizar con caracteres **son más limitadas, y no son nada similares a las operaciones matemáticas que conocemos**, por lo que es cuestión de práctica el tener que acostumbrarse a trabajar con caracteres.

Para ello, se liga mediante un **código internacional** llamado **ASCII** cada carácter con un valor numérico determinado. Un anexo aparte mostrará algunos de los **256 caracteres ASCII los cuales están ligados a un índice cada uno**. Así por ejemplo, la letra 'A' está ligada al índice 65.

Es importante fijarse en esta relación entre carácter e índice (no memorizarla, se dará el correspondiente ASCII del carácter en el enunciado), ya que entonces, se entenderá mucho mejor la aritmética de caracteres. **A la hora de denotar caracteres, los denotaremos entre comillas simples**, para no confundir la variable a con el carácter 'a'

En primer lugar, viene bien saber que **se pueden utilizar operaciones relacionales en todos los lenguajes de programación. El orden de caracteres lo establece el propio código ASCII, y tiene la ventaja de ser intuitivo, puesto se sigue el orden numérico para los dígitos y el alfabético para las letras**, tanto mayúsculas como minúsculas, aunque siendo “mayores” las letras minúsculas al aparecer después de las mayúsculas.

Algunos enunciados booleanos verdaderos con estos operadores relacionales son p.ej. 'a' = 'a'; 'a' <> 'A'; 'a' > 'A'; 'A' < 'a'; 'b' < 'g'; 'z' > 'm'; '3' < '4'; '5' > '0' y otros enunciados menos intuitivos como '!' < '+'; '&' > '\$'; '"' <> "'"; '2' > '%'; '7' < '^'; '0' < 'o'; 'D' < 'c'; 'd' > 'c'; '' <> ''.

Otras operaciones que pueden aplicarse a los caracteres son: **el ordinario de un carácter** (ord()), que devuelve el índice ASCII ligado al carácter; su inverso, **el carácter de un número** (chr()), en el que dado un número dentro del rango de índices ASCII, devuelve el carácter correspondiente según el índice; las operaciones booleanas **isAlpha()**, **isDigit()**, en las que dado un carácter, indica si este es una letra del alfabeto o un dígito numérico, respectivamente; y las operaciones **upper()**, **lower()**, que transforman una letra del alfabeto a mayúscula y a minúscula, respectivamente. En algunos lenguajes y en pseudocódigo, existen dos operaciones extra válidas, **succ()** y **pred()**, en las que dado un carácter, devuelve el siguiente y el anterior carácter al dado, respectivamente.

Por ejemplo, ord('B') = 66, chr(66) = 'B'; isAlpha('B') = true; isDigit('B') = false; upper('B') = 'B'; lower('B') = 'b'; succ('B') = 'C'; pred('B') = 'A'.

Evidentemente, existen bastantes más caracteres que los dados, pero nos vamos a abstener a utilizar exclusivamente los caracteres del inglés, ya que **existen muchos lenguajes que no aceptan caracteres especiales**, es decir, caracteres de otras lenguas distinta a la inglesa. Luego, a la hora de crear nuestros programas, **debemos procurar no utilizar caracteres especiales, lo que incluye caracteres comunes del español como letras con tilde, la u con diéresis y la letra ñe**, por lo que les suprimiremos sus tildes, diéresis y virgulillas. Así, en vez de escribir á, é, í, ó, ú, ü, ñ; escribiremos a, e, i, o, u, u, nn. **Los signos de apertura de interrogación (?) y de exclamación (!) simplemente se omiten.**

Conforme a los Strings, en este libro se les considerarán como una secuencia de caracteres y no como un TPD, por lo que se impartirán en el Tema 8: Arrays.

Los punteros también son otro TPD, pero se impartirán mucho más adelante, cuando se usen en nuestros programas más sofisticados y con estructuras de datos más complejas.

**Las distintas operaciones** para los distintos TPDs **pueden combinarse entre sí siempre y cuando los dos operandos resultantes no sean de distinto tipo**. P.ej, 'A' + 3 es una operación inválida pero ord('A') + 3 sí lo sería, pues el ordinario de 'A' devuelve un resultado del mismo tipo de dato que 3, un entero. De la misma forma, 3 + 5.2 sería inválido, pero 3.0 + 5.2 sí lo sería. **También sería inválido operar con un tipo de dato que ese operador concreto no admite**, por ejemplo, expresiones como succ(true), 'd' - 'c', 3 or 4 son inválidas.

La **jerarquía de operaciones** que se sigue al operar expresiones combinadas es la siguiente:

- |   |  |
|---|--|
| 1. Paréntesis   | 5. Negación                                |
| 2. Exponenciación y funciones matemáticas                         | 6. Producto, división, módulo y conjunción |
| 3. Operadores de caracteres (y enumeraciones, rodajas)            | 7. Suma, resta y disyunción                |
| 4. Desreferenciación (punteros y accesos de arreglos o registros) | 8. Operadores relacionales                 |
|   | 9. De izquierda a derecha                  |

P.ej:	$5 * (3 + 4) = \text{ord}('B') / 2 + 2 \text{ or not } (5^2 > \text{sqrt}(\text{ord}('d')) \text{ and isDigit}(\text{chr}(50)))$
Paréntesis	$5 * 7 = \text{ord}('B') / 2 + 2 \text{ or not } (5^2 > \text{sqrt}(100) \text{ and isDigit}('2'))$
Exponenciación y funciones mat.	$5 * 7 = \text{ord}('B') / 2 + 2 \text{ or not } (25 > 10 \text{ and isDigit}('2'))$
Operadores de caracteres	$5 * 7 = 66 / 2 + 2 \text{ or not } (25 > 10 \text{ and true}) =$ $5 * 7 = 66 / 2 + 2 \text{ or not true}$ {Por preferencia del paréntesis}
Negación	$5 * 7 = 66 / 2 + 2 \text{ or false}$
Producto, división, módulo y conjunción	$35 = 33 \pm 2 \text{ or false}$ {La disyunción depende del valor de la expresión $35 = 33 + 2$ }
Suma, resta y disyunción	$35 = 35$
Operaciones relacionales	true {Resultado final}

Para finalizar, se muestran a modo de resumen los distintos operadores que hemos visto para manejar diferentes TPDs para pseudocódigo y para los lenguajes de programación con los que trabajaremos en otro anexo aparte.

## 3.2 Tipos Derivados de Datos

Los **tipos derivados de datos** se pueden clasificar en:

1. **Enumeraciones:** es un tipo de dato compuesto por varios elementos llamados **enumeradores o identificadores**, los cuales están **etiquetados**. Por ejemplo, una enumeración de los posibles mazos que pueden obtenerse en una carta de la baraja española es Mazos = (Oros, Copas, Bastos, Espadas), y la de sus posibles valores es Valores = (As, 2, 3, 4, 5, 6, 7, Sota, Caballo, Rey) Como se puede observar, **la enumeración actúa como una variable que solo puede tomar como valores un conjunto de elementos** ordenado o no, finito y determinado por el usuario, por ello, a los TDDs se les conoce también como tipos definidos por el usuario. **Esto confiere mayor seguridad a las variables**, ya que si por ejemplo, declaro Sexo como el TPD carácter, se espera que el usuario introduzca 'M' para masculino y 'F' para femenino, pero puede introducir otro carácter distinto que no nos interese. Por ello, declaramos Sexo como el TDD enumeración, para limitar el nº discreto de valores que puede tener una variable de ese tipo a M, F (ojo, aquí son etiquetas, no caracteres). Si entonces se introdujera otra etiqueta, entonces se imprimirá en pantalla un

mensaje de error avisándole al usuario de que se ha equivocado. Normalmente, las enumeraciones agrupan una serie de identificadores que comparten algo en común, p.ej. Sexo = (Masculino, Femenino), Estaciones = (Otoño, Invierno, Primavera, Verano) Además, podemos utilizar como etiquetas un número, una letra, una palabra o una serie de palabras, a nuestra libre disposición, llegando a mezclar dichos formatos de etiquetado incluso para la misma enumeración.

2. **Rodajas:** aunque sólo están disponibles como TDD ya implementado para algunos lenguajes, en este libro introduciremos este concepto a nuestro pseudocódigo ya que es una óptima herramienta para los principiantes en la programación que ayuda a simplificar el código. En otros manuales, esta herramienta no se reconoce y se opta por diseños más formales y concurrentes típicos de la mayoría de lenguajes. La rodaja es un tipo de dato que **deriva de un rango de valores perteneciente a un tipo primitivo**. El nombre rodaja sugiere como si de tener un solomillo y tomar la parte del solomillo que más nos gustara se tratara. **En este símil, el solomillo entero es el TPP**, mientras que **la porción de solomillo** que queremos servir en nuestro plato **es la rodaja**. Por ello es que también se le conoce como **subrango o subtipo**. Por ejemplo, el conjunto de los  $\mathbb{N}$  naturales es una rodaja o subconjunto del conjunto de los  $\mathbb{Z}$  enteros. Sabemos que todos los naturales son enteros, pero no todos los enteros son naturales. Por lo tanto, decimos que los naturales son un TDD rodaja del TPD entero. Así mismo, el número 3 pertenece tanto al TPD entero como al TDD natural. Las rodajas se indican bajo la notación a..b, siendo a el primer valor del subrango y b, el último. De esta forma, siendo el rango de los enteros  $\text{int} = \text{MinInt}..\text{MaxInt}$ , el subrango de los naturales se denotará como  $\text{nat} = 0..\text{MaxInt}$ .

3. **Arreglos:** es un tipo derivado que almacena **una serie de valores contiguos del mismo tipo**. Se subdividen en dos tipos principales: las **secuencias o vectores**, cuando este arreglo está organizado en una única fila como por ejemplo el vector u; y las **tablas o matrices**, cuando este arreglo está organizado en filas y en columnas como por ejemplo la matriz A. Sus nombres de variable se denotan con uno o dos parámetros de dimensión encerrados entre corchetes a su derecha: u[n] y A[m, n], siendo m el número de filas que posee el arreglo (en el caso del vector,  $m=1$ , se omite) y n el número de columnas. A la hora de asignar los valores del arreglo, estos deben ir entre corchetes y separados por comas. En el caso de las matrices, cada fila debe ir separada por punto y coma. En este ejemplo,  $u[3] = [3, 1, -1]$  y  $A[3,2] = [1, 2; 0, -1; 3, 1]$  Debajo se muestran lo que serían sus equivalentes en notación matemática:

$$u = (3 \quad 1 \quad -1); A = \begin{pmatrix} 1 & 2 \\ 0 & -1 \\ 3 & 1 \end{pmatrix}$$

4. **Registros:** es un tipo de dato derivado que almacena **una serie de valores contiguos pero que estos pueden ser de diferente tipo**. Este conjunto de valores denominados **campos forman una estructura de datos** que definen un objeto cotidiano. Son **uno de los tipos más comunes que nos podemos encontrar como representación de conceptos más complejos**. P.ej. un documento de identidad, la etiqueta de una prenda, el ticket de la compra, una carta de la baraja española, etc. Para declarar un nuevo tipo de registro, se denota escribiendo cada campo entre llaves, separados por comas. P.ej.  $\text{Carnet} = \{\text{Nombre}, \text{Apellidos}, \text{DNI},$

Letra}, y para crear un objeto de ese tipo, lo mismo pero dando ya los valores de esos campos. P.ej. `Carnet1 = {"Carmen", "Muestra Muestra", 123456789, 'A'}` Como se puede observar, las comillas son imprescindibles, puesto diferencian claramente entre lo que es el nombre del campo, y el valor y tipo de ese campo.

A continuación, se exponen las **operaciones de conjunto** que se pueden realizar con los TDDs enumeración y rodajas, así como con cualquier TPD. Trataremos más adelante y más detalladamente con los TDDs arreglos y registro en sus temas, el 8 y el 10, respectivamente.

Dados los TDDs enumeración `Días = (L, M, X, J, V, S, D)` y rodaja `Alfabeto = 'A'..'Z'` (compuesta por los caracteres ASCII que pertenecen al alfabeto y son mayúsculos), las operaciones que se pueden realizar son las siguientes:

- **Operadores relacionales:** de la misma forma que `'A' < 'F'` (al aparecer A antes que F en el alfabeto y en la lista de caracteres ASCII), y `'W' > 'E'`, con las enumeraciones podemos hacer lo mismo, por lo que `L < X`, `V > M`, `S <> D`, son enunciados correctos. Aunque cabe añadir que no en todas las enumeraciones tiene sentido hablar de orden, p.ej. en la enum. Mazos anterior tiene lógica para el computador enunciados como `Oros < Espadas` o `Espadas > Bastos`, porque el orden con el que hemos declarado esta enumeración es `Mazos = (Oros, Copas, Bastos, Espadas)`, pero el orden de mazos en realidad no tiene ningún fin práctico. En el caso de `Días`, tiene su fin porque sigue el orden de los días de la semana, empezando por el Lunes.
- **Extremos:** se realiza bajo las operaciones **first()** y **last()**, que devuelven respectivamente el primer y último valor del conjunto de valores que forma el tipo de dato. Por ejemplo, `first(int) = MinInt`, `first(nat) = 0`, `first(char) = 'A'`, `first(Días) = L`, `first(Alfabeto) = 'A'`, `last(Días) = D`, `last(Alfabeto) = 'Z'`, `last(Mazos) = Espadas`.
- **Posicionamiento:** la operación **pos()** devuelve la posición relativa de un valor de la enumeración o de la rodaja en el conjunto que lo define. Por ejemplo, `pos(L) = 1`, `pos(S) = 6`, `pos(V) = 5`, `pos(Bastos) = 3`, `pos('L') = 12`. Por otra parte, su operación inversa es **val()**. Así para el tipo de dato `Día`, `val(1) = L`, `val(6) = S`, `val(3) = X`; para el tipo de dato `Alfabeto`, `val(12) = 'L'`, `val(5) = 'E'`.
- **Anterior y posterior:** se utilizan las operaciones **pred()** y **succ()**, que devuelven respectivamente el anterior y el posterior de un valor. P.ej. `pred(0) = -1`, `succ(0) = 1`, `pred('C') = 'B'`, `succ('P') = 'Q'`, `pred(S) = V`, `succ(X) = J`, `pred(Bastos) = Copas`, `succ(Oros) = Copas`. Hay que tener **especial cuidado con no introducir como operando un extremo del tipo**. Para cualquier tipo T, las expresiones `pred(first(T))` y `succ(last(T))` darán lugar a un error de desbordamiento. De este modo, expresiones como `pred(L)` o `succ(D)` son ilegales.

- **Pertenencia:** El operando (in) recibe como operandos un valor y el nombre de un tipo de dato y **devuelve un booleano que indica si ese dato pertenece a dicho tipo de dato**. P.ej. estos enunciados que utilizan operador de pertenencia son verdaderos: 3 in nat, 3 in int, -1 not in nat, -1 in int, ‘&’ in char, ‘G’ in char, ‘&’ not in Alfabeto, ‘G’ in Alfabeto, Bastos in Mazos, Corazones not in Mazos, J in Días, W not in Días.

Las principales diferencias entre estos cuatro TDDs radican en si se permiten que los tipos de dato que componen estos sean distintos o no y en si almacenan sólo un valor de su rango (valores discretos) o más de un valor (valores contiguos).

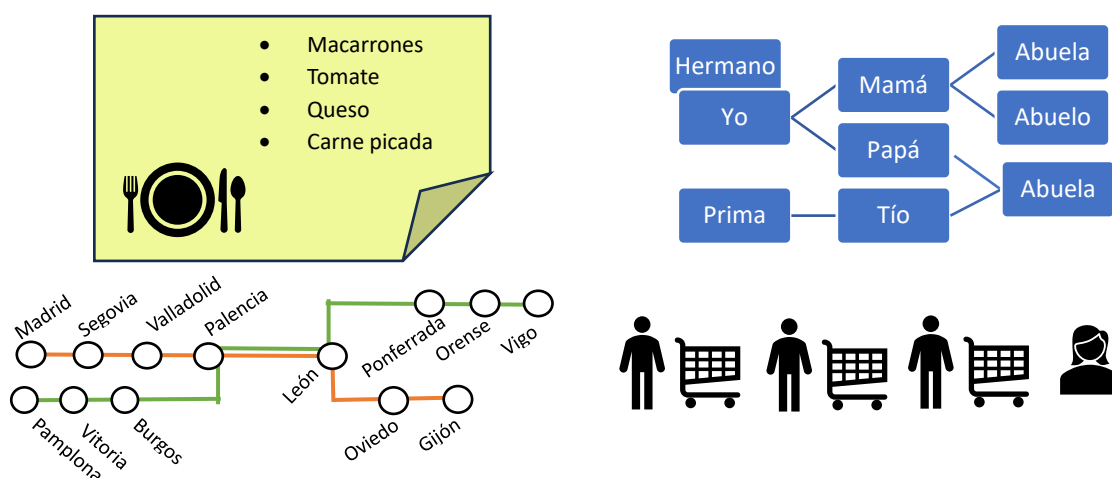
	Valores discretos	Valores contiguos
Valores del mismo tipo	Rodajas	Arreglos
Valores de distinto tipo	Enumeraciones	Registros

### 3.3 Tipos Abstractos de Datos

El resto de **estructuras de datos definidas por modelos matemáticos**, siendo estos **definidos a la vez por el punto de vista del usuario** y cómo operan con dichas estructuras, son los denominados **TADs**.

No vamos a desarrollar este tipo de datos en este curso ya que entonces complicaríamos bastante el temario, y este curso se supone que debe introducirnos al mundo de la programación.

Simplemente se deben dar a conocer este tipo de datos para hacer la siguiente objeción: **“Toda forma de organización de los datos que al ser humano se le ocurra es inherentemente un tipo abstracto de dato”** Como ejemplos, se muestran aquí los más comunes, una lista de la compra, un árbol genealógico, un plano de líneas de transporte, la fila del supermercado, etc.



Nos centraremos únicamente y de forma superficial en las **listas**, al ser la estructura más parecida a la secuencia, en el tema 11.