Práctica 1

Ejercicio 1: Los lenguajes de programación más representativos son:

1951 - 1955: Lenguajes tipo assembly

Programación realizada por una sola persona. Se utilizaba para aplicaciones científicas.

1956 - 1960: FORTRAN, ALGOL 58, ALGOL 60, LISP

Hardware extremadamente caro. Estas herramientas resolvían problemas científicos numéricos (problemas que involucraban cálculos complejos en relativamente pocos datos y simples). FORTRAN introdujo la modularidad a través de sistemas desarrollados y subprogramas compilados y posible intercambio de datos entre módulos a través de un entorno global. ALGOL 60 introdujo la noción de bloque estructura y procedimientos recursivos.

ALGOL 60, no sobrevivió, pero sigue estando presente en sus idiomas herederos, que se encuentran entre los idiomas más utilizados en la práctica.

LISP se basó en la teoría de funciones recursivas y cálculo lambda, y dio fundamento a una nueva clase de lenguajes llamados lenguajes funcionales (o aplicativos).

LISP puro esta desligado de los conceptos de Von Neumann de variables modificables, asignación sentencias, sentencias goto, etc. Los programas LISP son exactamente como estructuras de datos LISP generales y, por lo tanto, el intérprete LISP se puede especificar en LISP de una manera bastante simple. Tiene énfasis en la computación simbólica

1961 - 1965: COBOL, ALGOL 60, SNOBOL, JOVIAL

Se incorpora la resolución de problemas de procesamiento de datos comerciales, es decir, problemas que involucra cálculos simples en grandes cantidades de datos. La herramienta que comenzó haciendo esto fue COBOL Estos idiomas se encuentran entre los principales logros en toda la historia de la informática, porque pudieron demostrar que la idea de un lenguaje de alto nivel era una idea técnicamente sólida y económicamente viable. COBOL a su vez introdujo archivos y descripciones de datos, y una noción muy preliminar de programación en lenguaje casi natural.

SNOBOL4 es un lenguaje que proporciona facilidades para la manipulación de cadenas y búsqueda de patrones. El estilo de programación que soporta es altamente declarativo. Tiene énfasis en la computación simbólica.

ALGOL 60 proporcionaba estructura de bloques y procedimientos recursivos.

1966 - 1970: APL, FORTRAN 66, BASIC, PL/I, SIMULA 67, ALGOL-W

APL admite un estilo de programación funcional. Tiene un conjunto muy grande de operadores, especialmente en arreglos, libera al programador de utilizando manipulaciones de matrices iterativas elemento por elemento de nivel inferior.

LISP, APL y SNOBOL4 son grandes consumidores de recursos de máquina (tiempo y memoria). Todos ellos requieren una gestión de recursos muy dinámica que sea difícil de hacer eficientemente en máquinas convencionales. Sin embargo, estos lenguajes tuvieron mucho éxito en áreas de aplicación especializadas.

PL/I fue diseñado a mediados de la década de 1960 con un objetivo ambicioso: integrar los conceptos más fructíferos y originales de los lenguajes anteriores en una forma de propósito general, un lenguaje de programación universal. Además de tomar conceptos de FORTRAN (como módulos separados), ALGOL 60 (estructura de bloques y procedimientos recursivos), COBOL (facilidades de descripción de datos) y LISP (estructuras de datos dinámicas), PL/I introdujo características menos consolidadas, como manejo de excepciones y algunas funciones multitarea primitivas.

FORTRAN, ALGOL 60 y APL están mayormente orientados hacia la resolución de problemas numéricos.

PL/I probablemente fue muy adelantado a su época. Incorporaba diferentes características, pero no las integraba de una manera uniforme. Además, las características más nuevas necesitaban más investigación y experimentación antes de incorporarse al lenguaje. Como resultado de esto, el lenguaje era extremadamente extenso y complejo. Entonces, con el paso del tiempo, fue desapareciendo gradualmente.

ALGOL 68 fue diseñado como sucesor de ALGOL 60. Se basa en el principio de ortogonalidad: las características del lenguaje se pueden componer de manera libre, uniforme y sin interferencias con efectos predecibles. ALGOL 68 es un buen caso de estudio para ver cómo los diferentes conceptos del lenguaje pueden interactuar para proporcionar potencia computacional. Otro concepto importante planteado por el esfuerzo de ALGOL 68 es la necesidad de una especificación formal del lenguaje.

La "pureza" de ALGOL 68, las complejidades resultado de una combinación ortogonal de características del lenguaje, y la ausencia de compromisos con aspectos tan mundanos como una notación sintáctica fácil de usar fueron responsables del temprano declive de ALGOL 68.

SIMULA 67 también fue un sucesor de ALGOL 60, diseñado para resolver problemas discretos problemas de simulación. Además de construcciones ad hoc para simulación y rutinas que proporcionan una forma primitiva de ejecución paralela, el lenguaje introdujo el concepto de clase, un mecanismo de modularización que puede agrupar juntos un conjunto de rutinas relacionadas y una estructura de datos. Las clases se pueden organizar como jerarquías de especialización creciente. El concepto de clase ha influido en la mayoría de los lenguajes diseñados después de SIMULA 67.

BASIC tiene una sintaxis algebraica simple como FORTRAN y estructuras de control y datos limitadas. Esta simplicidad, y la facilidad y eficiencia de las implementaciones BASIC, hicieron que el lenguaje fuera extremadamente popular. El lenguaje en sí no introducia nuevos conceptos lingüísticos, pero fue una de las primeras herramientas disponibles que admitían un estilo de programación interpretativo altamente interactivo.

1971 - 1975: Pascal, C, Scheme, Prolog

Pascal, concebido principalmente como un vehículo para la enseñanza de la programación estructurada, tuvo una rápida expansión de su interés con la llegada de las computadoras personales de bajo costo. El principal atractivo del lenguaje era la sencillez y el apoyo a disciplinados programación. El idioma también sufrió grandes cambios y modernización, por lo que hoy en día existen muchos dialectos de Pascal.

En la década de 1970, quedó claro que las necesidades de soporte de software confiable y mantenible imponían fuertes requisitos a los lenguajes de programación. Esto dio ímpetu a nuevas investigaciones, experimentos y evaluaciones del lenguaje. Entre los conceptos lingüísticos más importantes investigados en este período fueron: tipos de datos abstractos y control de visibilidad para módulos, tipado fuerte y verificación de programa estático, relación entre construcciones de lenguaje y pruebas formales de corrección, módulos genéricos, manejo de excepciones, concurrencia y comunicación y

sincronización entre procesos. Entre los experimentos de lenguaje más influyentes se encuentran CLU, Mesa, Concurrent Pascal, Euclid y Gypsy. Otros lenguajes diseñados en la década de 1970, que sobrevivieron después de su etapa experimental y ahora se usan ampliamente, son C y Modula-2. En particular, C tuvo mucho éxito, en parte debido a la creciente disponibilidad de computadoras ejecutando el sistema operativo UNIX, cuyo desarrollo motivó los primeros diseños del lenguaje.

En el lado no convencional, la familia de lenguajes funcionales continuó floreciendo, produciendo varios dialectos LISP. Entre ellos, Scheme fue ampliamente adoptado con fines didácticos en cursos de introducción a la programación como alternativa a los lenguajes convencionales.

PROLOG fue el punto de partida de una nueva familia de lenguajes: los lenguajes de programación lógica.

1976 - 1980: Smalltalk, Ada, FORTRAN 77, ML

Los orígenes de la programación orientada a objetos se remontan a Simula 67. Sin embargo, el enfoque se hizo popular debido al éxito de Smalltalk a finales de los 70.

El deseo de unificar los lenguajes de programación utilizados en las aplicaciones informáticas integradas y la necesidad de un software más confiable y mantenible llevaron al Departamento de Defensa de los EE. UU. en 1978 a establecer los requisitos para que un lenguaje de programación se usara como lenguaje común en todo el Departamento de Defensa.

Debido a que ningún idioma existente cumplió con los requisitos, el D.O.D. patrocinó el diseño de un nuevo lenguaje. El resultado de este proceso es el lenguaje de programación Ada, que puede verse como la síntesis de los conceptos más avanzados de los lenguajes de programación convencionales.

Los desarrollos en programación funcional continuaron en los años 80, produciendo lenguajes como Miranda y ML. La importante contribución conceptual de ML fue mostrar que los lenguajes de programación pueden hacerse muy poderosos computacionalmente y, sin embargo, pueden preservar la capacidad de probar la ausencia de ciertos tipos de errores sin ejecutar programas.

1981 - 1985: Smalltalk 80, Turbo Pascal, Postscript

La palabra Smalltalk se usa a menudo para indicar el lenguaje Smalltalk-80 y la máquina virtual compatible, la primera versión que se puso a disposición del público y se creó en 1980

Aunque nacio a principios de los 70, Pascal cobra auténtica vida a partir de principios/mediados de los 80, popularizado por el fabuloso Turbo Pascal de MS-DOS para PC. Borland lanzó siete versiones de Turbo Pascal: 1.0 a 5.5 (Orientado a Objetos), 6 y 7 para MS-DOS. Fue sustituido por Borland Delphi.

Postscript es un lenguaje de descripción de páginas. Su aparición está relacionada con la necesidad de un medio estándar para definir las imágenes de las páginas de la impresora láser (creada recientemente en la época)

1986 - 1990: FORTRAN 90, C++, SML

C++ logró implementar las características orientadas a objetos en un lenguaje exitoso y ampliamente disponible como C. Esto permitió que una gran población de programadores cambiara gradualmente de un paradigma de programación convencional a uno mejor. Eiffel es otro lenguaje orientado a objetos, cuyo objetivo es respaldar la programación con principios de ingeniería de software disciplinados subyacentes.

1991 - 1995: TCL, PERL, HTML

Los lenguajes de secuencias de comandos, como TCL/TK, que especifican patrones de activación para fragmentos de herramientas existentes, son un soporte cada vez más popular para el desarrollo rápido de aplicaciones, lo que puede ser útil para desarrollar prototipos.

Perl fue originalmente desarrollado para la manipulación de texto. Se previó que fuera práctico (facilidad de uso, eficiente, completo) en lugar de hermoso (pequeño, elegante, mínimo). Sus principales características son que es fácil de usar, soporta tanto la programación estructurada como la programación orientada a objetos y la programación funcional. La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas. Perl también toma características de la programación shell.

En 1989 existían dos técnicas que permitían vincular documentos electrónicos, por un lado los hipervínculos o enlaces (hiperlinks o links) y por otro lado un poderoso lenguaje de etiquetas denominado SGML. Por entonces, Tim Berners-Lee, quien trabajaba en el Centro Europeo de Investigaciones Nucleares CERN da a conocer a la prensa que estaba trabajando en un sistema que va a permitir acceder a ficheros en línea que funcionaba sobre redes de computadoras o máquinas electrónicas basadas en el protocolo TCP/IP. Inicialmente fue desarrollado para que se pudiera compartir fácilmente información entre científicos de distintas universidades e institutos de investigación de todo el mundo. A principios de 1990, define por fin el HTML como un subconjunto del conocido SGML y crea algo más valioso incluso, el World Wide Web.

1996 - 2000: Java, Javascript, XML

Como se dijo, HTML se definió en el marco de SGML y fue de lejos la aplicación más conocida de este estándar. Los navegadores web sin embargo siempre han puesto pocas exigencias al código HTML que interpretan y así las páginas web son caóticas y no cumplen con la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos.

Otra limitación del HTML es que cada documento pertenece a un vocabulario fijo, establecido por el DTD. No se pueden combinar elementos de diferentes vocabularios. Asimismo, es imposible para un intérprete (por ejemplo, un navegador) analizar el documento sin tener conocimiento de su gramática (del DTD). Por ejemplo, el navegador sabe que antes de una etiqueta debe haberse cerrado cualquier <p> previamente abierto. Los navegadores resolvieron esto incluyendo lógica ad hoc para el HTML, en vez de incluir un analizador genérico. Ambas opciones, de todos modos, son muy complejas para los navegadores.

Se buscó entonces definir un subconjunto del SGML que permita:

- Mezclar elementos de diferentes lenguajes. Es decir que los lenguajes sean extensibles.
- La creación de analizadores simples, sin ninguna lógica especial para cada lenguaje.
- Empezar de cero y hacer hincapié en que no se acepte nunca un documento con errores de sintaxis.

Para hacer esto XML deja de lado muchas características de SGML que estaban pensadas para facilitar la escritura manual de documentos. XML en cambio está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

XML y sus extensiones han sido regularmente criticadas por su nivel de detalle y complejidad.

Indique para cada uno de los períodos presentados cuales son las características nuevas que se incorporan y cuál de ellos la incorpora.

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada the Green Project en Sun Microsystems en 1991.

Los objetivos eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratoniana de tres días entre John Gage, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable.

La promesa inicial de Gosling era Write Once, Run Anywhere (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares, de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro, y los principales navegadores web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar

Ejercicio 2: Escriba brevemente la historia del lenguaje de programación que eligió en la encuesta u otro de su preferencia

JAVASCRIPT

JavaScript se introdujo en 1995 como una forma de agregar programas a páginas web en el navegador Netscape Navigator. En su momento fue una idea novedosa. En los primeros días de la World Wide Web, HTML era bastante simple, y bastante fácil de aprender casi todo lo que se necesitaba saber para agrupar páginas web. Cualquiera podía hacer una Web juntando tablas, texto y añadiendo alguna imagen.

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. Esa velocidad era más que suficiente para la época salvo que quisieras descargar imagenes de cierto tamaño. Lo cierto era que la web en aquel entonces no ofrecía gran cosa más que servir como una inmensa biblioteca donde los usuarios consultaban mayormente contenido basado en texto pero la evolución que conocemos hoy estaba por llegar y podían verse los primeros pasos.

En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos. Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Desde entonces, el lenguaje ha sido adoptado por todos los demás navegadores gráficos principales. Ha hecho posibles las aplicaciones web modernas, aplicaciones con las que puede interactuar directamente sin hacer una recarga de página para cada acción.

En la actualidad los navegadores web no son las únicas plataformas en las que se utiliza JavaScript. También es posible ejecutar código JavaScript en un entorno servidor. De hecho las bases de datos, como MongoDB y CouchDB, usan JavaScript como su lenguaje de scripting y consulta. Varias plataformas para la programación de escritorio y servidor, en particular el proyecto Node.js proporcionan un entorno para la programación de JavaScript fuera del navegador.

Antes de continuar hay que aclarar que aunque se parecen JavaScript no tiene nada que ver con el lenguaje de programación llamado Java. El nombre en realidad viene por una cuestión más de marketing. Cuando se introdujo JavaScript, el lenguaje Java se estaba comercializando en gran medida y estaba ganando popularidad. Alguien pensó que era una buena idea tratar de avanzar en esto y hasta nuestros días. :)

Aunque JavaScript surgió como un lenguaje de script para mejorar las capacidades de la web de la época allá por 1995 por la extinta Netscape, JavaScript no ha dejado de evolucionar desde entonces. Originalmente el lenguaje se basaba a su vez basaba en CEnvi desarrollado a su vez por Nombas.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar las limitaciones de la web de entonces, adaptando otras tecnologías existentes (como ScriptEase) al navegador Netscape Navigator 2.0, que iba a lanzarse en aquel año. Inicialmente, Eich denominó a su lenguaje LiveScript y fue un éxito.

Fue entonces cuando, justo antes del lanzamiento, Netscape decidió cambiar el nombre por el de JavaScript y firmó una alianza con Sun Microsystems para continuar el desarrollo del nuevo lenguaje de programación.

Microsoft, al ver el movimiento de uno de sus principales competidores, también decidió incorporar su propia implementación de este lenguaje, llamada JScript, en la versión 3 de su navegador Internet Explorer.

Esto contribuyó todavía más al empuje y popularización del lenguaje, pero comenzaron a presentarse pequeños problemas por las diferencias entre implementaciones. Por lo que se decidió estandarizar ambas mediante la versión JavaScript 1.1 como propuesta a ECMA, que culminó con el estándar ECMA-262. Este estándar dicta la base del lenguaje ECMAScript a través de su sintaxis, tipos, sentencias, palabras clave y reservadas, operadores y objetos, y sobre la cual se pueden construir distintas implementaciones. La versión JavaScript 1.3 fue la primera implementación completa del estándar ECMAScript.

ECMAScript 3, que data de 1999, y cuyas últimas mejoras han sido:

- Soporte de expresiones regulares.
- Nuevas sentencias de control.
- Manejo de excepciones (bloque try-catch).
- Definición de errores más precisa.
- Formateo de salidas numéricas de datos.
- Manejo de strings más avanzado.

ECMAScript 4, que aparece en 2004:

- Pretende convertir JavaScript en un nuevo lenguaje con nuevas reglas.
- Introduce el tipado de variables
- introduce el concepto tradicional de clases e interfaces al estilo de lenguajes como Java.

Las características que se incluyen en ECMAScript5 son las siguientes:

- Getters y setters.
- Array extras y reductions.
- Rediseño de los atributos internos de las propiedades.
- Introducción de métodos estáticos de Object, que permiten:
- Acceder a la información del prototipo.
- Manipular las propiedades de un objeto.
- Crear objetos de forma dinámica.
- Obtener los nombres de las propiedades.
- Impedir que un objeto sea modificado.
- Cambios a las funciones:
- Soporte nativo del function currying a través del método bind.
- Cambios en el objeto Date.
- Soporte nativo de JSON.

ECMAScript 6:

- Módulos.
- Ámbito a nivel de bloque (sentencia let).
- Generators.
- Proxys.
- Destructuring assignments.
- Rest y default arguments.
- Name objects.
- Iterators.
- Array comprehensions.
- String templates.
- Hash tables y weak maps.

JavaScript fue ideado para dotar a la web de capacidades interactivas que le ayudarán a dar el salto al siguiente nivel permitiendo crear una interfaz de usuario activa, lo que ofrece retroalimentación a los visitantes según navegan por sus páginas. Por ejemplo, es común usar JavaScript en la validación de formularios para asegurarnos que la información introducida es válida. Sin necesidad de enviar ninguna información al servidor, el programa realiza los cálculos necesarios ahorrando tiempo y recursos del lado del servidor.

Con JavaScript podemos crear sobre la marcha páginas HTML personalizadas, dependiendo de las acciones ejecutadas por el usuario. Supongamos que estamos en una web de seguros, con JavaScript podemos realizar consultas en el servidor sin necesidad de recargar la página, mostrar opciones personalizadas, etc y lanzar eventos en función del día y hora en donde nos encontremos.

Ejercicio 3: ¿Qué atributos debería tener un buen lenguaje de programación? Por ejemplo, ortogonalidad, expresividad, legibilidad, simplicidad, etc. De al menos un ejemplo de un lenguaje que cumple con las características citadas.

El cumplimiento de los atributos se basa en una evaluación subjetiva y son difíciles de establecer de manera precisa, y mucho menos cuantitativa. Además, no son conceptos independientes: en algunos casos se superponen, en otros están en conflicto.

SIMPLICIDAD Y LEGIBILIDAD – Ejemplo: Javascript o Python.

- Los lenguajes de programación deberían:
    - Poder producir programas fáciles de escribir y de leer.
    - Resultar fáciles a la hora de aprenderlo o enseñarlo
    - Ejemplo de cuestiones que atentan contra esto:
        - Muchas componentes elementales
        - Conocer subconjuntos de componentes
        - El mismo concepto semántico – distinta sintaxis
        - Distintos conceptos semánticos – la misma notación sintáctica
        - Abuso de operadores sobrecargados

CLARIDAD EN LOS BINDINGS – Ejemplo: Java

- Los elementos de los lenguajes de programación pueden ligarse a sus atributos o propiedades en diferentes momentos:
    - Definición del lenguaje
    - Implementación del lenguaje
    - En escritura del programa
    - Compilación
    - Cargado del programa
    - En ejecución
- La ligadura en cualquier caso debe ser clara

CONFIABILIDAD – Ejemplo: Java

- La confiabilidad está relacionada con la seguridad
    - Chequeo de tipos: Cuanto antes se encuentren error menos costoso resulta realizar los arreglos que se requieran.
    - Manejo de excepciones: La habilidad para interceptar errores en tiempo de ejecución, tomar medidas correctivas y continuar.

SOPORTE – Ejemplo: Java, Javascript, Python, C

- Debería ser accesible para cualquiera que quiera usarlo o instalarlo
    - Lo ideal sería que su compilador o intérprete sea de dominio público
    - Debería poder ser implementado en diferentes plataformas
    - Deberían existir diferentes medios para poder familiarizarse con el lenguaje: tutoriales, cursos textos, etc.

ABSTRACCIÓN – Ejemplo: Java, Javascript.

- Capacidad de definir y usar estructuras u operaciones complicadas de manera que sea posible ignorar muchos de los detalles.
    - Abstracción de procesos y de datos

ORTOGONALIDAD

- Se refiere al significado de las palabras reservadas o símbolos.
    - Si una palabra reservada siempre tiene el mismo significado independientemente del contexto en que se use el lenguaje tiene mayor ortogonalidad que en otro donde no sucede eso.
        - Un ejemplo es el significado de "+" si siempre representa la suma aritmética es ortogonal, por el contrario, si representa a la suma para variables numéricas y concatenación para variables de tipo cadena entonces es menor ortogonal.

EFICIENCIA – Java)?

- Con respecto a:
    - Tiempo y Espacio
    - Esfuerzo humano
    - Optimizable

Ejercicio 4: Tome uno o dos lenguajes de los que ud. Conozca y

- **Describa los tipos de expresiones que se pueden escribir en el/ellos**
- **Describa las facilidades provistas para la organización del programa**
- **Indique cuáles de los atributos del ejercicio anterior posee el/los lenguaje/s elegidos y cuáles no posee, justifique en cada caso.**

Javascript

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

Ventajas

1. Es un lenguaje muy sencillo.
2. Es rápido, por lo tanto tiende a ejecutar las funciones inmediatamente.
3. Cuenta con múltiples opciones de efectos visuales.
4. Es soportado por los navegadores más populares y es compatible con los dispositivos más modernos, incluyendo iPhone, móviles y PS3.
5. Es muy versátil, puesto que es muy útil para desarrollar páginas dinámicas y aplicaciones web.
6. Es una buena solución para poner en práctica la validación de datos en un formulario.
7. Es multiplataforma, puede ser ejecutado de manera híbrida en cualquier sistema operativo móvil.
8. Es el único lenguaje que permite trabajar modo FullStack en cualquier tipo de desarrollo de programación.

Desventajas

1. En el FrontEnd sus códigos son visibles, por lo tanto pueden ser leídos por cualquier usuario.
2. Tiende a introducir gran cantidad de fragmentos de código en los sitios web.
3. Sus opciones 3D son limitadas, si se quiere utilizar este lenguaje de programación para crear un juego, deben emplearse otras herramientas.
4. No es compatible en todos los navegadores de manera uniforme.
5. Los usuarios tienen la opción de desactivar JavaScript desde su navegador.

6. Sus script son limitados por razones de seguridad y no es posible realizar todo con JavaScript, por lo tanto es necesario complementarlo con otros lenguajes evolucionados y más seguros. Esta es una de las características de JavaScript que algunos expertos lo contemplan como una ventaja y otros como una desventaja.

SIMPLICIDAD Y LEGIBILIDAD → Posee, es sencillo de leer, aprender y escribir.

CLARIDAD EN LOS BINDINGS → Ni idea

SEGURIDAD → Sip, throw.

ABSTRACCION → Gran cantidad de funciones que abstraen detalles, ej Math.Max() , reduce()

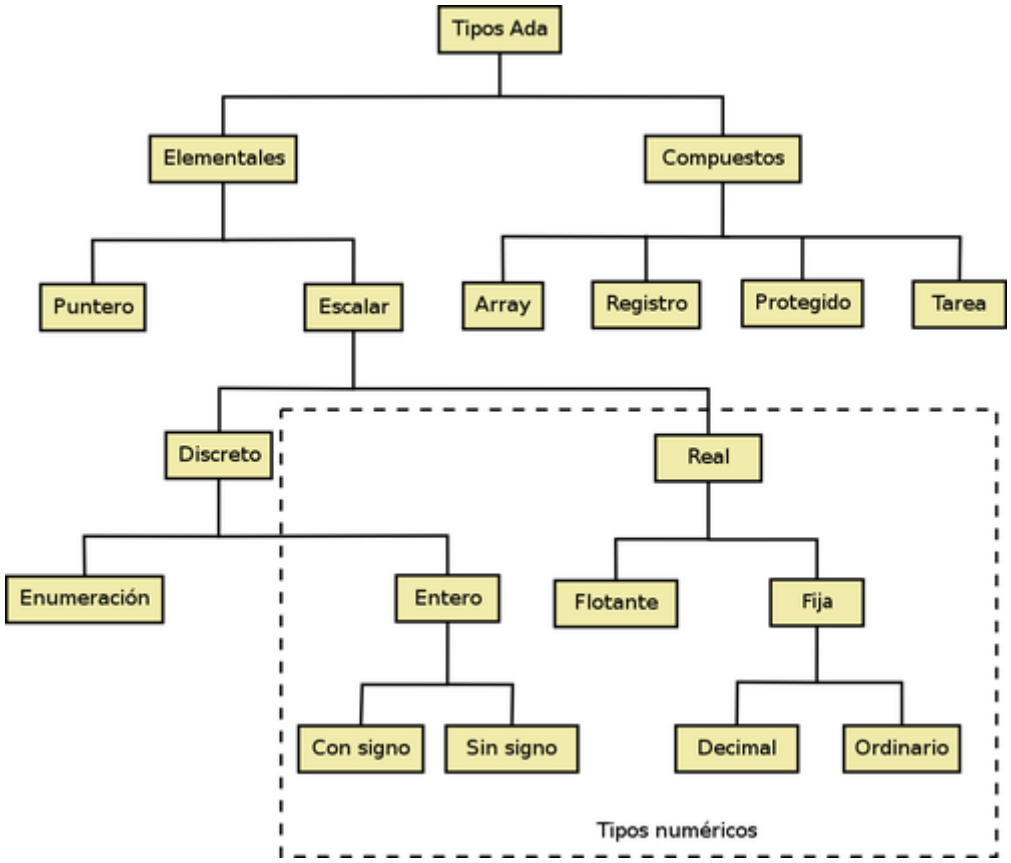
ORTOGONALIDAD → No tanta, el + se puede usar para concatenar números o strings

EFICIENCIA → Si, JavaScript tiene mayor rendimiento y memoria eficiente.

Lenguajes – ADA

Ejercicio 5: Describa las características más relevantes de Ada, referida a:

- **Tipos de datos**



Los tipos de *Ada* se pueden clasificar en:

- Escalar:
    - Discreto:

        - Enteros: Integer, Natural, Positive.
        - Enumeraciones: Boolean, Character.
    - Real:
        - Coma flotante.
        - Coma fija.

- Compuesto:
    - Vector: Arrays, Strings.
    - Registros: record.

- Puntero: access
    - Puntero a objetos.
    - Punteros a subprogramas.

- Privado: private.
- Tareas: task.

- **Tipos abstractos de datos – paquetes**

  Los paquetes exportan mediante una interfaz bien definida tipos, objetos y operaciones y permiten ocultar su implementación, lo que proporciona al programador tipos abstractos de datos y subprogramas de manera transparente.

  Los paquetes de Ada proporcionan:

    - Abstracción de datos.
    - Encapsulamiento.
    - Compilación separada y dependiente.

El paquete consta de especificación (parte visible) y cuerpo (implementación que se oculta) y pueden compilarse por separado.

La sintaxis de su especificación es la siguiente:

```
especificación_paquete ::=
  package [ identificador_unidad_padre . ] identificador is
    { declaración_básica }
  [ private  { declaración_básica } ]
  end [ [ identificador_unidad_padre . ] identificador ] ;
```

La sintaxis del cuerpo de un paquete es la siguiente:

```
cuerpo_paquete ::=
  package body [ identificador_unidad_padre . ] identificador is
    [ parte_declarativa ]
  [ begin
    secuencia_de_sentencias
  [ exception
```

```
    manejador_de_excepción
    { manejador_de_excepción } ] ]
  end [ [ identificador_unidad_padre . ] identificador ] ;
```

Un paquete permite agrupar declaraciones y subprogramas relacionados.

- **Estructuras de datos**

Arreglos:

Tipo arreglo

La forma de declarar un arreglo en Ada es

vec: array (1..5) of integer

La forma de acceder a cada componente de vec es similar a Pascal:

vec(3):=0;

vec(i):=10; --siendo i integer

Es posible iniciar arreglos sin la necesidad de utilizar un for:

vec:=(0,0,0,0,0);

vec:=(1=>0, 2=>0, 3=>0, 4=>0, 5=>0);

vec:=(1|2|3|4|5=>0);

vec:=(1..5=>0);

vec:=(others=>0);

Arreglos irrestrictos (var. semidinámicas)

Definen una clase de tipos de array posibles. Cada miembro de la clase tiene el mismo tipo, el mismo número de índices y el mismo tipo de índice.

Ejemplo:

type matriz is (integer range<>, integer range<>) of integer;

si quiero declarar una variable matriz debería limitar el índice

x: matriz(1..5, 1..5);

y: matriz(10..20, 2..3);

Atributos de los arreglos:

first

last

lenght

range

Ejemplo:

```
procedure imprimir (m:matriz) is
begin
 for i in m'range(1)
 for j in m'range(2)
 put(x(i,j));
 end loop;
 end loop;
end imprimir;
```

Registros

```
type alumno is record
 nombre: string(10);
 numero: integer;
 end record
 a: alumno;
```

La forma de acceder a los campos es similar a Pascal:

```
a.numero:=10;
```

Además es posible realizar las siguientes asignaciones:

```
a:=("pepe",10);
a:=(nombre=>"pepe", numero=>10);
```

- **Manejo de excepciones**

En Ada, cuando se produce algún error durante la ejecución de un programa, se eleva una excepción. Dicha excepción puede provocar la terminación abrupta del programa, pero se puede controlar y realizar las acciones pertinentes. También se pueden definir nuevas excepciones que indiquen distintos tipos de error.

En Ada, dentro del paquete Standard, existen unas excepciones predefinidas, éstas son:

Constraint_Error

cuando se intenta violar una restricción impuesta en una declaración, tal como indexar más allá de los límites de un array o asignar a una variable un valor fuera del rango de su subtipo.

Program_Error

se produce cuando se intenta violar la estructura de control, como cuando una función termina sin devolver un valor.

Storage_Error

es elevada cuando se requiere más memoria de la disponible.

Tasking_Error

cuando hay errores en la comunicación y manejo de tareas.

Numeric_Error

en Ada 83 se podía presentar cuando ocurría un error aritmético. A partir del estándar Ada 95, desaparece por motivos de portabilidad y pasa a ser un renombrado de Constraint_Error. Por ejemplo, en Ada 83 al dividir entre cero podía saltar Constraint_Error o Numeric_Error (dependiendo del compilador). En Ada 95 este error siempre levanta Constraint_Error.

Name_Error

se produce cuando se intenta abrir un fichero que no existe.

Cuando se espere que pueda presentarse alguna excepción en parte del código del programa, se puede escribir un manejador de excepciones en las construcciones que lo permitan (bloques o cuerpos de subprogramas, paquetes o tareas), aunque siempre está el recurso de incluir un bloque en cualquier lugar del código.

Su sintaxis sería:

```
manejador_excepción ::=
  when [ identificador : ] elección_excepción { | elección_excepción } =>
    secuencia_sentencias
elección_excepción ::= identificador | others
```

A la sentencia que comienza por when, se le denomina manejador de excepción.

La palabra reservada others indica cualquier otra excepción y debe ser la única y última opción

- **Manejo de concurrencia**

La concurrencia es la simultaneidad de hechos. Un programa concurrente es aquel en el que ciertas unidades de ejecución internamente secuenciales (procesos o threads), se ejecutan paralela o simultáneamente.

Existen 3 formas básicas de interacción entre procesos concurrentes:

Sincronización (p.e. las citas o paso de mensajes).

Señalización (p.e. los semáforos).

Comunicación (p.e. uso de memoria compartida).

La concurrencia o procesamiento paralelo se ha implementado en leguajes de programación de distinta manera:

Programación concurrente clásica: se basa en la utilización de variables compartidas. Es el caso de Modula-2 o Concurrent Pascal. Para ello, se emplean herramientas como semáforos, regiones críticas y monitores.

Programación concurrente distribuida: se basa en la transferencia de mensajes entre los procesos o threads. Es el caso de C/POSIX, Occam o Ada. Se emplean herramientas como canales, buzones y llamadas a procedimiento remoto.

En Ada se emplea una programación concurrente distribuida y la principal forma de sincronizar las unidades de ejecución, conocidas como tareas, son los puntos de entrada a la tarea o citas.

Mas información sobre ADA:

http://www.frlp.utn.edu.ar/materias/sintaxis/ApunteTadAda.pdf

http://www.gedlc.ulpgc.es/docencia/mp_i/GuiaAda

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Ada

Lenguajes - JAVA

Ejercicio 6: Diga para qué fue, básicamente, creado Java. ¿Qué cambios le introdujo a la Web? ¿Java es un lenguaje dependiente de la plataforma en dónde se ejecuta? ¿Por qué?

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada the Green Project en Sun Microsystems en 1991.

Los objetivos eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratoniana de tres días entre John Gage, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable.

Java introdujo cualquier aplicación escrita en Java podría ejecutarse de la misma forma en cualquier hardware, independientemente de la plataforma.

Ejercicio 7: ¿Sobre qué lenguajes está basado?

En C++.

Ejercicio 8: ¿Qué son los applets? ¿Qué son los servlets?

Applet

El applet es un programa Java™ diseñado para incluirse en un documento Web HTML. Podrá escribir un applet Java y luego incluirlo en una página HTML de una manera muy parecida a cómo se incluye una imagen. Al utilizar un navegador habilitado para Java para ver una página HTML que contiene un applet, el código del applet se transfiere al sistema y la máquina virtual Java del navegador lo ejecuta.

El documento HTML contiene códigos que especifican el nombre del applet Java y su localizador uniforme de recursos (URL). El URL es la ubicación en la que residen los bytecodes del applet en Internet. Cuando se visualiza un documento HTML que contiene un código de applet Java, un navegador Web habilitado para Java descarga los bytecodes Java de Internet y utiliza la máquina virtual

Java para procesar el código desde el documento Web. Estos applets Java son los que permiten que las páginas Web contengan gráficos animados o información interactiva.

Servlet

Los servlets son modulos java que nos sirven para extender las capacidades de los servidores web. Aunque es una definición un poco ambigua los servlets son programas para los servidores, mientras que los applets son programas para los clientes y los middlets los programas para microdispositivos.

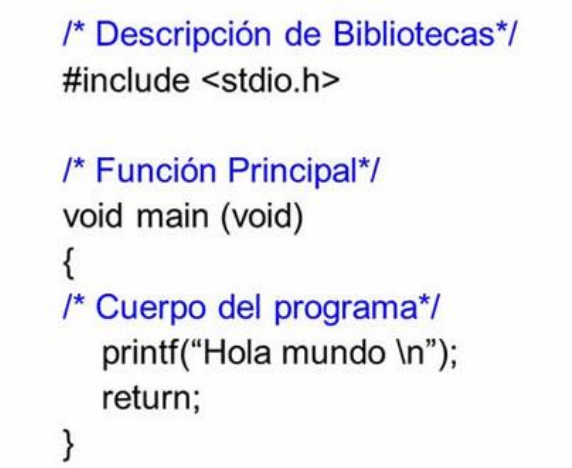
Podremos desarrollar desde un simple servlet que nos muestre una página web simple saludandonos hasta uno que se conecte a una base de datos utilizando un pool de conexiones, encriptando la información en su envio, accediendo a bases de datos distribuidas y manteniendo su información de forma persistente en un EJB. Todo ello para conseguir una información dinámica. A partir de aqui las posibilidades son “infinitas”.

Lenguajes – C

Ejercicio 9: ¿Cómo es la estructura de un programa escrito en C? ¿Existe anidamiento de funciones?

Todo programa en C consta de una o más funciones, una de las cuales se llama main. El programa comienza en la función main, desde la cual es posible llamar a otras funciones.

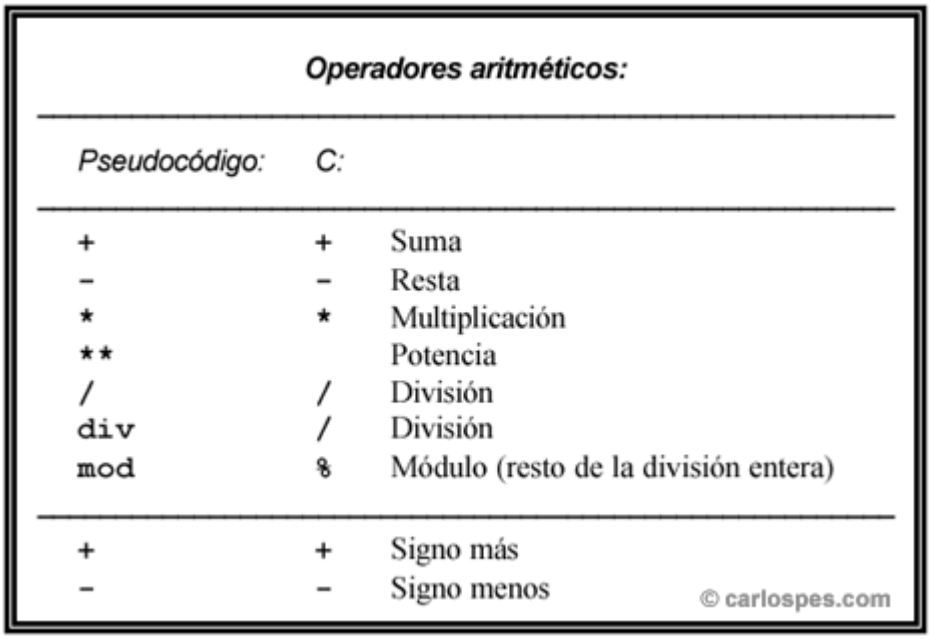
Cada función estará formada por la cabecera de la función, compuesta por el nombre de la misma y la lista de argumentos, la declaración de las variables a utilizar y la secuencia de sentencias a ejecutar.



```
/* Descripción de Bibliotecas*/
#include <stdio.h>

/* Función Principal*/
void main (void)
{
/* Cuerpo del programa*/
   printf("Hola mundo \n");
   return;
}
```

```
declaraciones globales

        main( ) {
                variables locales
                bloque
        }

        funcion1( ) {
                variables locales
                bloque
        }
```

Ejercicio 10: Describa el manejo de expresiones que brinda el lenguaje.



Operadores aritméticos:

Pseudocódigo:	*C:*	
+	+	Suma
-	-	Resta
*	*	Multiplicación
**		Potencia
/	/	División
div	/	División
mod	%	Módulo (resto de la división entera)
+	+	Signo más
-	-	Signo menos

En un programa, el tipo de un dato determina las operaciones que se pueden realizar con él. Por ejemplo, con los datos de tipo entero se pueden realizar operaciones aritméticas, tales como la suma, la resta o la multiplicación.

Ejemplo 1: Algunos ejemplos son:

111 + 6 (operación suma)

19 - 72 (operación resta)

24 * 3 (operación multiplicación)

Todas las operaciones del ejemplo constan de dos operandos (constantes enteras) y un operador. La mayoría de las veces es así, pero, también es posible realizar operaciones con distinto número de operadores y/u operandos.

Ejemplo 2: Por ejemplo:

111 + 6 - 8 (tres operandos y dos operadores)

-((+19) + 72) (dos operandos y tres operadores)

-(-72) (un operando y dos operadores)

En las operaciones del ejemplo se puede observar que los caracteres más (+) y menos (-) tienen dos usos:

1. Operadores suma y resta.

2. Signos de un número (también son operadores).

Los operadores de signo más (+) y menos (-) son operadores monarios, también llamados unarios, ya que, actúan, solamente, sobre un operando.

Los caracteres abrir paréntesis "(" y cerrar paréntesis ")" se utilizan para establecer la prioridad de los operadores, es decir, para establecer el orden en el que los operadores actúan sobre los operandos.

Un operador indica el tipo de operación a realizar sobre los operandos (datos) que actúa. Los operandos pueden ser:

Constantes (expresadas por su valor o con un nombre (identificador)).

Variables.

Llamadas a funciones.

Elementos de formaciones (arrays).

Un operador siempre forma parte de una expresión, en la cual, el operador siempre actúa sobre al menos un operando. Por el contrario, un operando sí puede aparecer solo en una expresión.

En programación, de la evaluación de una expresión siempre se obtiene un valor. Dicho valor puede ser de tipo: entero, real, lógico, carácter o cadena. Por consiguiente, una expresión puede ser:

Aritmética (devuelve un número entero o real).

Lógica (devuelve un valor lógico: verdadero o falso)

De carácter (devuelve un carácter representable por el ordenador).

De cadena (devuelve una cadena).

Mas información sobre C:

https://www.carlospes.com/curso_de_lenguaje_c/

Lenguajes - Python - RUBY - PHP

Ejercicio 11: ¿Qué tipo de programas se pueden escribir con cada uno de estos lenguajes? ¿A qué paradigma responde cada uno? ¿Qué características determinan la pertenencia a cada paradigma?

Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Python es multiparadigma: orientado a objetos, imperativo, funcional, reflexivo

Ruby es un lenguaje de programación muy popular que se utiliza para muchas cosas, desde el desarrollo de aplicaciones web hasta el análisis de datos. Ruby es un lenguaje multiparadigma: orientado a objetos, reflexivo

PHP es un lenguaje de programación de código abierto del lado del servidor que se utiliza principalmente para crear páginas web dinámicas. PHP es multiparadigma: imperativo, funcional, orientado a objetos, procedural, reflexivo

Ejercicio 12: Cite otras características importantes de Python, Ruby, PHP, Gobstone y Processing. Por ejemplo: tipado de datos, cómo se organizan los programas, etc.

Los tipos de datos básicos de Python son los booleanos, los numéricos (enteros, punto flotante y complejos) y las cadenas de caracteres.

Python también define otros tipos de datos, entre los que se encuentran:

- Secuencias: Los tipos list, tuple y range
- Mapas: El tipo dict
- Conjuntos: El tipo set
- Iteradores
- Clases
- Instancias
- Excepciones

Los principales tipos de datos (considerados tipos de datos escalares) en PHP son:

- boolean: almacenan valores verdadero o falso (true / false).
- integer: números enteros.
- float: números con decimales, usando el punto como separador decimal.
- string: cadenas de texto.

PHP también incluye los siguientes tipos de datos compuestos (los trataremos en profundidad más adelante):

- array: usados para almacenar varios valores.
- object: almacena objetos en PHP.

Incluye también los siguientes tipos de datos especiales:

- resource: hace referencia a recursos tales como archivos abiertos, conexiones establecidas con bases de datos, etc.
- null: indique el valor está vacío (no contiene nada).

En Ruby, todo es tratado como un objeto, eso no excluye a los números, en forma general, Ruby cuenta con diferentes clases para manejar cada tipo de números, por ejemplo:

- Integer: la clase base de donde derivan todos los enteros.
- Fixnum: clase para números enteros, su tamaño depende de la arquitectura de donde se interprete el código, sin embargo, su tamaño es eso -1 bit y usa complemento 2 para su representación en memoria, si un número excede el tamaño asignado, automaticamente se convierte en bignum.
- Bignum: Contiene valores mayores a fixnum, la restricción depende de la arquitectura pero pueden guardarse números muy grandes, tanto como de nuestra memoria, si el número ingresado cabe en un fixnum, automáticamente se convierte a esta clase.
- Float: Almacena números con punto flotante con la arquitectura de doble precisión nativa.
- Rational: Almacena números racionales, es decir, números con un valor de numerador y un denominador.

Para las cadenas de caracteres, su uso es bastante similar al de cualquier lenguaje orientado a objetos, con la clase String, sin embargo cabe mencionar algunas caracteristicas mas emocionantes de Ruby es que se puede multiplicar cadenas.

“Hola ” * 5 => “Hola Hola Hola Hola Hola”

A su vez, Ruby trae librerias para manejar los tipos date y time, sin embargo para usarlos, se debe llamar a la librería correspondiente, esto con la palabra reservada require.

En Gobstones los tipos de datos son: colores, direcciones, números y booleanos.

Los distintos tipos de datos en Processing son: int, float, boolean, true, false, = (assign), width, height

Lenguaje Javascript

Ejercicio 13: ¿A qué tipo de paradigma corresponde este lenguajes? ¿A qué tipo de Lenguaje pertenece?

Javascript es Multiparadigma: programación funcional, programación basada en prototipos, imperativo, interpretado (scripting)

Es débilmente tipado y dinámico.

Ejercicio 14: Cite otras características importantes de javascript. Tipado de datos, excepciones, variables, etc.

- Seis tipos de datos primitivos, controlados por el operador typeof
    - Undefined: typeof instance === "undefined"
    - Boolean: typeof instance === "boolean"
    - Number: typeof instance === "number"
    - String: typeof instance === "string"
    - BigInt: typeof instance === "bigint"
    - Symbol: typeof instance === "symbol"
- Null: typeof instance === "object". Tipo primitivo especial que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra null;
- Object: typeof instance === "object". Tipo estructural especial que no es de datos pero para cualquier instancia de objeto construido que también se utiliza como estructuras de datos: new Object, new Array, new Map (en-US), new Set, new WeakMap, new WeakSet, new Date y casi todo lo hecho con la palabra clave new;
- Function: una estructura sin datos, aunque también responde al operador typeof: typeof instance === "function". Esta simplemente es una forma abreviada para funciones, aunque cada constructor de funciones se deriva del constructor Object.

Cuando ocurre un error, JavaScript crea un objeto de error con dos propiedades: nombre y mensaje. Entonces se dice que JavaScript lanza una excepción.

Puedes lanzar excepciones (errores) usando la sentencia throw y manejarlas usando las sentencias try…catch.

<u>Sentencia throw</u>

trow permite personalizar un error, y puede o no ser usado junto con try y catch:

throw expresión; // 'expresión' es el valor personalizado que será lanzado.

Podemos especificar un objeto cuando lanzamos una excepción. En este caso, manejaremos las propiedades del objeto desde el bloque catch.

/* Crea un tipo de objeto llamado 'excepcionUsuario' /

function excepcionUsuario(mensaje) {

this.mensaje = mensaje;

this.nombre = 'excepcionUsuario';

}

/ Convierte la excepción en una cadena, /

excepcionUsuario.prototype.aCadena = function() {

return this.nombre + ': ' + this.mensaje;

}

/ Crea una instancia del tipo de objeto y lánzala */

throw new excepcionUsuario('La cadena introducida no es una dirección de e-mail válida');

Sentencia try…catch

La sentencia try … catch consiste en:

- Un bloque try, que contiene una o más instrucciones. Si una de estas no se ejecuta con éxito, se lanza una excepción que es manejada por el bloque catch. En caso contrario, catch se ignora.
- Ninguno o varios bloques catch, que especifican qué hacer si una excepción es lanzada en el bloque try, (o en una función llamada dentro del bloque try).
- El bloque finally, que se ejecuta inmediatamente después de try…catch, permite ejecutar código sea cual sea el resultado, y terminar de «pulir» el manejo del error.

El bloque finally

El bloque finally contiene sentencias a ejecutar después de las sentencias try…catch. El bloque finally se ejecuta siempre: tanto si se lanza una excepción como si no, y aunque no exista un bloque catch que maneje la excepción.

Si el bloque finally retorna un valor, este valor se convierte en el valor de retorno de toda la sentencia try-catch-finally, independientemente de que hayan sentencian return en el bloque try o el bloque catch:

Sentencias try…catch anidadas

Es posible anidad una o más sentencias try…catch. Si una sentencia try…catch interna no tiene un bloque catch, pasa a verificarse la sentencia try…catch del bloque exterior.