NLN

PROGRAMAÇÃO

JAVASCRIPT

JAVASCRIPT PARA INICIANTES

2021

JOHN BACH

1ª edição
2021

Por John Bach

"A programação não tem a ver com o que se sabe; é sobre o que pode descobrir. Chris *Pine*

Introdução, Ano Novo

Começar é fácil, mas há um longo caminho a percorrer se quiser expressar todas as possibilidades da linguagem. O Javascript é uma das línguas padrão da web e, portanto, ideal para muitos dos profissionais que planeiam dedicar-se a este meio. Mas mesmo que não queira desenvolver especificamente para a web, o Javascript é uma excelente alternativa para fazer aplicações móveis ou aplicações de desktop.

Talvez nem todos os desenvolvedores precisem de atingir um nível avançado de Javascript, mas um conhecimento básico será certamente de grande ajuda em muitos pontos das suas carreiras profissionais. Em todo o caso, uma vez concluída a aprendizagem, terás infinitas oportunidades na ponta dos dedos.

Começaremos por olhar para o estado do Javascript hoje (2017) e as razões pelas quais o Javascript vale a pena usar em geral. Então vamos cuidar de explicar como pode aprender Javascript e chegar a qualquer nível que nos propusemos.

Javascript está em todo o lado

O Javascript tornou-se uma linguagem ideal para aprender, uma vez que tem muitas e variadas aplicações, além de proporcionar simplicidade às pessoas que estão a começar. Para executá-lo só precisamos de um navegador, embora no momento o Javascript tenha ultrapassado o âmbito dos clientes web, para estar localizado em qualquer lugar.

Javascript na web

O ambiente onde Javascript apareceu pela primeira vez foi a web. A sua execução focou-se no âmbito de uma página web e permitiu que os desenvolvedores fornecessem interatividade, manipular o documento ou a janela do navegador, realizar cálculos, etc. Netscape Navigator, um navegador que desapareceu hoje, tem a honra de ter introduzido o Javascript como um idioma, embora hoje seja suportado por todos os clientes web com os quais um utilizador pode navegar.

A utilização do Javascript para a web, no ambiente do navegador, também é conhecida como programação do lado do cliente. Hoje em dia ainda é o ambiente mais comum para a execução javascript, mas na verdade é apenas mais um entre as suas possibilidades.

Javascript no backend

Alguns desenvolvedores tiveram a ideia de que poderiam extrair o tempo de execução javascript, que até então apenas estava disponível no âmbito do navegador, para usá-lo para qualquer outro propósito fora do cliente web. Foi assim que o NodeJS nasceu, o que não é mais do que o Javascript fora do navegador.

Com o NodeJS podemos programar com aplicações Javascript que funcionam diretamente no sistema operativo e que são capazes de resolver qualquer tipo de problema. Com esta tecnologia, o Javascript tornou-se uma linguagem de programação de propósito geral.

Um dos usos mais comuns do NodeJS é a programação de backend. Permite programar aplicações que sejam capazes de executar no servidor, fornecendo acesso a bases de dados, ao sistema de ficheiros e a quaisquer outros recursos do lado do servidor. No entanto, o NodeJS é tão abrangente que pode ser usado para muitas outras tarefas, como automação, otimização ou implementação de aplicações, entre outras operações.

Javascript como idioma para aplicativos de dispositivos

Durante anos também foi possível utilizar o Javascript como um idioma para criar aplicações para dispositivos (aplicações móveis, tablets, televisores...). As aplicações desenvolvidas são instaladas a partir das correspondentes lojas de aplicações dos principais sistemas móveis e o utilizador em princípio não percebe qualquer diferença em relação às aplicações desenvolvidas com as línguas autóctones. Mas, devido ao facto de serem programados com Javascript e HTML5, abrem um novo campo de ação para as pessoas experientes no desenvolvimento para a web e dispensam a aprendizagem de línguas nativas para cada plataforma móvel.

Inicialmente, para executar aplicações feitas em Javascript e HTML5, era necessário um "Web View". Basicamente, a função da vista web é oferecer um enquadramento para a execução da app, de modo que seja executado dentro de um navegador, mesmo que o utilizador não a perceba. Esta situação tem várias vantagens e desvantagens que não iremos entrar, mas hoje também existe a possibilidade de usar o Javascript como uma linguagem de desenvolvimento para aplicações nativas, que não requerem uma visão web para funcionar.

Exemplos de quadros para o desenvolvimento de aplicações baseadas na visão web temos Apache Cordova, Phone Gap ou Ionic. Exemplos de quadros para o desenvolvimento de aplicações nativas usando Javascript incluem O Script Nativo e O Nativo de Reagir. Todas as alternativas têm a importante vantagem de produzir apps para Android e iOS com a mesma base de código, bem como para outros sistemas minoritários.

Javascript como um idioma para aplicações de desktop

Outra área em que o Javascript penetrou fortemente está no desenvolvimento de aplicações para computadores pessoais. Com o Javascript somos capazes de criar aplicações avançadas, capazes de usar todos os recursos de um computador e também executar em qualquer sistema operativo que precisamos.

A utilização do Javascript para aplicações de desktop é fácil graças a projetos como o Electron, que nos permite produzir aplicações multi-plataformas, ou seja, podem ser instaladas no Windows, Mac OS X e Linux. Existem algumas aplicações bem conhecidas desenvolvidas com a Electron, tais como Átomo, Visual Studio Code, Slack, Hyper, etc.

Javascript na web

O campo de ação do Javascript é muito vasto, fornecendo alternativas para quase qualquer ambiente de execução. No entanto, se queremos começar a aprender Javascript, o lugar mais adequado é a web.

Dentro da disciplina de desenvolvimento do lado do cliente (Javascript executado dentro do ambiente do navegador) podemos encontrar diferentes tipos de projetos, que também requerem diferentes abordagens e conhecimentos.

Websites

Quando nos referimos a sites aqui, referimo-nos a sites onde a parte mais importante são o conteúdo, sejam blogs, páginas de notícias e até mesmo e-commerce.

O Javascript nestes casos dedica-se a fornecer funcionalidade e interação, permitindo interfaces dinâmicas de utilizador, resposta a ações do utilizador, validação de formulários, etc.

Aplicações de página única

Nos últimos anos, a web tornou-se popular como uma plataforma para aplicações empresariais. As aplicações denominadas "gestão", que eram anteriormente executadas com programas de desktop, têm hoje frentes web que nos permitem utilizá-las a partir da nuvem, ou seja, de qualquer navegador ligado à Internet e sem a necessidade de instalar software na máquina. Neste tipo de aplicação, é comum transportar uma grande parte da carga de processamento do lado do servidor para o cliente. Neste novo paradigma, o navegador é responsável por fazer muito mais coisas do que em sites tradicionais, como criar o código HTML para visualizar os dados ou navegar entre ecrãs ou rotas de aplicações.

Nas Aplicações de Página Única, também conhecidas como SPA, é normal que o servidor entregue apenas os dados de negócios brutos e que o navegador faça todo o trabalho de apresentação desses dados num formato adequado (o HTML é produzido no navegador para representar esses dados). Mas o que mais caracteriza um SPA é que a navegação é sempre feita dentro da mesma página e o Javascript está encarregado de apresentar um ou outro ecrã ao utilizador sem ter de recarregar toda a página.

Os dois principais fatores que caracterizam as SPAs, 1) o facto de trazer dados brutos do servidor (mais leve) e 2) toda a navegação é feita dentro do mesmo documento, produzem aplicações web com uma resposta muito rápida, proporcionando uma experiência de utilização próxima da de uma aplicação de ambiente de trabalho.

Bibliotecas, quadros e desenvolvimento "Vanilla Javascript" para a web

A criação de um SPA é uma tarefa muito mais avançada do que o desenvolvimento de um website e para realizar este trabalho é importante que a equipa de desenvolvimento se baseie num quadro javascript, como angular, reagir, VueJS, Ember ou Polymer. .

Nota: Reagir e Polímero são considerados mais bibliotecas, mas com uma série de complementos, que eles próprios fornecem em muitos casos, oferecem tantas funcionalidades como as encontradas num quadro. Caso alguém ainda não saiba, uma biblioteca é um conjunto de funções, ou classes e objetos, que nos permite executar uma série de tarefas comuns para o desenvolvimento de determinadas necessidades de aplicação. Um quadro distingue-se principalmente de uma biblioteca porque, além de fornecer código para resolver problemas comuns, oferece uma arquitetura que os desenvolvedores devem seguir para produzir aplicações e garantir uma melhor qualidade de código e uma maior facilidade de manutenção. Por outras palavras, o quadro, além de oferecer vários utilitários, marca um estilo e fluxo de trabalho no desenvolvimento de aplicações.

Para o desenvolvimento de websites, seria geralmente suficiente para usar o Javascript também, sem a necessidade de depender de qualquer biblioteca adicional. Esse desenvolvimento javascript "puro" é geralmente conhecido como "Javascript de Baunilha".

Nota: Deve ficar claro que "Baunilha" não é uma marca registada ou qualquer sabor de Javascript para além da própria língua. É como uma piada para indicar que com o Javascript (e nada mais) você

pode resolver todas as coisas que bibliotecas e quadros prontos oferecem- lhe.

No entanto, também é comum que bibliotecas como o jQuery sejam usadas no desenvolvimento de websites. jQuery é um conjunto de objetos e funções (código de utilidade geral) que tem como objetivo ajudá-lo a manipular a página, guardando as diferenças entre diferentes navegadores e permitindo-lhe escrever um único código que funciona corretamente em qualquer cliente web. Além disso, o jQuery oferece-lhe muitas funções que são realmente úteis para o desenvolvimento de muitas tarefas comuns do site, que você pode usar mais rápido do que se você apenas trabalhar com Javascript.

Nota: No que diz respeito ao jQuery, vale a pena referir que existe uma corrente de desenvolvedores que avisam que usar o jQuery não é absolutamente necessário na maioria dos casos. Usar jQuery não é mau, mas muitas pessoas implementam-no para resolver necessidades que um pouco de Javascript "Baunilha" é capaz de fazer. Às vezes jQuery é carregado por padrão, talvez por conveniência, para se deixar levar ou simplesmente devido à ignorância da própria língua javascript, e no entanto muito poucas das suas funções são usadas.

Existem bibliotecas mais especializadas para resolver cada uma das coisas que uma biblioteca geral como o jQuery oferece e que assumem muito menos peso de download e tempo de processamento para os navegadores.

Mas para além das bibliotecas e dos quadros, temos de saber que hoje, para além da própria linguagem, existem muitas outras tecnologias baseadas em Javascript nos navegadores para resolver uma vasta gama de necessidades.

HTML5 APIs

Com o advento do HTML5, houve uma maior normalização dos navegadores, alcançando um compromisso por parte dos fabricantes de apoiar os idiomas web (HTML + CSS + Javascript) como ditam as suas especificações. Mas também produziu um fluxo abundante de novas especificações para trabalhar com a mais ampla gama de navegadores, computadores ou recursos de dispositivos.

HTML5 oferece APIs para trabalhar com elementos como a câmara, geolocalização, armazenamento, bitmap ou desenho de vetor, áudio, vídeo, etc. Tudo o que o HTML5 oferece está disponível em todos os navegadores e faz parte do kit de ferramentas do desenvolvedor Javascript, sem utilizar qualquer biblioteca ou estrutura.

Componentes Web

A próxima revolução de desenvolvimento para a Internet, que ainda não foi explorada em meados de 2017, chama-se Web Components. Baseia-se numa nova API (com várias especificações em conjunto) destinada a criar componentes. Os componentes são como novas tags HTML que qualquer desenvolvedor pode criar para resolver problemas comuns ou específicos de aplicação.

Com componentes web os desenvolvedores podem estender o HTML criando novos componentes capazes de fazer qualquer coisa e com capacidades avançadas de encapsulamento, para respeitar a sua autonomia e capaz de ser usado em qualquer projeto, maximizando as capacidades de reutilização de códigos sem a necessidade de se basear em qualquer tipo de biblioteca ou enquadramento.

Tal como o HTML5, os Componentes Web fazem parte das possibilidades oferecidas pelos navegadores por padrão, embora o seu suporte neste momento não seja tão universal.

Nota: Nos próximos meses, todos os navegadores devem suportar os Componentes Web V1 e podemos usá-lo sem quaisquer restrições. Entretanto existe um Polyfill que permite alargar o suporte aos Componentes Web aos navegadores que ainda não o têm. Os polyfills são literalmente "preenchimentos de lacunas", que compõem as deficiências dos navegadores antigos no que diz respeito às normas. Nas APIs HTML5 já começaram a ser utilizadas intensivamente para permitir a utilização de novas funcionalidades das idiomas webem clientes antigos ou ligeiramente atualizados.

Javascript como a primeira língua

Se está a aprender programação, o Javascript é uma excelente aposta para começar. Basicamente por três razões principais:

Fácil de usar

Só precisamos de um navegador para poder executar o Javascript. Não requer qualquer tipo de compilação (processo para criar um ficheiro binário executável para um determinado sistema operativo) mas é interpretado. Isto envolve um fluxo de trabalho mais racionalizado, facilitando os primeiros passos. Além disso, o Javascript é uma dactilografia dinâmica e a sua sintaxe é menos elaborada do que a de outras línguas e permite que as coisas sejam feitas de diferentes maneiras, de acordo com a habilidade, preferências ou costumes de cada programador.

Amplas áreas de aplicação

Como vimos, o Javascript tem utilizações praticamente ilimitadas. Significa que pode usar a linguagem praticamente para o que precisa. Ao aprender uma única língua, poderá alcançar qualquer propósito que definir para si mesmo.

É uma linguagem aberta e padrão para aweb:

Em algum momento da sua atividade profissional, a maioria dos desenvolvedores trabalhará no ambiente web. A programação de aprendizagem com o Javascript assegura-nos que o conhecimento será aplicado diretamente em vários pontos da carreira de procissão dos alunos.

Estas são as razões pelas quais ensinamos a programar com o Javascript há mais de 10 anos e porque é que em 2017 as prestigiadas universidades de Stanford abandonaram o seu curso introdutório para programação com Java a favor do Javascript.

Várias profissões começam com Javascript

Outra razão pela qual vale a pena começar pelo Javascript é porque a linguagem é um dos pilares fundamentais para realizar múltiplas profissões na procura no âmbito da web.

Desenvolvedor de frontend:

É o profissional que lida com o desenvolvimento do lado do cliente, embora possa ter várias atividades para além da programação, tais como design, layout, desenvolvimento de interface de utilizador, aplicações DE SPA do lado do cliente, etc. O conhecimento javascript é essencial em todas as áreas de ação de um desenvolvedor de frontend.

Desenvolvedor debackend:

Como backend, compreendemos a programação do lado do servidor. Para backend, a verdade é que Javascript (com NodeJS) é apenas uma das muitas possibilidades. No entanto, o NodeJS oferece muitas vantagens, como a sua assíncrodo, velocidade e otimização, tornando-o ideal para muitos tipos de projetos.

Desenvolvedorfullstack:

O desenvolvedor fullstack é capaz de trabalhar tanto no lado do cliente como do servidor. Na realidade é uma raça rara, tanto que muitas vezes se diz que ela não existe realmente, porque requer muito conhecimento do desenvolvedor e, portanto, é muito complicado, ou impossível, que um único perfil os una a todos. No entanto, na minha opinião, é uma realidade em muitos postos de trabalho, uma vez que há profissionais que devem ter conhecimento global em várias áreas (os freelancers autónomos são o principal exemplo) e, portanto, é um número que realmente existe. Hoje em dia, o facto de o Javascript servir tanto o cliente como o lado do servidor, torna a figura do desenvolvedor fullstack muito mais fácil e facilita a vida a milhares de desenvolvedores, uma vez que os poupa a fadiga mental de passar da linguagem para a linguagem. outro constantemente.

Nota: A especialização também está presente nestas profissões. É verdadeiramente incrível ver como os perfis se desenvolvem no mundo da web e como novas

profissões especializadas são geradas ano após ano. O que costumávamos chamar de frontend, simplesmente, hoje podemos subdividir em dezenas de perfis ou profissões específicas como "frontend engineer", "frontend web designer", "CSS architect", "mobile frontend developer", "frontend devops" ...

Como aprender Javascript

Aprender Javascript a um nível avançado leva meses de estudo, mas começar é fácil e em pouco tempo você será capaz de fazer coisas incríveis com pouco esforço. No manual Javascript poderá conhecer o idioma, mas para começar queremos dar-lhe algumas dicas.

Aprenda Javascript e não um derivado

A nossa primeira dica se está apenas a começar é aprender Javascript e não uma biblioteca ou enquadramento. No final, todos os projetos na web usam Javascript e tudo o que você pode fazer com uma biblioteca ou estrutura também pode ser feito com Javascript, por isso a coisa certa a fazer é começar por dominar a linguagem, e depois definir novos objetivos a médio ou longo prazo.

Em resumo, não aprenda jQuery: aprenda Javascript. Não aprendas Angular: aprenda Javascript, não aprenda Reagir: aprenda Javascript... Com uma base sólida de Javascript será muito mais fácil para você aprender mais tarde qualquer biblioteca ou enquadramento em que você deseja basear. Além disso, não terá problemas no futuro, quando quiser fazer coisas para as quais essa biblioteca não se destina ou pretende personalizar qualquer detalhe da sua aplicação.

O que deve acompanhar o conhecimento e a utilização do Javascript são as suas habilidades em tudo o que o navegador (a plataforma web) lhe oferece por padrão: HTML5 e Web Components. Tudo o que faz com base no Javascript "Vanilla", HTML5 e Componentes Web tem a certeza de que pode ser usado em qualquer projeto onde possa trabalhar, independentemente da biblioteca ou enquadramento que está a ser usado em cada caso.

Conhecimento da plataforma web

O nosso segundo conselho é que, se vai trabalhar neste meio, deve ter em conta cada uma das peculiaridades da plataforma web. Há muito conhecimento geral que precisa adquirir que lhe dará uma base para construir as suas habilidades no mundo do desenvolvimento web.

Conhecimento do meio: Deve saber o que é a Internet, a Web, o protocolo HTTP, o sistema de nome de domínio e, claro, as línguas fundamentais para especificar o conteúdo e forma: HTML e CSS.

Conhecimento de design: Embora o seu perfil possa ser mais um programador, é ideal ter uma visão, pelo menos técnica, das características de design para a web. A experiência do utilizador, a usabilidade, o design gráfico geral ou a acessibilidade são pontos importantes.

Conhecimento de programação: Se se vai dedicar à profissão de programação, não basta ter um conhecimento básico de código e programação estruturada, como é ideal interessar-se por programação, análise e design orientados para objetos, padrões de design, dados de bases, etc.

Conhecimento de ferramentas: O profissional deve também conhecer um grande grupo de ferramentas para o trabalho diário, desde o terminal da linha de comando e administração básica dos servidores, até à automatização de tarefas, através das ferramentas de controlo de versão (preferência Git) e otimização.

Diferenças entre Java e Javascript

Estamos a contar várias questões e curiosidades interessantes que servem de introdução ao Manual Javascript e queremos discutir uma das associações mais típicas que são feitas quando se ouve falar do Javascript. Queremos relacioná-lo com outra linguagem de programação, chamada Java, que não tem muito a ver com isso.

O Javascript foi realmente chamado assim porque o Netscape (o navegador que lançou o Javascript), que era aliado aos criadores da Java na época (Sun Microsystems), queria aproveitar a imagem de marca e popularidade de Java, como uma manobra para a expansão da sua nova linguagem. Ao todo, foi criado um produto com certas semelhanças, como a sintaxe linguística ou de nome. Fez-no entender que era um irmão mais novo e especificamente orientado para fazer coisas nas páginas web, mas também fez com que muitas pessoas caíssem no erro de pensar que são as mesmas.

Queremos deixar claro que o Javascript não tem nada a ver com Java, exceto com o acordo de colaboração dos criadores de ambas as línguas, como foi lido. Sempre foram produtos totalmente diferentes que não têm mais relação uns com os outros do que a sintaxe idêntica, embora, na verdade, a sintaxe de ambos seja herdada de outra língua popular chamada C.

Diferenças entre Java e Javascript

Diferenças mais notáveis entre Java e Javascript

Algumas das diferenças mais representativas entre estas duas línguas são as seguintes:

Compilador. Para programar em Java precisamos de um Kit de Desenvolvimento e um compilador. No entanto, o Javascript não é um idioma que exija que os seus programas sejam compilados, mas sim interpretados pelo navegador quando lê a página.

Orientado para objetos. Java é uma linguagem de programação orientada

aos objetos. (Mais tarde veremos o que significa orientado para objetos, para o qual ainda não sabe). O Javascript é uma linguagem "multi-paradigma", não requer programação orientada a objetos, embora o permita. Isto significa que poderemos programar em Javascrip sem criar aulas, como é feito em linguagens de programação estruturadas como C ou Pascal.

Finalidade. Java em princípio é muito mais poderoso que o Javascript, porque é uma língua de propósito geral. Com Java pode fazer aplicações das mais variadas, no entanto, com o Javascript só podemos escrever programas para executar em páginas web. Embora para dizer a verdade, desde a última década o Javascript expandiu-se tanto que hoje podemos usar o idioma para fazer aplicações de todos os tipos, como programas de consola, aplicações móveis, aplicações de desktop, etc. Portanto, hoje em dia é Javacript podemos dizer que é quase tão poderoso, ou mais, do que Java é.

Dactilografiaestática. Java é uma linguagem de programação fortemente dactilografada (também chamada de dactilografia estática). Isto significa que ao declarar uma variável em Java

teremos de indicar o seu tipo e não será capaz de mudar de um tipo para outro ao longo da execução do programa. Por seu lado, o Javascript não tem esta funcionalidade, mas é uma linguagem de dactilografia dinâmica (ou ligeiramente dactilografada) e podemos colocar a informação que queremos numa variável, independentemente do seu tipo. Além disso, podemos alterar o tipo de dados de uma variável quando queremos.

Língua aberta / linguagem proprietária

. Outra diferença importante é que o Javascript se baseia numa norma aberta, que não tem um proprietário específico e, portanto, qualquer fabricante pode implementá-lo livremente nos seus sistemas. No entanto, a Java é uma língua detida por uma empresa (atualmente Oracle), pelo que é dirigida com uma abordagem particular e comercial.

Outras características. Como vemos, Java é muito mais complexa, mas também mais poderosa e robusta. Inicialmente, Java tem mais funcionalidades do que o Javascript e requer uma aprendizagem muito mais intensa para dominá-lo. O Javascript facilita a aprendizagem, mesmo para pessoas sem experiência de programação, e permite-lhe fazer programas rapidamente, obtendo resultados bastante atrativos com pouco código e esforço.

As diferenças que separam Java do Javascript são, portanto, notáveis. Também são usados para coisas muito diferentes. A Java é mais pensada para fazer aplicações complexas, orientadas para o mundo dos negócios, ou aplicações para telefones Android. O Javascript destina-se a tornar as aplicações mais pesadas, principalmente orientadas para a Web.

Queremos salientar que atualmente a abordagem web Javascript não tem de ser considerada única. Como dissemos, é possível fazer quase todo o tipo de programa usando o Javascript e nos últimos anos tem vindo a ocupar cada vez mais parcelas. Além disso, à medida que foram introduzidas novas versões da norma Javascript, tornou-se mais robusto e adequado também para grandes aplicações.

Antes de começar

Há vários pontos que queremos comentar como uma introdução ao Manual javascript e que talvez queira saber antes de começar a programar. Em primeiro lugar, seria bom ter uma ideia mais concreta das possíveis aplicações que a língua poderia ter e que podem ser encontradas em inúmeros sites. Além disso, queremos também discutir as ferramentas e conhecimentos prévios de que precisamos para começar a trabalhar.

Utilizações do Javascript

Talvez hoje seja supérfluo dizer para que é o Javascript, mas vamos ver brevemente alguns usos desta linguagem que podemos encontrar na web para ter uma ideia das possibilidades que tem.

Sem ir mais longe, DesarrolloWeb.com usa o Javascript para o menu superior, que mostra diferentes links dentro de cada opção principal. Estamos a mudar a página de vez em quando, mas no design atual deste site, elementos como a caixa "Login" também têm o seu dinamismo com o Javascript.

Atualmente quase todas as páginas ligeiramente avançadas usam Javascript, uma vez que se tornou uma das insígnias do que é chamado Web 2.0 e a rica experiência do utilizador. Por exemplo, sites populares como Facebook, Twitter ou Youtube usam Javascript em abundância. Para ser mais específico, quando clicamos num link na rede social para comentar algo, um pequeno formulário é exibido na página que aparece como se fosse magia e depois é enviado sem sair da própria página. Mesmo quando votamos num vídeo no YouTube ou quando os personagens que escrevemos nas mini-publicações do Twitter são contados, o Javascript é usado para executar pequenas funcionalidades que não podem ser feitas apenas com HTML. Pode ver exemplos javascript dentro de qualquer página ligeiramente complexa. Alguns que teremos visto inúmeras vezes são calendários dinâmicos para selecionar datas, calculadoras ou conversores de moeda, editores de texto ricos, navegadores dinâmicos, etc.

É muito mais comum encontrar Javascript para realizar efeitos simples em páginas web, ou não tão simples, como navegadores dinâmicos, abrir janelas secundárias, validação de formulários, etc. Atrevemo-nos a dizer que esta linguagem é realmente útil nestes casos, uma vez que estes efeitos típicos têm complexidade

suficiente para serem implementados em questão de minutos sem a possibilidade de erros. No entanto, para além destes exemplos simples, podemos encontrar muitas aplicações na Internet que baseiam parte do seu funcionamento no Javascript, que fazem com que uma página web se torne um verdadeiro programa interativo para gerir qualquer recurso. Exemplos claros são aplicações de escritório on-line, tais como Google Docs, Office Online ou Google Calendar.

O que precisa para trabalhar com Javascript

Para programar em Javascript precisamos basicamente o mesmo que desenvolver páginas web com HTML. Um editor de texto e um navegador compatível com Javascript. Qualquer computador minimamente atual tem tudo o que precisa para ser capaz de programar em Javascrip. Por exemplo, um utilizador do Windows tem dentro da sua instalação típica do sistema operativo, um editor de texto, um Notepad e um navegador: Internet Explorer.

Conhecimento prévio recomendado

A verdade é que não é preciso muita base de conhecimento para começar a programação no Javascript. Muito provavelmente, se ler estas linhas já sabe tudo o que precisa para trabalhar, já que já terá tido alguma relação com o desenvolvimento de websites e terá detetado que para fazer certas coisas é bom saber um pouco de Javascript.

No entanto, seria bom ter um comando avançado de HTML, pelo menos o suficiente para escrever código nessa língua sem ter que pensar no que está a fazer. Um conhecimento médio de CSS e talvez alguma experiência anterior em alguma linguagem de programação também será útil, embora neste manual de DesarrolloWeb.com tentaremos explicar o Javascript mesmo para pessoas que nunca programaram.

Capítulo I

Versões de navegador e Javascript

Para continuar com a introdução ao Javascript, também é apropriado introduzir as diferentes versões do Javascript que existem e que evoluíram juntamente com as versões dos navegadores. A linguagem tem progredido durante os seus anos de vida e aumentando as suas capacidades.

Este é um conhecimento interessante, uma vez que quando nos desenvolvemos com o Javascript dependemos diretamente da plataforma de execução e da compatibilidade ou não com alternativas linguísticas modernas.

Já para o alertar desde o início, neste Manual Javascript vamos trabalhar com versões totalmente alargadas do idioma em todos os navegadores, pelo que tudo o que explicarmos pode ser aplicado sem qualquer problema.

Versões de linguagem javascript

A evolução do Javascript

No artigo da introdução ao Javascript já explicámos algumas das histórias da língua. Vimos que no início os seus objetivos eram simples e que hoje em dia com o Javascript podemos conseguir criar páginas realmente complexas, interfaces de utilizador e efeitos. Portanto, à medida que as exigências dos desenvolvedores cresciam, também a própria linguagem.

É importante mencionar que a linguagem evoluiu de duasformas:

Por um lado, a própria linguagem tem vindo a incorporar operadores, estruturas de controlo, regras de sintaxe para fazer coisas repetitivas com menos código. Devemos estas melhorias na língua à norma ECMASCript, de que falaremos imediatamente.

Por outro lado, os navegadores têm vindo a incorporar novas instruções, para serem capazes de manipular elementos modernos da página, tais como divisões, estilos e sistemas CSS tais como armazenamento local, trabalho de tela completa, geolocalização e um longo etc. Todas estas melhorias são definidas de acordo com os padrões de HTML 5 e as suas APIs Javascript.

Em todo o caso, o que deve ser claro é que o Javascript tem vindo a melhorar as suas funcionalidades como idioma, enquanto os navegadores têm vindo a oferecer um maior suporte a funcionalidades avançadas para o controlo de elementos de página. É por isso que no Javascript geralmente temos de ter cuidado com o mercado do navegador e com a compatibilidade com as funcionalidades que queremos usar.

Norma ECMASCript

O javascript como língua é padronizado pela organização "ECMA International". Esta empresa dedica-se a criar padrões de comunicação e processamento de informação. Um dos seus padrões mais conhecidos é o ECMAScript.

ECMAScript é a definição padrão da língua Javascript, que qualquer cliente que suporte o Javascript deve suportar. Ao longo do tempo, foram publicadas diferentes versões do ECMAScript, com o mais recente ECMAScript 2015, também conhecido como ES6.

O ES6 é suportado por todos os navegadores atuais, exceto o Internet Explorer. Portanto, se apoiar programas IE, não poderia utilizar o ES6 em princípio.

Isto é meio verdade, uma vez que existem sistemas que permitem traduzir o código de uma versão para outra da norma, embora de momento não queiramos entrar nestes detalhes. Se estiver interessado, recomendamos a leitura do Manual do Webpack, que é uma das muitas ferramentas que pode utilizar para realizar esta tarefa.

HTML 5 APIs

Uma API é uma interface de programação de aplicações. Basicamente, indica como as funcionalidades ou serviços de um sistema vão ser acedidos, através das instruções e que procedimentos.

Os navegadores definem uma API para aceder aos recursos que temos para manipular o estado da página e de acordo com periféricos, armazenamento, etc.

Os navegadores APIs são geralmente especificados pelo W3C e devem ser fielmente implementados por todos os clientes web.

Acontece um pouco o mesmo que com as versões linguísticas. Se o navegador é muito antigo, pode não ter suporte para algumas APIs, por isso temos de ser cautelosos. Na prática, mais do que tudo com o Internet Explorer e algum navegador móvel antigo.

Nesta primeira parte do manual não vamos cobrir as APIs HTML 5 ou o acesso aos recursos do navegador. Esta área é objeto de estudo na segunda parte, dedicada ao Desenvolvimento Javascript do lado do Cliente.

Versões linguísticas

Na primeira parte deste manual o objetivo é aprender bem o Javascript, pelo que nos limitaremos ao estudo da linguagem em particular. Neste sentido, vamos usar o ES5, que é a versão do idioma que funciona em todos os navegadores atuais.

Como informação geral, resumiremos que comentaremos as diferentes versões do Javascript:

Realmente, qualquer navegador moderadamente moderno terá agora todas as funcionalidades Javascript que vamos precisar e acima de tudo, aquelas que podemos usar nos nossos primeiros passos com o idioma. No entanto, pode ser útil conhecer as primeiras versões do Javascript que discutimos neste artigo, como uma curiosidade.

Javascript 1: Nasceu com Netscape 2.0 e apoiou um grande número de instruções e funções, quase tudo o que existe agora já foi introduzido no primeiro padrão.

Javascript 1.1: É a versão Javascript que foi desenhada com a chegada de navegadores 3.0. Implementou pouco mais do que a sua versão anterior, como processar imagens dinamicamente e criar matrizes.

Javascript 1.2: A versão dos navegadores 4.0. Isto tem a desvantagem de ser um pouco diferente nas plataformas da Microsoft e netscape, uma vez que ambos os navegadores cresceram de forma diferente e estavam em plena luta pelo mercado.

Javascript 1.3: Versão implementada pelos navegadores 5.0. Nesta versão, algumas diferenças e arestas ásperas entre os dois

navegadores foram arquivadas.

Javascript 1.5: Versão atual, no momento desta escrita, implementada pela Netscape 6.

Por seu lado, a Microsoft também evoluiu para apresentar a sua versão 5.5 do JScript (é o que chamam de javascript usado pelos navegadores da Microsoft).

Em 2009 foi publicada a versão do ECMAScript 5. Esta é a versão com que vamos lidar principalmente no manual, que funciona em todos os navegadores do mercado.

Em 2015 é lançada a sexta versão do ECMAScript, com um número muito significativo de novas funcionalidades. Esta versão é totalmente utilizável, exceto no Internet Explorer. As suas melhorias são tão importantes que todos os desenvolvedores profissionais as utilizam e, se necessário, o código é traduzido para suportar navegadores mais antigos. Esta versão é explicada no Manual ES6.

Existe atualmente o compromisso de lançar uma versão da norma ecmascript por ano. Portanto, existem vários padrões com pequenas alterações incrementais, que foram incorporadas a diferentes taxas entre navegadores.

Conclusão das versões javascript e compatibilidade

É óbvio que, depois de escrever estas linhas, muitas outras versões do Javascript serão apresentadas ou terão sido apresentadas, porque, à medida que os navegadores são melhorados e as versões HTML são lançadas, surgem novas necessidades para a programação de elementos dinâmicos. No entanto, tudo o que vamos aprender neste manual, usos ainda mais avançados, já está implementado em qualquer Javascript que exista hoje.

Capítulo II

Efeitos rápidos com Javascript

Antes de mergulhar no assunto, podemos ver uma série de efeitos rápidos que podem ser programados com Javascript, o que pode nos dar uma ideia mais clara das capacidades e poder da linguagem. Abaixo veremos vários exemplos, que realçamos para esta introdução no Manual Javascript, por terem um mínimo de complexidade e, apesar de serem muito básicos, virão a calhar para terem uma ideia mais precisa do que é o Javascript quando se trata de percorrer os capítulos seguintes.

Abra uma janela para crianças

Primeiro vamos ver que com uma linha javascript podemos fazer coisas bastante atraentes. Por exemplo, podemos ver como abrir uma janela secundária sem barras de menu que mostram o motor de pesquisa do Google. O código seria o seguinte.

```
<script>

window.open ("http://www.google.com", "", "width =
550, height = 420, menubar = no")

</script>
```

Uma mensagem de boas-vindas

Poderemos apresentar uma caixa de texto pop-up após a conclusão do carregamento da capa do nosso site, que poderá receber os visitantes.

```
<script>

window.alert ("Welcome to my website. Thanks ...")

</script>
```

Data atual

Agora vamos olhar para um guião simples para mostrar a data de hoje. Por vezes é muito interessante mostrá-lo nos websites para dar um efeito de que a página está "atualizada", ou seja, é atualizada.

```
<script> document.write (new Date ()) </script>
```

Estas linhas devem ser inseridas dentro do corpo da página onde queremos que a última data de atualização apareça. Podemos ver o exemplo a correr aqui.

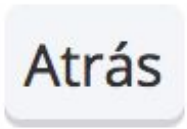
Nota: Um detalhe a destacar é que a data aparece num formato um pouco estranho, indicando também a hora e outros atributos do mesmo, mas vamos aprender a obter exatamente o que queremos no formato correto.

Botão de trás

Outro exemplo rápido pode ser visto abaixo. É um botão para voltar, como o que temos na barra de ferramentas do navegador. Agora veremos uma linha de código que mistura HTML e Javascript para criar este botão que mostra a página anterior na história, se houver.

```
<input type = button value = Back onclick = "history.go
(-1)">
```

O botão seria semelhante ao seguinte, um botão normal com a aparência padrão que o navegador e o sistema operativo que utiliza dão os botões. Abaixo está uma imagem de como seria o botão no meu sistema.



Em diferença com os exemplos anteriores, deve notar-se que neste caso a instrução Javascript está dentro de um atributo HTML, onclick, o que indica que esta instrução tem de ser executada em resposta ao clique do botão.

A facilidade com que algumas ações interessantes podem ser levadas a cabo foi verificada. Como podem imaginar, haveria muitas outras amostras javascript simples que reservamos para capítulos posteriores.

Língua javascript

Vamos ver como inserir o código Javascript num documento HTML. Neste artigo vamos explicar algumas regras importantes e as formas mais básicas de executar o Javascript no contexto de uma página.

Nesta parte do manual sobre Javascript vamos conhecer a forma mais básica de trabalhar com a linguagem. Neste artigo daremos as primeiras informações sobre como incluir scripts, misturando o próprio código Javascript com o HTML.

Então também veremos como o código deve ser colocado para que o nosso site seja compatível com todos os navegadores, mesmo aqueles que não suportam Javascript. Muitas ideias de como o Javascript funciona já foram descritas em capítulos anteriores, mas com o objetivo de não deixar nada no oleoduto, tentaremos agarrar daqui todos os dados importantes desta língua.

Javascript é escrito para o documento HTML

A coisa mais importante e básica que podemos destacar neste momento é que a programação javascript é feita dentro do próprio documento HTML. Ou seja, o código Javascript, na maioria dos casos, é misturado com o próprio código HTML para gerar a página.

Isto significa que temos de aprender a misturar as duas linguagens de programação e rapidamente veremos que, para que estas duas línguas possam coexistir sem problemas entre elas, devem ser incluídos os delimiters que separam as etiquetas HTML das instruções javascript. Estes delimiters são as etiquetas <SCRIPT> e </SCRIPT> Todo o código Javascript que colocamos na página tem de ser inserido entre estas duas tags.

A colocação dos scripts importa

Podemos introduzir vários scripts na mesma página, cada um que poderia ser inserido dentro de diferentes etiquetas <SCRIPT> A colocação destes scripts não é irrelevante. No início, com o que sabemos até agora e os scripts que fizemos para testes, dá-nos um pouco o mesmo onde colocá-los, mas em certos casos esta colocação será muito importante. Em cada caso, e quando chegar a hora, será reportado em conformidade.

O Javascript também pode ser escrito dentro de certos atributos de página, como o atributo onclick. Estes atributos estão relacionados com as ações do utilizador e são chamados manipuladores de eventos.

Abaixo vamos olhar mais de perto para estas duas formas de escrever scripts, que têm como principal diferença o momento em que as frases são executadas.

Formas de escrever scripts javascript

Até agora, no Manual Javascript, já tivemos a oportunidade de experimentar alguns scripts simples, no entanto, ainda temos de aprender uma das bases para poder trabalhar com a língua e que é aprender as duas formas de executar o código Javascript. Há duas maneiras fundamentais de executar scripts na página. A primeira destas formas é a execução direta do script, a segunda é a execução em resposta à ação do utilizador.

Vamos agora explicar cada uma destas formas de execução disponíveis, mas para aqueles que o querem, também recomendamos ver o vídeo sobre Formas de incluir e executar scripts.

Execução direta do código Javascript

É o método mais básico de executar scripts. Neste caso, as instruções estão incluídas no <SCRIPT>tag, como já comentamos anteriormente. Quando o navegador lê a página e encontra um script, interpreta as linhas de código e executa-as uma após a outra. Chamamos assim execução direta porque quando a página é lida os scripts são executados diretamente.

```
<! DOCTYPE html>

<html lang = "is">

<head>

  <meta charset = "UTF-8">

  <meta name = "viewport" content = "width = device-width,
initial-scale = 1.0">

  <meta http-equiv = "X-UA-Compatible" content = "ie = edge">

  <title> Example of direct execution </title>

</head>

<body>

    <h1> Page with Javascript </h1>

    <p> This page has a dialog box, which will be displayed
as soon as the browser processes it. </p>



    <script>

      var people = 4;

      var amountEntrance = 9.50;

      alert ('You need' + people * ticket amount + 'euros for
everyone to enter the cinema');

    </script>



    <p> When the user clicks accept in the dialog box, the
browser will display the full page. </p>

</body>

</html>
```

O código Javascript será executado à medida que a página abrir. Quando o navegador, ao processar a página, encontrar este código, deixará de ler a página para executar o script Javascript. Como resultado, vai mostrar numa caixa de diálogo "É preciso 38 euros para que todos entrem no cinema". Quando o utilizador clicar no botão "aceitar", continuarão a ler o resto da página e a exibir o conteúdo de toda a página na janela do navegador. É claro que veremos muitos outros exemplos ao longo do manual.

Este método será o que utilizamos preferencialmente na maioria dos exemplos nesta parte do Manual javascript. Na segunda parte do Manual javascript podemos aprender muitas coisas e entre elas veremos em detalhe a segunda forma de executar scripts que vamos relacionar abaixo.

Correndo javascript em resposta a um evento

É a outra maneira de executar scripts, mas antes de vermos, temos de falar sobre os acontecimentos. Os eventos são ações realizadas pelo utilizador. Programas como o Javascript estão preparados para capturar certas ações realizadas, neste caso na página, e realizar ações em resposta. Desta forma, podem ser realizados programas interativos, uma vez que controlamos os movimentos do utilizador e respondemos a eles. Existem muitos tipos diferentes de eventos, por exemplo, o premir de um botão, o movimento do rato, ou a seleção de texto na página.

As ações que queremos realizar em resposta a um evento podem ser indicadas de várias maneiras, por exemplo dentro do mesmo código HTML, em atributos que são colocados dentro da etiqueta que queremos responder às ações do utilizador. No capítulo em que vimos um exemplo rápido, já verificamos que se quiséssemos um botão para executar ações quando foi clicado, tínhamos de as indicar dentro do atributo onclick do botão.

Por conseguinte, verificamos se o código Javascript pode ser inserido em certos atributos de tags HTML. No entanto, não é o único método possível. Também podemos selecionar elementos de página diretamente com javascript e funções associadas que serão executadas em resposta a eventos. Embora seja um pouco cedo para que possa entender todo este processo em detalhe, vamos ver um exemplo simples.

```
<! DOCTYPE html>

<html lang = "is">

<head>

  <meta charset = "UTF-8">

  <meta name = "viewport" content = "width = device-width,
initial-scale = 1.0">

  <meta http-equiv = "X-UA-Compatible" content = "ie = edge">

  <title> Example hover over </title>

</head>

<body>

    <h1> Javascript Example </h1>

    <span id = "honeydew"> Mouse over here </span>

    <script>

    var passes = 0;

    function announcePast () {

      passes = passes + 1;

      alert ('Mouse over' + passes + 'times');

    }

    document.getElementById ('my element'). addEventListener
('mouseenter', announcePast);

    </script>

</body>

</html>
```

Desta vez tem o código para uma página inteira. Nele pode encontrar o código Javascript incorporado dentro do corpo da página. Ao contrário do código anterior, quando o navegador lê a página não executa nada. Simplesmente memoriza o script e associa a função ao elemento que foi definido. Especificamente, definimos um evento do tipo "mouseenter" num elemento da página que tem um identificador "o meu elemento". Isto significa que cada vez que o utilizador coloca o ponteiro do rato sobre o elemento "o meu elemento", executará a função "anunciarPasses".

Como dizemos, talvez seja um pouco cedo para ver exemplos deste estilo, por isso não se preocupe se não entendeu tudo. Estamos apenas a começar o manual Javascript e poderás aprender tudo isto e muito mais pouco a pouco. Assim, veremos mais tarde este tipo de execução aprofundada e os tipos de eventos que existem. Para lá chegar, ainda temos de aprender muitas outras coisas com o Javascript. No próximo artigo, mostraremos como podemos esconder o código Javascript para navegadores antigos.

Esconda scripts javascript em navegadores antigos

Ao longo dos capítulos anteriores do Manual Javascript, já vimos que a língua foi implementada a partir do Netscape 2.0 e do Internet Explorer 3.0. Mesmo para quem não sabe, não faz mal dizer que existem navegadores que funcionam em sistemas operativos, onde apenas é possível exibir texto e onde certas tecnologias e aplicações não estão disponíveis, como o uso de imagens, diferentes fontes tipográficas ou o próprio Javascript. .

Assim, nem todos os navegadores que podem ser utilizados pelos utilizadores que visitam a nossa página entendem o Javascript. Nos casos em que os scripts não são interpretados, os navegadores assumem que o seu código é texto da própria página e, consequentemente, apresentam os scripts no corpo do documento, como se fosse texto normal. Para evitar que o texto do script seja escrito na página quando os navegadores não o entendem, tem de escondê-los com comentários HTML (<! - comentário HTML ->). Além disso, neste artigo também veremos como exibir uma mensagem que é vista apenas em navegadores que não suportam Javascript.

Atualizado: Neste momento podemos dizer que quase 100% dos navegadores disponíveis suportam o Javascript, ou pelo menos reconhecem as etiquetas de script, pelo que, mesmo que seja desativado, não mostrarão o texto dos nossos programas Javascript. Portanto, atualmente já não é essencial realizar o funcionamento da ocultação do código dos scripts de página para navegadores antigos. No entanto, se quisermos fazer uma página completamente correta, será útil aprender a esconder um script para que nunca seja exibido como texto na página.

Ocultar código Javascript com comentários HTML

Vejamos um exemplo de código onde os comentários HTML têm sido usados para esconder o Javascript, ou melhor dito, o código dos scripts Javascript.

```
<SCRIPT>

<! -

Javascript code

// ->

</SCRIPT>
```

Vemos que o início do comentário HTML é idêntico ao que o conhecemos em HTML, mas o encerramento do comentário tem uma peculiaridade, que começa com um duplo corte. Isto porque o final do comentário contém vários caracteres que o Javascript reconhece como operadores e ao tentar analisá-los, lança uma mensagem de erro de sintaxe. Para que o Javascript não lance uma mensagem de erro, esta barra dupla é colocada antes do comentário HTML, que não é mais do que um comentário Javascript, que saberemos mais tarde quando falarmos de sintaxe.

O início do comentário HTML não é necessário comentá-lo com a barra dupla, uma vez que o Javascript entende bem que se destina simplesmente a ocultar o código. Um esclarecimento neste momento: se colocarmos as duas barras nesta linha, elas seriam vistas em navegadores antigos como estando fora dos comentários html. <SCRIPT> as tags não são entendidas por navegadores mais

antigos, portanto não são interpretadas, pois são com qualquer etiqueta desconhecida.

Mostrar uma mensagem para navegadores antigos com <NOSCRIPT>

Existe a possibilidade de indicar um texto alternativo para navegadores que não compreendam o Javascript, para informá-los de que um script deve ser executado nesse local e que a página não está funcionando a 100% das suas capacidades. Também podemos sugerir que os visitantes atualizem o seu navegador para uma versão compatível com o idioma. Para isso usamos a etiqueta <NOSCRIPT> e entre esta etiqueta e o seu fecho correspondente podemos colocar o texto alternativo ao script.

```
<SCRIPT>

javascript code

</SCRIPT>

<NOSCRIPT>

This browser does not understand the scripts that are
running, you must update your browser version to a
newer one.

<br> <br>

<a href=http://netscape.com> Netscape </a>. <br>

<a href=http://microsoft.com> Microsoft </a>.

</NOSCRIPT>
```

Mais sobre scripting

Neste artigo vamos mostrar um dos atributos que podem ser indicados na etiqueta SCRIPT, o que indica o idioma que vamos usar. Além disso, mostraremos outra forma muito útil de associar o código Javascript na página, através de um ficheiro externo. Este ponto é fundamental e teremos de prestar especial atenção, uma vez que é sem dúvida a forma de trabalhar que é mais utilizada.

Métodos interessantes de colocação de scripts javascript

Indicar a linguagem que estamos a usar

Atualizado: Hoje já não é necessário especificar o idioma de script que estamos a usar. A razão é que o Javascript é a única linguagem aceite pela indústria para scripts em páginas web. Portanto, o atributo "linguagem" é realmente desnecessário.

A etiqueta <SCRIPT> tem um atributo que é usado para indicar o idioma que estamos a usar, bem como a sua versão. Por exemplo, podemos indicar que estamos a programar no Javascript 1.2 ou no Visual Basic Script, que é outra linguagem para programar scripts no navegador cliente que só é compatível com o Internet Explorer.

O atributo em questão é "linguagem" e o mais comum é simplesmente indicar a língua em que os scripts foram programados. O idioma predefinido é Javascript, por isso, se não usarmos este atributo, o navegador entenderá que o idioma com o qual está a ser programado é Javascript. Um detalhe em que as pessoas muitas vezes cometem erros sem se aperceberem é que a linguagem é escrita com dois -g- e não com -g- e com -j- como em espanhol.

```
<script language = "javascript">
```

Utilização do atributo "tipo":

Quando colocamos uma etiqueta SCRIPT, devemos usar o atributo "tipo" para indicar que tipo de codificação de script estamos a fazer e o idioma utilizado.

<script type = "text / javascript">

O atributo "tipo" é necessário para validar corretamente o seu documento nas versões mais atuais de HTML.

Nota nas versões HTML: Com o advento do HTML5, a necessidade de especificar o idioma e o tipo (atributos de idioma e tipo) foi removido, portanto, com a etiqueta de script é mais do que suficiente para abrir um bloco de código Javascript. No entanto, nas versões anteriores de HTML somos obrigados a definir o atributo "tipo". Embora em HTML 4.01 transitório validemos corretamente o atributo idioma, não validará se estivermos a fazer HTML rigoroso, pelo que não recomendamos a utilização de "linguagem" em qualquer caso. Nos exemplos de DesarrolloWeb.com onde a língua foi usada, por favor ignore-a.

Inclui ficheiros javascript externos

Outra forma de incluir scripts em páginas web, implementados a partir do Javascript 1.1, é incluir ficheiros externos onde pode colocar muitas funções que são usadas na página. Os ficheiros geralmente têm uma extensão .js e são incluídos desta forma.

```
<script src = "external_file.js"> </script>
```

Dentro das etiquetas <SCRIPT> qualquer texto pode ser escrito e será ignorado pelo navegador, no entanto, os navegadores que não entenderem o atributo SRC terão este texto por instruções. É muito raro que isso aconteça na paisagem do navegador. Portanto, não é realmente aconselhável colocar qualquer código Javascript dentro de um bloco de scripts que estamos usando para incluir um ficheiro externo.

O ficheiro que incluímos (neste caso "external_file.js") deve conter apenas declarações javascript. Não devemos incluir código HTML de qualquer tipo, nem mesmo as etiquetas </SCRIPT> e </SCRIPT>

A importância da utilização de ficheiros de código javascript externos

Ao nível didático neste manual usamos um monte de prática de

Inclui o código Javascript dentro do próprio documento HTML. É algo que vem muito confortável para expressar pequenos exemplos javascript. No entanto, a nível profissional não é uma boa prática, mas a recomendação é colocar todos, ou a maioria do seu código Javascript em ficheiros ".js" que inclui externamente. Existem várias razões para recomendar esta prática, entre as quais podemos destacar:

Do ponto de vista da separação de códigos, cada ficheiro deve ter apenas uma língua. Os documentos HTML são colocados em ficheiros ".html", documentos CSS em ficheiros ".css" e Javascript também separados nos seus ficheiros ".js".

O ficheiro externo permite que o navegador o cache, por isso, quando a página é revisitada, o código já está no navegador e não tem de o rebaalar. Este ponto é especialmente importante quando temos muitas páginas que carregam o mesmo código Javascript, uma vez que o navegador irá realmente descarregar o ficheiro ".js" uma vez e as outras vezes irão consumi-lo a partir da cache, poupando transferência e aumentando a velocidade de carregamento.

Quando tiver noções mais avançadas sobre o Javascript verá que na etiqueta de script pode usar atributos como "adiar" ou "async". Ambos os atributos fazem com que o navegador não pare de executar um script externo que esteja no meio do conteúdo HTML, mas continue a analisar a página e a renderizar o seu conteúdo enquanto o ficheiro é descarregado. Portanto, ambos os atributos otimizam a carga da página e melhoram a velocidade com que o navegador apresenta o conteúdo ao utilizador.

Nota: Quando se utiliza "adiar" a carga do ficheiro é feita em paralelo com a análise do HTML e de outros ficheiros externos, mas a execução é adiada até que o navegador termine de analisar e renderizar a página. Quando se usa "async" significa que o script é descarregado ao mesmo tempo que a página é analisada e que, uma vez descarregado, é executado, embora não tenha demorado muito a analisar completamente a página.

Um exemplo de carga e execução do Javascript diferido que obtemos assim.

```
<script src = "deferred_external_file.js" defer> </script>
```

Tendo em conta estes outros usos interessantes que existem no Javascript e que temos de saber para tirar partido das possibilidades da tecnologia, devemos ter aprendido tudo o que é essencial para começar a trabalhar com o Javascript no contexto de uma página web.

Capítulo III

Sintaxe javascript

Finalmente começamos a ver código fonte Javascript! Esperamos que todas as informações anteriores do Manual Javascript tenham sido assimiladas, nas quais basicamente aprendemos várias formas de incluir scripts em páginas web. Até agora, tudo o que vimos neste manual pode ter parecido muito teórico, mas a partir de agora esperamos que seja mais agradável começar a ver mais prático e diretamente relacionado com a programação.

A língua javascript tem uma sintaxe muito semelhante a Java porque é baseada nele. É também muito semelhante ao da língua C, por isso, se o leitor conhecer alguma destas duas línguas, será fácil de manusear com o código. De qualquer forma, nos seguintes capítulos vamos descrever cuidadosamente toda a sintaxe, para que os novatos não tenham qualquer problema com isso.

Comentários no código

Um comentário é uma parte do código que não é interpretado pelo navegador e cujo utilitário está em facilitar a leitura do programador. O programador, à medida que desenvolve o guião, deixa frases ou palavras individuais, chamadas de comentários, que o ajudam a ler o guião mais facilmente ao modificá-lo ou depurá-lo.

Alguns comentários javascript já foram vistos anteriormente, mas agora vamos contá-los novamente. Há dois tipos de comentários na língua. Um deles, o bar duplo, é usado para comentar uma linha de código. O outro comentário pode ser usado para comentar em várias linhas e é indicado com os sinais / * para iniciar o comentário e * / para ter fim. Vamos ver alguns exemplos.

```
<SCRIPT>
// This is a one line comment
/ * This comment can be extended
along several lines.
The ones you want * /
</SCRIPT>
```

Maiúsão e minúscula

Nas letras maiúsculas e minúsculas javascript devem ser respeitadas. Se cometermos um erro ao usá-los, o navegador responderá com uma mensagem de erro, sintaxe ou referência indefinida.

Por exemplo, a função alerta () não é a mesma da função Alerta (). O primeiro exibe texto numa caixa de diálogo, e o segundo (com o primeiro A maiúscula) simplesmente não existe, a menos que nós mesmos o definimos. Como pode ver, para que a função reconheça o Javascript, tem de escrever todas as minúsculas. Veremos outro exemplo claro quando lidarmos com variáveis, uma vez que os nomes que damos às variáveis também são sensíveis a casos.

Regra geral, os nomes das coisas em Javascript são sempre escritos em minúsculas, a menos que seja usado um nome com mais de uma palavra, uma vez que, neste caso, as iniciais das palavras seguintes à primeira serão capitalizadas. Por exemplo, document.bgColor (que é um lugar onde a cor de fundo da página web é guardada), é escrito com o maiúscula "C", uma vez que é a primeira letra da segunda palavra. Também pode usar letras maiúsculas nas iniciais das primeiras palavras em alguns casos, como os nomes das aulas, embora veremos mais tarde quais são estes casos e quais são as aulas.

Separação de instruções

As diferentes instruções que os nossos scripts contêm devem ser convenientemente separadas para que o navegador não indique os erros de sintaxe correspondentes. Javascript tem duas formas de separar as instruções. ;): O primeiro é através do ponto e vírgula personagem e a segunda é através de uma quebra de linha.

Por esta razão, as declarações javascript não precisam de terminar com um ponto e vírgula a menos que coloquemos duas instruções na mesma linha.

Não é má ideia, de qualquer forma, habituarmo-nos a usar o ponto e vírgula após cada instrução, porque outras línguas como Java ou C obrigam-nos a serem usadas e vamos habituar-nos a fazer uma sintaxe mais semelhante à habitual em ambientes de programação avançados.

Variáveis em Javascript

Este é o primeiro dos artigos que vamos dedicar a variáveis em Javascript dentro do Manual javascript. Veremos, se ainda não sabemos, que as variáveis são um dos elementos fundamentais na realização de programas, no Javascript e na maioria das linguagens de programação existentes.

Então vamos começar por conhecer o conceito de uma variável e vamos aprender a declará-los em Javascript, juntamente com explicações detalhadas sobre o seu uso na língua.

Conceito variável

Uma variável é um espaço na memória onde os dados são armazenados, um espaço onde podemos armazenar qualquer tipo de informação que precisemos para realizar as ações dos nossos programas. Podemos pensar nisso como uma caixa, onde armazenamos dados. Essa caixa tem um nome, para que mais tarde possamos nos referir à variável, recuperar os dados e atribuir um valor à variável sempre que quisermos.

Por exemplo, se o nosso programa realizar somas, será muito normal para nós armazenar em variáveis as diferentes adições que participam na operação e o resultado da soma. O efeito seria algo assim.

adding1 = 23

adding2 = 33

sum = adding1 + adding2

Neste exemplo temos três variáveis, adicionando1, adicionando2 e soma, onde poupamos o resultado. Vemos que a sua utilização para nós é como se tivéssemos uma secção onde guardar dados e que eles podem ser acedidos apenas colocando o seu nome.

Regras para o nome variável em Javascript

Os nomes variáveis devem ser construídos com caracteres alfanuméricos (números e letras), o sublinhado ou sublinhado (_) e o caractere dólar $. Para além disso, existem várias regras adicionais para a construção de nomes para variáveis. O mais importante é que não podem começar com um carácter numérico. Não podemos usar caracteres raros como o sinal +, um espaço, ou um sinal. Os nomes suportados para variáveis podem ser:

Age

country of birth

_Name

$ element

Other $ _Names

Temos também de evitar a utilização de nomes reservados como variáveis, por exemplo, não poderemos chamar as nossas palavras variáveis como o regresso ou para, para os quais veremos que são utilizadas para estruturas da própria língua. Vamos agora olhar para alguns nomes variáveis que não está autorizado a usar:

12 months

your name

return

for

more or less

pe% pe

Nomes variáveis em Javascript são casos sensíveis

Lembre-se que o Javascript é uma linguagem sensível a casos, por isso as variáveis também são afetadas por essa distinção. Portanto, a variável denominada "meu nome" não é a mesma que a variável "myName". "Idade" não é a mesma que "idade".

Tenha este detalhe em mente, uma vez que é uma fonte comum de problemas de código que por vezes são difíceis de detetar. Isto porque, por vezes, pensamos que estamos a usar uma variável, que deve ter um determinado dado, mas se cometer um erro ao escrevê-lo e colocar maiússão ou minúscula onde não deve, então será outra variável diferente, que não terá os dados esperados. Uma vez que o Javascript não o obriga a declarar variáveis, o programa funcionará sem produzir um erro, no entanto, a execução não produzirá os efeitos desejados.

Declaração de variáveis em Javascript

Declarar variáveis consiste em definir, e incidentalmente informar o sistema, que você vai usar uma variável. É um costume comum em linguagens de programação especificar explicitamente as variáveis a serem usadas em programas. Em muitas linguagens de programação existem algumas regras rígidas quando se trata de declarar variáveis, mas a verdade é que o Javascript é bastante permissivo.

Javascript ignora muitas regras por ser uma linguagem um pouco livre na programação e um dos casos em que dá um pouco de liberdade é quando declara as variáveis, uma vez que não somos obrigados a fazê-lo, ao contrário do que acontece em outras linguagens de programação como Java, C, C # e muitas outras.

Declaração variável com var

Javascript tem a palavra "var" que usaremos quando quisermos declarar uma ou mais variáveis. Sem surpresas, esta palavra é usada para definir a variável antes de usá-la.

Nota: Embora o Javascript não nos obrigue a declarar explicitamente variáveis, é aconselhável declará-las antes de as usar e veremos a partir de agora que também é um bom hábito. Além disso, em artigos posteriores veremos que, em alguns casos especiais, um guião em que declaramos uma variável e outro em que não temos, não produzirá exatamente os mesmos resultados, uma vez que a declaração não afeta o âmbito das variáveis.

```
var operand1

var operand2
```

Também pode atribuir um valor à variável quando está a ser declarada

```
var operand1 = 23

var operand2 = 33
```

Também é permitido declarar várias variáveis na mesma linha, desde que sejam separadas por vírgulas.

```
var operand1, operand2
```

Declaração de variáveis Javascript com let and const

A partir do Javascript em versões modernas (lembre-se que o Javascript é um padrão e que à medida que evolui ao longo do tempo), especificamente no Javascript na sua versão ES6, existem outras formas de declarar variáveis:

Declaração: Esta nova forma de declarar variáveis afeta o seu âmbito, uma vez que são locais ao bloco onde estão a ser declaradas.

Const Declaration: Na verdade "const" não declara uma variável, mas uma constante, que não pode alterar o seu valor durante a execução de um programa.

Talvez neste momento no manual javascript não seja necessário aprofundar demasiado estes modelos de declaração e para aqueles que estão a aprender recomendamos que se concentrem na utilização de declarações com "var". No entanto, se quiser ter mais informação sobre estes novos tipos de variáveis, explicamo-las no artigo Vamos e const: variáveis no ECMAScript 2015. Se quiser ver mais notícias sobre o padrão Javascript que chegou em 2015, recomendamos a leitura do Manual ES6.

Âmbito de variáveis em Javascript

O âmbito das variáveis é um dos conceitos mais importantes que devemos saber quando trabalhamos com variáveis, não só no Javascript, mas na maioria das linguagens de programação.

No artigo anterior já começámos a explicar quais são as variáveis e como as declarar. Neste artigo do Manual Javascript pretendemos explicar em detalhe o que é este campo de variáveis e oferecer exemplos para que possa ser bem compreendido.

No início do artigo vamos referir-nos ao âmbito das variáveis declaradas com "var". No entanto, há uma nova forma de declarar variáveis, com "let", que afeta diretamente o seu âmbito. No final, também ofereceremos algumas notas e referências para compreendê-la.

Conceito de âmbito variável

O âmbito das variáveis é chamado o local onde estão disponíveis. Em geral, quando declaramos uma variável que a disponibilizamos no local onde foi declarada, isso ocorre em todas as linguagens de programação e uma vez que o Javascript é definido dentro de uma página web, as variáveis que declaramos na página serão acessíveis dentro dela.

No Javascript não seremos capazes de aceder a variáveis que foram definidas noutra página. Portanto, a página onde é definida é o âmbito mais comum de uma variável e chamaremos este tipo de variáveis globais a página. Veremos também que as variáveis podem ser feitas com diferentes âmbitos do global, ou seja, variáveis que iremos declarar e serão válidas em lugares mais limitados.

Variáveis globais

Como dissemos, as variáveis globais são aquelas que são declaradas no âmbito mais amplo possível, que no Javascript é uma página web. Para declarar uma variável global para a página vamos simplesmente fazê-lo em um script, com a palavra var.

```
<SCRIPT>

global variable var

</SCRIPT>
```

As variáveis globais são acessíveis a partir de qualquer lugar da página, isto é, a partir do script onde foram declarados e todos os outros scripts na página, incluindo manipuladores de eventos, como onclick, que já vimos poderia ser incluído dentro de certas tags HTML.

Variáveis locais

Também podemos declarar variáveis em lugares mais estreitos, como uma função. Vamos chamar estas variáveis locais. Quando as variáveis locais são declaradas, só podemos acessá-las dentro do local onde foram declaradas, ou seja, se as declarássemos em função, só podemos aceder a elas quando estamos nessa função.

As variáveis podem ser locais para uma função, mas também podem ser locais para outros âmbitos, como um loop. Em geral, qualquer área delimitada por chaves é uma área local.

```
<SCRIPT>

function myFunction () {

    var variableLocal

}

</SCRIPT>
```

No script anterior declaramos uma variável dentro de uma função, de modo que a variável só será válida dentro da função. Pode ver como os aparelhos são usados para delimitar o local onde essa função é definida ou o seu âmbito.

Não há qualquer problema em declarar uma variável local com o mesmo nome que uma global, neste caso a variável global será visível a partir de toda a página, exceto no âmbito onde a variável local é declarada uma vez que neste site esse nome variável é ocupado pelo local e é ela que tem validade. Em resumo, a variável que será válida em qualquer lugar da página é a global. Menos na área onde a variável local é declarada, que será a que tem validade.

```
<SCRIPT>

var number = 2

function myFunction () {

    var number = 19

    document.write (number) // print 19

}

document.write (number) // print 2

</SCRIPT>
```

Nota: Para compreender este código, certamente será útil consultar o capítulo sobre a criação de funções no Javascript.

Uma dica para principiantes pode ser não declarar variáveis com os mesmos nomes, por isso nunca há confusão sobre qual variável é válida em qualquer momento.

Diferenças entre declarar variáveis com var, ou não declará-las

Como dissemos, no Javascript somos livres de declarar ou não as variáveis com a palavra VAR, mas os efeitos que alcançaremos em cada caso serão diferentes. Especificamente, quando usamos o VAR estamos a fazer o varible que estamos a declarar local ao âmbito onde é declarado. Por outro lado, se não usarmos a palavra VAR para declarar uma variável, será global para toda a página, qualquer que seja o âmbito em que foi declarada.

No caso de uma variável declarada na página web, fora de uma função ou de qualquer outro âmbito menor, não é importante para nós se é ou não declarada com var, do ponto de vista funcional. Isto porque qualquer variável declarada fora de um âmbito é global para toda a página. A diferença pode ser vista numa função, por exemplo, já que se usarmos o VAR a variável será local para a função e se não a usarmos, a variável será global para a página. Esta diferença é fundamental quando se trata de controlar corretamente o uso de variáveis na página, uma vez que se não o fizermos numa função, poderíamos substituir o valor de uma variável, perdendo os dados que pode conter anteriormente.

```
<SCRIPT>
var number = 2
function myFunction () {
    number = 19
    document.write (number) // print 19
}
document.write (number) // print 2
// call the function
myFunction ()
document.write (number) // print 19
</SCRIPT>
```

Neste exemplo, temos uma variável global para a página chamada número, que contém um 2. Temos também uma função que usa o número variável sem o ter declarado com var, pelo que o número variável da função será o mesmo número variável global declarado fora de função. Numa situação como esta, a execução da função substituirá o número variável e os dados que estavam lá antes de executar a função serão perdidos.

Declaração variável com let

Desde o ECMAScript 2015 existe a declaração de arrendamento. A sintaxe é a mesma que o VAR ao declarar variáveis, mas no caso de deixar a declaração afetar o bloco.

Bloco significa qualquer espaço delimitado por teclas, tais como as declarações dentro das teclas de um loop.

```
for (let i = 0; i <3; i ++) {

  // in this case the variable i only exists inside the for
loop

  alert (i);

}

// outside the for block there is no variable i
```

Se essa variável "i" tivesse sido declarada no cabeçalho para loop usando "var", existiria fora do bloco de código para o código.

O que podemos poupar em variáveis

Numa variável podemos introduzir vários tipos de informação. Por exemplo, poderíamos introduzir texto simples, números inteiros ou reais, etc. Estes diferentes tipos de informação são conhecidos como tipos de dados. Cada um tem características e usos diferentes.

Vamos ver quais são os tipos de dados mais comuns de Javascript.

Números, 19

Para começar temos o tipo numérico, para poupar números como 9 ou 23,6

Correntes, 20

O tipo de corda de caracteres armazena um texto. Sempre que escrevemos uma corda de carácter, temos de usar aspas (").

Propriedade Booleans

Também temos o tipo booleano, que armazena informações que podem ser válidas (verdadeiras) ou não (falsas).

Finalmente, seria relevante salientar aqui que as nossas variáveis podem conter coisas mais complicadas, como um objeto, uma função, ou vazio (nulo), mas veremos mais tarde.

Na verdade, as nossas variáveis não são forçadas a guardar um tipo de dados específico e, portanto, não especificamos nenhum tipo de dados para uma variável quando estamos a declará-lo. Podemos

introduzir qualquer informação numa variável de qualquer tipo, podemos até alterar o conteúdo de uma variável de um tipo para outro sem qualquer problema. Veremos isto com um exemplo.

```
var city_name = "Valencia"

revised var = true

city_name = 32

revised = "no"
```

Esta leveza ao atribuir tipos a variáveis pode ser uma vantagem no início, especialmente para pessoas inexperientes, mas a longo prazo pode ser uma fonte de erros, uma vez que, dependendo do tipo de variáveis, eles se comportarão de uma forma ou de outra e se não controlarmos exatamente o tipo de variáveis podemos encontrar-nos adicionando um texto a um número. O Javascript funcionará perfeitamente, e devolverá um dado, mas em alguns casos pode não ser o que esperávamos. Assim, embora tenhamos liberdade com tipos, esta mesma liberdade faz-nos estar mais atentos a possíveis desencontros que são difíceis de detetar ao longo dos programas. Vamos ver o que aconteceria se adicionássemos letras e números.

```
var adding1 = 23

var adding2 = "33"

var sum = adding1 + adding2

document.write (sum)
```

Este guião mostrar-nos-ia na página o texto 2333, que não corresponde à soma dos dois números, mas à sua concatenação, um após o outro.

Tipos de dados em Javascript

Nos nossos scripts vamos lidar com vários tipos de variáveis de informação, como textos ou números. Cada um destes tipos de informação são os tipos de dados. Javascipt distingue entre três tipos de dados e toda a informação que pode ser armazenada em variáveis será incorporada num desses tipos de dados. Vamos ver mais de perto o que são estes três tipos de dados.

Tipo de dados numéricos

Há apenas um tipo de dados numéricos nesta língua, ao contrário da maioria das línguas mais conhecidas. Todos os números são, portanto, do tipo numérico, independentemente da precisão que têm ou se são números reais ou inteiros. Os números inteiros são números não-vírgulas, tais como 3 ou 339. Os números reais são números fracionados, como 2,69 ou 0,25, que também podem ser escritos em notação científica, por exemplo 2.482e12.

Com o Javascript também podemos escrever números noutras bases, como o hexadecimal. As bases são sistemas de numeragem que usam mais ou menos dígitos para escrever números. Há três bases com as quais podemos trabalhar

A Base 10 é o sistema que normalmente usamos, o sistema decimal. Qualquer número, por defeito, é entendido como sendo escrito na base 10.

Base 8, também chamado de sistema octal, que usa dígitos de 0 a 7. Para escrever um número em octal, basta escrever esse número precedido por 0, por exemplo 045.

Base 16 ou sistema hexadecimal, é o sistema de numerações que utiliza 16 dígitos, aqueles entre 0 e 9 e as letras de A a F, para os dígitos em falta. Para escrever um número em hexadecimal, devemos escrevê-lo precedido por um zero e um x, por exemplo 0x3EF.

Tipo booleano

O tipo bolean, boolean em inglês, é usado para salvar um sim ou não ou por outras palavras, um verdadeiro ou um falso. É usado para executar operações lógicas, geralmente para executar ações se o conteúdo de uma variável for verdadeiro ou falso.

Se uma variável é verdadeira então ----- eu executar algumas instruções

Se não ----- eu executo outros

Os dois valores que as variáveis booleanas podem ter são verdadeiros ou falsos.

```
myBoleana = true

<br>

myBoleana = false
```

Cadeia de caracteres de tipo de dado

O último tipo de dados é o usado para guardar um texto. O Javascript tem apenas um tipo de dados para guardar texto e qualquer número de caracteres pode ser introduzido. Um texto pode ser composto por números, letras e qualquer outro tipo de caracteres e sinais. Os textos são escritos em aspas, duplo ou único.

```
myText = "Pepe is going to fish"

myText = '23 %% $ Letters & * - * '
```

Tudo o que está incluído em aspas, como nos exemplos anteriores, é tratado como uma corda de carácter, independentemente do que colocamos dentro das aspas. Por exemplo, numa variável de texto podemos poupar números e, nesse caso, temos de ter em conta que as variáveis de texto e numéricas não são a mesma coisa e enquanto as variáveis numéricas são usadas para fazer cálculos matemáticos, as variáveis de texto não são.

Personagens de fuga em cordas de texto

Há uma série de caracteres especiais que são usados para expressar certos controlos numa cadeia de texto, como uma quebra de linha ou um separador. Estes são os caracteres de fuga e são escritos com notação especial que começa com um backslash (um corte para a frente em frente ao normal ") e, em seguida, o código para o personagem ser exibido é colocado.

Um caráter muito comum é a quebra de linha, que é alcançada por dactilografia n. Outro personagem muito comum é colocar aspas, porque se colocarmos aspas sem o seu caráter especial, as aspas que colocamos começarão a fechar a cadeia de caracteres. As aspas devem então ser inscritas com "o" (aspas duplas ou únicas). Existem outros personagens de fuga, que veremos na tabela abaixo, que são mais resumidos, embora o usado para escrever um backslash também deve ser destacado como o personagem habitual. , para não confundi-lo com o início de um personagem de fuga, que é o duplo backslash.

Mesa com todos os personagens de fuga

```
Line break: \ n

Single quote: \ '

Double quote: \ "

Tab: \ t

Carriage return: \ r

Page advance: \ f

Move back space: \ b

Counterbar: \\
```

Alguns destes personagens provavelmente nunca serão usados por si, uma vez que a sua função é uma parte estranha e às vezes pouco clara.

Capítulo IV

Operadores javascript

Ao desenvolver programas em qualquer idioma, os operadores são utilizados para fazer os cálculos e operações necessários para realizar os seus objetivos. Mesmo o programa mais pequeno que se possa imaginar precisa que os operadores façam as coisas, uma vez que um programa que não realiza operações limitar-se-ia apenas a fazer sempre o mesmo.

É o resultado de operações que fazem com que um programa varie o seu comportamento de acordo com os dados com os qual tem de trabalhar e nos ofereça resultados relevantes para o utilizador que o utiliza. Existem operações mais simples ou complexas, que podem ser realizadas com operações de diferentes tipos, como números ou textos, veremos neste capítulo, e os seguintes, em detalhe, todos estes operadores disponíveis no Javascript.

Exemplos de utilização de operadores

Antes de ir enumerar os diferentes tipos de operadores, vamos ver alguns exemplos destes para nos ajudar a ter uma ideia mais precisa do que são. No primeiro exemplo, vamos realizar uma soma utilizando o operador de soma.

3 mais 5

Esta é uma expressão muito básica que não faz muito sentido por si só. Faz a soma entre os dois óperas número 3 e 5, mas não ajuda muito porque nada é feito com o resultado. Normalmente, mais do que um operador é combinado para criar expressões mais úteis. A expressão a seguir é uma combinação entre dois operadores, um executa uma operação matemática e o outro é usado para salvar o resultado.

```
myVariable = 23 * 5
```

No exemplo anterior, o operador * é utilizado para realizar uma multiplicação e o operador = é usado para atribuir o resultado numa variável, por isso guardamos o valor para posterior utilização.

Os operadores podem ser classificados de acordo com o tipo de ações que realizam. Em seguida, olharemos para cada um destes grupos de operadores e descreveremos a função de cada um.

Operadores aritméticos

São aqueles utilizados para realizar operações matemáticas simples, tais como adição, subtração ou multiplicação. No javascript são os seguintes:

+ Soma de dois valores

- Subtração de dois valores, também pode ser usada para alterar o sinal de um número se o usarmos com uma única ópera -23

* Multiplicação de dois valores

Divisão de dois valores

% O restante da divisão de dois números (3% 2 devolveria 1, o restante da divisão de 3 por 2)

++ Incremento numa unidade, usado com um único operand

- Decremento numa unidade, usado com um único operand

Exemplos, 19

```
price = 128 // I introduce a 128 in the price variable

units = 10 // another assignment, then we will see assignment
operators

invoice = price * units // multiply price by units, I get the invoice
value

remainder = invoice% 3 // I get the remainder from dividing the
invoice variable by 3

price ++ // increase the price by one unit (now worth 129)
```

Operadores de atribuição

São usados para atribuir valores a variáveis, já usamos o operador de atribuição = em exemplos anteriores, mas há outros operadores deste tipo, que vêm da língua C e que muitos dos leitores já saberão.

= Atribuição. Atribua a parte direita dos iguais à parte esquerda. À direita são colocados os valores finais e à esquerda uma variável é geralmente colocada onde queremos guardar os dados.

+ = Atribuição com soma. Adicione a parte direita à parte esquerda e guarde o resultado para a parte esquerda.

- = Atribuição com subtração

* = Alocação de multiplicação

/ = Atribuição de divisão

% = O resto é obtido e atribuído

Exemplos, 19

```
savings = 7000 // assign a 7000 to the savings variable

savings + = 3500 // increase savings variable by 3500, now worth
10500

savings / = 2 // divide my savings by 2, now there are 5250
```

No seguinte artigo continuaremos a conhecer outros operadores Javascript: operadores de cordas, operadores lógicos e operadores condicional.

Operadores em cadeia

As cadeias de caracteres, ou variáveis de texto, também têm os seus próprios operadores para executar ações típicas em cordas. Embora o javascript tenha apenas um operador para cordas, outras ações podem ser executadas com uma série de funções predefinidas no idioma que veremos mais tarde.

+ Concatenate duas cordas, cole a segunda corda após a primeira.

Exemplo

```
string1 = "hello"

string2 = "world"

concatenated string = string1 + string2 // concatenated string is
valid "helloworld"
```

Um detalhe importante que pode ser visto neste caso é que o operador + serve dois usos diferentes, se os seus operos são números, ele adiciona-os, mas se são cordas que os concatena. Isto acontece em geral com todos os operadores que são repetidos no idioma, o javascript é inteligente o suficiente para entender que tipo de operação deve ser efetuada verificando os tipos que estão envolvidos no mesmo.

ligados com texto e operadores numéricos misturados. Neste caso, o javascript assume que quer concatenar e trata os dois óperas como se fossem cordas de carácter, mesmo que a cadeia de texto tenha sido um número. Veremos isto mais facilmente com o seguinte exemplo.

```
myNumber = 23

myString1 = "pepe"

myString2 = "456"

result1 = myNumber + myString1 // result1 is worth "23pepe"

result2 = myNumber + myString2 // result2 is valid "23456"

myString2 + = myNumber // myString2 is now "45623"
```

Como vimos, também no caso do operador + =, se estivermos a lidar com cordas de texto e números misturados, tratará os dois operadores como se fossem cordas.

Nota: Como deve ter imaginado, muitas operações típicas a executar com cordas estão em falta, para as quais não existem operadores. É porque estas funcionalidades são obtidas através da classe String de Javascript, que veremos mais tarde.

Operadores lógicos

Estes operadores são usados para executar operações lógicas, que são aquelas que resultam num verdadeiro ou falso, e são usados para tomar decisões nos nossos scripts. Em vez de trabalhar com números, para realizar este tipo de operações, os óperas booleanos, que sabíamos anteriormente, são usados, que são verdadeiros (verdadeiros) e falsos (falsos). Os operadores lógicos relacionam óperas booleanas para resultar em outra ópera booleana, como podemos ver no exemplo seguinte.

Se estou com fome e tenho comida, então começo a comer.

O nosso programa Javascript usaria uma ópera booleana neste exemplo para tomar uma decisão. Primeiro vai ver se tenho fome, se é verdade (verdade) vai ver se eu tenho comida. Se ambos forem verdadeiros, pode ser comido. No caso de não ter comida ou não tiver fome, não comia, como se não tivesse fome ou comida. A ópera em questão é a ópera, o que será verdade se os dois óperas forem verdadeiros.

Nota: Para não ser enganoso, deve dizer-se que os operadores lógicos podem ser utilizados em combinação com tipos de dados que não o Boolean, mas neste caso devemos usá-los em expressões que os tornam Boolean. No próximo grupo de operadores que vamos tratar neste artigo falaremos de operadores condicional, que podem ser utilizados em conjunto com operadores lógicos para fazer todas as frases complexas de que precisamos. Por exemplo:

```
if (x == 2 && y! = 3) {

   // the variable x is worth 2 and the variable y is
different from three

}
```

Na expressão condicional acima, estamos a avaliar dois controlos relacionados com um operador lógico. Por um lado x == 2 voltará verdadeiro se a variável x for 2, e por outro, y! = 3 voltará verdadeiro quando a variável y tem um valor diferente de 3. Ambos os cheques devolvem um Boolean cada, que então o operador lógico & e é aplicado a ele para verificar se ambos os controlos foram cumpridos ao mesmo tempo.

Escusado será dizer que, para vermos exemplos de operadores condicionais, precisamos de aprender estruturas de controlo como se, que ainda não chegámos.

! Operador NO ou negação. Se fosse verdade, muda para falso e vice-versa.

& operador e, se ambos são verdadeiros, é verdade.

|| Operador O, verdade se pelo menos um deles for verdadeiro.

Exemplo

myBoleano = true

myBoleano =! myBoleano // miBoleano now worth false

hunger = true

gotFood = true

comoMeal = I am hungry && I have food

Operadores condicionais

São usados para fazer expressões condicionais tão complexas quanto quisermos. Estas expressões são usadas para tomar decisões com base na comparação de vários elementos, por exemplo, se um número é maior do que outro ou se são iguais.

Nota: Os operadores condicionados são, naturalmente, também utilizados para fazer expressões comparando outros tipos de dados. Nada impede a comparação de duas cadeias, para ver se são iguais ou diferentes, por exemplo. Podemos até comparar Booleans.

Os operadores condicionais são utilizados em expressões condicionais para a tomada de decisões. Uma vez que estas expressões condicionais serão estudadas mais tarde, será melhor descrever os operadores condicionais mais tarde. De qualquer forma aqui podemos ver a mesa dos operadores condicional.

==Verificar se dois valores são iguais

! = Verifique se dois valores são diferentes

> Maior do que, retorna verdadeiro se o primeiro operand é maior do que o segundo

<Menos do que, é verdade quando o item à esquerda é menor do que o da direita

> = Maior igual

<= Menos igual

Veremos exemplos de operadores condicionais quando explicamos estruturas de controlo, como o se for condicional.

Além disso, neste texto veremos uma questão de grande importância na programação em geral, que é a precedência dos operadores, que devemos ter em conta sempre que utilizamos diferentes operadores na mesma expressão, de modo a que

estejam relacionados uns com os outros e sejam resolvidos da forma como tínhamos planeado.

Operadores de bits

Estes são muito incomuns e talvez nunca possa usá-los. A sua utilização é feita para realizar operações com zeros e uns. Tudo o que um computador pega são zeros e uns, embora nós

Os números e letras para as nossas variáveis na verdade estes valores são escritos internamente sob a forma de zeros e uns. Em alguns casos, podemos precisar de realizar operações que tratem as variáveis como zeros e algumas e para isso vamos usar estes operands. Neste manual é um pouco grande demais para nós realizarmos uma discussão sobre este tipo de operadores, mas aqui você pode ver estes operadores no caso de precisar deles.

& Y-bit

^ Bit Xor

| Ou pedaços

<< >> >>>>>>> = >> = << = Vários tipos de mudanças

Precedência dos operadores

A avaliação de uma frase que vimos nos exemplos anteriores é bastante simples e fácil de interpretar, mas quando uma multiplicidade de operadores diferentes entra em jogo numa frase, pode haver confusão ao interpretá-la e elucidar quais os operadores que executam antes dos outros. Para definir algumas orientações na avaliação das sentenças e que são sempre executadas da mesma forma e com bom senso, há a precedência dos operadores, que não é mais do que a ordem em que as operações que representam serão executadas. Inicialmente, todos os operadores são avaliados da esquerda para a direita, mas existem regras adicionais, segundo as quais certos operadores são avaliados antes de outros. Muitas destas regras de precedência são derivadas da matemática e são comuns a outras línguas, podemos vê-las abaixo.

() []. Parênteses, suportes quadrados e o operador de pontos para objetos

! - ++ - negação, negativo e incrementos

*** /% Divisão de multiplicação e módulo**

Adicionar e subtrair

<< >> >>>> Alterações no nível do bit

<<=>> = Operadores condicionais

==! = Operadores condicionais da igualdade e da desigualdade

e ^ | Lógica de bit-level

& e || Lógicos booleanos

= + = = = = = = = == =% = << = >> = >>>> = e = ^ == =! = Atribuição

Nos exemplos que se seguem podemos ver como as expressões podem tornar-se confusas, mas com a tabela de precedência do

operador poderemos compreender sem erros qual é a ordem em que são executadas.

12 * 3 + 4 - 8/2% 3

Neste caso, os operadores * /e% são executados primeiro, da esquerda para a direita, com os quais estas operações seriam realizadas. Primeiro a multiplicação e depois a divisão porque é mais à esquerda do módulo.

36 + 4 - 4% 3

Agora o módulo.

36 mais 4 - 1

Finalmente, as adições e subtrações da esquerda para a direita.

40 - 1

O que nos dá o seguinte valor.

39

Em todo o caso, é importante perceber que o uso de parênteses pode poupar-nos muitas dores de cabeça e, acima de tudo, a necessidade de conhecer a tabela de precedência do operador de cor. Quando não sabemos a ordem em que as sentenças serão executadas, podemos usá-las e, assim, forçar que o pedaço de expressão encontrado dentro dos parênteses seja avaliado antes.

Operador de tipo javascript para controlo do tipo

Pudemos verificar que, para certos operadores, o tipo de dados que estão a tratar é importante, uma vez que, se os dados forem de um tipo, serão realizadas operações diferentes do que se forem de outro.

Por exemplo, quando usamos o operador +, se fossem números, acrescentou-os, mas se fossem cordas de carácter, concatenated-os. Vemos então que o tipo de dados que estamos a utilizar importa e que teremos de estar cientes deste pormenor se quisermos que as nossas operações sejam realizadas como esperado.

Para verificar o tipo de dados pode utilizar outro operador disponível a partir do javascript 1.1, o tipo de operador, que devolve uma cadeia de texto que descreve o tipo de operador que estamos a verificar.

Nota: ao longo da nossa experiência com o Javascript veremos que muitas vezes é mais útil alterar o tipo de dados de uma variável antes de fazer um check com tipeof para ver se podemos usá-lo como um operand. Existem várias funções para tentar alterar o tipo de variável, como o parseInt (), que veremos mais tarde na Segunda Parte do Manual javascript.

```
boolean var = true

numeric var = 22

floating_number = 13.56

var text = "my text"

var date = new Date ()

document.write ("<br> Boolean type is:" + boolean typeof)

document.write ("<br> The numeric type is:" + numeric typeof)

document.write ("<br> The type of floating_number is:" +
typeof floating_number)

document.write ("<br> The text type is:" + typeof text)

document.write ("<br> The date type is:" + typeof date)
```

Se executarmos este script, obteremos que o seguinte texto será escrito na página:

Tipo booleano é: boolean

O tipo numérico é: número

O tipo de floating_numeric é: número

O tipo de texto é: cadeia

O tipo de data é: objeto

Neste exemplo podemos ver que os diferentes tipos de variáveis são impressos na página. Estes podem ser os seguintes:

boolean, para dados boolean. (Verdadeiro ou falso)

número, para os numéricos.

corda, para cordas de caráter.

objeto, para objetos.

função, para funções.

indefinidamente, para variáveis declaradas que não foram atribuídas valores.

Queremos destacar apenas mais dois detalhes:

Os números, quer tenham ou não uma parte decimal, são sempre do tipo de dados numéricos.

Uma das variáveis é um pouco mais complexa, é a variável de data que é um objeto da classe Data () que é usada para lidar com datas em scripts. Vamos vê-lo mais tarde, assim como os objetos.

Note também que as funções javascript são tratadas como o

Recomendações com tipo de

Em nossa opinião, não é muito adequado pedir constantemente o tipo de variável, fazer coisas diferentes quando tem um tipo ou outro. O normal é que você é claro sobre que tipo de operação você quer realizar num dado momento e que, se você não tem certeza se a variável de um tipo ou outro chegou, você convertê-lo com alguma função como parseInt () ou parseFloat (), que veremos mais tarde.

O caso mais útil que agora me lembro é perguntar se o tipo de variável é indefinido, o que pode dar-lhe a resposta sobre se a variável foi inicializada ou não.

```
variable let;

if (typeof variable == 'undefined') {

console.log ('The variable is undefined, it has no
defined value);

}
```

Outro caso que é frequentemente usado muito é quando se depura, quando é preciso encontrar erros no seu código, face a operações erráticas e inesperadas. Nesses casos é útil consultar o tipo de variáveis, porque às vezes no Javascript trazem-lhe surpresas que fazem com que os seus programas não funcionem bem.

Estruturas de controlo javascript

Os scripts vistos até agora no Manual Javascript têm sido tremendamente simples e lineares: declarações simples foram executadas uma após a outra do início ao fim. No entanto, isso nem sempre tem de ser o caso e, na verdade, na maioria dos casos as coisas são muito mais complexas.

Se tivermos alguma experiência em programação saberemos que em programas será geralmente necessário fazer coisas diferentes dependendo do estado das nossas variáveis ou realizar o mesmo processo muitas vezes sem escrever as mesmas linhas de código uma e outra vez. As estruturas de controlo são usadas para fazer coisas mais complexas nos nossos scripts. Com eles podemos tomar decisões e loops. Nos capítulos seguintes aprenderemos sobre as diferentes estruturas de controlo que existem no Javascript.

Tomada de decisão

São usados para executar algumas ações ou outras dependendo do estado das variáveis. Ou seja, tomar decisões para executar algumas instruções ou outras dependendo do que está acontecendo naquele momento nos nossos programas.

Por exemplo, dependendo se o utilizador que entra na nossa página tem idade legal ou não, podemos ou não permitir-lhe ver o conteúdo da nossa página.

Se a idade é maior que 18, então

Deixei-te ver o conteúdo de um adulto.

Se não, não sou.

Vou mandá-lo para fora da página.

No Javascript podemos tomar decisões usando duas declarações diferentes.

Função IF

Mudar

Loops

Loops são usados para executar certas ações repetidamente. São amplamente utilizados a todos os níveis na programação. Com um loop podemos, por exemplo, imprimir os números de 1 a 100 numa página sem ter que escrever a instrução de impressão uma centena de vezes.

De 1 a 100

Imprima o número atual

No javascript existem vários tipos de loops, cada um é indicado para um tipo diferente de iteração e são os seguintes:

Para

EN, 20

FAZER ENQUANTO

Como já referimos, as estruturas de controlo são muito importantes no Javascript e em qualquer linguagem de programação. É por isso que nos capítulos seguintes veremos cada uma destas estruturas cuidadosamente, descrevendo a sua utilização e oferecendo alguns exemplos.

Estrutura IF em Javascript

No Manual Javascript de DesarrolloWeb.com já começamos a explicar o que são estruturas de controlo. Neste artigo, vamos mostrar como funciona a afirmação, que é a estrutura mais comum usada para tomar decisões em programas de computador.

If é uma estrutura de controlo usada para tomar decisões. É um condicional que é usado para realizar uma ou outra operações com base numa expressão. Funciona da seguinte forma, uma expressão é avaliada primeiro, se der um resultado positivo as ações relacionadas com o caso positivo são realizadas.

A sintaxe da estrutura IF é a seguinte.

Nota: Todas as estruturas de controlo são minúsculas em Javascript. Embora às vezes possamos escrever o nome da estrutura no texto do manual em letras maiúsculas, no código dos nossos scripts temos sempre que colocá-lo em minúsculas. Caso contrário, receberemos uma mensagem de erro.

```
if (expression) {
  // actions to be taken in the positive case
  // ...
}
```

Opcionalmente, as ações podem ser indicadas caso a avaliação da

sentença devolva resultados negativos.

```
if (expression) {
  // actions to be taken in the positive case
  // ...
} else {
  // actions to be taken in the negative case
  // ...
}
```

Vamos ver várias coisas. Para começar, vemos como as chaves incluem as ações que queremos realizar no caso de as expressões serem cumpridas ou não. Estas teclas devem ser sempre colocadas, exceto no caso de existir apenas uma instrução como ações a serem executadas, que são opcionais.

Nota: Embora as chaves para englobar as declarações a serem executadas tanto nos casos positivos como negativos sejam opcionais quando queremos executar uma única declaração, a recomendação é colocá-las sempre, pois assim obteremos um código fonte mais claro. Por exemplo:

```
if (it rains)
  alert ("Water falls");
```

Seria exatamente como este código:

```
if (it rains) {
  alert ("Water falls");
}
```

Ou mesmo, como este outro:

```
if (it rains) alert ("Water falls");
```

No entanto, quando usamos as teclas, o código é muito mais claro, porque pode ver de uma só vez quais as instruções que dependem do caso positivo do se. Este é um detalhe que pode não ser muito importante agora, mas será apreciado quando o programa é mais complexo ou quando vários programadores estão encarregados de tocar no mesmo código.

Outro detalhe que é óbvio é o recuo (margem) que colocamos em cada um dos blocos de instruções a executar nos casos positivos e negativos. Este entalhe é totalmente opcional, só o fizemos desta forma para que a estrutura IF seja entendida de uma forma mais visual. As quebras de linha também não são necessárias e foram também colocadas para uma melhor visão da estrutura. Poderíamos colocar a declaração da IF na mesma linha de código, mas isso não vai ajudar as coisas a ficarem claras.

Nota: Nós, bem como qualquer pessoa com alguma experiência na área de programação, aconselhamos a utilização dos travões necessários e quebras de linha para que as instruções possam ser melhor compreendidas. Talvez no dia em que fizer um código seja claro sobre o que fez e por que é assim, mas num mês, quando tiver que reler esse código, poderá lembrar-se menos do que fez nos seus scripts e apreciará que eles têm um formato amigável para que possa ser facilmente lido pelas pessoas. Se trabalhar em equipa, estas recomendações serão ainda mais importantes, uma vez que é ainda mais difícil ler o código fonte que outras pessoas fizeram.

Vamos ver algum exemplo de condicionals if.

```
if (dia == "monday")
   document.write ("Have a happy start to the week")
```

Se for segunda-feira, vai nos desejar uma semana feliz. Não vai fazer de outra forma. Como neste exemplo apenas indicamos uma instrução para o caso positivo, não será necessário utilizar as chaves (embora seja aconselhável colocá-las). Note também o operador condicional composto por dois sinais iguais.

Vamos agora ver outro exemplo, um pouco mais.

```
if (credit> = price) {

   document.write ("you have bought the article" + new
article) // show purchase

   cart + = newArticle // insert the item in the shopping
cart

   credit - = price // I decrease the credit according to the
price of the item

} else {

   document.write ("you have run out of credit") // I report
that you are short of money

   window.location = "shoppingcart.html" // I go to the cart
page

}
```

Este exemplo é um pouco mais complexo, e também um pouco ficcional. O que faço é verificar se tenho crédito para fazer uma alegada compra. Para isso, olho para se o crédito é maior ou igual ao preço do artigo, em caso afirmativo, informo sobre a compra, insiro o artigo no carrinho e repouso o preço do crédito acumulado. Se o preço do artigo for superior ao dinheiro disponível, reporto a situação e envio o navegador para a página onde o seu carrinho de compras é apresentado.

Expressões condicionais

A expressão para avaliar é sempre colocada em parênteses e é composta por variáveis que são combinadas entre si usando operadores condicionais. Recordamos que os operadores condicionais estão relacionados com duas variáveis e sempre devolveram um resultado Boolean. Por exemplo, um operador condicional é o operador "é igual" (==), que retorna verdadeiramente se os dois operáticos forem iguais ou falsos se forem diferentes.

```
if (age> 18)

   document.write ("you can see this page for
adults")
```

Neste exemplo usamos o operador condicional "é maior" (>). Neste caso, retorna verdadeiramente se a idade variável for superior a 18 anos, pelo que a seguinte linha seria executada, informando-nos de que o conteúdo adulto pode ser visto.

Expressões condicionais podem ser combinadas com expressões lógicas para criar expressões mais complexas. Lembramo-nos que expressões lógicas são aquelas que têm Booleans como óperas e que devolvem outro valor booleano. São os operadores lógicos de negação, lógica e or.

```
if (battery <0.5 && redElectrica == 0)

   document.write ("your laptop will shutdown in
seconds")
```

O que fazemos é verificar se a bateria do nosso suposto computador é inferior a 0,5 (está quase acabada) e também verificamos se o

computador não tem rede elétrica (está desligado da tomada). Em seguida, o operador lógico combina-os com um Y, então se estiver quase sem bateria e sem energia, informa que o computador vai desligar.

Nota: A lista de operadores que podem ser utilizados com estruturas IF pode ser vista no capítulo sobre operadores condicionais e operadores lógicos.

A estrutura se é uma das mais utilizadas em linguagens de programação, para tomar decisões com base na avaliação de uma frase. No artigo anterior do Manual javascript já começamos a explicar a estrutura e agora vamos ver alguns usos mais avançados.

Declarações aninhadas do IF

Para tornar as estruturas condicionais mais complexas podemos nidificar as declarações do IF, ou seja, colocar estruturas IF dentro de outras estruturas IF. Com um único IF podemos avaliar e executar uma ou outra ação de acordo com duas possibilidades, mas se tivermos mais possibilidades de avaliar, temos de nidificar os IFs para criar o fluxo de código necessário para decidir corretamente.

Por exemplo, se eu quiser verificar se um número é inferior ou igual a outro, tenho que avaliar três possibilidades diferentes. Em primeiro lugar, posso verificar se os dois números são iguais, se forem, já resolvi o problema, mas se não forem iguais, ainda terei de ver qual dos dois é maior. Vejamos este exemplo no código Javascript.

```
var number1 = 23

var number2 = 63

if (number1 == number2) {

   document.write ("The two numbers are the same")

} else {

   if (number1> number2) {

      document.write ("The first number is greater
than the second")

   } else {

      document.write ("The first number is less
than the second")

   }

}
```

O fluxo do programa é como mencionamos anteriormente, primeiro é avaliado se os dois números são iguais. Se for positivo, é apresentada uma mensagem informando-o disto. Caso contrário, já sabemos que são diferentes, mas ainda temos de descobrir qual dos dois é maior. Para isso, outra comparação é feita para descobrir se a primeira é maior do que a segunda. Se esta comparação der resultados positivos, mostramos uma mensagem a dizer que a primeira é maior do que a segunda, caso contrário indicaremos que a primeira é menos do que a segunda.

Notamos mais uma vez que as chaves são opcionais neste caso, uma vez que apenas uma declaração é executada para cada caso. Além disso, quebras de linha e travessão também são opcionais em

qualquer caso e servem apenas para ver o código de uma forma mais ordenada. Manter o código bem estruturado e escrito de uma forma compreensível é muito importante, uma vez que tornará a vida mais agradável na programação e mais tarde quando tivermos de rever os programas.

Nota: Neste manual utilizarei uma notação como a que viu nas linhas anteriores. Além disso, vou manter sempre essa notação. Isto tornará, sem dúvida, os códigos com exemplos mais fáceis de compreender, se não o fizéssemos, seria um verdadeiro incómodo lê-los. Esta mesma receita aplica-se aos códigos que tem de criar e o principal beneficiário será você e os colegas que lerem o seu código.

Operador IF

Há um operador que ainda não vimos e é uma forma mais esquemática de fazer alguns IFs simples. Vem da língua C, onde muito poucas linhas de código são escritas e onde quanto menos escrevermos, mais elegantes seremos. Este operador é um claro exemplo de guardar linhas e caracteres ao escrever scripts. Vamos vê-lo rapidamente, porque a única razão pela qual o incluo é para que saiba que existe e se o encontrar em alguma ocasião lá fora pode identificá-lo e como funciona.

Um exemplo de utilização do operador IF pode ser visto abaixo.

```
Variable = (condition)? value1: value2
```

Este exemplo não só realiza uma comparação de valor, como também atribui um valor a uma variável. O que faz é avaliar a condição (colocada em parênteses) e, se for positiva, atribui o valor1 à variável e de outra forma atribui-lhe o valor2. Vejamos um exemplo:

```
moment = (current_time <12)? "Before noon": "After noon"
```

Este exemplo olha para ver se a hora atual é inferior a 12. Se assim for, é agora antes do meio-dia, por isso atribua "Antes do Meio-Dia" ao tempo variável. Se a hora for maior ou igual a 12, já é depois do meio-dia, pelo que o texto "Depois do meio-dia" é atribuído à variável de momento.

Estrutura javascript SWITCH

As estruturas de controlo são a forma como se pode controlar o fluxo de programas, para fazer coisas diferentes dependendo dos estados das variáveis. No Manual Javascript já começamos a ver as estruturas de controlo e agora é a vez da SWITCH, uma estrutura ligeiramente mais complexa que permite múltiplas operações dependendo do estado de uma variável.

Neste artigo veremos que a switch nos ajuda a tomar decisões com base em diferentes estados das variáveis. Esta expressão é usada quando temos múltiplas possibilidades como resultado da avaliação de uma frase.

A estrutura SWITCH foi incorporada a partir da versão 1.2 do Javascript (Netscape 4 e Internet Explorer 4). A sua sintaxe é a seguinte.

```
switch (expression) {

   case value1:

      Statements to execute if the expression has a
value of value1

      break

   case value2:

      Statements to execute if the expression has a
value of value2

      break

   case value3:

      Statements to execute if the expression is
valued at value3

      break

   default:

      Statements to execute if the value is not one
of the above

}
```

A expressão é avaliada, se valer a pena valor1 as frases relacionadas com esse caso são executadas. Se a expressão valer valor 2, as instruções relacionadas com esse valor são executadas, e assim por diante, para quantas opções quisermos. Finalmente, para todos os casos não previamente contemplados, o caso por defeito é executado.

A palavra break é opcional, mas se não a colocarmos depois de um valor ser encontrado, todas as frases relacionadas com isso e todas

as seguintes serão executadas. Ou seja, se no nosso esquema anterior não houvesse quebras e a expressão fosse de valor 1, as frases relacionadas com o valor1 seriam executadas, bem como as relacionadas com o valor2, o valor3 e o incumprimento.

A opção por predefinição ou opção por defeito também é opcional.

Vamos ver um exemplo de usar esta estrutura. Suponha que queremos indicar que dia da semana é. Se o dia for 1 (segunda-feira) receber uma mensagem indicando que, se o dia for 2 (terça-feira) temos de receber uma mensagem diferente e assim por diante para cada dia da semana, exceto nos dias 6 (sábado) e 7 (domingo) que queremos mostrar a mensagem "é fim de semana". Por dias superiores a 7 indicaremos que este dia não existe.

```
switch (day_of_the_week) {
   case 1:
      document.write ("It's Monday")
      break
   case 2:
      document.write ("It's Tuesday")
      break
   case 3:
      document.write ("It's Wednesday")
      break
   case 4:
      document.write ("It's Thursday")
      break
   case 5:
      document.write ("It's Friday")
      break
   case 6:
   case 7:
      document.write ("It's weekend")
      break
   default:
      document.write ("That day does not exist")
}
```

O exemplo é relativamente simples, só pode ter uma pequena dificuldade, consistindo em interpretar o que acontece nos casos 6 e 7, que tínhamos dito que tínhamos de mostrar a mesma mensagem. No caso 6, não indicamos nenhuma instrução, mas como não fazemos uma pausa, a seguinte frase ou frases serão executadas, que correspondem à sentença indicada no caso 7, que é a mensagem que informa que é fim de semana. Se o caso for 7, é simplesmente indicado que é um fim de semana, como pretendido.

Nota: Também temos um tutorial de vídeo no SWITCH em Javascript que pode ser muito útil para entender tudo muito melhor.

FOR loops em Javascript

O laço FOR é utilizado para repetir uma ou mais instruções num determinado número de vezes. Entre todos os loops, o FOR é geralmente usado quando sabemos com certeza o número de vezes que queremos que corra. A sintaxe para o loop é mostrada abaixo.

```
for (initialization; condition; update) {
  // statements to execute in each iteration
}
```

O loop FOR tem três partes incluídas nos parênteses, que nos ajudam a definir como queremos que as repetições sejam realizadas. A primeira parte é a inicialização, que só é executada quando se inicia a primeira iteração do loop. Nesta parte, a variável que usaremos para manter a contagem das vezes que o loop é executado é geralmente colocada.

A segunda parte é a condição, que será avaliada sempre que uma iteração do laço começa. Contém uma expressão para decidir quando parar o loop, ou melhor, a condição que deve ser cumprida para que o loop continue a funcionar.

Finalmente temos a atualização, que serve para indicar as alterações que queremos executar nas variáveis cada vez que a iteração do loop termina, antes de verificar se deve continuar a executar.

Após o for, as frases que queremos executar em cada iteração são colocadas, fechadas em aparelhos encaracolados.

Um exemplo de utilização deste laço pode ser visto abaixo, onde os números de 0 a 10 serão impressos.

```
var i
for (i = 0; i <= 10; i ++) {
   document.write (i)
   document.write ("<br>")
}
```

Neste caso, a variável i é inicializada para 0. Como condição para realizar uma iteração, deve ser cumprido que a variável i é inferior ou igual a 10. Como atualização, a variável i será aumentada em 1.

Como podem ver, este ciclo é muito poderoso, uma vez que numa única linha podemos indicar muitas coisas diferentes e muito variadas, o que permite uma configuração rápida do loop e uma enorme versatilidade.

Por exemplo, se quisermos escrever os números de 1 a 1.000 dois por dois, o seguinte loop será escrito.

```
for (i = 1; i <= 1000; i + = 2)
   document.write (i)
```

Se olharmos, em cada iteração atualizamos o valor do i aussi-lo em 2 unidades.

Nota: Outro detalhe, não utilizamos as teclas que englobam as instruções do lacete FOR porque tem apenas uma declaração e neste caso não é forçada, como aconteceu com as instruções IF.

Se quisermos contar de 343 para 10, usaremos este ciclo.

```
for (i = 343; i> = 10; i--)
    document.write (i)
```

Neste caso, diminuímos a variável i por uma unidade em cada iteração, começando pelo valor 343 e desde que a variável tenha um valor superior ou igual a 10.

Exercício de amostra para loop

Vamos parar para assimilar o loop com um exercício que não levanta quaisquer dificuldades se tivermos entendido o funcionamento do loop.

Este é um loop que escreve os títulos de <H1> para <H6> numa página web com um texto que diz "Level x heading".

O que queremos escrever para uma página web usando o Javascript é o seguinte:

```
<H1> Level 1 Header </H1>
<H2> Level 2 Header </H2>
<H3> Level 3 Header </H3>
<H4> Level 4 Header </H4>
<H5> Level 5 Header </H5>
<H6> Level 6 Header </H6>
```

Para isso temos de fazer um loop que começa a 1 e termina a 6 e em cada iteração vamos escrever o título em que toca.

```
for (i = 1; i <= 6; i ++) {
   document.write ("<H" + i + "> Level header" + i
+ "</ H" + i + ">")
}
```

Agora que estamos familiarizados com o loop for, estamos em posição de aprender a lidar com outras estruturas de controlo para repetir, como o tempo e fazer ... enquanto loops.

ENQUANTO e DO WHILE loops

Estamos a lidar com as diferentes estruturas de controlo que existem na língua javascript e especificamente olhando para os diferentes tipos de loops que podemos implementar nesta linguagem de programação. Em artigos anteriores do Manual Javascript já vimos o primeiro dos ciclos que devemos saber, o loop e agora vamos lidar com os outros dois tipos de estruturas de controlo para fazer repetições. Vamos agora olhar para os dois tipos de loops TEMPO que podemos usar no Javascript e os usos de cada um.

ENQUANTO loop

Estes ciclos são usados quando queremos repetir a execução de algumas frases um número indefinido de vezes, desde que uma condição seja cumprida. É mais fácil compreender que o loop FOR, uma vez que não incorpora na mesma linha a inicialização das variáveis a sua condição para continuar a executar e a sua atualização. Apenas a condição que deve ser cumprida para que uma iteração seja realizada é indicada, como veremos abaixo.

```
while (condition) {
  // statements to execute
}
```

An example of code where this loop is used can be seen below.

```
var color = ""
while (color! = "red") {
  color = prompt ("give me a color (write red to exit)", "")
}
```

Este é um exemplo da coisa mais simples a fazer com um loop de tempo. O que faz é pedir ao utilizador para introduzir uma cor e fá-lo repetidamente, desde que a cor inserida não seja vermelha. Para executar um loop como este, primeiro temos que rubricar a variável que vamos usar na condição de iteração em loop. Com a variável inicializada podemos escrever o loop, que verificará para executar que a variável de cor é diferente de "vermelho". Em cada iteração do

loop é solicitada ao utilizador uma nova cor para atualizar a variável de cor e a iteração está terminada, por isso voltamos ao início do loop, onde temos que reavaliar se o que está na variável de cor é "vermelho" e assim por diante enquanto o texto "vermelho" não tiver sido introduzido como uma cor.

Nota: Neste exemplo, utilizamos a função pronta javascript, que ainda não vimos neste manual. Esta função é utilizada para apresentar uma caixa de diálogo onde o utilizador deve escrever um texto. Esta função pertence ao objeto da janela Javascript e discutimos no artigo Métodos de janela em Javascript.

Fazer... ENQUANTO loop

O fazer... enquanto loop é a última das estruturas para implementar repetições disponíveis em Javascript e é uma variação do loop de tempo visto acima. É geralmente usado quando não sabemos quantas vezes o loop será executado, assim como o loop WHILE, com a diferença que sabemos com certeza que o loop será executado pelo menos uma vez.

Este tipo de loop foi introduzido no Javascript 1.2, pelo que nem todos os navegadores o suportam, apenas a versão 4 ou superior. De qualquer forma, qualquer código que queira escrever com o DO... O WHILE também pode ser escrito usando um loop WHILE, por isso em navegadores mais antigos terá de traduzir o seu DO ... ENQUANTO loop em um loop DE TEMPO.

A sintaxe é a seguinte:

```
do {
  // loop statements
} while (condition)
```

O loop é sempre executado uma vez e no final a condição é avaliada para dizer se o loop é executado novamente ou a sua execução termina.

Vejamos o exemplo que escrevemos para um loop DE WHILE neste outro tipo de loop.

```
var color

do {

  color = prompt ("give me a color (write red to exit)", "")

} while (color! = "red")
```

Este exemplo funciona exatamente igual ao anterior, exceto que não tivemos que rubricar a variável de cor antes de entrar no loop. Pede uma cor enquanto a cor inserida é diferente de "vermelho".

Exemplo do uso de loops de enquanto

Vejamos abaixo um exemplo mais prático sobre como trabalhar com um loop WHILE. Como é muito difícil dar exemplos práticos com o pouco que sabemos sobre o Javascript, vamos avançar com uma instrução que ainda não conhecemos.

Neste exemplo vamos declarar uma variável e inicializá-la para 0. Em seguida, adicionaremos um número aleatório de 1 a 100 a essa variável até adicionarmos 1.000 ou mais, imprimindo o valor da variável de soma após cada operação. Será necessário utilizar o loop WHILE porque não sabemos exatamente o número de iterações que teremos de realizar (dependerá dos valores aleatórios obtidos).

```
var sum = 0

while (sum <1000) {

   sum + = parseInt (Math.random () * 100)

   document.write (sum + "<br>")

}
```

Assumimos que, no que diz respeito ao ciclo WHILE, não haverá problemas, mas onde pode haver problemas está na declaração utilizada para levar um número aleatório. No entanto, não é necessário explicar a frase aqui, porque planeamos fazê-lo mais tarde. De qualquer forma, se quiser, pode ver este artigo que fala sobre números aleatórios em Javascript.

Quebrar e continuar

O Javascript tem diferentes estruturas de controlo para implementar loops, tais como FOR, WHILE e DO ... WHILE, que já pudemos explicar em capítulos anteriores do Manual Javascript. Como vimos, com estes ciclos podemos cobrir um grande número de necessidades, mas talvez com o tempo descubra que está a perder algumas possibilidades de controlar as repetições dos ciclos.

Imagine, por exemplo, que está a fazer um ciclo muito longo para encontrar algo em centenas ou milhares de sites. Mas coloque-se no caso de que durante as primeiras iterações você encontra o valor que você estava procurando. Então não faria sentido passar pelo resto do ciclo para procurar esse item, como já o tinhas encontrado. Nestas situações, é conveniente para nós saber para o loop cancelar o resto das iterações. Obviamente, este é apenas um exemplo de como podemos precisar de controlar um pouco mais o loop. Na vida real, como programador, encontrará muitas outras ocasiões em que estará interessado em fazer isto ou outras coisas com eles.

Assim, existem duas instruções que podem ser usadas nas diferentes estruturas de controlo e principalmente nos ciclos, o que o ajudará a controlar dois tipos de situações. São as instruções de rutura e continuação:

rutura: Significa parar a execução de um loop e deixá-lo.

continuar: Usado para parar a iteração atual e voltar ao início do loop para efetuar outra iteração, se aplicável.

Pausa

Um loop é parado usando a quebra de palavras. Parar um loop significa sair e deixar tudo como é para continuar o fluxo do programa imediatamente após o loop.

```
for (i = 0; i <10; i ++) {
  document.write (i)
  type = prompt ("tell me if I keep asking ...", "yes")
  if (type == "no")
    break
}
```

Este exemplo escreve os números de 0 a 9 e em cada iteração do loop pergunta ao utilizador se quer continuar. Se o utilizador disser alguma coisa, continua, exceto quando diz "não", situação em que sai do circuito e deixa a conta onde tinha deixado de lado.

Contínuo

É usado para voltar ao início do loop a qualquer momento, sem executar as linhas abaixo da palavra continuar.

```
var i = 0

while (i <7) {

  increase = prompt ("The account is at" + i + ", tell me if I
increase", "if")

  if (increment == "no")

    continue

  i ++

}
```

Este exemplo normalmente contaria de i = 0 a i = 7, mas cada vez que o laço é executado pergunta ao utilizador se quer aumentar a variável ou não. Se introduzir "não", a declaração contínua é executada, que regressa ao início do ciclo sem aumentar a variável i em 1, uma vez que as declarações abaixo da continuação seriam ignoradas.

Exemplo adicional da declaração de rutura

Um exemplo mais prático destas instruções pode ser visto abaixo. É um loop FOR planeado para ir até 1.000, mas vamos detê-lo com uma pausa quando chegarmos a 333.

```
for (i = 0; i <= 1000; i ++) {
   document.write (i + "<br>")
   if (i == 333)
      break;
}
```

Loops aninhados em Javascript

No Manual Javascript já abordámos vários artigos para falar sobre loops. Neste momento não deverá haver qualquer problema em criar os diferentes tipos de loops sem problemas, no entanto, queremos dedicar um artigo inteiro para discutir um dos usos mais comuns dos loops, que podemos encontrar quando estamos a fazer programas mais complexos : ciclos de nidificação.

Nidificar um laço consiste em colocar esse laço dentro de outro. A nidificação em loop é necessária para tornar o processamento um pouco mais complexo do que o que vimos nos exemplos anteriores. Se, na sua experiência como programadores, ainda lhes deu um loop, certifique-se de que, mais cedo ou mais tarde, irá satisfazer essa necessidade.

Um laço aninhado tem uma estrutura como a de baixo. Tentaremos explicá-lo tendo em conta estas linhas:

```
for (i = 0; i <10; i ++) {
  for (j = 0; j <10; j ++) {
  document.write (i + "-" + j)
  }
}
```

A execução funcionará da seguinte forma. Para começar, o primeiro loop é inicializado, por isso variável eu vou ser 0, e então o segundo loop é inicializado, então variável j também será 0. Em cada iteração, o valor da variável i é impresso, um hífen (" - ") e o valor da

variável j, uma vez que as duas variáveis valem 0, o texto" 0-0 "será impresso na página web.

Devido ao fluxo do programa em esquemas de nidificação como o que vimos, o loop que está aninhado (mais adiante) é o que é executado mais vezes. Neste exemplo, para cada iteração do laço ultraperiferdo, o laço aninhado funcionará completamente uma vez, ou seja, fará as suas 10 iterações. Estes valores seriam escritos na página web, na primeira iteração do lacete externo e desde o início:

0-0

0-1

0-2

0-3

0-4

0-5

0-6

0-7

0-8

0-9

Para cada iteração do laço exterior, serão executadas 10 iterações do laço interior ou aninhado. Vimos a primeira iteração, agora vamos ver as seguintes iterações do laço exterior. Em cada um acumula-se uma unidade em variável i, com o que estes valores viriam a sair.

1-0

1-1
1-2
1-3
1-4
1-5
1-6
1-7
1-8
1-9

E depois estes:

2-0
2-1
2-2
2-3
2-4
2-5
2-6
2-7
2-8
2-9

Assim, até que as duas voltas estejam terminadas, o que seria quando o valor 9-9 é alcançado.

Vejamos um exemplo muito semelhante ao anterior, embora um pouco mais útil. O objetivo é imprimir todas as tabelas de multiplicação na página. De 1 a 9, isto é, a mesa de 1, a mesa de 2, de 3 ...

```
for (i = 1; i <10; i ++) {
   document.write ("<br> <b> The" + i + "table:
</b> <br>")
   for (j = 1; j <10; j ++) {
      document.write (i + "x" + j + ":")
      document.write (i * j)
      document.write ("<br>")
   }
}
```

Com o primeiro laço controlamos a tabela atual e com o segundo laço desenvolvemo-la. No primeiro ciclo escrevemos um cabeçalho, em negrito, indicando a tabela que estamos escrevendo, primeiro o de 1 e depois os outros em ordem ascendente até 9. Com o segundo laço escrevo cada um dos valores em cada tabela. Pode ver o exemplo em curso neste link.

Nota: Veremos mais coisas com loops aninhados em capítulos posteriores, embora se quisermos ir um pouco à frente para ver um

novo exemplo que fortaleça este conhecimento, podemos ver um exemplo no Javascript Workshop em loops aninhados, onde a mesa é construída com todas as cores charutos em 256 definições de cores.

Funções javascript

Continuamos a trabalhar e a expandir o nosso conhecimento do Javascript. Com o que vimos até agora no Manual Javascript, já temos uma certa fluência para trabalhar nesta interessante linguagem de programação. Mas ainda temos um longo caminho a percorrer.

Agora vamos ver um tema muito importante, especialmente para aqueles que nunca programaram e com o Javascript estão a dar os primeiros passos no mundo da programação, uma vez que veremos um novo conceito, o de função e os usos que tem. Para quem já conhece o conceito de função, será também um capítulo útil, pois também veremos a sintaxe e o funcionamento das funções no Javascript.

O que é uma função

Ao fazer um programa ligeiramente grande, existem certos processos que podem ser concebidos de forma independente, e que são mais fáceis de resolver do que todo o problema. Além disso, estes são geralmente realizados repetidamente ao longo da execução do programa. Estes processos podem ser agrupados numa função, definida para que não tenhamos de repetir esse código repetidamente nos nossos scripts, mas simplesmente chamar a função e é responsável por fazer tudo o que deve.

Assim, podemos ver uma função como uma série de instruções que incluímos dentro do mesmo processo. Este processo pode então ser executado a partir de qualquer outro site apenas chamando-o. Por exemplo, numa página web pode haver uma função para alterar a cor do fundo e a partir de qualquer ponto da página poderíamos chamá-lo para mudar a cor quando quisermos.

Nota: Se quisermos, podemos expandir esta descrição das funções no conceito de função do artigo.

As funções são constantemente utilizadas, não só aquelas que escreve, mas também aquelas que já estão definidas no sistema, uma vez que todas as linguagens de programação geralmente têm muitas funções para realizar processos comuns, como obter o tempo, imprimir uma mensagem no ecrã ou converter variáveis de um tipo para outro. Já vimos alguma função nos nossos exemplos simples acima. Por exemplo, quando estávamos a fazer um documento.write () estávamos na verdade a chamar a função de escrita () associada ao documento da página, que escreve texto para a página.

Nos capítulos de funções veremos primeiro como desempenhar as nossas próprias funções e como chamá-las mais tarde. Ao longo do manual veremos muitas das funções definidas no Javascript que devemos usar para executar diferentes tipos de ações comuns.

Como escrever uma função

Uma função deve ser definida com uma sintaxe especial que saberemos a seguir.

```
function functionname () {
  function instructions
  ...
}
```

Primeiro escreva a função palavra, reservada para este uso. Em seguida, o nome da função é escrito, que, como nomes variáveis, pode ter números, letras e algum caráter adicional como em sublinhado. Em seguida, as diferentes instruções da função são colocadas em aparelhos. Os aparelhos no caso de funções não são opcionais, e também é útil colocá-los sempre como visto no exemplo, de modo que a estrutura das instruções incluídas na função seja facilmente reconhecida.

Vamos ver um exemplo de uma função para escrever uma mensagem de boas-vindas na página dentro <H1> tags para que seja mais destacada.

```
function write Welcome () {
   document.write ("<H1> Hello everyone </H1>")
}
```

Basta escrever texto na página. Admitimos que é uma função tão simples que o exemplo não expressa suficientemente o conceito de

função, mas veremos outras mais complexas. As tags H1 não são mostradas na página, mas são interpretadas como o significado da página, neste caso escrevemos um cabeçalho de nível 1. Uma vez que estamos a escrever numa página web, ao colocar tags HTML são interpretadas como o que são.

Como chamar uma função

Para executar uma função temos que invocá-la em qualquer lugar da página. Com isto, receberemos todas as instruções que a função tem entre as duas teclas a serem executadas.

Para executar a função usamos o seu nome seguido pelos parênteses. Por exemplo, isto chamaria a função writeWelcome () que acabamos de criar.

escrever Bem-vindo ()

Onde colocamos as funções Javascript

As funções são um dos principais componentes dos programas, na maioria das linguagens de programação. No Manual Javascript já começamos a explicar o que é uma função e como podemos criá-la e invocá-la nesta língua. Agora vamos lidar com um tópico que não tem tanto a ver com sintaxe e programação, mas tem mais a ver com o uso correto e habitual que é feito das funções no Javascript, que é nada mais nada menos do que a colocação do código das funções na página Web.

Em princípio, podemos colocar as funções em qualquer lugar da página, sempre entre <SCRIPT> tags, claro. No entanto, existe uma limitação na sua colocação em relação aos locais a partir dos quais é chamado. Prevemos que o mais fácil é colocar a função antes de qualquer chamada e, portanto, certamente nunca cometeremos um erro.

Existem duas opções possíveis para colocar o código de uma função:

a) Colocar a função no mesmo bloco de script: Especificamente, a função pode ser definida no bloco <SCRIPT> bloco onde está a chamada de função, embora não importe se a chamada é antes ou depois do código de função, dentro do mesmo bloco <SCRIPT>

```
<SCRIPT>
myFunction ()
function myFunction () {
   // I do something ...
   document.write ("This is fine")
}
</SCRIPT>
```

Este exemplo funciona corretamente porque a função é declarada no mesmo bloco que a sua chamada.

b) Coloque a função noutro bloco de script: Também é válido que a função esteja num bloco <SCRIPT> bloqueio antes do bloco onde está a chamada.

```
<HTML>
<HEAD>
   <TITLE> MY PAGE </TITLE>
<SCRIPT>
function myFunction () {
   // I do something ...
   document.write ("This is fine")
}
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT>
myFunction ()
</SCRIPT>

</BODY>
</HTML>
```

Vemos um código completo sobre como uma página web pode estar onde temos funções Javascript. Como pode ver, as funções estão no topo da página (dentro da CABEÇA). Este é um excelente lugar para colocá-los, porque se presume que eles ainda não serão usados no cabeceamento e podemos sempre apreciá-los no corpo porque sabemos com certeza que já foram declarados.

Para tornar esta questão da colocação da função clara, vamos olhar para o seguinte exemplo, o que daria um erro. Olhe atentamente para o seguinte código, que irá lançar um erro, porque fazemos uma chamada para uma função que é declarada num bloco de <SCRIPT>

```
<SCRIPT>
myFunction ()
</SCRIPT>


<SCRIPT>
function myFunction () {
   // I do something ...
   document.write ("This is fine")
}
</SCRIPT>
```

Com isto esperamos ter resolvido todas as dúvidas sobre a colocação do código das funções Javascript. Nos seguintes artigos veremos outros tópicos interessantes, como os parâmetros das funções.

Parâmetros de função

No Manual Javascript já falámos sobre funções. Concretamente, este é o terceiro artigo que abordamos sobre o assunto.

As ideias que já explicámos sobre funções não são as únicas que temos de aprender a lidar com elas com todo o seu potencial. As funções também têm entrada e saída de dados. Neste artigo veremos como podemos enviar dados para funções Javascript.

Propriedade de parâmetros

Os parâmetros são usados para enviar valores para funções. Uma função funcionará com os parâmetros para executar as ações. Por outra forma, os parâmetros são os valores de entrada que uma função recebe.

Para dar um exemplo fácil de entender, uma função que executa uma soma de dois números teria esses dois números como parâmetros. Os dois números são a entrada, assim como a saída seria o resultado da soma, mas veremos isso mais tarde.

Vejamos um exemplo anterior em que criámos uma função para exibir uma mensagem de boas-vindas na página web, mas agora vamos passar um parâmetro que conterá o nome da pessoa a ser saudada.

```
function write Welcome (name) {
  document.write ("<H1> Hello" + name + "</H1>")
}
```

Como podemos ver no exemplo, para definir um parâmetro na função temos que colocar o nome da variável que vai armazenar os dados que lhe passamos. Essa variável, que neste caso é chamada de nome, terá como valor os dados que transmitimos para a função quando a chamamos. Além disso, a variável onde recebemos o parâmetro terá vida durante a execução da função e deixará de existir quando a função terminar a sua execução.

Para chamar uma função que tenha parâmetros, coloque o valor do parâmetro nos parênteses. Para chamar a função de exemplo, teria de escrever:

```
writeWelcome ("Alberto García")
```

Ao chamar a função desta forma, o parâmetro do nome leva o valor "Alberto García" e ao escrever a saudação no ecrã escreverá "Olá Alberto García" entre <H1> tags.

Os parâmetros il podem receber qualquer tipo de dados, numéricos, textuais, booleanos ou um objeto. Não especificamos realmente o tipo de parâmetro, pelo que temos de ter especial cuidado ao definir as ações que realizamos dentro da função e passar valores para o mesmo, para garantir que tudo seja consistente com os tipos de dados que esperamos que as nossas variáveis ou parâmetros tenham.

Múltiplos parâmetros

Uma função pode receber quantos parâmetros quisermos e expressá-lo os nomes dos parâmetros são separados por vírgulas, dentro dos parênteses. Vamos rapidamente ver a sintaxe para a função de antes, mas feita para que receba dois parâmetros, o primeiro o nome a cumprimentar e o segundo a cor do texto.

```
function writeWelcome (name, colorText) {
   document.write ("<FONT color = '" + colorText + "'>")
   document.write ("<H1> Hello" + name + "</H1>")
   document.write ("</FONT>")
}
```

Chamaríamos a função com esta sintaxe. Entre os parênteses colocaremos os valores dos parâmetros.

```
var myName = "Pepe"
var myColor = "red"
write Welcome (myName, myColor)
```

Coloquei duas variáveis em parênteses em vez de dois textos citados. Quando colocamos variáveis entre os parâmetros que estamos realmente passando para a função são os valores que as variáveis contêm e não as próprias variáveis.

Os parâmetros são passados pelo valor

Em linha com o uso de parâmetros nos nossos programas Javascript, temos de saber que os parâmetros das funções são passados por valor. Isto significa que estamos a passar valores e não variáveis. Na prática, mesmo que modifiquemos um parâmetro numa função, a variável original que tínhamos passado não mudará o seu valor. Pode ser facilmente visto com um exemplo.

```
function stepValue (myParameter) {

   myparameter = 32

   document.write ("I have changed the value to 32")

}

var myVariable = 5

stepValue (myVariable)

document.write ("the value of the variable is:" +
myVariable)
```

No exemplo temos uma função que recebe um parâmetro e modifica o valor do parâmetro que lhe atribui o valor 32. Também temos uma variável, que inicializamos para 5 e depois chamamos a função passando esta variável como um parâmetro. Como dentro da função modificamos o valor do parâmetro, pode acontecer que a variável original altere o seu valor, mas como os parâmetros não modificam o valor original das variáveis, não altera o seu valor.

Desta forma, uma vez executada a função, ao imprimir o valor do miVariable no ecrã, o número 5 será impresso, que é o valor original da variável, em vez de 32, que foi o valor com o qual tínhamos atualizado o parâmetro.

No Javascript, só se pode passar variáveis por valor.

Valores de retorno nasfunções Javascript

Estamos a aprender sobre o uso de funções no Javascript e neste momento podemos já ter percebido a grande importância que eles têm em fazer programas mais ou menos avançados. Neste artigo do Manual Javascript continuaremos a aprender coisas sobre funções e especificamente que com elas também pode devolver valores. Além disso, veremos algum caso de uso interessante sobre as funções que podem clarificar um pouco o âmbito das variáveis locais e globais.

Valores de retornoem funções

As funções javascript também podem devolver valores. De facto, esta é uma das utilidades mais essenciais das funções, que devemos conhecer, não só no Javascript, mas em geral em qualquer linguagem de programação. Assim, invocando uma função, pode executar ações e oferecer um valor como saída.

Por exemplo, uma função que calcule o quadrado de um número terá esse número como entrada e como saída terá o valor resultante de encontrar o quadrado desse número. A entrada de dados nas funções que vimos anteriormente no artigo sobre parâmetros de função. Agora temos de aprender sobre a saída.

Vejamos um exemplo de uma função que calcula a média de dois números. A função receberá ambos os números e devolverá o valor da média.

```
function media (value1, value2) {
  var result
  result = (value1 + value2) / 2
  return result
}
```

Para especificar o valor que a função irá devolver, a palavra devolução é utilizada, seguida do valor que pretende devolver. Neste caso, o conteúdo da variável de resultados é devolvido, que contém a média calculada dos dois números.

Podemos agora perguntar-nos como receber um dado devolvido por uma função. Na verdade, no código fonte dos nossos programas podemos invocar as funções no lugar que queremos. Quando uma função devolve um valor, a chamada de função é simplesmente substituída pelo valor que devolve. Assim, para armazenar um valor de retorno de uma função, temos que atribuir a chamada a essa função como conteúdo numa variável, e faríamos isso com o operador de atribuição =.

Para ilustrar isto, pode ver este exemplo, que chamará a função média () e salvará o resultado da média numa variável e, em seguida, imprima-o na página.

```
var my media

myAverage = mean (12.8)

document.write (myMedia)
```

Devoluções múltiplas

Na verdade, as funções Javascript só podem devolver um valor, por isso, em princípio, não podemos fazer funções que devolvam dois dados diferentes.

Nota: na prática nada nos impede de ter uma função de retorno superior a um valor, mas como só podemos devolver uma coisa, teríamos de colocar todos os valores que queremos devolver numa estrutura de dados, como um conjunto. No entanto, essa seria uma utilização mais ou menos avançada que não vamos ver neste momento.

Agora, embora só possamos devolver um dado, na mesma função podemos colocar mais do que um retorno. Como dizemos, só poderemos devolver uma coisa, mas dependendo do que aconteceu na função pode ser de um tipo ou de outro, com alguns dados ou outros.

Nesta função podemos ver um exemplo de utilização de retorno múltiplo. Esta é uma função que devolve 0 se o parâmetro recebido foi uniforme e o valor do parâmetro se fosse estranho.

```
function multipleReturn (number) {
  var remainder = number% 2
  if (remainder == 0)
    return 0
  else
    return number
}
```

Para descobrir se um número é mesmo, encontramos o resto da divisão dividindo-o por 2. Se o resto é zero é que foi par e nós devolvemos 0, caso contrário - o número é estranho - devolvemos o parâmetro recebido.

Âmbito de variáveis em funções

Dentro das funções podemos declarar variáveis. Nesta matéria, temos de saber que todas as variáveis declaradas numa função são locais para essa função, ou seja, só serão válidas durante a execução da função.

Nota: Mesmo que pensemos nisso, podemos perceber que os parâmetros são como variáveis que são declaradas no cabeçalho da função e que são inicializadas ao chamar a função. Os parâmetros também são locais para a função e só serão válidos quando a função estiver em funcionamento.

Pode ser que possamos declarar variáveis em funções que têm o mesmo nome que uma variável global para a página. Assim, dentro da função, a variável que será válida é a variável local e fora da função a variável global para a página será válida.

Por outro lado, se não declararmos as variáveis nas funções, entender-se-á pelo javascript que estamos a referir-nos a uma variável global para a página, por isso, se a variável não for criada, cria-a, mas sempre global para a página em vez de local para a função.

Vamos ver o seguinte código.

```
function variables_glogales_y_locales () {
  var local variable = 23
  globalGlobal = "qwerty"
}
```

Neste caso, a variávelLocal é uma variável que foi declarada na função, pelo que será local para a função e só será válida durante a sua execução. Por outro lado, a variávelGlobal não foi declarada (porque antes de a utilizar, a palavra VAR não foi usada para o declarar). Neste caso, a variável variávelGlobal é global a toda a página e continuará a existir mesmo que a função termine a sua execução. Além disso, se a variável variávelGlobal existisse antes de chamar a função, como resultado da execução desta função, um valor hipotético dessa variável seria batido e substituído por "qwerty".

Nota: Podemos encontrar mais informações sobre o âmbito variável num artigo anterior.

Biblioteca de funções javascript

Em todas as linguagens de programação existem bibliotecas de funções que servem para fazer coisas diferentes e muito repetitivas na programação. As bibliotecas em linguagens de programação poupam-lhe o trabalho de escrever funções comuns que os programadores podem normalmente precisar. Uma linguagem de programação bem desenvolvida terá uma boa quantidade delas. Às vezes é mais difícil conhecer bem todas as bibliotecas do que aprender a programar na língua.

O Javascript contém um bom número de funções nas suas bibliotecas. Como é uma linguagem que funciona com objetos, muitas das bibliotecas são implementadas através de objetos. Por exemplo, as funções de manipulação de matemática ou de cordas são implementadas utilizando objetos de matemática e cordas. No entanto, existem algumas funções que não estão associadas a nenhum objeto e são as que veremos neste capítulo, uma vez que ainda não conhecemos os objetos e não precisaremos que os estudem.

Funções javascript incorporadas

Estas são as funções que o Javascript disponibiliza aos programadores.

método de eval (cadeia)

Esta função recebe uma corda de carácter e executa-a como se fosse uma declaração javascript.

parseInt (corda, base)

Receba uma corrente e baseada. Devolve um valor numérico resultante da conversão da cadeia para um número na base indicada.

método parseFloat (cadeia)

Converte a corda num número e devolve-a.

fuga (carácter)

Devolve um personagem que recebe como parâmetro numa codificação ISO Latin 1.

unescape (carácter)

Faz exatamente o oposto da função de fuga.

isNaN (número)

Devolve um booleano dependendo do que recebe por parâmetro. Se não for um número, retorna verdadeiro, se for um número, retorna falso.

As bibliotecas que são implementadas através de objetos e as do manuseamento do navegador, que também são geridas com objetos, veremos mais tarde.

Nota: Não queremos induzir as pessoas em erro com esta pequena lista de funções javascript nativas. Há realmente muitas outras funções que vamos ver ao longo deste manual, o que acontece é que eles estão associados com objetos. Por exemplo, como referimos, existem funções de cadeia de caracteres, que estão associadas a objetos de corda, funções para trabalhar com cálculos

matemáticos avançados, que estão associados à classe Matemática, funções para trabalhar com o objeto da janela do navegador, com o documento, etc.

Exemplos de utilização das funções incorporadas no Javascript

Até agora, conhecemos simplesmente uma lista das funções nativas da língua javascript. Agora podemos ver vários exemplos do uso de funções javascript nativas, que temos disponíveis em qualquer navegador e em qualquer versão do Javascript.

Veremos três funções de diferentes áreas que são bastante fundamentais no trabalho habitual com esta linguagem, explicadas através de exemplos.

Função Eval

Esta função é muito importante, tanto que existem algumas aplicações Javascript que não poderiam ser executadas se não a usarmos. A sua utilização é muito simples, mas pode ser um pouco mais complexa para entender em que casos usá-lo porque às vezes a sua aplicação é um pouco subtil.

Com o conhecimento atual não podemos dar um exemplo muito complicado, mas pelo menos podemos ver a função funcionando. Vamos usá-lo numa declaração um pouco estranha e bastante inútil, mas se conseguirmos compreendê-la, também compreenderemos a função eval.

```
var myText = "3 + 5"

eval ("document.write (" + myText + ")")
```

Primeiro criamos uma variável com um texto, na linha seguinte usamos a função eval e como parâmetro passamos uma instrução javascript para escrever no ecrã. Se concatenarmos as cordas que estão dentro dos parênteses da função eval, ficamos com isto.

```
document.write (3 + 5)
```

A função de avaliação executa a instrução que lhe foi passada por parâmetro, pelo que executará esta declaração, o que resultará na escrita de um 8 para a página web. Primeiro a soma em parênteses é resolvida, então nós temos o 8 e depois a escrita no ecrã instruções é executada.

Função ParseInt

Esta função recebe um número, escrito como uma corda, e um número que indica uma base. Pode realmente receber outros tipos de variáveis, uma vez que as variáveis não têm tipo em Javascript, mas geralmente é usado passando uma cadeia para converter a variável de texto para um número.

As diferentes bases que a função pode receber são 2, 8, 10 e 16. Se não passarmos qualquer valor como base, a função interpreta a base como decimal. O valor devolvido pela função tem sempre a base 10, por isso, se a base não for 10, converte o número para essa base antes de o devolver.

Vamos olhar para uma série de chamadas para a função parseInt para ver o que ele retorna e entender a função um pouco mais.

```
document.write (parseInt ("34"))
```

Returns the number 34

```
document.write (parseInt ("101011", 2))
```

Returns the number 43

```
document.write (parseInt ("34", 8))
```

Returns the number 28

```
document.write (parseInt ("3F", 16))
```

Returns the number 63

Esta função é usada na prática para muitas coisas diferentes no manuseamento de números, por exemplo, obter a parte mais pequena de um decimal.

```
document.write (parseInt ("3.38"))
```

Returns the number 3

Também é muito comum usá-lo para saber se uma variável é numérica, porque se passarmos um texto para a função que não é numérica, devolverá NaN (Não um Número) o que significa que não é um Número.

```
document.write (parseInt ("develoloweb.com"))

Returns the NaN number
```

Este mesmo exemplo é interessante com uma modificação, porque se passarmos uma combinação de letras e números, ele nos dará o seguinte.

```
document.write (parseInt ("16XX3U"))

Returns the number 16
```

```
document.write (parseInt ("TG45"))

Returns the NaN number
```

Como pode ver, a função tenta converter a corda para o número e se não puder, devolve a NaN.

Todos estes exemplos algo não relacionados de como o parseInt funciona será revisto mais tarde em exemplos mais práticos quando se trata de trabalhar com formulários.

Função IsNaN

Esta função devolve um Boolean dependendo se recebe um número ou não. A única coisa que pode receber é um número ou a expressão NaN. Se receber uma NaN, retorna verdadeiramente e se

receber um número, retorna falsa. É uma função muito simples de compreender e usar.

A função geralmente funciona em combinação com a função parseInt ou parseFloat, para saber se o retorno destas duas funções é um número ou não.

```
miInteger = parseInt ("A3.6")

isNaN (myInteger)
```

Na primeira linha atribuímos à variável miInteger o resultado de tentar converter o texto A3.6 para um inteiro. Uma vez que este texto não pode ser convertido para um número, a função parseInt retorna NaN. A segunda linha verifica se a variável anterior é NaN e como se fosse, retorna verdadeira.

```
miFloat = parseFloat ("4.7")

isNaN (miFloat)
```

Neste exemplo, convertemos um texto para um número com decimais. O texto converte-se perfeitamente porque corresponde a um número. Ao receber um número, a função isNa devolve um falso.

Dispomos de um Workshop Javascript muito interessante que tem sido realizado para fortalecer o conhecimento destes capítulos. É um script para validar um campo de formulários para que tenhamos

a certeza de que dentro do campo há sempre um número inteiro. Pode ser muito interessante lê-lo agora, uma vez que usamos as funções isNaN () e parseInt (). Veja o workshop

Esperamos que os exemplos vistos neste artigo tenham sido interessantes. No entanto, como já vimos anteriormente, existem algumas outras funções javascript nativas que devemos estar cientes, mas que estão associadas a classes e objetos javascript nativos. Mas antes de avançarmos para esse ponto, queremos oferecer um pequeno guia básico para trabalhar com programação orientada para objetos em Javascript.

Matrizes em Javascript

Nas linguagens de programação existem estruturas de dados especiais que nos ajudam a armazenar informação mais complexa do que variáveis simples. Uma estrutura típica em todas as línguas é a Matriz, que é como uma variável onde podemos introduzir vários valores, em vez de apenas um como acontece com variáveis normais.

As matrizes permitem-nos guardar várias variáveis e acedê-las de forma independente, é como ter uma variável com compartimentos diferentes onde podemos introduzir diferentes dados. Para isso usamos um índice que nos permite especificar o compartimento ou posição a que nos referimos.

Nota: As matrizes foram introduzidas nas versões Javascript 1.1 ou superior, ou seja, só podemos usá-las a partir dos navegadores 3.0. Para navegadores mais antigos pode simular a matriz usando sintaxe de programação orientada a objetos, mas a verdade é que atualmente esta limitação não deve preocupar-nos. Além disso, dada a complexidade da tarefa de simular uma matriz através de objetos, pelo menos no momento em que nos encontramos e nas poucas ocasiões em que vamos precisar, pensamos que é melhor esquecer essa matéria e simplesmente trabalhar com matrizes normalmente. Assim, neste artigo e no seguinte vamos ver como usar a autêntica matriz javascript.

Criação de matrizes javascript

O primeiro passo para usar uma matriz é criá-la. Para isso utilizamos um objeto Javascript já implementado no navegador. Veremos agora um tópico para explicar o que é a orientação dos objetos, embora não seja necessário compreender a utilização de matrizes. Esta é a afirmação para criar um objeto de matriz:

```
var myArray = new Array ()
```

Isto cria uma matriz na página que está em execução. A matriz é criada sem qualquer conteúdo, ou seja, não terá nenhuma caixa ou compartimentos criados. Também podemos criar a matriz Javascript especificando o número de compartimentos que terá.

```
var myArray = new Array (10)
```

Neste caso, indicamos que a matriz terá 10 posições, ou seja, 10 caixas onde guardar dados.

É importante notar que a palavra Código Matriz em Javascript é escrita com a primeira letra capitalizada. Como em javascript maiúscula e minúscula, se for importante, se o escrevermos em minúsculas, não funcionará.

Quer o número de caixas na matriz javascript seja indicado, podemos introduzir quaisquer dados na matriz. Se a caixa for criada é simplesmente inserida e se a caixa não foi criada é criada e então os dados são introduzidos, então o resultado final é o mesmo. Esta criação de caixa de verificação é dinâmica e ocorre ao mesmo tempo que os scripts são executados. Vamos ver abaixo como introduzir valores nas nossas matrizes.

```
myArray [0] = 290

myArray [1] = 97

myArray [2] = 127
```

São introduzidas indicando em parênteses quadrados o índice da posição onde queríamos guardar os dados. Neste caso, entramos 290 na posição 0, 97 na posição 1 e 127 na posição 2.

As matrizes em Javascript começam sempre na posição 0, por isso uma matriz que tenha 10 posições, por exemplo, terá caixas de 0 a 9. Para recolher dados de uma matriz fazemos o mesmo: colocar o índice de posição a em parênteses quadrados a que queremos aceder. Vamos ver como o conteúdo de uma matriz seria impresso no ecrã.

```
var myArray = new Array (3)


myArray [0] = 155

myArray [1] = 4

myArray [2] = 499


for (i = 0; i <3; i ++) {

  document.write ("Position" + i + "of the array:" + myArray [i])

  document.write ("<br>")

}
```

Criámos uma matriz com três posições, depois introduzimos um valor em cada uma das posições da matriz e finalmente imprimimo-las. Em geral, a travessia de matrizes para imprimir as suas

posições, ou qualquer outra coisa, é feita com loops. Neste caso, usamos um loop FOR que vai de 0 a 2.

Podemos ver o exemplo a correr noutra página.

Tipos de dados em matrizes

Nas caixas das matrizes podemos guardar dados de qualquer tipo. Podemos ver uma matriz onde introduzimos dados de caracteres.

miArray [0] = "Hello"

myArray [1] = "a"

miArray [2] = "all"

Mesmo no Javascript podemos guardar diferentes tipos de dados nas caixas da mesma matriz. Ou seja, podemos introduzir números em algumas caixas, textos noutras, Booleans ou qualquer outra coisa que quisermos.

miArray [0] = "develoloweb.com"

myArray [1] = 1275

miArray [1] = 0.78

myArray [2] = true

Declaração de matriz e inicialização sumária

No Javascript temos à nossa disposição uma forma resumida de declarar uma matriz e valores de carga no mesmo passo. Vejamos o seguinte código:

```
var arrayRapido = [12,45, "array initialized in its declaration"]
```

Como podem ver, estamos a definir uma variável chamada fastArray e estamos a indicar nos parênteses quadrados vários valores separados por vírgulas. Isto é o mesmo que ter declarado a matriz com a função Array () e, em seguida, carregar os valores um por um.

Comprimento da matriz

No artigo anterior do Manual Javascript começamos a explicar o conceito de matriz e a sua utilização no Javascript. Neste artigo, continuaremos com o tema, mostrando o uso da sua propriedade de comprimento.

Todas as matrizes em javascript, além de armazenar o valor de cada uma das suas caixas, também armazenam o número de posições que têm. Para isso, eles usam uma propriedade do objeto matriz, a propriedade de comprimento. Veremos em objetos o que é uma propriedade, mas no nosso caso podemos imaginar que é como uma variável, além das posições, que armazena um número igual ao número de caixas que a matriz tem.

Para aceder a uma propriedade de um objeto, o operador de pontos deve ser utilizado. O nome da matriz a que queremos aceder está escrito ao número de posições que tem, sem parênteses ou parênteses, seguido de um período e do comprimento da palavra.

```
var myArray = new Array ()


myArray [0] = 155

myArray [1] = 499

myArray [2] = 65


document.write ("Array length:" + myArray.length)
```

Este código imprimiria no ecrã o número de posições da matriz, que neste caso é 3. Lembramo-nos que uma matriz com 3 posições varia entre a posição 0 e a 2.

É muito comum que a propriedade de comprimento seja usada para atravessar uma matriz através de todas as suas posições. Para o ilustrar, vamos ver um exemplo de uma caminhada por esta matriz para mostrar os seus valores.

```
for (i = 0; i <myArray.length; i ++) {

  document.write (myArray [i])

}
```

Note que o laço for for é executado sempre que eu for inferior ao comprimento da matriz, retirado da sua propriedade de comprimento.

O exemplo seguinte ajudar-nos-á a compreender melhor as rotas através das matrizes, o funcionamento da propriedade de comprimento e a criação dinâmica de novas posições. Vamos criar

uma matriz com 2 posições e preencher o seu valor. Mais tarde, introduziremos um valor na posição 5 da matriz. Finalmente, vamos imprimir todas as posições da matriz para ver o que acontece.

```
var myArray = new Array (2)


miArray [0] = "Colombia"
miArray [1] = "United States"


miArray [5] = "Brazil"


for (i = 0; i <myArray.length; i ++) {
  document.write ("Position" + i + "of the array:" + myArray [i])
  document.write ("<br>")
}
```

O exemplo é simples. Pode ver-se que caminhamos através da matriz de 0 para o número de posições na matriz (indicada pela propriedade de comprimento). Ao longo do caminho estamos a imprimir o número da posição seguida pelo conteúdo da matriz nessa posição. Mas podemos ter dúvidas quando nos perguntamos qual será o número de elementos nesta matriz, uma vez que o declarámos com 2 e depois introduzimos um terceiro na posição 5. Quando virmos a saída do programa, poderemos responder às nossas perguntas. Vai ficar mais ou menos assim:

Array position 0: Colombia

Array position 1: United States

Array position 2: null

Array position 3: null

Array position 4: null

Array position 5: Brazil

Pode ver-se claramente que o número de posições é de 6, de 0 a 5. O que aconteceu é que ao introduzir um dado na posição 5, todas as caixas que não foram criadas até à quinta também são criadas.

As posições 2 a 4 não são ininitializadas. Neste caso, o nosso navegador escreveu a palavra nulo para expressar isto, mas outros navegadores podem usar a palavra indefinido. Veremos mais tarde o que é este nulo e onde podemos usá-lo, o importante agora é que compreenda como as matrizes funcionam e as utilize corretamente.

Matrizes multidimensionais em Javascript

Como estamos a ver, as matrizes são muito importantes no Javascript e também na maioria das linguagens de programação. Especificamente, já aprendemos a criar matrizes e usá-las em artigos anteriores no Manual Javascript. Mas ainda temos algumas coisas importantes para explicar, como matrizes de várias dimensões.

As matrizes multidimensionais são uma estrutura de dados que armazena valores em mais de uma dimensão. As matrizes que vimos até agora armazenam valores numa dimensão, pelo que só usamos um índice para aceder a posições. As matrizes bidimensionais armazenam os seus valores, por assim dizer, em filas e colunas e, portanto, precisaremos de dois índices para aceder a cada uma das suas posições.

Por outras palavras, uma matriz multidimensional é como um recipiente que armazena mais valores para cada posição, isto é, como se os elementos da matriz fossem, por sua vez, outras matrizes.

No Javascript não há um verdadeiro objeto multidimensional. Per utilizzare questi estruturas podemos definir matrizes que em cada uma das suas posições haverá outra matriz. Nos nossos programas podemos usar matrizes de qualquer dimensão, veremos abaixo como trabalhar com matrizes bidimensionais, que serão as mais comuns.

Neste exemplo, vamos criar uma matriz bidimensional onde teremos cidades, por um lado, e a temperatura média em cada uma durante os meses de inverno, por outro.

```
var city_media_temperatures0 = new Array (3)
city_media_temperatures0 [0] = 12
city_media_temperatures0 [1] = 10
city_media_temperatures0 [2] = 11

var city_media_temperatures1 = new Array (3)
city_media_temperatures1 [0] = 5
city_media_temperatures1 [1] = 0
city_media_temperatures1 [2] = 2

var city_media_temperatures2 = new Array (3)
city_media_temperatures2 [0] = 10
city_media_temperatures2 [1] = 8
city_media_temperatures2 [2] = 10
```

Com as linhas anteriores, criámos três matrizes unidimensionais e três elementos, como os que já conhecíamos. Agora vamos criar uma nova matriz de três elementos e vamos introduzir em cada uma das suas caixas as matrizes criadas anteriormente, com as quais teremos uma matriz, ou seja, uma matriz bidimensional.

```
var cities_temperatures = new Array (3)

temperatures_city [0] = average_city_city0

city_temperatures [1] = city_average_ temperatures1

city_temperatures [2] = city_average_ temperatures2
```

Vemos que para introduzir toda a matriz nos referimos a ela sem parênteses ou parênteses, mas apenas com o seu nome. A matriz temperatures_cities é a nossa matriz bidimensional.

Também é interessante ver como um passeio por uma matriz bidimensional é realizado. Para isso temos de fazer um loop que percorre cada um dos quadrados da matriz bidimensional e dentro destes fazem um novo caminho para cada um dos seus quadrados internos. Isto é, uma visita de uma matriz dentro de outra.

O método de loop através de outro é colocar um laço dentro de outro, que é chamado de loop aninhado. Neste exemplo, vamos colocar um FOR dentro de outro. Além disso, vamos escrever os resultados numa tabela, o que vai complicar um pouco o guião, mas desta forma poderemos ver como construir uma mesa a partir de Javascript à medida que fazemos a digressão aninhada do loop.

```
document.write ("<table width = 200 border = 1 cellpadding =
1 cellspacing = 1>");

for (i = 0; i <cities_temperatures.length; i ++) {

   document.write ("<tr>")

   document.write ("<td> <b> City" + i + "</b> </td>")

   for (j = 0; j <temperatures_cities [i] .length; j ++) {

      document.write ("<td>" + cities_temperatures [i] [j] +
"</td>")

   }

   document.write ("</tr>")

}

document.write ("</table>")
```

Este guião é um pouco mais complexo do que os vistos anteriormente. A primeira ação é escrever o cabeçalho de mesa, que é, o <TABLE> tag juntamente com os seus atributos. Com o primeiro loop fazemos uma viagem até à primeira dimensão da matriz e usamos a variável i para acompanhar a posição atual. Para cada iteração deste ciclo escrevemos uma linha e para começar a linha abrimos a etiqueta <TR> Além disso, escrevemos numa caixa o número da cidade que visitamos naquele momento. Mais tarde colocamos outro laço que passa por cada uma das células da matriz na sua segunda dimensão e escrevemos a temperatura da cidade atual em cada um dos meses, dentro da sua <TD> tag. Uma vez terminada a segunda volta, as três temperaturas foram impressas e, portanto, a linha está terminada. O primeiro ciclo continua a repetir-se até que todas as cidades sejam impressas e uma vez terminadas fechamos a mesa.

Nota: Pode ter reparado que, por vezes, gerar código HTML a partir de Javascript torna-se complexo. Mas o problema não é só que o código é difícil de produzir, mas o pior é que se cria um código que é difícil de manter, no qual tanto a parte da programação em

Javascript é misturada com a parte da apresentação em HTML. O que também viu é apenas um código muito simples, com uma tabela realmente elementar, imagine o que aconteceria quando a tabela ou os dados fossem mais complexos. Felizmente, existem melhores formas de gerar a produção de HTML do que temos visto agora, embora esteja um pouco avançado neste momento. De qualquer forma, deixamos-lhe um link para o manual do sistema de modelos Javascript Handlebars, que é uma alternativa simples de biblioteca para gerar saída HTML a partir de Javascript.

Podemos ver o exemplo de execução e examinar todo o código do script.

Inicialização de matriz

Para terminar com o tema das matrizes, vamos ver uma forma de inicializar os seus valores ao mesmo tempo que o declaramos, para que possamos realizar o processo de introdução de valores em cada uma das posições da matriz de uma forma mais rápida.

O método normal de criar uma matriz que vimos foi através do objeto Array, colocando o número de caixas na matriz em parênteses ou não colocando nada, de modo que a matriz é criada sem qualquer posição. Para introduzir valores a uma matriz é feito o mesmo, mas colocando os valores com os quais queremos preencher as células separadas por vírgulas entre os parênteses. Vamos vê-lo com um exemplo que cria uma matriz com os nomes dos dias da semana.

```
var diasWeek = new Array ("Monday", "Tuesday", "Wednesday,",
"Thursday", "Friday", "Saturday", "Sunday")
```

A matriz é criada com 7 caixas, de 0 a 6 e em cada caixa o dia correspondente da semana é escrito (Em aspas porque é um texto).

Agora vamos ver algo mais complicado, trata-se de declarar a matriz bidimensional que usamos antes para as temperaturas das cidades nos meses numa única linha, entrando nos valores ao mesmo tempo.

```
var cities_temperatures = new Array (new Array (12,10,11), new
Array (5,0,2), new Array (10,8,10))
```

No exemplo introduzimos em cada caixa da matriz outra matriz que tem como valores as temperaturas de uma cidade em cada mês.

Javascript ainda tem uma forma mais resumida do que a que acabamos de ver, o que explicamos no primeiro artigo onde cobrimos matrizes. Para isso, simplesmente colocamos os dados da matriz que estamos a criar em parênteses quadrados. Para terminar, mostraremos um exemplo sobre como usar esta sintaxe para declarar matrizes de mais de uma dimensão.

```
var arrayMuchasDimensiones = [1, ["hello", "que", "tal", ["these",
"we are", "I am"], ["good", "bad"], "I just"], 2, 5];
```

Neste exemplo criámos uma matriz muito desigual, porque tem caixas com conteúdo de inteiros simples e outras com conteúdo de cordas e outras que são outras matrizes. Poderíamos aceder a algumas das suas caixas e mostrar os seus valores assim:

```
alert (arrayMuchasDimensiones [0])

alert (arrayMuchasDimensiones [1] [2])

alert (arrayMuchasDimensiones [1] [3] [1])
```

Introdução geral a objetos em Javascript

Vamos apresentar-nos a um tema javascript muito importante, como objetos. É um tema que ainda não vimos e sobre o qual vamos lidar constantemente, uma vez que a maioria das coisas no Javascript, mesmo a mais simples, vamos fazer através do manuseamento de objetos. De facto, nos exemplos feitos até agora, fizemos grandes esforços para não utilizar objetos e, mesmo assim, já os utilizamos ocasionalmente, uma vez que é muito difícil encontrar exemplos no Javascript que, embora simples, não os utilizam.

A Programação Orientada para o Objeto (OOP) representa uma nova forma de pensar ao fazer um programa. O Javascript não é uma linguagem de programação pura orientada a objetos porque, embora use objetos muitas vezes, não precisamos de programar todos os nossos programas com base neles. Na verdade, o que geralmente vamos fazer com o Javascript é usar objetos e não tanto programação orientada para objetos. Portanto, a forma de programação não vai mudar muito em comparação com o que vimos até agora no Manual javascript. Em resumo, o que vimos até agora no que diz respeito à sintaxe, funções, etc. ainda é perfeitamente válido e pode ser utilizado da mesma forma indicada. Só vamos aprender uma espécie de nova estrutura, como objetos.

Nota: Para começar a ficar um pouco encharcado sobre objetos, temos um pequeno artigo publicado no DesarrolloWeb sobre a programação orientada para os objetos. Seria altamente recomendável que o lesse, pois são explicados vários conceitos em que não entraremos em tantos detalhes. Se já conhece o POO, continue a ler sem pausa, mas se quiser ir mais fundo, lembre-se que também temos um Manual de Orientação completa do Objeto. Se gosta de ver vídeos, recomendamos também a classe de objetos ministrados no Curso de Programação de Vídeo.

O que é um objeto

Embora não vamos entrar em detalhes com os concetos, como são muito bem explicados em referências que já indicámos, os objetos são uma ferramenta de linguagem de programação em que duas coisas fundamentais se juntam: dados e funcionalidades. Todos os programas de computador lidam com essas duas coisas de alguma forma. Com o que vimos até agora tivemos os dados em variáveis e a funcionalidade em funções, certo? porque no mundo dos objetos, tanto os dados como a funcionalidade estão na mesma estrutura, o objeto.

A questão é que agora precisa de aprender novos nomes para se referir aos dados e funcionalidade agrupados num objeto:

Propriedades: Em objetos as propriedades referem-se aos dados

Métodos: Em objetos, os métodos referem-se à funcionalidade

Imagine que tem um objeto de botão (um botão de navegador, algo que pode pressionar para realizar uma ação). O botão tem um texto escrito, porque esse texto seria um dado e, portanto, nós chamaríamos de propriedade. Outra propriedade de um botão seria se é ou não ativado. Por outro lado, um botão poderia ter uma funcionalidade associada, que estaria num método, como processar a ação de um clique. Imagina algo mais genérico como um telefone. O telefone pode ter propriedades como fazer, modelo, sistema operativo e métodos como ligar, desligar, ligar um número, etc.

Em linguagens de programação puras orientadas a objetos, como Java, tem sempre de programar com base em objetos. Para programar teria de criar "classes", que são uma espécie de "moldes" a partir dos quais os objetos são criados. O programa resolveria qualquer necessidade criando objetos baseados nesses moldes (classes), existindo vários (dezenas, centenas ou milhares) de

objetos de diferentes classes. Os objetos teriam de colaborar entre si para resolver qualquer tipo de ação, tal como em sistemas como um avião existem vários objetos (o motor, hélices, controlos...) que colaboram entre si para resolver a necessidade de transportar passageiros ou mercadorias em viagens aéreas.

No entanto, como temos vindo a dizer, no Javascript não se trata tanto de uma programação orientada para objetos, mas de objetos. Muitas vezes serão objetos já criados pelo próprio navegador (a janela do navegador, um documento HTML a ser exibido, uma imagem ou um formulário dentro desse documento HTML, etc.), e outras vezes serão objetos criados por si ou por outros. desenvolvedores que o ajudam a fazer coisas específicas. Portanto, o que queremos saber para começar é a sintaxe que você precisa para usar objetos, basicamente aceder às suas propriedades e executar os seus métodos.

Nota: Para saber quais são os objetos do navegador, que temos disponíveis no Javascript para resolver as necessidades das páginas web, tem de ler o manual sobre o trabalho com o Javascript para utilizar e manipular os recursos do navegador.

Como aceder a propriedades e métodos de objetos

No Javascript podemos aceder às propriedades e métodos de objetos de forma semelhante à que fazemos noutras linguagens de programação, com o ponto do operador (".").

As propriedades são acedidas colocando o nome do objeto seguido de um período e o nome da propriedade a ser acedida. Desta forma:

myObject.myProperty propriedade

Para chamar os métodos usamos uma sintaxe semelhante, mas colocando os parâmetros que passamos para os métodos em parênteses. Da seguinte forma:

myObject.myMethod (parâmetro1, parâmetro2) método

Se o método não receber parâmetros, colocamos os parênteses também, mas sem nada dentro.

myObject.myMethod () método

Como criar objetos

Como dissemos, a maioria dos objetos com os quais vai trabalhar no Javascript para criar interação, efeitos e comportamentos diferentes nas páginas web, eles dão-lhe prontos. O navegador em si oferece-os a si, pelo que simplesmente tem de os utilizar. É material de estudo do Manual Javascript e objetos do navegador. Tendo esclarecido este ponto, é de notar que o Javascript é um pouco particular na criação de objetos, basicamente porque tradicionalmente não existe um conceito de "classe".

Para ser mais exato, no Javascript, ES5, as aulas são criadas através de funções e com o novo operador cria-se objetos a partir dessas funções, mas não há classes como as que conhecemos noutras línguas mais tradicionais.

Nota: As aulas já existem no ES6 e o Javascript é capaz de gerar classes e deles produzir objetos, como outras línguas. Obtém mais informações no Manual ES6.

A outra alternativa para criar objetos no Javascript é através de objetos literais, que não são mais do que a definição do objeto através de código incluído em chaves encaracoladas, indicando as suas propriedades ou métodos como está.

```
{
  name: 'Miguel Angel Alvarez',
  SitioWeb: 'DesarrolloWeb.com'
}
```

No código anterior temos apenas propriedades definidas, mas temos outros artigos onde você pode ver como definir métodos também. Para

saber mais sobre objetos literais recomendamos a leitura do artigo sobre objetos javascript literais, onde explicaremos com mais cuidado esta sintaxe habitual para criar objetos nesta linguagem.

No Javascript tradicional dissemos que as aulas não existem, mas podemos criar casos de objetos a partir de funções, como veremos no próximo ponto.

Criar e instantaneaizar objetos a partir de funções

Para quem não sabe, instantaneamente um objeto é a ação de criar um exemplo de classe, para que possa trabalhar com ele mais tarde. Classe é a definição das características e funcionalidades de um objeto. As aulas não funcionam diretamente, são apenas definições. Para trabalhar com uma classe, temos de ter um objeto instantâneo dessa classe. Lembramo-nos que no Javascript não há aulas, mas podemos usar funções.

Esta função simples poderia ser usada como um modelo para construir objetos da classe Pessoa:

```
function Person (name) {
  this.name = name;
}
```

Vais reparar que estamos a usar a palavra "isto" lá dentro. Esta palavra é uma referência ao objeto a ser criado com esta função. No javascript para criar um objeto a partir de uma função a nova declaração é usada, desta forma.

```
var miguel = new Persona ('Miguel Angel Alvarez');
```

Numa variável a que chamamos "miguel" atribuo uma nova (nova) instância da classe Pessoa. Os parênteses são preenchidos com os dados que a classe precisa para inicializar o objeto, se não houver parâmetros para entrar, os parênteses são colocados vazios. Na verdade, o que se faz quando se cria um objeto é chamar a função que o constrói e a função em si cuidará de inicializar as propriedades do objeto (que é o que a referência "esta" utiliza).

Modificar propriedade

Quando a propriedade já estiver presente, o Object.defineProperty () tentará modificá-lo de acordo com os valores no descritor de propriedade e ajustar o objeto atual. Se o descritor de propriedade no objeto tiver definido o valor configurável para falso, a propriedade não está definida e não pode ser configurada Modifique-as (apenas modifique o valor da chave escritural para falso [e vice-versa]); não é possível alternar entre propriedades de propriedades (propriedades de acesso e propriedades de dados) quando a propriedade não é ajustável.

O erro TypeError será chamado quando tentar alterar propriedades não ajustáveis (exceto por um valor writable como mostrado acima).

A chave é writable

Quando se define o valor da chave escrita para falso, a propriedade não é credível, ou seja, os valores não podem ser retribuídos:

```
var o = {};
Object.defineProperty (o, 'a', {
  value: 1,
  enumerable: true
});
Object.defineProperty (o, 'b', {
 value: 2,
 enumerable: false
});
Object.defineProperty (o, 'c', {
  value: 3
}); // property not enumerated
o.d = 4; // statistic property because we created it via
        // Assign value to a property directly
```

```
for (var i in o) {
  console.log (i);
}
// 'a' and 'd' (unordered)

Object.keys (o); // ['a', 'd']

o.propertyIsEnumerable ('a'); // true
o.propertyIsEnumerable ('b'); // false
o.propertyIsEnumerable ('c'); // false
```

A chave é configurável

A chave configurável controla se a propriedade pode ser eliminada do objeto (através do parâmetro de eliminação) e se o resto das teclas podem ser alteradas (exceto que o valor da chave é writable a falso):

```
var o = {};
Object.defineProperty (o, 'a', {
 get: function () {return 1; },
 configurable: false
});

Object.defineProperty (o, 'a', {
 configurable: true
}); // throws a TypeError
Object.defineProperty (o, 'a', {
 enumerable: true
}); // throws a TypeError
Object.defineProperty (o, 'a', {
 set: function () {}
}); // throws a TypeError (set was undefined previously)
Object.defineProperty (o, 'a', {
 get: function () {return 1; }
}); // throws a TypeError
Object.defineProperty (o, 'a', {
 value: 12
}); // throws a TypeError

console.log (o.a); // 1
delete o.a; // Nothing will happen
console.log (o.a); // 1
```

Se o valor chave configurável da propriedade o.a for verdadeiro, nenhum dos erros anteriores será lançado, e a propriedade será eventualmente eliminada.

Adicione os atributos e valores predefinidos das teclas

É importante ter em conta os valores predefinidos das teclas. Existe uma grande diferença entre atribuir um valor a uma propriedade do objeto diretamente e utilizar a função Object.defineProperty () como mostrado no exemplo seguinte:

```
var o = {};


oa = 1;
// Rewards expression
Object.defineProperty (o, 'a', {
  value: 1,
  writable: true,
  configurable: true,
  enumerable: true
});
"But in return
Object.defineProperty (o, 'a', {value: 1});
// Rewards
Object.defineProperty (o, 'a', {
  value: 1,
  writable: false,
  configurable: false,
  enumerable: false
});
```

Funções personalizadas de getter e setter

O exemplo a seguir demonstra como criar um objeto que valore automaticamente os seus valores atribuídos. Quando se define um valor para a temperatura da propriedade, é adicionado ao arquivo:

```
function Archiver () {
  var temperature = null;
  var archive = [];

  Object.defineProperty (this, 'temperature', {
    get: function () {
      console.log ('get!');
      return temperature;
    },
    set: function (value) {
      temperature = value;
      archive.push ({val: temperature});
```

```
};


function TestDefineSetAndGet () {
    Object.defineProperty (this, 'myproperty', pattern);
}


var instance = new TestDefineSetAndGet ();
instance.myproperty = 'test';
console.log (instance.myproperty);
// I always return this string, whatever you have assigned

console.log (instance.myname); // this is my name string
```

Suporta todos os navegadores

- **Objeto.defineProperties ()**

A função Object.defineProperties () define novas propriedades num objeto diretamente, modifica propriedades que já existem num objeto e, em seguida, reabre esse objeto.

Estrutura geral

```
Object.defineProperties (obj, props)
```

Obj

O i objeto em che si quer definir propriedades ou modificá-las.

Adereços

Um objeto com propriedades enumeradas que especificam os descritores de propriedade que serão adicionados ou modificados pelo objeto e que serão associados com nomes de atributos. Existem dois tipos de descritores: descritores de dados e descritores acessórios, consulte a página de função Object.defineProperty () para obter mais detalhes.

Os descritores de dados e os descritores de acesso são objetos, partilhando as seguintes teclas necessárias:

* configurável: Se o valor desta chave for verdadeiro, pode modificar o descritor de propriedade (não o valor do imóvel) e a propriedade pode ser eliminada do objeto. O valor predefinido é falso.

enumerado: Se o valor desta chave for verdadeiro, esta propriedade aparecerá ao enumerar as propriedades de enumeração do objeto.

O valor predefinido é falso.

Podem existir as seguintes chaves opcionais nos metadados:

* valor: Especifica o valor associado à propriedade, e qualquer valor pode ser permitido em JavaScript (tais como números, funções, objetos, etc.). O valor predefinido é indefinido.

writable: Se o valor desta chave for verdadeiro, o valor associado a esta propriedade pode ser modificado usando os coeficientes de atribuição. O valor predefinido é falso.

As seguintes chaves opcionais podem ser encontradas nos descritores de acesso:

* obtenha: A função que será definida como uma função de getter para esta propriedade, ou o valor indefinido se não houver nenhuma função getter associada a esta propriedade; o valor devolvido por esta função será usado como um valor para o imóvel. O valor predefinido é indefinido.

Conjunto: A função que será definida como função de setter para esta propriedade, ou o valor indefinido se não houver nenhuma função de setter associada a esta propriedade; esta função aceita um argumento que é o novo valor que o utilizador tentou atribuir a esta propriedade. O valor predefinido é indefinido.

Valor devolvido

O objeto que passou para a função.

a propriedade descrição

A função Object.defineProperties () define todas as propriedades do objeto obj, que dependem do objeto (e também devem ser enumeradas).

Exemplos, 19

Um exemplo de definição de propriedades propriedade1 e propriedade2 em objeto obj, uma dessas propriedades é escrita (atribuindo o verdadeiro valor da chave writable) e outra não-writable (atribuindo o valor falso à chave dedispoable):

```
var obj = {};
Object.defineProperties (obj, {
 'property1': {
  value: true,
  writable: true
 },
 'property2': {
  value: 'Hello',
  writable: false
 }
 // ...
});
```

Suporte reduzido para navegadores

Assumindo que o ambiente de funcionamento não é modificado, uma vez que todos os nomes e propriedades indicam os seus valores iniciais, a função seguinte é equivalente à função object.defineProperties ():

```
function defineProperties (obj, properties) {
  function convertToDescriptor (desc) {
    function hasProperty (obj, prop) {
      return Object.prototype.hasOwnProperty.call (obj, prop);
    }

    function isCallable (v) {
      // NB: modify as necessary if other values than functions are
callable.
      return typeof v === 'function';
    }

    if (typeof desc! == 'object' || desc === null)
      throw new TypeError ('bad desc');

    var d = {};

    if (hasProperty (desc, 'enumerable'))
      d.enumerable = !! desc.enumerable;
    if (hasProperty (desc, 'configurable'))
      d.configurable = !! desc.configurable;
    if (hasProperty (desc, 'value'))
      d.value = desc.value;                                    se
(temProperty (desc, 'writable'))
d.writable = !! desc.writable;
se (temProperty (desc, 'get')) {
```

```
    var g = desc.get;

    if (! isCallable (g) && typeof g! == 'undefined')
      throw new TypeError ('bad get');
    d.get = g;
  }
  if (hasProperty (desc, 'set')) {
    var s = desc.set;
    if (! isCallable (s) && typeof s! == 'undefined')
      throw new TypeError ('bad set');
    d.set = s;
  }

  if (('get' in d || 'set' in d) && ('value' in d || 'writable' in d))
    throw new TypeError ('identity-confused descriptor');

  return d;
}

if (typeof obj! == 'object' || obj === null)
  throw new TypeError ('bad obj');

properties = Object (properties);
```

```
var keys = Object.keys (properties);
 var descs = [];

 for (var i = 0; i <keys.length; i ++)
  descs.push ([keys [i], convertToDescriptor (properties [keys [i]])]);

 for (var i = 0; i <descs.length; i ++)
  Object.defineProperty (obj, descs [i] [0], descs [i] [1]);

 return obj;
}
```

Suporta todos os navegadores

- **Object.entradas ()**

A função Object.entries () devolve uma matriz que contém as propriedades estatísticas de um objeto sob a forma de pares [chave, valor] na mesma ordem que o loop para ... Para dentro.

A diferença entre esta função e o loop para ... é que o para ... loop irá sobrepor as propriedades estatisticamente significativas na cadeia do protótipo também.

Estrutura geral

Objeto.entradas (.obj)

Obj

Ii objeto cujas propriedades dependentes serão devolvidas como pares [chave, valor].

Valor devolvido

Uma matriz que contenha os valores das propriedades dependentes do objeto obj sob a forma de pares [chave, valor].

Exemplos, 19

Exemplo de um objeto simples:

```
const obj = {foo: 'bar', baz: 42};

console.log (Object.entries (obj)); // ['foo', 'bar'], ['baz', 42]]
```

Exemplo de um objeto semelhante à aritmética:

```
const obj = {0: 'a', 1: 'b', 2: 'c'};

console.log (Object.entries (obj)); (I), [b], [ii], [c]]]
```

Um exemplo de um objeto semelhante a matrizes, mas a ordem das chaves é aleatória:

```
const anObj = {100: 'a', 2: 'b', 7: 'c'};

console.log (Object.entries (anObj)); ['(Ii),' b '], [7', 'c'], [100 ',' a ']]
```

A propriedade getFoo é uma propriedade não estatística no objeto myObj:

```
const myObj = Object.create ({}, {getFoo: {value () {return
this.foo;}}});

myObj.foo = 'bar';

console.log (Object.entries (myObj)); // [['foo', 'bar']]
```

Note que se passar por um corretor que não represente um objeto, será convertido para um objeto, como na seguinte cadeia de texto:

```
console.log (Object.entries ('foo')); (l), [o]], [ii], 'o']]
```

Mas uma matriz vazia será devolvida para qualquer tipo de dado inicial, porque os valores iniciais não têm quaisquer atributos associados a eles:

```
console.log (Object.entries (100)); // []

console.log (Object.entries (true)); // []
```

Passando propriedades e valores de objeto através do loop para ... de:

```
const obj = {a: 5, b: 7, c: 9};

for (const [key, value] of Object.entries (obj)) {

  console.log (`$ {key} $ {value}`); // "a 5", "b 7", "c 9"

}
```

Ou utilizando funções matricias:

```
Object.entries (obj) .forEach (([key, value]) => {
console.log (`$ {key} $ {value}`); // "a 5", "b 7", "c 9"
});
```

A função Object.entries () pode ser usada para converter um objeto do tipo Objeto para Mapa:

```
const obj = {foo: 'bar', baz: 42};
const map = new Map (Object.entries (obj));
console.log (map); // Map {foo: "bar", baz: 42}
```

Suporte reduzido para navegadores

Para adicionar suporte para a função Object.entries () em ambientes que não suporta, pode utilizar o seguinte código:

```
if (! Object.entries)
  Object.entries = function (obj) {
    var ownProps = Object.keys (obj),
      i = ownProps.length,
      resArray = new Array (i); // preallocate the Array
    while (i--)
      resArray [i] = [ownProps [i], obj [ownProps [i]]];

    return resArray;
  };
```

Se pretender suportar navegadores IE que sejam lançados menos de 9, tem de fornecer uma função alternativa para a função Object.keys (que encontrará na sua página).

- Objeto.freeze ()

A função Object.freeze () bloqueia um objeto, impede a adição de novas propriedades, impede a supressão de propriedades no

mesmo, e impede a modificação do valor, escalabilidade, escalabilidade ou escrita das suas propriedades; "Congelado".

Estrutura geral

```
Object.freeze (obj)
```

Obj

O objeto que vai congelar.

Valor devolvido

Objeto congelado.

a propriedade descrição

Quaisquer propriedades de objetos congelados não podem ser adicionadas ou eliminadas; qualquer tentativa de o fazer falhará silenciosamente ou provocará a exceção TypeError (este erro será normalmente no estilo rigoroso).

Não é possível alterar os valores de atributos que têm funções de dados, getter e setter, note que os valores dos objetos atribuídos a atributos ainda podem ser modificados a menos que também estejam congelados.

As matrizes podem ser congeladas como objetos, uma vez que não podemos modificar os valores dos seus elementos, ou adicionar ou apagar elementos dos mesmos.

Note que no ECMAScript 5 a utilização desta função num meio não-objecto resultaria num lançamento typeError, mas a partir do

ECMAScript 2015 (ou seja, ES6), os meios que não representam objetos serão tratados como objetos congelados, ou seja, o seu valor será devolvido como é:

```
> Object.freeze (1)

TypeError: 1 is not an object // ES5



> Object.freeze (1)

1 // ES2015
```

Exemplos, 19

Congelar objetos

No exemplo seguinte, criaremos um objeto chamado obj, adicionaremos novas propriedades, modificaremos propriedades e eliminaremos o outro:

```
var obj = {
  prop: function () {},
  foo: 'bar'
};

obj.foo = 'baz';
obj.lumpy = 'woof';
delete obj.prop;
```

Note que a função Object.freeze () irá retornar o objeto congelado, mas também congelará o objeto original, pelo que não é necessário salvar o objeto devolvido da função para congelar o objeto original:

```
var o = Object.freeze (obj);


o === obj; // true

Object.isFrozen (obj); // === true
```

Note que o processo de modificação dos valores de atributos ou de adição de novas propriedades falhará silenciosamente, mas se o estilo rigoroso for desencadeado, o erro TypeError será chamado:

```
obj.foo = 'quux'; // Nothing will happen

// The new property will not be added

obj.quaxxor = 'the friendly duck';


function fail () {

  'use strict';

  obj.foo = 'sparky'; // TypeError

  delete obj.quaxxor; // TypeError

  obj.sparky = 'arf'; // TypeError

}


fail ();
```

Tentar fazer modificações utilizando a função Object.defineProperty () fará com que o erro do TypeError seja lançado:

```
  Object.defineProperty (obj, 'ohai', {value: 17}); // TypeError

Object.defineProperty (obj, 'foo', {value: 'eit'}); // TypeError
```

É impossível modificar o valor da propriedade do protótipo, e ambas as seguintes expressões lançarão o erro TypeError:

```
Object.setPrototypeOf (obj, {x: 20})

obj .__ proto__ = {x: 20}
```

Congelar matrizes

Criaremos uma matriz chamada A, depois congelá-la e tentar modificar os seus valores, e essas tentativas falharão silenciosamente; se estivermos no padrão rigoroso, o erro do TypeError será chamado:

```
let a = [0];

Object.freeze (a); // The matrix can no longer be modified


a [0] = 1; // Will fail silently

a.push (2); // Will fail silently


// The TypeError error will be called

function fail () {

  "use strict"

  a [0] = 1;

  a.push (2);

}


fail ();
```

Congelamento profundo

É verdade que o objeto congelado não será ajustável, mas não é necessariamente "fixo"; no seguinte exemplo, o congelamento será superficial:

```
obj1 = {
  internal: {}
};

Object.freeze (obj1);
obj1.internal.a = 'aValue';

obj1.internal.a // 'aValue'
```

Para deixar o objeto adormecido, cada uma das propriedades desse objeto deve ser congelada. Isto chama-se "congelação profunda":

```
function deepFreeze (obj) {

  // Get the names of properties in the object
  var propNames = Object.getOwnPropertyNames (obj);

  // Freeze those properties before freezing the object
  propNames.forEach (function (name) {
    var prop = obj [name];
```

```
    // Freeze the property if it is an object
    if (typeof prop == 'object' && prop! == null)
      deepFreeze (prop);
  });

  // Freeze the object itself
  return Object.freeze (obj);
}

obj2 = {
  internal: {}
};

deepFreeze (obj2);
obj2.internal.a = 'anotherValue';
obj2.internal.a; // undefined
```

Estes atributos gerais devolvem um valor simples, não representam as funções ou atributos de um objeto.

- Infinito
- Rio Nan

- Propriedade indefinida
- Nulo

1.1 Infinito

A propriedade geral infinita é um valor numérico que representa o infinito.

Atributos de propriedade infinita

- Writeable - não
- Estatisticamente - não
- Ajustável - Não

Estrutura geral

Infinito

a propriedade descrição

A propriedade Infinity é uma propriedade do objeto global, que é, é uma variável no domínio público.

O valor da propriedade Infinity é Number.POSITIVE_INFINITY, e o valor da propriedade Infinity é maior do que qualquer outro número. Este valor segue o comportamento do valor infinito na matemática. Por exemplo, o resultado da multiplicação de qualquer número no Infinito é o Infinito, e a saída de qualquer número no Infinity é 0.

Exemplos, 19

Os seguintes exemplos mostram o resultado de operações matemáticas sobre o valor do Infinito:

```
console.log (Infinity); // Infinity

console.log (Infinity + 1); // Infinity

console.log (Math.pow (10, 1000)); // Infinity

console.log (Math.log (0)); // -Infinity

console.log (1 / Infinity); // 0
```

1.2 NaN

A propriedade pública NaN é um valor que não representa um número (uma abreviatura para o Não-A-Number).

Propriedades da propriedade NaN

- Writeable – não
- Estatisticamente – não
- Ajustável - Não

Estrutura geral

Rio Nan

a propriedade descrição

A propriedade naN é uma propriedade do objeto global, que é, é uma variável no domínio público.

O valor inicial da propriedade NaN é Número.NaN. A propriedade naN não é configurável e não é concebível em navegadores modernos, mas tente evitar escrevê-la.

O valor da NaN raramente é usado em programas, mas é o valor que será devolvido quando as funções do objeto Math (como Math.sqrt (-1)) falham ou quando um valor numérico (como parseint ("blabla") falha).

Teste se o valor é NaN

O valor dos parâmetros NaN (via==,! == == e !==== ==) não é igual a qualquer outro valor, incluindo os outros valores de NNA; então use o Número.isNaN () ou isNaN) Para ver se o valor envolvido é NaN, ou tentar comparar o valor com si mesmo, o valor NaN é o único valor que não é igual a si mesmo.

```
NaN === NaN; // false

Number.NaN === NaN; // false

isNaN (NaN); // true

isNaN (Number.NaN); // true


function valueIsNaN (v) {return v! == v; }

valueIsNaN (1); // false

valueIsNaN (NaN); // true

valueIsNaN (Number.NaN); // true
```

indefinido em JavaScript

A propriedade pública indefinida representa o valor inicial indefinido no JavaScript.

Os atributos da propriedade são indefinidos

- **Writeable – não**
- **Estatisticamente – não**
- **Ajustável - Não**

Estrutura geral

propriedade indefinida

a propriedade descrição

A propriedade indefinida é uma propriedade do objeto global, isto é, é uma variável no domínio público, e o valor primário da propriedade indefinida é o tipo de dado inicial indefinido.

A propriedade indefinida não é configurável e não é legível em navegadores modernos (começando com o ECMAScript 5 padrão),

mas tente evitar a sobreescrita.

As variáveis que não foram atribuídas a um valor terão um valor indefinido e as expressões serão indefinidas se um valor não for atribuído à variável de valor. Uma função devolverá o valor indefinido se não especificarmos um valor para devolver essa função através da expressão de retorno.

É verdade que podemos usar a palavra indefinido como identificador em qualquer campo que não seja o domínio público (porque indefinido não é uma palavra reservada), mas não é apropriado fazê-lo, o que vai dificultar a manutenção e depurar erros do programa.

```
// Do not do this at all

// "foo string"

(function () {var undefined = 'foo';

console.log (undefined, typeof undefined);}) ();


// "foo string"

(function (undefined)

{

console.log (undefined, typeof undefined);

}

)

 ('foo');
```

Exemplos, 19

Coeficiente de correspondência com indefinido

È pode usar o coeficiente de correspondência (===) com indefinido para ver se uma variável tem um valor. No seguinte código, o valor da variável x não é conhecido, e o resultado da expressão é verdadeiro:

```
var x;

if (x === undefined) {

  // The software expressions will be implemented here

}

else {

  // The software expressions will not be implemented here

}
```

Note que usamos o coeficiente de correspondência (=== em vez de usar o equalizador normal (==), porque x == indefinido voltará verdadeiro se x for nulo, enquanto o coeficiente correspondente não é nulo. Consulte a página de Transações de Comparação para obter mais detalhes.

O tipo de parâmetro é o valor indefinido

Em alternativa, podemos usar o tipo de parâmetro para ver se o valor da variável é indefinido:

```
var x;

if (typeof x === 'undefined') {

  // The software expressions will be implemented here

}
```

Uma das razões para a utilização do tipo de parâmetro é que não lançará um erro quando a variável não for declarada:

```
// The variable is not known yet
if (typeof x === 'undefined') {   // without problems
  // The software expressions will be implemented here
}

if (x === undefined) {   // Directs ReferenceError

}
```

O coeficiente é nulo e o valor é indefinido

Uma terceira forma de verificar se o valor de uma variável indefinida é utilizar o parâmetro nulo:

```
var x;

if (x === void 0) {

  // The software expressions will be implemented here

}


// This variable has not been known before

if (y === void 0) {

  // The error will be called a - Uncaught ReferenceError: y is not
defined

}
```

Nulo em JavaScript

O nulo representa a ausência intencional de valor do objeto, que é um tipo de valor inicial em JavaScript.

Estrutura geral

Nulo

a propriedade descrição

O nulo é utilizado por nulo, note-se que o nulo não é um identificador de um imóvel no objeto público (ou seja, não é como a propriedade indefinida), mas o nulo indica que não há definição da linha de base, e indica que a variável não se refere a qualquer objeto.

```
// The variable does not exist, as it has not been known or created
before

console.log (foo); // ReferenceError: foo is not defined


// The variable is present but has no type and no value

var foo = null;

console.log (foo); // null
```

A diferença entre nulo e indefinido

Ao verificar se há nulo ou indefinido, considere a diferença entre o coeficiente de igualdade (==) e o coeficiente de correspondência (===). O primeiro fator converterá a espécie antes da comparação:

```
typeof null // "object" (not "null" for historical reasons)

typeof undefined // "undefined"

null === undefined // false

null == undefined // true

null === null // true

null == null // true

! null // true

isNaN (1 + null) // false

isNaN (1 + undefined) // true
```

Processamento de palavras

Estes objetos representam cordas de texto e as formas como são processados e modificados.

- Corda
- Regexp

O objeto de corda em JavaScript

O objeto string é uma função de sintaxe para cordas de texto, ou um conjunto de caracteres.

Estrutura geral

Os valores iniciais das cordas de texto podem assumir a seguinte forma:

```
'string text'

"string text"

"中文 español deutsch English हिन्दी العربية português বাংলা русский
日 日本語ਪੰਜਾਬੀ ਪੰਜਾਬੀ 국어 மழழழ் Hebrew"
```

As cadeias de texto podem ser criadas utilizando diretamente o objeto público de corda:

Corda (coisa)

Coisa

Qualquer valor que queiramos converter para uma cadeia de texto.

Propriedade de modelos

A partir do modelo ECMAScript 2015 podem ser criados literais do modelo:

```
`hello world`

`hello!

 world! `

`hello $ {who}`

escape `<a> $ {who} </a>`
```

Contrabando de caracteres

Além dos caracteres regulares, os caracteres especiais podem ser incluídos contrabandeando-os da seguinte forma:

Code / output

\ 0 ** The character is NULL

\ ' ** Single quotation mark

\ " ** double quotation mark

\\ ** backslash

\ N ** New lineface (new line)

\ R ** Return carriage to the beginning of the line (carriage return)

\ V ** Vertical tab space

\ t ** Scheduling distance (tab)

\ B ** Backspace

\ F ** Move to a new page (form feed)

\ UXXXX ** Unicode codepoint format

\ U {X} ... \ u {XXXXXX} ** Unicode codepoint format

\ XXX ** Latin-1 (Latin-1)

Em contraste com algumas outras linguagens de programação, o JavaScript não diferencia entre as cordas de texto incluídas em aspas únicas e cordas de texto rodeadas por marcas duplas de aspas; portanto, os caracteres de contrabando anteriores funcionarão independentemente do tipo de aspas à sua volta.

Longas cordas de texto

Por vezes, o código conterá algumas cordas de texto muito longas, e em vez da linha longa ou do invólucro de linha no editor, podemos dividir estas cadeias de texto em várias linhas de código sem afetar o conteúdo das cordas de texto em si. Há duas maneiras de fazer isto.

Podemos usar o operador + para adicionar cordas de texto uns aos outros da seguinte forma:

```
let longString = “This is a very long string which needs" +
        "to wrap across multiple lines because" +
        "otherwise my code is unreadable.";
```

Também pode utilizar a contrala detrás no final de cada linha para indicar que a cadeia de texto será completada na linha seguinte; mas certifique-se de que não há espaço ou qualquer outro personagem após o retrocesso (exceto a nova linha), caso contrário não funcionará. Aqui está um exemplo deste método:

```
let longString = "This is a very long string which needs \
to wrap across multiple lines because \
otherwise my code is unreadable. ";
```

Ambos os exemplos anteriores produzirão a mesma cadeia de texto.

a propriedade descrição

As cadeias de texto são úteis para a retenção de dados que representam palavras texuais. Uma das operações mais utilizadas nas cordas de texto é verificar o seu comprimento (através da propriedade de comprimento), recolher cordas de texto utilizando + e +, e verificar ou colocar cadeias de subtexto usando a função indexOf () função ,A função sub-atrogés em comprimento ().

Aceda aos caracteres

Há duas maneiras de aceder aos caracteres numa cadeia de texto. A primeira é usar charAt ():

```
return 'cat'.charAt (1); // "a"
```

Outro método adicionado à norma ECMAScript 5.1 é tratar a cadeia de texto como objetos semelhantes a matriz. Os caracteres estão associados a um índice numérico:

```
return 'cat' [1]; // "a"
```

Quando aceder a caracteres que usam suportes quadrados, tentar apagar ou atribuir um valor a esses atributos não funcionará; estes atributos não serão writable ou definido

Compare as cadeias de texto

Os desenvolvedores de cordas usam strcmp () para comparar cordas de texto; no JavaScript, pode utilizar operadores maiores e menores do que:

```
var a = 'a';

var b = 'b';

if (a <b) {  // true

  console.log (a + 'is less than' + b);

} else if (a> b) {

  console.log (a + 'is greater than' + b);

} else {

  console.log (a + 'and' + b + 'are equal.');

}
```

Um resultado semelhante ao acima pode ser obtido utilizando a função localeCompare () que está disponível nas cópias do objeto string.

Diferenças entre cordas de texto primárias e objetos de corda

Note que a linguagem JavaScript distingue entre objetos de corda e cordas de texto iniciais (o mesmo se aplica aos objetos Boolean e Number).

As cadeias de texto definidas por marcas de citações duplas ou únicas e cordas de texto devolvidas da função String sem a utilização da nova palavra reservada são cordas primitivas, e o JavaScript converte automaticamente estes valores iniciais em objetos de corda, permitindo que ao utilizar a função (ou tentar aceder a uma propriedade) numa corda inicial, o JavaScript encapsula a cadeia de texto inicial dentro de um objeto e chama a respetiva função (ou propriedade obrigatória) :

```
var s_prim = 'foo';

var s_obj = new String (s_prim);


console.log (typeof s_prim); // "string"

console.log (typeof s_obj); // "object"
```

Note que os objetos de corda e as cordas de texto iniciais produzem resultados diferentes quando usados com a função de eval () como a cadeia de texto passada será tratada como eval, e os objetos de corda comportar-se-ão como outros objetos, ou seja, o objeto será devolvido. O exemplo que se segue ilustra-o:

```
var s1 = '2 + 2'; // Create a text string

var s2 = new String ('2 + 2'); // Create an object

console.log (eval (s1)); // Figure 4

console.log (eval (s2)); // Text string 2 + 2
```

Por estas razões, os códigos não funcionam quando se encontram objetos de corda em vez de cordas de texto primárias, mas os desenvolvedores não deitam fora essas diferenças. Podemos converter o objeto String para a cadeia de texto inicial que é equivalente usando o valorOf () função:

```
console.log (eval (s2.valueOf ())); // Figure 4
```

Funções da função String

- String.protótipo

Esta propriedade permite adicionar atributos que estão disponíveis para todos os objetos cujo tipo é String.

Funções da função String

*** String.fromCharCode ()**

Crie uma cadeia de texto utilizando uma cadeia de valores Unicode.

String.fromCodePoint () *

Crie uma cadeia de texto utilizando uma cadeia de valores de caracteres (pontos de código).

*** Corda.raw ()**

Crie uma cadeia de texto a partir de um modelo cru.

- ## String.fromCharCode ()

A função estática String.fromCharCode () devolve uma cadeia de texto construída utilizando uma cadeia de valores de caracteres Unicode.

Estrutura geral

```
String.fromCharCode (num1 [, ... [, numN]])
```

número1, ..., numN

Uma série de números que representam valores do Unicode. O domínio permitido é entre 0 e 65535 (ou seja, 0xFFFF), e o maior número de 0xFFFF será truncado.

Valor devolvido

Uma cadeia contendo caracteres que estão ligados aos valores do Unicode passados para a função.

a propriedade descrição

Esta função devolve uma cadeia de texto primitiva e não devolve um objeto de corda.

Uma vez que a função deCharCode é um método estático do objeto string, deve sempre usá-lo como String.fromCharCode (), uma vez que não pode usá-lo como parte do seu objeto de corda.

Exemplos, 19

O exemplo a seguir mostra a utilização da funçãoCharCode ()

```
String.fromCharCode (65, 66, 67); // "ABC"

String.fromCharCode (0x2014) // "-"

String.fromCharCode (0x12014) // "-" The character 1 will be ignored
```

Manusear valoresmaiores do que o máximo

É verdade que os valores do Unicode são geralmente representados por um carácter de 16 bits (como anteriormente esperado durante o processo de redação JavaScript). Poderíamos usar a função deCharCode () para recriar um único personagem a partir da maioria dos valores (valores UCS-2, que são um subconjunto de UTF 16 - onde os caracteres mais comuns).

Para todos os valores válidos do Unicode (até 21 bits), a utilização da função deCharCode () por si só não é adequada, porque os caracteres com valores grandes utilizam dois dígitos para representar um único carácter; portanto, String.fromCodePoint () Parte da norma ECMAScript 2015 (ES6) para devolver tais pares, permitindo a representação de caracteres com valores superiores ao máximo.

Suporta todos os navegadores

- String.fromCodePoint ()

A função estática String.fromCodePoint () devolve uma cadeia de texto construída utilizando uma cadeia de pontos de código.

Estrutura geral

```
String.fromCodePoint (num1 [, ... [, numN]])
```

número1, ..., numN

Uma série de números que representam os valores dos caracteres (pontos de código).

Valor devolvido

Uma cadeia de texto que contém caracteres associados aos valores dos caracteres passados para a função.

Exceções, 19

A exceção RangeError será chamada se o valor de caracteres unicode inválido for passado (por exemplo: "RangeError: NaN não é um ponto de código válido").

a propriedade descrição

Esta função devolve uma cadeia de texto primitiva e não devolve um objeto de corda.

Uma vez que a função do CodePoint é um método estático do objeto string, deve sempre usá-lo como String.fromCodePoint (), uma vez que não pode usá-lo como parte do seu objeto de corda.

Exemplos, 19

Os seguintes exemplos mostram a utilização da função CodePoint ()

```
String.fromCodePoint (42); "" * "

String.fromCodePoint (65, 90); // "AZ"

String.fromCodePoint (0x404); // "\ u0404"

String.fromCodePoint (0x2F804); // "\ uD87E \ uDC04"

String.fromCodePoint (194564); // "\ uD87E \ uDC04"

String.fromCodePoint (0x1D306, 0x61, 0x1D307) // "\ uD834 \
uDF06a \ uD834 \ uDF07"


String.fromCodePoint ('_'); // RangeError

String.fromCodePoint (Infinity); // RangeError

String.fromCodePoint (-1); // RangeError

String.fromCodePoint (3.14); // RangeError

String.fromCodePoint (3e-2); // RangeError

String.fromCodePoint (NaN); // RangeError
```

A função String.fromCharCode () não consegue encontrar caracteres cujos valores sejam superiores ao máximo. O exemplo a seguir devolverá um personagem de 4 bytes, mais caracteres de 2 bytes (ou seja, devolve um personagem que representa uma cadeia de texto de 2 bits em vez de 1):

consola.log (String.fromCodePoint (0x2F804));

Suporte reduzido para navegadores

A função String.fromCodePoint foi adicionada ao ECMAScript 2015 (ES6) e ainda não está suportada em todos os navegadores, pelo que o seguinte código pode ser usado para identificá-lo:

(This code for professionals / skip it and return to it after the book is finished)

```
/ *! http://mths.be/fromcodpoint v0.1.0 by @mathias * /
if (! String.fromCodePoint) {
 (function () {
  var defineProperty = (function () {
   // IE 8 only supports `Object.defineProperty` on DOM elements
   try {
    var object = {};
    var $ defineProperty = Object.defineProperty;
    var result = $ defineProperty (object, object, object) && $
defineProperty;
   } catch (error) {}
   return result;
  } ());
  var stringFromCharCode = String.fromCharCode;
```

```
var floor = Matemática.floor;
var do CodePoint = função () {
var MAX_SIZE = 0x4000;
var codeUnnits = [];
var highSurrogate;
var lowSurrogate;
índice var = -1;
comprimento do var = argumentos.comprimento;
se (! comprimento) {
devolução '';
}
resultado var = '';
enquanto (++ índice <comprimento) {
código varPoint = Número (argumentos [índice]);
se (
! isFinite (codePoint) || // 'NaN',' + Infinito', ou '-Infinity'
codePoint <0 || // não um código unicode válido
codePoint> 0x10FFFF || // não um código unicode válido
piso (codePoint)! = codePoint // não um inteiro
) {
lançar RangeError('Ponto de código inválido:' + codePoint);
}
```

```
      if (codePoint <= 0xFFFF) {  // BMP code point
        codeUnits.push (codePoint);
      } else {  // Astral code point; split in surrogate halves
        // http://mathiasbynens.be/notes/javascript-
encoding#surrogate-formulae
        codePoint - = 0x10000;
        highSurrogate = (codePoint >> 10) + 0xD800;
        lowSurrogate = (codePoint% 0x400) + 0xDC00;
        codeUnits.push (highSurrogate, lowSurrogate);
      }
      if (index + 1 == length || codeUnits.length> MAX_SIZE) {
        result + = stringFromCharCode.apply (null, codeUnits);
        codeUnits.length = 0;
      }
    }
    return result;
  };
  if (defineProperty) {
    defineProperty (String, 'fromCodePoint', {
      'value': fromCodePoint,
      'configurable': true,
      'writable': true
```

```
  });
    } else {
      String.fromCodePoint = fromCodePoint;
    }
  } ());
}
```

Apoiar navegadores

Chrome // Firefox // Internet Explorer // Ópera // Safari

41 // 29 // não suportado //28 // 10

Corda.raw ()

A função String.raw () é uma função de tag para literais de modelo, semelhante ao r anterior em Python ou ao anterior @ em C # (mas existem diferenças entre eles). Usado para obter a cadeia de texto em bruto do modelo (ou é, texto inexplicável).

Estrutura geral

```
String.raw (callSite, ... substitutions)
```

String.raw'templateString'

Callite

Objeto para modelo de chamada como: {cru: ['foo', 'bar', 'baz']}}

... propriedade substituições

Os valores a substituir.

modeloDese de propriedade

Uma cadeia de texto que representa o modelo e os valores a substituir ($ {...}).

Valor devolvido

Uma cadeia de texto em bruto para a cadeia de texto que representa o modelo.

Exceções, 19

A exceção TypeError será chamada se o primeiro corretor não for um objeto devidamente definido.

a propriedade descrição

Na maioria dos casos, a função String.raw () é usada com cordas de texto que representam os modelos. O primeiro formato acima mencionado raramente é usado, porque o motor JavaScript ligará automaticamente para este formato com os argumentos necessários.

Exemplos, 19

Os seguintes exemplos mostram que a função bruta () é utilizada:

```
String.raw`Hi \ n $ {2 + 3}! `;
// 'Hi \ n5!', The character after 'Hi'
// is not a newline character,
// '\' and 'n' are two characters.

String.raw`Hi \ u000A! `;
// 'Hi \ u000A!', Same here, this time we will get the
// \, u, 0, 0, 0, A, 6 characters.
// All kinds of escape characters will be ineffective
// and backslashes will be present in the output string.
// You can confirm this by checking the .length property
```

```
// of the string.


let name = 'Bob';

String.raw`Hi \ n $ {name}! `;

// 'Hi \ nBob!', Substitutions are processed.


// Normally you would not call String.raw () as a function,

// but to simulate `t $ {0} e $ {1} s $ {2} t` you can do:

String.raw ({raw: 'test'}, 0, 1, 2); // 't0e1s2t'

// Note that 'test', a string, is an array-like object

// The following is equivalent to

// `foo $ {2 + 3} bar $ {'Java' + 'Script'} baz`

String.raw ({

  raw: ['foo', 'bar', 'baz']

}, 2 + 3, 'Java' + 'Script'); // 'foo5barJavaScriptbaz'
```