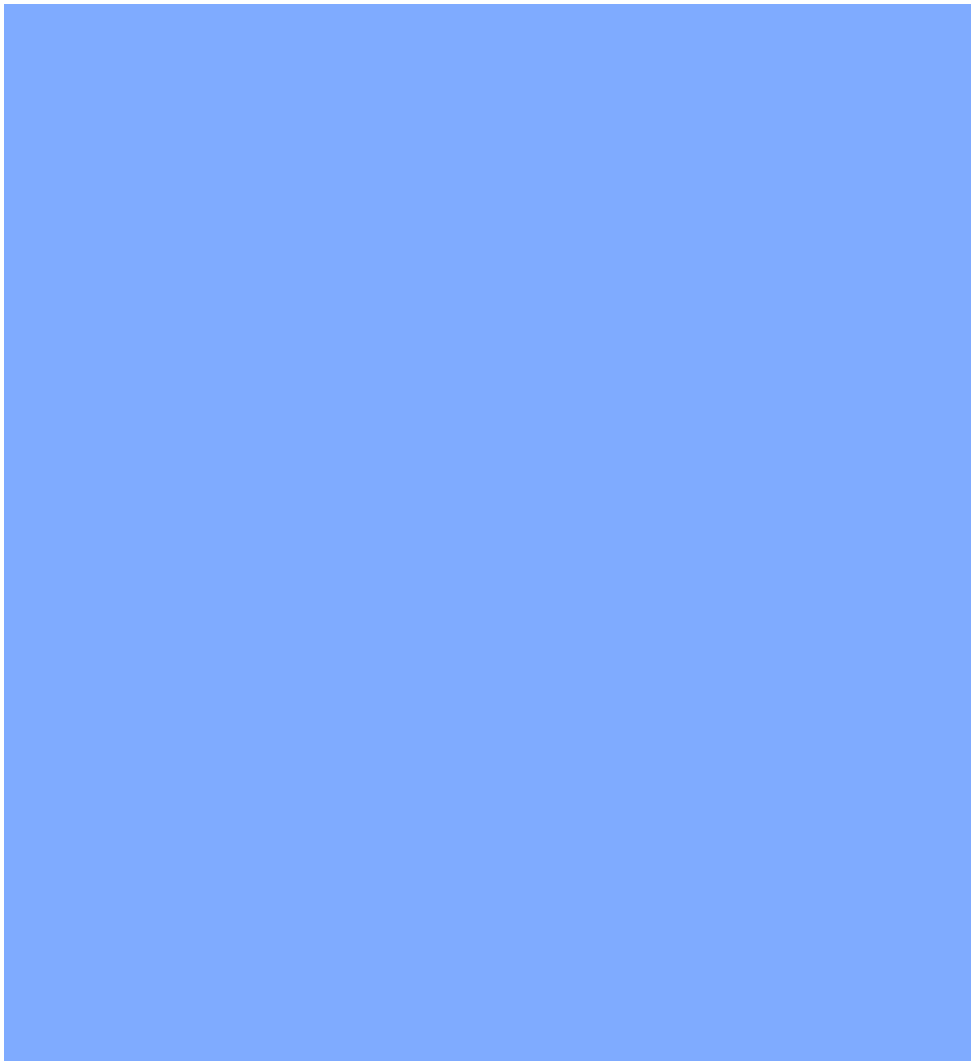
EDEM - Tech Knowledge



Introducción a la programación II

Martín San José de Vicente
Director de Contenidos de Imagina Formación

OBJETIVO

En estas diapositivas hablaremos de qué son los tipos de datos y cuáles son los más comunes dentro de todos los lenguajes de programación.

También hablaremos de qué son los espacios de memoria y cómo podemos almacenar en éstos valores a través de variables y constantes.

Por último destacaremos la importancia de una buena organización de código y de la necesidad intrínseca del desarrollo de reutilización.

ÍNDICE

- Tipos de datos comunes entre lenguajes
- Los espacios de memoria ¿Tengo que controlarlos?
- Variables y constantes
- Módulos, paquetes y librerías. ¿Pero cómo me organizo?
- Reutilización de código: Procedimientos, funciones y métodos

Tipos de datos comunes entre lenguajes

Picar código, tirar líneas, desarrollar...

¿Realmente en qué consiste?

> Tipos de datos comunes entre lenguajes

¿Qué son los tipos de datos?

En programación, como en la vida misma, almacenamos y tratamos datos.

Estos datos pueden ser representados de distintas formas: textos o números por ejemplo.

En todos los lenguajes de programación contamos con algunos tipos de datos que son comunes a todos y que debemos conocer previamente para después ver cómo se implementan sintácticamente en cada uno de ellos.

> Tipos de datos comunes entre lenguajes

¿Qué son los tipos de datos?

Los datos con los que trabajamos en los lenguajes de programación son **representados en lenguaje máquina** de formas que no llegaríamos a entender.

Es por ello que en muchos lenguajes es necesario, y en otros no tanto, decir el tipo de dato con el que estamos trabajando.

> Tipos de datos comunes entre lenguajes

¿Qué son los tipos de datos?

Dependiendo del tipo de dato con el que trabajemos, podremos realizar unas operaciones u otras sobre éstos.

Por ejemplo:
> Podemos sumar números enteros, pero no concatenarlos.
> Podemos concatenar cadenas de caracteres, pero lo que no podríamos hacer es sumarlas.

> Tipos de datos comunes entre lenguajes

¿Qué son los tipos de datos?

Prácticamente todos los lenguajes hacen uso de los mismos tipos de datos a los que llamamos tipos de datos primitivos.

Digamos que se han estandarizado una serie de tipos de datos y todos los lenguajes los comparten.

> Tipos de datos comunes entre lenguajes

¿Qué son los tipos de datos?

Vamos a enumerar algunos de los tipos de datos **primitivos** y para qué se usan.

Así, en el siguiente módulo, puedas ver rápidamente cómo usarlos en el lenguaje de programación Python.

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Cadena de Caracteres**:
  - Comunmente llamadas **String**
  - Datos que son de tipo **texto**
  - *Por ejemplo: el nombre de un usuario.*
- **Números enteros**
  - Comunmente llamados **Int** o **Integer**
  - Datos que son números enteros. Es decir que no tienen decimales.
  - Pueden ser **negativos o positivos**
  - *Por ejemplo: la edad de un usuario.*

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Números decimales**
  - Comunmente llamados **Float y Double.**
  - **Float** es la forma de representar números decimales más habitual.
  - **Double** sirve para dar mayor precisión a los datos decimales, ya que tiene el doble de precisión de un **Float.**
  - El valor decimal al que alcanza **Float** es de **3e38**, mientras que un **Double** puede llegar hasta **1.7e308**.
  - Cuando no es suficiente precisión, se puede llegar a usar tipos **Long** que abarcan aún más información.
  - *Por ejemplo: La temperatura de un termómetro.*

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Valores Binarios - Verdadero o Falso**
  - Comunmente llamados **Boolean**
  - Estos valores son imprescindibles para realizar algoritmos donde existan comprobaciones, validaciones, condiciones, bucles, etc**.**
  - Sus valores sólo pueden ser **True** (verdadero) o **False** (falso).
  - El valor **True** también suele estar representado por el **1**, mientras que **False** es el 0.
  - *Por ejemplo: Saber si un usuario tiene o no tiene coche propio.*

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Fechas**
  - Comunmente llamado **Date**
  - Se usan para representar fechas
  - Permiten acceder al día, mes y año de forma independiente.
  - Además, pueden incluir un **timestamp** que es la forma en la que representamos la **hora** con sus minutos, segundos y uso horario.
  - *Por ejemplo: Para guardar la fecha de nacimiento de un usuario*.

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Listas**
  - Comunmente llamados **Arrays o Lists**
  - Sirven para almacenar listas de valores de otros tipos. Por ejemplo, una lista de textos sería un **Array** de **Strings.**
  - En muchos lenguajes, el tipo de datos que se puede almacenar en una lista es único, mientras que en otros se puede llegar a combinar.
  - Permiten leer, añadir, editar y borrar elementos de la lista.
  - *Por ejemplo: Lista de contactos de un usuario*

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Enumerados**
  - Son listas cerradas de datos en las que podemos almacenar datos de diferentes tipos.
  - Estas listas son inmutables y son muy útiles.
  - *Por ejemplo: Colores disponibles para un modelo de coche.*

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

- **Diccionarios**
  - Comunmente llamados **Dictionaries**
  - Se usan para almacenar datos de diferentes tipos a través del método **clave - valor**.
  - A través de las claves, que son únicas, se puede obtener el valor de los elementos.
  - Este tipo de datos no está presente en todos los lenguajes, pero en aquellos en los que está, otorga una flexibilidad genial al código haciendo que todo sea más sencillo.

> Tipos de datos comunes entre lenguajes

¿Qué tipos de datos hay?

De forma particular, cada lenguaje luego aporta otros tipos de datos como podrían ser los **número irreales** o las **tuplas** (conjuntos de datos ordenados del mismo o diferente tipo).

> Tipos de datos comunes entre lenguajes

Tipado Débil o Inferido

Se dice de **tipado débil** cuando el lenguaje **no exige declarar el tipo de un valor o dato**.

El tipo del valor se decidirá en lo que llamaremos **tiempo de ejecución**.

En en el siguiente módulo veremos a lo que nos referimos cuando hablamos de tiempo de ejecución. De momento, quédate que el **tipo del valor se decide cuando el programa se ejecuta**.

> Tipos de datos comunes entre lenguajes

Tipado Dinámico

Se dice de **tipado dinámico** cuando el lenguaje **no exige declarar el tipo de un valor o dato**, pero en caso de que se **declare** pasaría a ser **estricto.**

> Tipos de datos comunes entre lenguajes

Tipado Fuerte o Estricto

Se dice de **tipado fuerte** cuando el lenguaje **exige declarar el tipo de un valor o dato.**

En caso de no declarar el tipo del valor, tendríamos **errores de compilación**. Es decir, que el programa no estaría siguiendo la sintaxis y **no podría ejecutarse**.

En el siguiente módulo hablaremos del **tiempo de compilación**. De momento, quédate con que **es el proceso que se realiza inmediatamente antes de ejecutar el código y que comprueba que todo esté bien escrito**.

Los espacios de memoria. ¿Tengo que controlarlos?

Qué son los espacios de memoria y qué implicaciones tiene a la hora de desarrollar.

> Los espacios de memoria. ¿Tengo que controlarlos?

¿Qué son los espacios de memoria?

En el mundo de la informática, la **memoria** es la parte encargada de almacenar datos. Digamos que es una de las partes imprescindibles de lo que entendemos como programación.

Cuando programamos y almacenamos datos (ya sea para persistirlos o tratarlos), estamos consumiendo **espacios de memoria**. Es decir, estamos diciendo a la máquina que en un lugar de su memoria debe alojar el contenido, por ejemplo, del nombre del usuario.

> Los espacios de memoria. ¿Tengo que controlarlos?

¿Qué son los espacios de memoria?

Consecuentemente, esto querría decir que cuantos más datos almacenamos, más espacio en memoria consumimos.

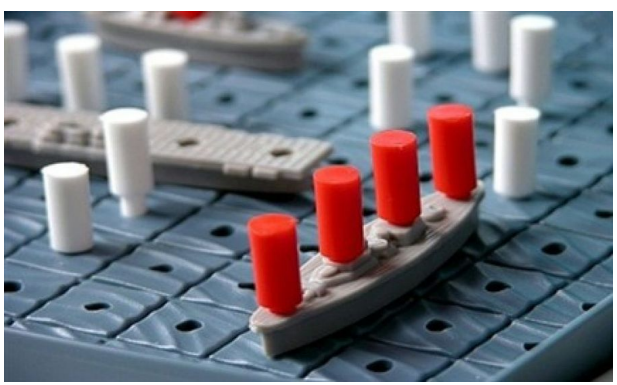
Por lo tanto, en algún momento deberíamos ser capaces de **liberar memoria** si esos datos almacenados ya nos son necesarios.

> Los espacios de memoria. ¿Tengo que controlarlos?

¿Qué son los espacios de memoria?

Imagínate la memoria como una maya parecida al de Hundir la Flota.

Básicamente, cuando programas y almacenas datos, estás colocando piezas (datos) en una de las cuadrículas disponibles.



> Los espacios de memoria. ¿Tengo que controlarlos?

¿Qué son los espacios de memoria?

Cuando llegues al límite de cuadrículas, ya no podrás colocar más piezas y esto puede suponer diversos problemas:

- Errores graves en base a una falta de espacio
- Que datos persistidos se borren aleatoriamente para introducir nuevos datos

> Los espacios de memoria. ¿Tengo que controlarlos?

¿Tengo entonces que controlar la memoria?

Al programar es imprescindible controlar la memoria usada y liberar aquella que no se esté usando.

En los lenguajes de más bajo nivel y primitivos, este control debía realizarse explícitamente, por ejemplo en **C** o en **Objective C** es necesario que lo tengamos en cuenta si buscamos un buen rendimiento.

En cambio, los lenguajes más modernos y de alto nivel, ya incorporan mecanismos automáticos que gestionan estas tareas por nosotros.

> Los espacios de memoria. ¿Tengo que controlarlos?

¿Tengo entonces que controlar la memoria?

A estos mecanismos se les suele llamar **recolectores de basura** o **Garbage Collector.**

Están presentes en lenguajes como **Java, C#, Javascript, Python, etc.**

Esto hace que el programador pueda "despreocuparse" de esta tarea en su día a día.

> Los espacios de memoria. ¿Tengo que controlarlos?

El Ciclo de vida de la Memoria

Independientemente del lenguaje de programación, el ciclo de vida de la memoria es siempre:

1. **Reserva de memoria necesaria**
    - Los tipos de datos ocupan distintas cantidades de memoria y por tanto debe reservarse el espacio necesario para que todo el contenido quepa adecuadamente.
    - Si no se reserva el espacio adecuado, se darán errores.

> Los espacios de memoria. ¿Tengo que controlarlos?

El Ciclo de vida de la Memoria

2. **Leer y Escribir en memoria**

- Una vez los datos están almacenados en memoria podemos consumirlos cuando sea necesario.

3. **Liberar la memoria cuando ya no es necesaria**

- Cuando los datos almacenados ya no son necesarios, deben ser eliminados de la memoria para así dejar espacio a nuevos datos que vayan a ser persistidos.

> Los espacios de memoria. ¿Tengo que controlarlos?

El Ciclo de vida de la Memoria

Los **dos primeros pasos del cliclo** se realizan de **forma explícita por el desarrollador de software.**

En cambio, el **tercer paso del ciclo** se realiza de **forma implícita** por el propio **lenguaje de alto nivel**.

La función del **Garbage Collector** es la de **rastrear la memoria y su utilización**, con la **intención de encontrar si alguna parte ya no es necesaria** y así **proceder a liberarla**.

> Los espacios de memoria. ¿Tengo que controlarlos?

¿Cuándo la memoria ya no es necesaria?

Encontrar de forma **automática** cuando la memoria **no es necesaria** es **indecidible** y esto implica que los **recolectores de basura** realizan **aproximaciones a la solución**.

> Los espacios de memoria. ¿Tengo que controlarlos?

La referencia

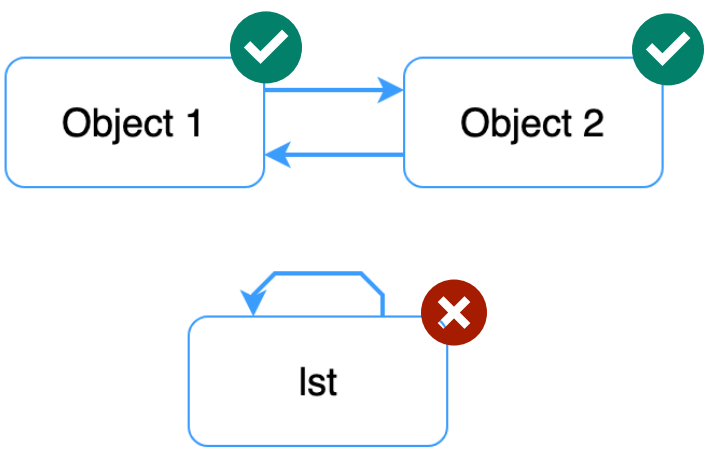
Los **recolectores de basura** buscarán aquellos espacios de memoria que están siendo **referidos desde el código** ya sea de forma **explícita o implícita** (de esto hablaremos más adelante durante el siguiente módulo).

Esto quiere decir que el gestor realizará un **conteo de referencias entre datos** y si encuentra **datos que no tienen referencias**, será eliminado (recolectado).

> Los espacios de memoria. ¿Tengo que controlarlos?

La referencia

A estos datos recolectables también se les llaman **datos inalcanzables**, ya que no se podría llegar a ellos a través de referencias de otros datos y por o tanto se entiende que se pueden eliminar.



> Los espacios de memoria. ¿Tengo que controlarlos?

Algoritmos de los recolectores

Cada lenguaje dispone de **algoritmos diferentes para sus recolectores de basura**.

Esto quiere decir que **aunque todos tienen el mismo objetivo, no todos se comportan igual**. Siendo algunos más eficientes que otros como cabe de esperar.

Algoritmos de los recolectores

Seleccionar bien el lenguaje de programación hace que la experiencia con los datos y problemas de memoria sea mejor.

Por ejemplo, uno de los más conocidos es el **recolector de basura de Java**, a pesar de ello, otros lenguajes como **Go** o **Python** son más eficientes a la hora de automatizar este proceso de limpieza de memoria.

> Los espacios de memoria. ¿Tengo que controlarlos?

Conclusiones

Concluiremos pues, que aunque hagamos uso de la memoria constantemente, existen desarrollos automáticos que permiten que el desarrollador se **abstraiga de este proceso**, lo deje en manos de algoritmos internos y se **centre en lo que realmente importa que es crear el programa**.

Aún así, siempre hay formas manuales avanzadas de gestionar la memoria cuando se dan casos problemáticos inesperados.

Variables y Constantes

Almacenar información variable o constante es una de las esencias de la programación.

> Variables y Constantes

¿Qué son las variables?

Se trata de una **manera de almacenar un valor o dato para poder hacer uso del mismo más adelante en el código**.

Se llaman variables, pues el **contenido puede variar** durante el desarrollo del programa.

Es decir, una variable sirve para **darle un nombre a un dato que almacenamos en la memoria**.

> Variables y Constantes

¿Qué son las variables?

Los nombres de las variables siguen algunas convenciones como:

- Se recomienda que **comiencen por minúscula** y luego estilo **camelCase** (intercalar minúsculas con mayúsculas al comienzo de una nueva palabra) para unir palabras
- **No pueden contener espacios**
- **No pueden empezar por números**
- **No deben contener caracteres considerados como "extraños".**
  - *Por ejemplo:* **=, +, *ñ*, ^**, *etc.*
  - Recuerda que el alfabeto que se emplea en la programación es el **inglés**.

> Variables y Constantes

¿Qué son las constantes?

Conceptualmente, es lo mismo que una variable con la particularidad de que **su información no cambia**.

Se llaman **constantes**, pues el **contenido NO puede variar** durante el desarrollo del programa.

Módulos, paquetes y librerías. ¿Pero cómo me organizo?

La modularidad y la organización del código es imprescindible.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

La necesidad de organización

Cuando creamos un programa, la organización sistemática es imprescindible.

Si el programa es pequeño, a penas se percibirán los beneficios de una buena arquitectura y estructura de software, pero **a medida que el proyecto crezca y crezca, será vital tener bien organizado** todo para poder **mantenerlo** y **hacerlo evolucionar** durante los siguientes años.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

La necesidad de organización

Los conceptos de **módulos**, **paquetes** y **librerías** son comunes a **todos los lenguajes y proyectos software**.

Cada lenguaje tiene sus particularidades a la hora de tratarlos, pero en esencia tienen el mismo objetivo: la **reutilización de código** ya sea **propio o de terceros**.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

¿Qué es un módulo?

Un módulo es una **porción de un programa que tiende a usarse repetidamente** y **se decide aislar para reutilizarse** cuando sea requerida.

Los módulos se usan muy a menudo para:

> Encapsular y modular las estructuras en los proyectos, consiguiendo así una **mayor independencia entre los bloques de código** y un **menos líneas de código por archivo**.

> Reutilizar código

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

¿Qué es un módulo?

Éstos suelen estar **estructurados u organizados en niveles**, teniendo un **módulo principal encargado de cargar el resto de módulos**.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

¿Por qué modular el software?

Nos permite tener un proyecto más estructurado y sobre todo, conseguir una **mayor velocidad de desarrollo al no tener que implementar cosas que ya se implementaron anteriormente**.

No es una buena práctica tener **archivos de código enormes**, ya que se vuelven **difíciles de mantener con el tiempo** y los **cambios** sobre éstos pueden resultar **ineficientes**.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

¿Qué es un paquete?

Un **paquete sería un conjunto de módulos** almacenados de forma **independiente** que permite **cargar aquellos módulos que vayamos a necesitar de forma aislada o en bloque**.

Sería como **una carpeta** que aloja **diferentes archivos independientes**.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

¿Qué es una librería?

Una **librería** (externa o interna) es un **conjunto de implementaciones ya desarrolladas** que nos ayudan a **simplificar tareas complejas**

Es decir, **permiten que no tengamos que implementar determinadas líneas de código nosotros.**

Va a un paso más allá que un módulo, ya que **nos ofrece nuevas funcionalidades antes inexistentes**.

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

Ventajas de usar Librerías

- Nos **ahorran tiempo de desarrollo**
- Nos **permiten realizar tareas que no podríamos realizar con facilidad**
- Nos **permiten modular el proyecto y mejorar nuestra arquitectura**
  - Esto conlleva a mejor **mantenimiento y extensibilidad futura**

> Módulos, Paquetes y Librerías. ¿Pero cómo me organizo?

Desventajas de usar Librerías

- Tenemos que estar **pendientes de versionados**
  - Puede que haya **cambios disruptivos que afecten a nuestros proyectos** si no nos percatamos.
- **Dependemos de** desarrollos de **terceros**
- **Hay que entender cómo hacer uso** de sus funcionalidades.
- **No podemos/debemos, en teoría, modificar las funcionalidades de las mismas**, por lo que podemos encontrarnos **atados** y **perder flexibilidad**.

Reutilización de código. Procedimientos, funciones y métodos

Si ya está hecho, ¿para qué duplicarlo?

La reutilización de código

En programación, el **código reutilizable** consiste en el uso de código similar en múltiples puntos de un programa.

No hablamos de copiar y pegar el mismo código de un archivo a otro, sino de establecer metodologías donde podamos usar algo ya escrito sin tener que escribirlo de nuevo.

La reutilización de código

La **reutilización de código** define la **metodología** que podemos usar para **implementar código reutilizable sin tener que volver a escribirlo en todas partes**.

Esta metodología sigue la filosofía general de programación de **Don't Repeat Yourself** (**DRY**).

> *Cada concepto debe tener una representación única e inequívoca dentro de un sistema.*

> Reutilización de código. Procedimientos, funciones y métodos

La reutilización de código

Seguir esta metodología **ayuda a los desarrolladores a mantener la estructura de sus aplicaciones.**

Evitamos pues **el desorden** y **la frustrante tarea de depuración de errores que conlleva un código mal estructurado**.

La reutilización de código

Hay muchos escenarios y **métodos** para hacerlo que iremos viendo a lo largo del siguiente módulo. De momento te enumeramos algunos de ellas:

- **Módulos**
- **Paquetes**
- **Librerías**
- **Clases** y **herencia**
- **Procedimientos**
- Las **funciones** y **métodos**

> Reutilización de código. Procedimientos, funciones y métodos

¿Qué es un procedimiento?

En programación, un procedimiento es un **módulo de código independiente que cumple una tarea concreta** y **se hace referencia dentro de otro bloque de código**.

La **objetivo principal de un procedimiento** es **ofrecer un único punto de referencia para alguna tarea pequeña** que el desarrollador pueda activar.

Un procedimiento también puede denominarse **función**, **subrutina**, **rutina**, **método** o **subprograma**.

> Reutilización de código. Procedimientos, funciones y métodos

¿Qué una función o un método?

Como se comenta en la diapositiva anterior, una **función o método** es un **procedimiento**.

Se suele distinguir entre **funciones y métodos**. Se usa la palabra **método para referirse a una función dentro del ámbito de una clase**.

Hablaremos en el siguiente módulo de lo que es una clase, pero de momento te puedes quedar con que es una **estructura típica de los lenguajes orientados a objetos** que permite personalizar y reutilizar bloques de código a través de **propiedades y métodos**.