PROGRAMAÇÃO FRONT END II

Maikon Lucian Lenz

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Objetos nativos do JavaScript

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Definir objetos nativos.
- Reconhecer os diferentes tipos de objetos nativos.
- Incluir objetos nativos em códigos JavaScript.

Introdução

A definição de objeto é fundamental na linguagem JavaScript, sendo que até mesmo as funções são tratadas como um objeto. Porém, cada implementação da linguagem dispõe de um conjunto de objetos nativos, sendo alguns deles naturais da própria linguagem.

Neste capítulo, você estudará a definição de objetos nativos, seus diferentes tipos e sua aplicação em códigos da linguagem JavaScript. Ao final, serão apresentados, detalhadamente, três objetos nativos de uso frequente (`Math`, `Date` e `RegExp`), com exemplos de seu uso. Esses objetos têm como função, respectivamente: efetuar operações matemáticas que não são possíveis por meio de operadores primitivos, apenas (`Math`); apresentar e manipular dados de datas e horas (`Date`); criar e executar expressões regulares para encontrar valores que obedeçam a um determinado padrão textual em `string`.

Objetos nativos

Em qualquer linguagem, os valores de um endereço de memória podem ter diferentes significados, dependendo do tipo de variável que ele representa ou é utilizada para compreendê-lo. Na linguagem JavaScript, existem os tipos primitivos — `number`, `string`, `symbol`, `boolean`, `undefined`

e `null` — e os tipos de objeto que definem coleções de valores, como o `Array` (SIMPSON, 2015a).

A diferença fundamental entre os diferentes tipos de variáveis é que, no primeiro grupo, são manipulados valores de forma direta (tipos primitivos), enquanto, no segundo, o que a variável armazena é o endereço de memória que aponta para outros valores (FLANAGAN, 2013).

Sabe-se, no entanto, que, apesar de apresentar tipos de bem-definidos, o JavaScript é uma linguagem fracamente tipada no sentido de que as variáveis não contêm um tipo rígido (FLANAGAN, 2013) e podem ter valores atribuídos de diferentes tipos de maneira extremamente dinâmica.

Os objetos podem ser compreendidos como conjuntos de chaves e valores de qualquer tipo, inclusive um objeto pode conter outro como chave ou valor. A cada par chave-valor é dado o nome de propriedade do objeto.

A forma mais simplificada de se definir um objeto pode ser vista no exemplo:

```
var obj = {}
```

A linha de código define uma variável cujo nome será associado a um endereço de memória de um objeto vazio, ou seja, sem qualquer propriedade.

Fique atento

Importante destacar que a palavra reservada `var` não é a única forma de se criar uma variável do tipo objeto, sendo possível utilizar `let`, `const` ou, mesmo, de forma direta como para qualquer outro tipo de variável.

As propriedades de um objeto são declaradas seguindo o padrão "`chave : valor`" e separadas umas das outras por vírgula, como na primeira linha do seguinte exemplo, em que o objeto de nome carro contém as propriedades: cor, do tipo `String` e valor azul, e marchas, de tipo `Number` e valor `5`. Outras formas de se criar propriedades podem incluir atribuições diretas (segunda linha) e literais (terceira linha).

```
let carro = { cor: 'Azul', marchas: 5 } // objeto com 2
propriedades
carro.velocidadeMaxima = 200              // adiciona nova
propriedade
carro['Marca'] = 'Desconhecida'           // adiciona nova
propriedade
```

As propriedades podem ser adicionadas e, também, excluídas dinamicamente por meio do código "`delete objeto.propriedade`".

Dentre os vários tipos de objetos, encontram-se os especiais, como vetores (`Array`) e funções (`function`), de forma que qualquer valor que não seja um tipo primitivo é considerado um objeto em JavaScript — daí a sua importância (FLANAGAN, 2013).

Há, no entanto, um conjunto de objeto predeterminados, denominados objetos nativos, que acompanham a linguagem para oferecer ferramentas e dados que facilitem o desenvolvimento de aplicações comuns. Esses são definidos pelo padrão ECMAScript, assim como o restante da linguagem. Os objetos especiais já citados anteriormente (`Array` e funções) são alguns exemplos desses objetos nativos (ECMA INTERNATIONAL, 2015).

Atualmente, a linguagem possui uma quantidade considerável de objetos nativos, de forma que eles já são agrupados e classificados pelo próprio padrão ECMAScript — conforme serão apresentados na sequência cada uma dessas categorias e os objetos que englobam.

Tipos de objetos nativos

Segundo o padrão ECMAScript, os objetos nativos podem ser classificados em (ECMA INTERNATIONAL, 2015):

- objeto global;
- objetos fundamentais;
- objetos de número e data;
- objetos de processamento de texto;
- objetos de coleção indexada;
- objetos de coleção chaveada;
- objetos de dados estruturados;
- objetos de controle de abstrações de objetos;
- objetos de reflexão.

Objeto global

É o principal objeto de qualquer execução, não dispõe de construtor, nem pode ser invocado como uma função, mas é criado logo antes de qualquer execução do código em si, sendo, portanto, um objeto único. Apresenta, entretanto, uma série de propriedades predefinidas que podem variar conforme a implementação da linguagem (ECMA INTERNATIONAL, 2015) .

Dentro do objeto global, são definidas as propriedades de valor: `Infinity` — de tipo numérico utilizada para representar valores infinitos; `NaN` (*Not a number* ou, em português, não é um número) — tipo numérico retornado em operações matemáticas que resultaram em algum erro; `undefined` — simboliza que a variável não teve seu valor definido ou a função não retorna um valor (SIMPSON, 2015a).

Por padrão, esse objeto apresenta, ainda, propriedades de funções:

- `eval()` — executa código representado por uma `string`;
- `uneval()` — representa um trecho de código por uma `string`;
- `isFinite()` — verifica se um número é finito;
- `isNaN` — verifica se um valor é do tipo `NaN`;
- `parseFloat()` — converte `String` para ponto flutuante;
- `parseInt()` — converte `String` para um número inteiro;
- `decodeURI()`, `decodeURIComponent()` `encodeURI()` e `encodeURIComponent()` — decodificam endereços de recursos.

O objeto global não tem um nome definido e é referenciado de diferentes formas, dependendo da implementação da linguagem (FLANAGAN, 2013).

Objetos fundamentais

Todos os demais objetos são derivados de algum objeto fundamental. Os mais utilizados são: `Object`, `Function`, `Boolean` e `Symbol`.

Repare que a própria função é, na verdade, um objeto para o JavaScript, e, com isso, diferentemente de outras linguagens, a função pode ser passada como parâmetro, atribuída a uma variável e declarar funções uma dentro da outra.

Sendo o `Boolean` um objeto, qualquer outro valor pode ser compreendido como verdadeiro (`true`) ou falso (`false`) em uma expressão lógica. Todos os valores diferentes de `undefined`, `null`, `± 0`, `false`, `NaN` e `Strings` vazias serão interpretados como verdadeiros (`true`).

Por fim, o objeto `Symbol` produz um valor único que pode ser utilizado como identificador de propriedades de um objeto.

Outros objetos fundamentais são objetos de erro: `Error`, `EvalError`, `InternalError`, `RangeError`, `ReferenceError`, `StopIteration`, `SyntaxError`, `TypeError` e `URIError`. Esses podem ser utilizados como base para outras classes de exceções a serem definidas pelo programador (ECMA INTERNATIONAL, 2015).

O primeiro, `Error`, é um objeto genérico do qual derivam os outros e que implementa as propriedades mensagem e nome para descrever o erro.

O objeto `EvalError` é instanciado quando se tenta utilizar a função global `eval()` com um parâmetro que seja possível de traduzir em um código executável para a linguagem JavaScript, normalmente por não obedecer à sintaxe da linguagem.

Um erro bastante comum é o `RangeError`, que ocorre sempre que uma operação qualquer extrapolar os limites definidos para um dado — por exemplo: ao referenciar tamanhos inválidos de um `Array`; ao tentar criar um objeto do tipo `Date` com horários ou datas inválidos; ao especificar quantidades inválidas de dígitos para um determinado valor numérico.

Ao se referenciar de forma inválida um endereço de memória qualquer, seja para objetos, propriedades ou qualquer outra variável, é gerado um erro do tipo `ReferenceError`. O exemplo mais recorrente é a tentativa utilizar identificadores cujos nomes não foram atribuídos a nenhuma variável ainda, sendo impossível para o interpretador traduzir aquele identificador para um endereço de memória.

Já o objeto `SyntaxError` é utilizado pelo interpretador quando este é incapaz de compreender o código redigido. Os casos podem envolver: função sem nome; bloco de código sem o finalizador `}`; uso indevido de palavras reservadas; entre outros.

O `TypeError` ocorre quando a utilização da variável não está de acordo com o tipo definido para ela e, apesar de o JavaScript ser fracamente tipado, não é possível ao interpretador encontrar uma conversão válida. Pode ser encontrado na tentativa de acessar propriedades inexistentes de um objeto, de se redefinir constantes.

Por fim, os endereços de recursos que não forem possíveis de ser codificados ou decodificados corretamente produzem um objeto do tipo `URIError`.

Alguns exemplos dos erros citados anteriormente podem ser vistos a seguir.

```
// RangeError
[].length = -1        // Comprimento deve ser sempre positivo
new Date('2019/30/30') // Data inválida.
(1.23).toFixed(-1)    // Número de casas decimais deve ser positivo

// ReferenceError
variavel;             // Referência a um nome não declarado

// SyntaxError
let var = 1             // var é uma palavra reservada
function(){ return 1   // função sem nome e bloco de código
sem fim }.

//TypeError
const obj = {}        // Declarada uma referência constante.
obj = {}              // Não pode receber o endereço de outro
objeto.
null.length           // Objeto null não tem a propriedade.

// URIError
encodeURI('\uD800')    // Não se pode codificar este caractere
```

Objetos de números e datas

Nesta categoria, constam três objetos: `Number`, `Math` e `Date`.

O objeto `Number` engloba todos os tipos numéricos e contém propriedades que delimitam valores de intervalor, máximo, mínimo, infinito positivo e negativo.

O objeto `Math` abrange funções e propriedades recorrentes em operações matemáticas, como logaritmo, exponencial, funções trigonométricas, além de constantes, como o número pi e número de Euler.

Já o objeto `Date` representa um instante de tempo em milissegundos, mas com propriedades e métodos que convertem, apresentam e facilitam operações entre datas e tempos.

Objetos de processamento de texto

Constituem-se dos objetos `String` e `RegExp`. O primeiro elenca propriedades e métodos para manipular cadeias de caracteres, que podem ser acessados diretamente a partir de um tipo primitivo `string`, já que o JavaScript converte a variável em um objeto. Já o segundo objeto, abreviação de *regular expression* (expressões regulares), é utilizado para representar e buscar padrões textuais em uma cadeia de caracteres, a fim de identificar elementos ou partes específicas de um texto.

Objetos de coleção

Dados podem ser agrupados e ordenados por um índice na forma de vetores, conhecidos pelo termo `Array` — neste caso, são ditos objetos de coleção indexada (SIMPSON, 2015b).

Outra forma de criar coleções de dados usa chaves para identificar cada um deles. São os objetos: `Map` — varre uma coleção retornando um `Array` com chave e valor; `Set` — similar ao `Map`, mas com valores únicos resultantes da iteração; `WeakMap` — o mesmo que um `Map` fracamente referenciado e não enumerável; e `WeakSet` — referência apenas objetos, enquanto o `Set` é capaz de utilizar valores de qualquer tipo.

Objetos de dados estruturados

Inclui os objetos `ArrayBuffer`, `DataView` e `JSON`, utilizados na troca de informações entre sistemas.

O `ArrayBuffer` tem tamanho fixo e estrutura de dados binária, enquanto o `DataView` funciona como um intérprete desse `Array`, tanto no procedimento de leitura quanto de escrita.

`JSON` é um objeto de comunicação no formato textual, similar à descrição literal de um objeto, mas apresentado em uma `String` (SIMPSON, 2015a).

Objetos de controle de abstrações

`Promise` e `Generators` são objetos que permitem o controle de fluxo assíncrono e habilitam operações com iterações infinitas ou processamento de dados paralelamente às demais instruções.

Objetos de reflexão

Objetos utilizados para criar armadilhas e capturar determinados eventos em outros objetos. O `Proxy` é construído para monitorar objetos em específico, enquanto o `Reflect` não tem construtor e serve para retornar informações a respeito do próprio código, como consulta às propriedades e aos protótipos de um determinado objeto.

Conforme o padrão da linguagem é atualizado, novos objetos nativos surgem para atender às necessidades comuns de propriedades e métodos dos programadores. Porém, mais importante do que conhecer cada um desses objetos é saber como fazer uso deles. Para tanto, alguns desses objetos serão detalhados e exemplificados a seguir.

Aplicação de objetos nativos em códigos JavaScript

Novos objetos estão sendo inseridos a cada ano pelo padrão ECMAScript, mas alguns deles são amplamente utilizados e já fazem parte da linguagem há mais tempo. A seguir, são detalhados os usos dos objetos `Math`, `Date` e `RegExp`.

`Math`

É um objeto estático, logo, não possui construtor. O objeto `Math` junto da notação "`.`" será usado para acessar as constantes definidas em propriedades desse objeto, bem como acessar os métodos de funções matemáticas comuns, passando os valores como parâmetros de uma função.

Há duas formas básicas de se utilizar o objeto: `Math.constante`, em que a palavra "constante" deverá ser substituída por um identificador válido existente no objeto (`PI`, `E`, `LN10`, ...); e `Math.função()`, substituindo a palavra "função" pelo nome de uma função válida (`abs()`, `ceil()`, etc.) (FLANAGAN, 2013).

Um problema simples de matemática, por exemplo, é o cálculo de uma área circular a partir do valor do seu raio. A expressão matemática para tanto corresponde a $\pi \cdot r^2$, sendo π (pi) uma constante cujo valor pode ser acessado pela propriedade `PI` do objeto `Math`. A operação de multiplicação pode ser solucionada pelo operador *, mas a operação de exponenciação do raio (r^2) terá de recorrer ao método `pow(x, y)` de `Math`, como no exemplo:

```
let r = 2
let area = Math.PI * Math.pow(r, 2)          // area = π.r²
```

Estão inclusas as funções trigonométricas: seno, cosseno, tangente, tangente hiperbólica e seus inversos, utilizadas conforme o exemplo:

```
let cos _ x = Math.cos(x)
let sen _ x = Math.sin(x)
let tan _ x = Math.tan(x)
let arc _ cos _ x = Math.acos(x)
let arc _ sen _ x = Math.asin(x)
let arc _ tan _ x = Math.atan(x)
```

Todos os valores de ângulo devem ser em radianos e, assim, também serão os resultados dessas funções.

Exponenciações, logaritmos e raízes também estão presentes:

```
let exp_x = Math.exp(x)        // e^x
let log_x = Math.log(x)        // ln x
let log10_x = Math.log10(x)    // log10 x
let log2_x = Math.log2(x)      // log2 x
let pow_x = Math.pow(x, y)     // x^y
let cbrt_x = Math.cbrt(x)      // ∛x
let sqrt_x = Math.sqrt(x)      // √x
```

Outras funções incluem truncamento e arredondamento de valores e obtenção de máximos e mínimos de um conjunto de números.

```
let trun _ x = Math.trunc(x)    // somente parte inteira de x
let abs _ x = Math.abs(x)       // |x|
let ceil _ x = Math.ceil(x)     // número inteiro superior mais
próximo
let fl _ x = Math.floor(x)      // número inteiro inferior mais
próximo
let rnd _ x = Math.round(x)     // arredonda para o inteiro mais
próximo
let max _ x = Math.max(x)       // maior número dentro da coleção
let min _ y = Math.min(y)       // menor número dentro da coleção
```

Por fim, o objeto `Math` apresenta, ainda, uma função para gerar números pseudoaleatórios entre 0 e 1: `Math.random()`.

Operações que resultariam em erro matemático contêm valores definidos para o JavaScript, utilizando os valores `Infinity` e `NaN` para os indicar. É o caso das divisões por zero que resultam em um infinito positivo, exceto quando o numerador também for igual a zero, cujo resultado será `NaN` (FLANAGAN, 2013).

Date

Auxilia na manipulação de datas e intervalos de tempo. Diferentemente do `Math`, o objeto `Date` apresenta método construtor. Logo, além das funções estáticas presentes no objeto, pode-se instanciar um novo objeto a partir da palavra reservada `new`. O construtor aceita argumentos de tipos e quantidades diferentes na criação de um novo objeto.

Se utilizado sem argumentos, o objeto é instanciado com os valores de data e hora atuais do sistema: um único parâmetro numérico, os milissegundos após a data e hora inicial de 01/01/1970. Já um parâmetro do tipo `string` pode especificar ano, mês, dia, hora, minutos, segundos e milissegundos, desde que utilizando um padrão de representação aceitável pelo objeto e seu método de conversão `Date.parse()`. Por fim, pode-se invocar o construtor, utilizando mais de um valor numérico e informando, pelo menos, os valores de ano e mês, sendo os demais opcionais: `new Date(ano, mês, dia, hora, minuto, segundo, milissegundo)`.

Após instanciado, o objeto não pode ter seus valores alterados por meio de propriedades, sendo toda a manipulação feita por métodos, que podem operar com base na hora local ou universal, quando o termo UTC estiver presente no método.

```
Date.getFullYear()      // ano no formato de 4 dígitos do objeto
Date.getMonth()          // mês do objeto
Date.getDate()          // dia do mês do objeto
Date.getDay()            // dia da semana do objeto
Date.getHours()          // hora do objeto
Date.getMinutes()        // minutos do objeto
Date.getSeconds()        // segundos do objeto
Date.getMilisseconds()   // milissegundos do objeto
Date.getTime()          // milissegundos desde 01/01/1970
Date.getTimezoneOffset()  // minutos entre hora local e UTC
```

Para todos os métodos de retorno de valores do exemplo anterior, há um equivalente para atribuição de valores, bastando substituir apenas os termos `get` por `set`, com exceção do método `getTimezoneOffset()`, que não pode ser alterado. Em ambos os casos, `get` e `set` possuem versões do mesmo método para manipulação em horário universal (UTC), bastando inserir o termo UTC após a palavra `get/set`, exceto para os métodos `getTime()` e `setTime()`.

Há, ainda, métodos de conversão de valores para `String` ou `JSON`.

```
Date.toDateString()         // formato textual da data local
Date.toUTCString()        // formato textual da data em UTC
Date.toISOString()         // padrão ISO-8601 da data/hora UTC
Date.toLocalDateString()   // formatação local de data em texto
Date.toLocalString()         // formatação local de data/hora
em texto
Date.toLocalTimeString()    // formatação local de horário em
texto
Date.toString()             // data/hora local em texto
Date.toTimeString()         // hora local em texto
Date.valueOf()              // objeto Date para milissegundos
```

Os métodos estáticos `Date.now()` e `Date.parse()` retornam, em milissegundos, o tempo decorrido desde a meia-noite do dia 01/01/1970, para o primeiro caso, e a conversão de uma `string` para uma data no segundo caso.

RegExp

Expressões regulares são utilizadas para localização de padrões textuais de forma a separar e identificar elementos específicos de um texto.

Um objeto `RegExp` pode ser criado a partir do construtor do objeto, passando como parâmetros duas `strings`: uma informando o padrão de expressão regular e outra de atributos globais. Pode-se usar como argumento outro objeto `RegExp` (SIMPSON, 2015a).

Ao objeto, estão associadas cinco propriedades:

- `global`, indicando que o índice de varredura da `RegExp` deverá ser cumulativo, permitindo executá-la várias vezes para encontrar cada situação em que o padrão se repete;

- `ignoreCase`, com valor `false`, para diferenciar entre maiúsculas e minúsculas;
- `lastIndex`, armazena a posição do último caractere em conformidade com a expressão;
- `multiline`, para varrer múltiplas linhas;
- `source`, armazena o texto da expressão.

Depois de criado o objeto com a expressão que será utilizada na busca por padrões, pode-se recorrer a dois métodos — `exec()` e `test()` —, inserindo como parâmetro a `string` que será pesquisada. O primeiro método retornará um `Array` contendo os resultados da comparação; enquanto o segundo apenas determinará a existência ou não dos padrões com `true` e `false`.

Em sua maioria, os caracteres de uma `string` utilizada na criação de um padrão `RegExp` representam diretamente um texto buscado, mas há, ainda, os metacaracteres com significados especiais:

```
 []       // delimita uma classe de caracteres
[^]       // negação de uma classe de caracteres
-         // delimita um intervalo de caracteres
\w        // somente caracteres alfabéticos
\W        // somente caracteres não alfabéticos
\s       // espaço em branco
\S       // exceto espaços em branco
\d       // dígitos de 0 a 9
\D       // exceto dígitos de 0 a 9
{n,m}    // ocorrência anterior no mínimo n e no máximo m vezes
{n,}     // ocorrência anterior no mínimo n vezes ou mais
{n}      // ocorrência anterior exatamente n vezes
?       // ocorrência anterior nenhuma ou uma única vez
+       // mais de uma ocorrência anterior
*       // ao menos uma ocorrência anterior
|       // ou a expressão da esquerda ou da direita
()      // agrupar metacaracteres especiais
(?:)    // agrupa itens sem recordar dos caracteres do grupo
^      // início de string
$      // final de string
(?=)   // caracteres a seguir devem corresponder a expressão
(?!)   // caracteres a seguir não devem corresponder com a
expressão
```

A combinação de caracteres e metacaracteres forma um padrão que, se atendido pela `string` buscada, retornará verdadeiro para o método `test()` e os próprios resultados do filtro para o método `exec()`. No exemplo a seguir, o padrão fornecido ao objeto `RegExp` permite a identificação de um horário no formato "hora:minuto" em um texto.

```
let padrao = "(10|11|12|[1-9]):[0-5][0-9]"
let atributos = "gmi"
let reg = new RegExp(padrao, atributos)
let texto = "Agora são 9:53"
let horario = RegExp.exec(texto)
```

O padrão buscado pode ser dividido em dois grupos, um antes e outro após o caractere "**:**". Os grupos estão entre colchetes para determinar que os caracteres pertencentes a cada um deles devem ser retornados individualmente no `Array`, e não apenas avaliado o padrão total. Dessa forma, é possível identificar o horário e, também, determinar os valores de hora e minuto pelo próprio `RegExp`.

A hora no texto pode ser de quatro maneiras diferentes: o número 10, 11, 12, ou apenas um caractere entre 1 e 9. Esse padrão será identificado desde que as horas sejam expressas apenas utilizando números significativos (sem zeros à esquerda) e estejam no padrão de 12 horas, e não de 24.

Já os caracteres após o "**:**" serão validados caso o primeiro esteja entre 0 e 5, e o segundo entre 0 e 9.

Utilizando o método `exec()` do `RegExp` criado, a variável horário receberá um `Array` como:

```
['9:53', '9', '53', index: 10, input: 'Agora são 9:53']
```

O primeiro membro do `Array` corresponde ao padrão inteiro detectado, incluindo o caractere "**:**". Os dois elementos seguintes correspondem a cada um dos grupos individuais determinados no `RegExp` pelos colchetes e aparecem na mesma ordem: primeiro a hora e, depois, os minutos. Os demais elementos que especificam o índice do último caractere correspondem a um padrão detectado e o texto analisado.

Referências

ECMA INTERNATIONAL. *ECMAScript® 2015 Language Specification*. 6. ed. Geneva: Ecma, 2015. 545 p. Disponível em: https://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf. Acesso em: 2 nov. 2019.

FLANAGAN, D. *JavaScript*: o guia definitivo. 6. ed. Porto Alegre: Bookman, 2013. 1080 p.

SIMPSON, K. *You don't know JS*: types & grammar. Sebastopol: O'Reilly, 2015a.

SIMPSON, K. *You don't know JS*: up & going. Sebastopol: O'Reilly, 2015b.

Leituras recomendadas

JAVASCRIPT. *MDN Web Docs*, Mountain View, 2019. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 2 nov. 2019.

SANDERS, B. *Smashing HTML5*: técnicas para a nova geração da web. Porto Alegre: Bookman, 212. 368 p.

SIMPSON, K. *JavaScript and HTML5 now*: a new paradigma for the open web. Sebastopol: O'Reilly, 2012. 12 p.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS