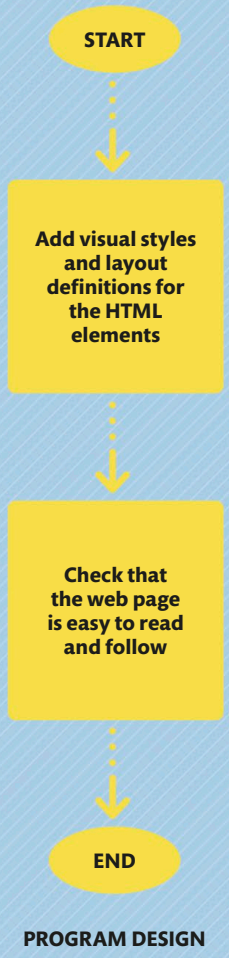
Styling the web page

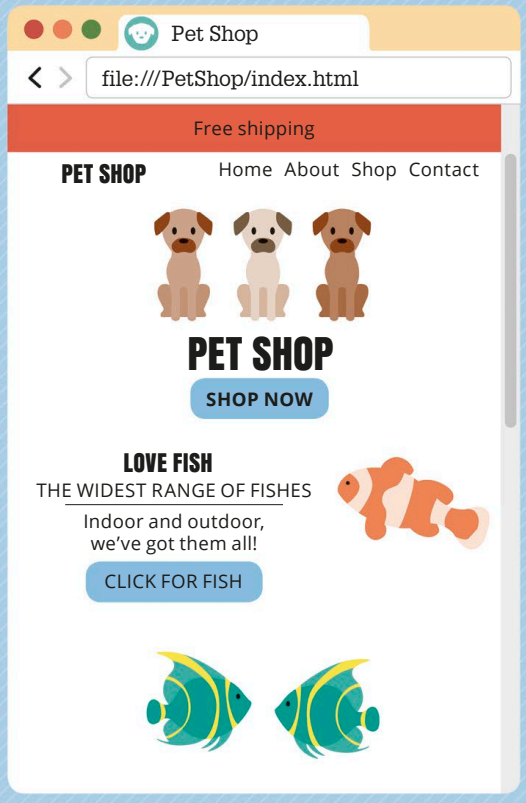
In this part of the project, styling will be applied to the framework created in HTML. CSS allows programmers to have much better control over the layout of their website. The look of the site can be kept consistent by using a single style sheet, which makes the maintenance of a website more efficient. This also saves time and makes it easy to update the design.

What the program does

Use the HTML elements of the web page created in Build a web page (see pp.216–233). You will use CSS to select and style these HTML elements individually in this second part of the project. Each element will be formatted according to its role and function, making the web page easier to navigate.



PROGRAM DESIGN



Styled website
The long vertical web page from the HTML part of this project will now appear styled, with clearly defined sections and formatted text and images. Adding CSS makes the web page more visual and individual.

Project requirements

To add styling to your web page, you will need the HTML file and images folder from the first part of the project. You can continue using Visual Studio as the development environment.

YOU WILL LEARN

- How to use CSS style sheets
- How to create buttons with rollovers
- How to add CSS animations and transitions

Time: 2–3 hours

Lines of code: 315

Difficulty level

WHERE THIS IS USED

CSS is used in all modern websites where visual appearance is important. Using CSS, it is possible to give every element of a website a distinct style. A well-presented web page will encourage interaction with the user and will be easier to navigate as well.

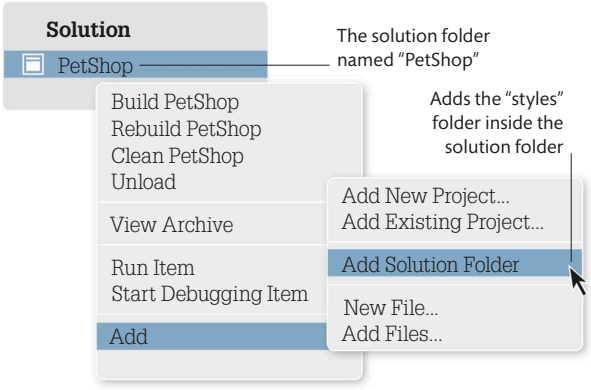
1 Setting up

Before you can start styling the website, it is necessary to create a special CSS file to contain the code and link it to the HTML file previously created. The following steps will create a dedicated "styles" folder and the CSS file inside it.

1.1 CREATE A NEW FOLDER

You will first need to create a new folder in Visual Studio to contain the CSS style sheets. In Windows, right-click on the project name "PetShop" in Solution Explorer. Next, select Add and choose New Folder. Name this folder "styles". The full path to this folder should be "C:\PetShop\styles".

On a Mac, open Finder and create a new folder called "styles" inside the website folder "PetShop". The full path should be "Users/[user account name]/PetShop/styles". Next, open Visual Studio, right-click on the project name "PetShop", select Add, and then choose Add Solution Folder.



1.2 ADD A CSS FILE

The next step is to create a new CSS file inside the styles folder. Make sure to name the CSS file "global.css". In Windows, the full path to the CSS file should be "C:\PetShop\styles\global.css". On a Mac, the full path to the file should be "Users/[user account name]/PetShop/styles/global.css". The website folder PetShop should now contain the images folder, the styles folder, and the HTML file.

LOCATING FOLDERS

In Windows, if you would like to see where a folder has been created from inside Visual Studio, go to the "Solution Explorer" window, right-click on the folder you want to locate, and select Open Folder in File Explorer. This will open an instance of File Explorer at the location of the folder.

On a Mac, to see the location of a folder from inside Visual Studio, go to the "Solution Explorer" window, command-click on the folder you would like to locate, and select Reveal in Finder. This will open an instance of Finder at the folder's location.

1.3 REFERENCE THE CSS FILE

HTML

It is necessary to link the newly created CSS file to the HTML document so that the styles can be applied to all of its elements. This reference to "global.css" file must be made using a <link> tag within the <head> tag of the "index.html" file. The fonts for this web page will be selected from the options available in Google Fonts. To do this, you will link the Google Fonts website to "index.html" and specify the fonts you want to use. The fonts Anton and Open Sans are used here, but you can pick any other font you like.

```
<head>
    <meta charset="utf-8" />
    <title>Pet Shop</title>
    <link rel="icon" type="image/png" href="images/favicon.png">
    <link href="styles/global.css" rel="stylesheet" />
    <link href="https://fonts.googleapis.com/css?
    family=Anton|Open+Sans" rel="stylesheet">
</head>
```

Title to be displayed in the browser tab

Link to the custom "global.css" file where all the styles are defined

Link to the CSS style sheets for the Google fonts

1.4 ADD COMMENTS

CSS

At the top of the "global.css" style sheet, add a comment with the names of the fonts and the list of colors to be used on the website. Add this information inside a comment block "/* */". Anything inside a comment block will be ignored by the browser. These comments are only included to help the programmer standardize the style of the website and provide an easy reference. Notice that the "font-family" definition contains the name of the primary font being employed and a secondary font type to use if the first font is not available. You can also choose a different color scheme for your website if you like.

```
/*
    font-family: "Anton", cursive;
    font-family: "Open Sans", sans-serif;

    Text color : #333;
    Dark blue : #345995;
    Light blue : #4392F1;
    Red : #D7263D;
    Yellow : #EAC435;
    Orange : #F46036;
*/
```

This font will be used for headings and other prominent text elements

This font will be used for normal paragraph text elements

Hex codes for the colors to be used

2 Styling the page elements

Now that the CSS file is ready to contain all of the style definitions, you can start by adding the styles that affect elements throughout the page. The next few steps will show you how to do this.

2.1 DEFINE THE HEADERS

Start by defining the "h1" and "h2" headers used throughout the website. The styles specified here will be applied to each instance of the two headers. Both headers will be styled the same but with a separate font-size definition.

```
body {
}

h1, h2 {
    margin: 0;
    padding: 0;
    font-family: "Anton", cursive;
    font-weight: normal;
}
h1 {
    font-size: 110px;
}
h2 {
    font-size: 30px;
}
```

The <body> tag appears by default

Applies the style definition to all "h1" and "h2" headers on the web page

The "font-family" property defines the preferred font to use and the fallback font type in case the preferred font is not available

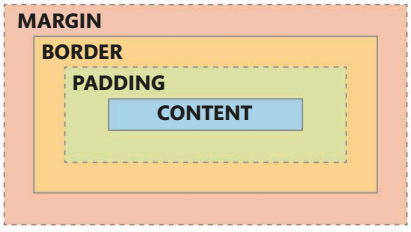
The headers will use the Anton cursive font

Only "h1" headers will have the font size 110px

Only "h2" headers will have the font size 30px

MARGIN AND PADDING

Margin is the space measured from the border, outside of the element. Padding is the space measured from the border, inside the element. There are different ways to describe the margin and padding styles.



STRUCTURE OF MARGIN AND PADDING IN A WEBSITE

MARGIN STYLES	
Code	**Output**
margin: 40px;	40px on top, bottom, left, and right
margin: 20px 40px;	20px on top and bottom; 40px on left and right
margin: 10px 20px 30px 40px;	10px on top, 20px on right, 30px on bottom, and 40px on left
margin: 0 auto;	0 on top and bottom and equal space on the left and right (same as center align)

Page elements
Every website is made up of various elements serving different purposes. CSS can add unique styles to every element.

2.2 ADD VERTICAL SPACERS

Each section of the web page can be separated with white space to make a website easy to follow. These white spaces are created through standardized vertical spacers throughout this website. In CSS, you can use compound style signatures to associate the required style definitions.

```
.spacer.v20 {
    height: 20px;
}
.spacer.v40 {
    height: 40px;
}
.spacer.v80 {
    height: 80px;
}
```

Compound style signature with the classes "spacer" and "v20"

The height will be 40px if the tag has both "spacer" and "v40" classes

The height will be 80px if the tag has both "spacer" and "v80" classes

2.3 ADDING ELEMENTS ON NEW LINES

Now create a style for "clear". This is required if the previous element does not have a fixed height or is set to float to one side. The "clear" property is used with values such as "left", "right", or "both". It prevents floating objects from appearing on the specified side of the element to which "clear" is applied.

```
.clear {
    clear: both;
}
```

Selects all of the elements with the class "clear"

This instruction will place the next HTML element on a new line

USE MULTIPLE CLASSES IN THE HTML TAG TO ACHIEVE MORE MEANINGFUL AND TARGETED STYLING

2.4 STYLE THE BODY TAG

Add the style definitions for the <body> tag from step 2.1. The style signature is "body". The style definition will set the values for the margin, padding, font, background color, and font color.

```
body {
    margin: 0;
    padding: 0;
    font-family: "Open Sans",
    sans-serif;
    font-size: 15px;
    background-color: white;
    color: #333;
}
```

- `padding: 0;` — No space between the browser window and the web page
- `font-size: 15px;` — The default font size for all of the text on the web page
- `background-color: white;` — The background color for the web page
- `color: #333;` — Hex code for dark gray, the font color for the body text

3 Styling the individual elements

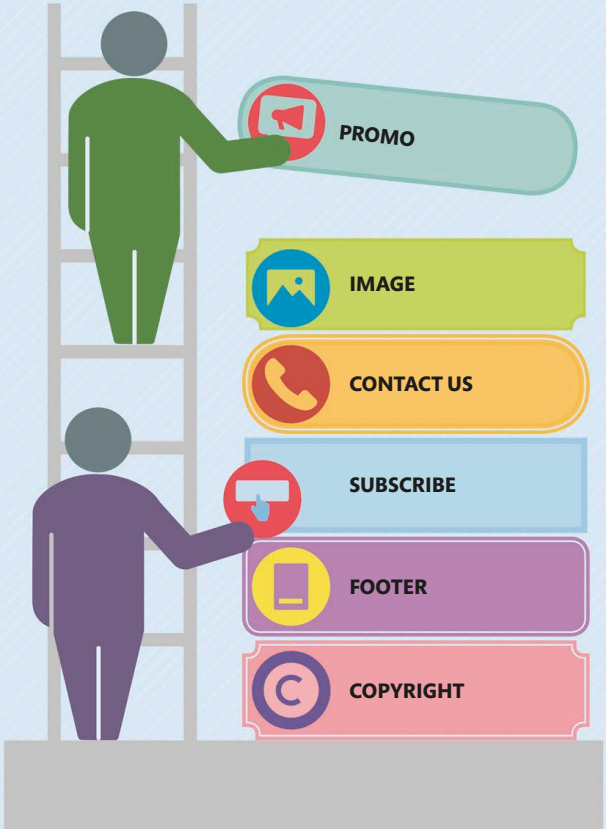
In this section, you can add styles to the individual HTML elements from the top of the web page down to the bottom. Each section can be seen as a horizontal layer with its own visual style and spacing requirements. Start by adding styles to the first element on the web page—the promo bar.

3.1 DEFINE STYLE SIGNATURES

The "Promotional Messages", "Subscribe", and "Footer" sections share some of the same style definitions. They all need to be center aligned, with white text and a minimum width of 1000px. Add this code below the code added in step 2.3 to give all three style signatures the same definition.

```
    clear: both;
}
#promo,
#subscribe,
#footer {
    text-align: center;
    color: #fff;
    min-width: 1000px;
}
```

- `#promo, #subscribe, #footer {` — The selectors apply the same style definition
- `color: #fff;` — Hex code for the color white
- `min-width: 1000px;` — The elements must be at least 1000px wide



3.2 BACKGROUND COLOR FOR PROMO

Next, you will add a background color to the Promotional Messages section to make it more visible on the web page. This style only applies to the promo section.

```
#promo {
    background-color: #F46036;
}
```

Hex code for orange in the color scheme

3.3 ADD PADDING TO THE PROMO DIV

Add padding around the text in the Promotional Messages to introduce some space from the border. The style will be applied to all of the promotional messages contained within the "promo" div.

```
#promo > div {
    padding: 15px;
}
```

Selects all tags within the <promo> tag

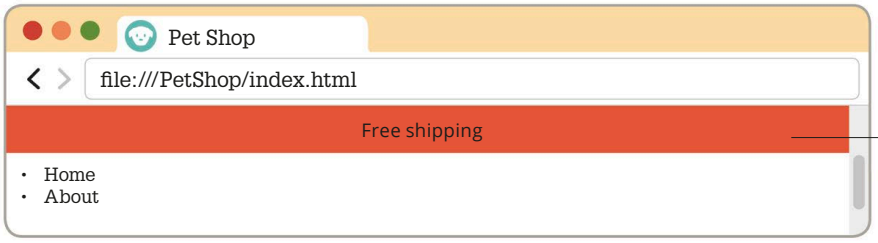
The space between the border and its text content

SAVE

3.4 VIEW THE PAGE

View the page by opening a browser and entering the URL into the address bar. In Windows, the URL to the file on your local computer will be "file:///C:/PetShop/index.html". On a Mac, the URL will be "file:///Users/[user account name]/PetShop/index.html".

If the page does not update in the browser when you refresh, the browser may have cached a previous version of the site. Empty the browser cache by going to the history settings and selecting "Clear browsing data". This will force the browser to get the latest files.



The promotional messages will now be displayed with the styles applied

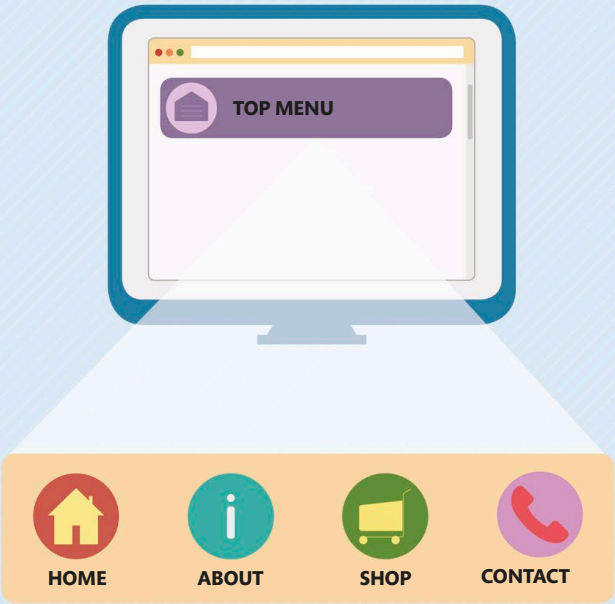
3.5 DEFINE THE WRAP CLASS

Add style definitions for the "wrap" class created in HTML. Most of the website's information is contained in this class. The "wrap" div has a fixed width of 1000px. The horizontal margins adjust automatically to keep the in the center of the screen if the screen width is more than 1000px wide.

```
    color: #333;
}
.wrap {
    margin: 0 auto;
    padding: 0;
    width: 1000px;
}
```

Add this after the code from step 2.4

Aligns the "wrap" element to the center of the page



3.6 DEFINE THE TOP MENU

Now add the style definitions for the Top Menu section. This panel will run across the full screen and will contain the menu items and logo. Set a fixed height for the menu and add padding in all directions for the list of menu items contained in the panel.

```
    padding: 15px;
}
#topMenu {
    height: 60px;
}
#topLinks {
    float: right;
    padding-top: 20px;
}
```

Add this after the code from step 3.3

Space between the "topLinks" border and the list it contains

3.7 DEFINE HORIZONTAL MENU LISTS

The menu lists in both Top Menu and the Footer are horizontal, so you can give them the same style definitions. The menu items should align left within their container lists so that they appear in a horizontal line with the first item on the left.

The two selectors share the same style definition

No bullet points will be added to the list items

```
#topLinks ul,
#footer ul {
        list-style-type: none;
        margin: 0;
        padding: 0;
        overflow: hidden;
}
#topLinks li {
        float: left;
}
```

Hides content that overflows the element's dimensions

The element floats to the left of its container

3.8 STYLE THE HYPERLINKS

The hyperlinks in the Top Menu will have one style for their normal state and a different style for when the mouse is hovering over them. The keyword ":hover" is a pseudo-class and instructs the browser to apply that style when the mouse is over the element. Add a "transition" instruction in both style definitions to make the mouse-over effect smoother. Here, three versions of the "transition" instruction have been included, each one intended for a different browser. Including multiple instructions for different browsers is not very common but is required occasionally.

```
#topLinks li a {
        color: #333;
        text-align: center;
        padding: 16px;
        text-decoration: none;
        -webkit-transition: all 250ms ease-out;
        -ms-transition: all 250ms ease-out;
        transition: all 250ms ease-out;
}
```

Center aligns the hyperlink contents

No underline beneath the hyperlink

Transition effect when a mouse moves off of the hyperlink

Transition definition required by older Microsoft browsers, such as Internet Explorer

Transition instruction for Google Chrome browser

```
#topLinks li a:hover {
    color: #4392F1;
    -webkit-transition: all 250ms ease-out;
    -ms-transition: all 250ms ease-out;
    transition: all 250ms ease-out;
    text-decoration: underline;
}
```

Hex code for the color blue

Transition effect when a mouse moves over the hyperlink

Underlines the hyperlink when a mouse hovers over it

TRANSITIONS

All major web browsers use different names for the "transition" property, so your CSS style definitions must include all three versions of the "transition" instruction to ensure that the transition effect renders correctly on the browsers. When a browser is implementing the CSS style definition, it will ignore the instructions intended for other browsers and apply the instructions it understands. A warning message about the invalid CSS properties may appear, but these can safely be ignored.

3.9 STYLE THE LOGO

The next step is to style the logo in the Top Menu. The logo is used three times on the page (in the Top Menu, the Banner, and the Footer), so you can encapsulate the logo font styles in its own class called "logo". The small logo in the Top Menu is a hyperlink back to the home page, so you will need to define both its normal and hover state.

```
#topMenu .logo {
    float: left;
    padding-top: 13px;
    font-size: 24px;
    color: #333;
    text-decoration: none;
}

    #topMenu .logo:hover {
        color: #4392F1;
    }

.logo {
    font-family: "Anton", cursive;
}
```

Places the logo on the left of the "topLinks" element

Sets the font size to 24px

No underline on the logo hyperlink

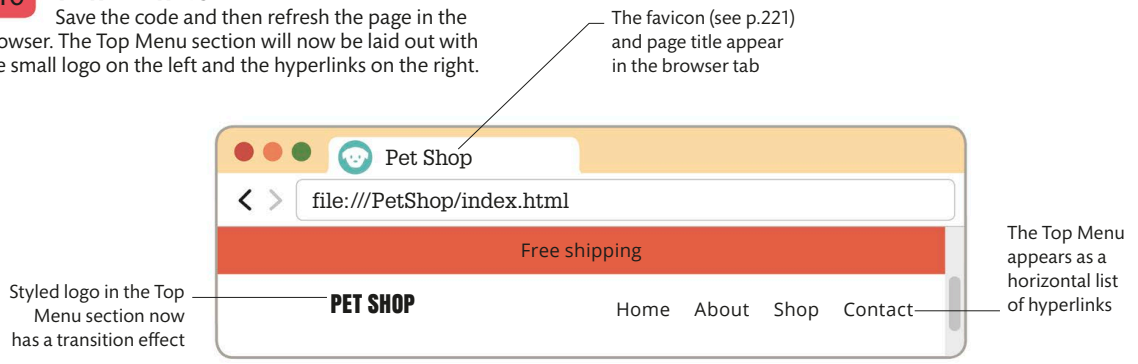
This will make the logo appear blue when the mouse hovers over it

The default font for the logo

SAVE

3.10 VIEW THE WEBSITE

Save the code and then refresh the page in the browser. The Top Menu section will now be laid out with the small logo on the left and the hyperlinks on the right.



3.11 STYLE THE BANNER

The next section to be styled is the Banner, which will contain the name of the website and an image. First, set the styles for the "banner" div by defining its width, height, and alignment. It should include a background image as well.

Link to the background image for the banner

```
#banner {
    background-image: url("../images/banner.jpg");
    background-repeat: no-repeat;
    background-position: center top;
    width: 100%;
    text-align: center;
    padding-top: 300px;
    color: #333;
}
```

The background image should not repeat vertically or horizontally

Center aligns the contents of the banner

Hex code for dark gray text

Space between the top border of the banner and the text inside

3.12 STYLE THE BANNER LOGO

Now you can add styles for the logo appearing inside the Banner section. In the HTML document, the logo appearing in the Banner section also has an <h1> tag. So this logo will receive style instructions from both "h1" and "logo" style definitions.

```
#banner .logo {
    margin-top: 20px;
}
```

Selects the "logo" elements inside of the banner

Space between the top border of the logo and the element above it

3.13 ADD STYLES TO THE HYPERLINK

The next step is to add styles for the "action" div and the "Shop Now" hyperlink. This link style definition will also contain the "transition" instructions to animate the change in styles between the normal and mouse-over states.

```
#banner #action {
        font-weight: bold;
        width: 200px;
        margin: 20px auto 0 auto;
}

        #banner #action a {
            -webkit-transition: all 250ms ease-out;
            -ms-transition: all 250ms ease-out;
            transition: all 250ms ease-out;
            padding: 20px;
            color: white;
            text-decoration: none;
            border-radius: 30px;
            background-color: #4392F1;
        }

            #banner #action a:hover {
                -webkit-transition: all 250ms ease-out;
                -ms-transition: all 250ms ease-out;
                transition: all 250ms ease-out;
                background-color: #F46036;
                padding: 20px 40px;
            }
```

Style definitions for the div that contains the hyperlink

Three versions of the transition instruction

Hex code for light blue

Transition effect when a mouse moves over the hyperlink

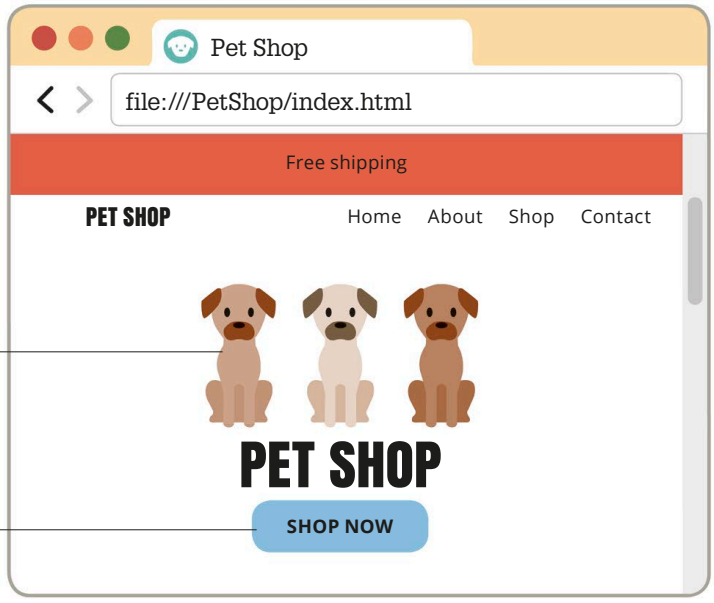
The horizontal padding will increase to 40px when a mouse hovers over the button

Hex code for the color orange

SAVE

3.14 **VIEW THE WEBSITE**

Refresh and view the page in the browser. You can now see the background image in the Banner and the animated rollover effect on the "Shop Now" link.



The banner background image will be displayed above the banner text

The banner call-to-action button with transition effect

4 Feature box styling

This section will add styles for the "feature box" control, which splits the page into a left column and a right column. The styles for the control are defined as classes and are applied to multiple elements on the web page. The class definitions also allow alternating the position of the images on the page.

4.1 **DEFINE THE LEFT COLUMN**

First, you will define the left column of the feature box. This will take up half of the space available. By default, every new div occupies a new line. However, since this element must float left, the next element (the right column) will appear on the same line.

Selects all of the elements that have both the "feature" and "leftColumn" classes

```
.feature .leftColumn {
    width: 50%;
    float: left;
    text-align: center;
}
```

The left column width is set at 50% of the container's width

Aligns the contents to the center of the left column

4.2 **DEFINE THE RIGHT COLUMN**

Add the code below to define the right column. This definition instructs the browser to include a margin on the left of the space available, where the left column will sit.

Selects all of the elements that have both the "feature" and "rightColumn" classes

```
.feature .rightColumn {
    margin-left: 50%;
    width: 50%;
    text-align: center;
}
```

Aligns the contents to the center of the right column

4.3 STYLE THE NONPICTURE ELEMENTS

Now set styles to define the nonpicture side of the feature box. In HTML, you used a div with class="text" (see p.225) to indicate the nonpicture elements. You can now define the left and the right text columns with the same definition.

```
.feature .leftColumn .text,
.feature .rightColumn .text {
        padding: 80px 20px 20px 20px;
        min-height: 260px;
}
```

Selectors for the "text" divs in the left and right columns

4.4 DEFINE THE NORMAL AND MOUSE-OVER STATE

Add this code to define the normal and mouse-over state for the hyperlinks that appear in the "text" divs. Similar to the Shop Now button that you styled earlier, this will also be styled as a button that changes color when the mouse hovers over it.

Selectors for the <a> tags in the "text" div in the left and right columns

"ease-out" defines the speed of the transition effect

```
.feature .leftColumn .text a,
.feature .rightColumn .text a {
        -webkit-transition: all 250ms ease-out;
        -ms-transition: all 250ms ease-out;
        transition: all 250ms ease-out;
        padding: 20px;
        background-color: #4392F1;
        color: white;
        text-decoration: none;
        border-radius: 30px;
}
    .feature .leftColumn .text a:hover,
    .feature .rightColumn .text a:hover {
            -webkit-transition: all 250ms ease-out;
            -ms-transition: all 250ms ease-out;
            transition: all 250ms ease-out;
            background-color: #F46036;
            text-decoration: none;
            padding: 20px 40px;
        }
```

Transition effect when a mouse moves off of the hyperlink

Sets rounded corners for the border of the hyperlink

Transition effect when a mouse moves over the hyperlink

The text is not underlined when the mouse hovers over the hyperlink

4.5 DEFINE THE HORIZONTAL RULE

You will also need to add a style definition for the horizontal rule appearing in the feature box. This rule will separate the heading of the column from the text below it.

```
.feature hr {
    background-color: #333;
    height: 1px;
    border: 0;
    width: 50px;
}
```

Sets the color for the horizontal rule to dark gray

Sets the width of the horizontal rule

4.6 DEFINE THE IMAGES

Now that you have styled the text columns, it is time to define how the images in the "featureImage" div will be styled.

Selects all <img> tags inside the "featureImage" div

```
.featureImage img {
    width: 500px;
}
```

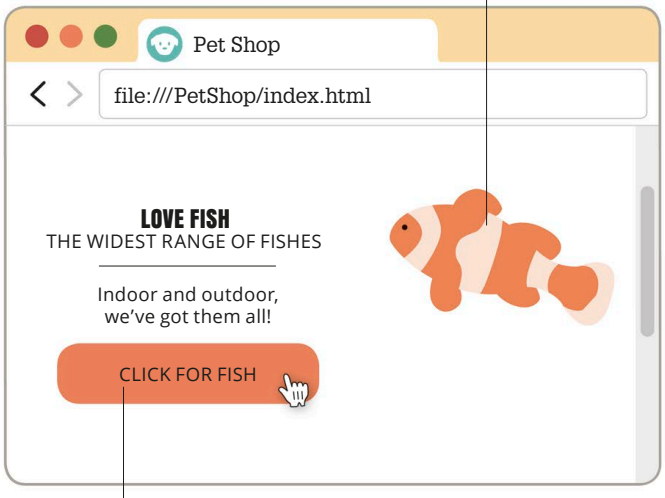
Resizes the image width to 500px. The height will adjust automatically

SAVE

4.7 VIEW THE STYLE DEFINITIONS

Refresh the browser to see the new style definitions being applied to the web page. All of the feature boxes will now have the correct styling, with images alternately appearing on the left and right.

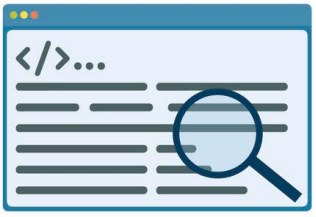
The image will appear in the right column with the correct width



The call-to-action button will now be styled, with transition effect

BROWSER TEST

New CSS features are constantly being added to browsers. However, there is no point in using these features unless you are sure that your website users will be able to take advantage of them. Old browsers will ignore modern CSS instructions and the styling of the HTML document will not conform to the expected layout. Fortunately, all modern browsers accept CSS3, though there may be small differences in the way they process some instructions. It is advisable always to test your web page in several different browsers to find the set of functionality that they all have in common.



4.8 ADD STYLE TO EMAIL HYPERLINK

In the "index.html" file, the "feature" class is used not only to advertise the three product categories, but also for the Contact Us section appearing farther down. In this step, you will use the "feature" layout definitions to style the Contact Us section, which includes a hyperlink that opens a new email in the user's email program.

```
.feature .leftColumn .text a.emailLink,
.feature .rightColumn .text a.emailLink {
        color: white;
        text-decoration: none;
        transition: none;
        padding: 10px;
        border: 0;
        background-color: #4392F1;
}
            .feature .leftColumn .text a.emailLink:hover,
            .feature .rightColumn .text a.emailLink:hover {
                -webkit-transition: all 250ms ease-out;
                -ms-transition: all 250ms ease-out;
                transition: all 250ms ease-out;
                background-color: #F46036;
            }
```

Selects the "emailLink" hyperlinks in both the left and right columns

The ":hover" pseudo-class selects the hyperlinks when a mouse hovers over them

Hex code for the color orange

4.9 DEFINE MIDDLE IMAGES

The next section that needs to be defined are the images that sit in the middle of the page. The "middleImage" div containers must align their contents to the center of the div to make them appear in the middle of the page. The <img> tags should also display the images with a consistent maximum width.

```
.middleImage {
    text-align: center;
}

    .middleImage img {
        max-width: 1000px;
    }
```

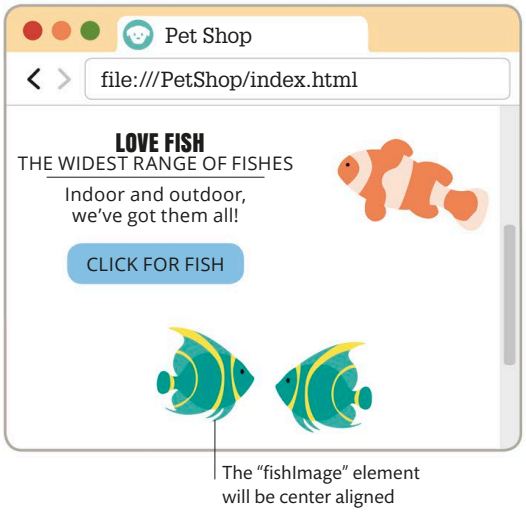
Aligns the "middleImage" contents to the center of the page

Selects all <img> tags inside divs with the "middleImage" class

SAVE

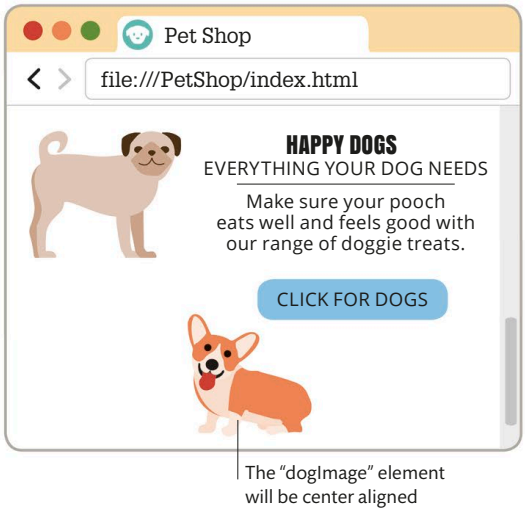
4.10 **VIEW THE IMAGES**

Save the code and then refresh the browser in order to see the updated web page. The image will now appear center aligned on the page, just below the feature box.



4.11 **CHECK THE IMAGE STYLING**

You will notice that the other instances of the "feature" and "middleImage" divs are all styled correctly throughout the page. This is because you defined those styles as classes, so they can be used multiple times on the same page.



5 Styling the remaining elements

Now that you have defined the styles for the main elements of the web page, you can continue adding style definitions for the remaining sections. In the next few steps, you will style the scroll button, the map, the Subscribe section, and the footer.

5.1 **STYLE THE SCROLL BUTTON**

Now you need to add style definitions to the "scroll to top" button. The button should have 50 percent opacity in its normal state and should display at 100 percent opacity when the mouse hovers over it. The "scroll to top" button should be set to invisible when the page opens. The button will be activated in the third part of this project, using JavaScript (see pp.292–295).

```
#scrollToTop {
    display: none;          /* The button is invisible when the page opens */
    opacity: .5;
    background-color: #F46036;   /* Sets the color of the button to orange */
    padding: 0 20px;
    color: white;
    width: 26px;
    font-size: 40px;        /* Sets the size of the up-arrow text in the button */
    line-height: 48px;
```

```
    position: fixed;
    right: 10px;
    bottom: 10px;
    border: 1px solid white;
    border-radius: 30px;
}
    #scrollToTop:hover {
        opacity: 1;
        -webkit-transition: all 250ms linear;
        -ms-transition: all 250ms linear;
        transition: all 250ms linear;
        cursor: pointer;
    }
```

The button stays in a fixed position in the bottom right corner of the page

A white border around the orange button makes it easier to see

The hover state is active when a mouse moves over the button

The cursor is displayed as a pointer

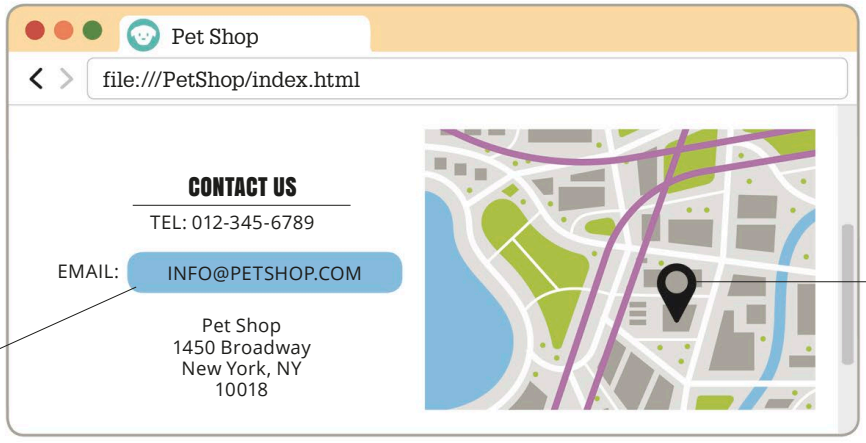
5.2 STYLE THE CONTACT US SECTION

The next section of the page that needs styling is Contact Us. The "feature" div has previously defined its two columns. The left column contains the text elements while the right has an embedded map from Google Maps. You will need to add an instruction to format the map's "iframe" element correctly. Save the code and refresh the browser to check if the section is displaying correctly.

```
.contactMap {
    width: 100%;
    height: 400px;
}
```

The map width will be **100%** of the space available in the right column

SAVE



The email call-to-action button, styled in step 4.8

A dot on the Google map shows the exact location of the address

5.3 STYLE THE SUBSCRIBE SECTION

The next element to be styled is the Subscribe section. This will appear below the Contact Us section on the web page. Add this code to set the style definitions for the Subscribe panel and the heading appearing inside it.

```
#subscribe {
    background-color: #4392F1;
    height: 160px;
    padding-top: 40px;
}

    #subscribe h2 {
        margin: 15px 0 20px 0;
        color: white;
        font-size: 24px;
        font-family: "Open Sans", sans-serif;
        font-weight: bold;
    }
```

Hex code for light blue. Sets the background color of the Subscribe section

Distance between the text and the top border of the Subscribe panel

Sets the text to white

Specifies the font used for the "h2" headers

5.4 STYLE THE INPUT FIELD

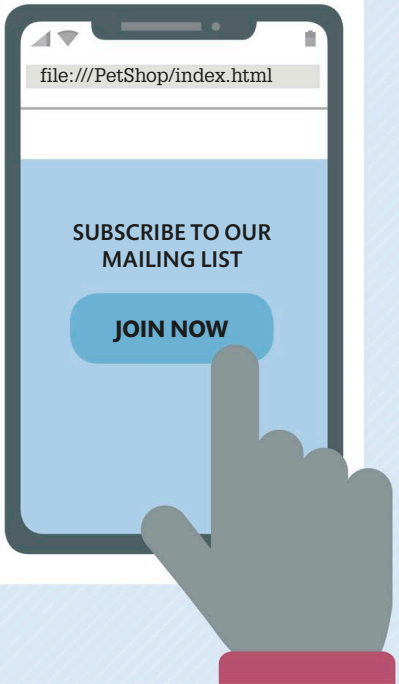
The Subscribe section has a text field where users can enter their email address. Add styles for this text input field to define its size and appearance, as well as the style of the placeholder text that will appear inside of it.

```
#subscribe input[type=text] {
        border: 0;
        width: 250px;
        height: 28px;
        font-size: 14px;
        padding: 0 10px;
        border-radius: 30px;
}
```

Ensures that only text fields are selected

No border around the email address text input box

Adds space to the sides of the text input box



5.5 STYLE THE SUBSCRIBE BUTTON

Add the code below to define styles for the "subscribe" <input> button and its hover state. The button will implement a transition on the "background-color" from dark blue to orange when a mouse moves over it, then back to dark blue when the mouse moves off of it.

Selects the input button in the "subscribe" div

```
#subscribe input[type=submit] {
        border: 0;
        width: 80px;
        height: 30px;
        font-size: 14px;
        background-color: #345995;
        color: white;
        border-radius: 30px;
        -webkit-transition: all 500ms ease-out;
        -ms-transition: all 500ms ease-out;
        transition: all 500ms ease-out;
        cursor: pointer;
}

        #subscribe input[type=submit]:hover {
            background-color: #F46036;
            -webkit-transition: all 250ms ease-out;
            -ms-transition: all 250ms ease-out;
            transition: all 250ms ease-out;
        }
```

Sets the button width to 80px

Hex code for the color dark blue

Transition instructions

The cursor will be displayed as a pointer when the mouse is over the button

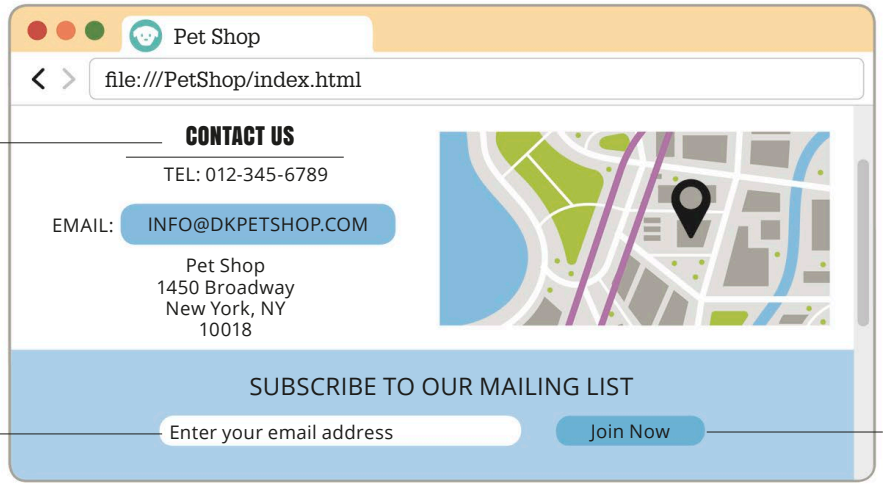
Hex code for orange in the color scheme

Repeat the transition instructions for the mouse-over effect

SAVE

5.6 VIEW THE WEBSITE

Save the code and refresh the browser to view the updated web page. Ensure that the panel is appearing below the Contact Us section and rendering correctly.



The text is aligned to the center of the left column

A placeholder text hint inside the email text input box

The button with rounded borders and transition effect will be displayed

5.7 STYLE THE FOOTER

You can now style the footer for the web page. Start by adding styles for the "footer" div, then add styles for the unordered list and the list items containing the links.

```
#footer {
    background-color: #F46036;
    height: 80px;
}
#footer ul {
    list-style-type: none;
    margin: 28px 0 0 0;
    padding: 0;
    overflow: hidden;
    display: inline-block;
}

    #footer li {
    float: left;
    }
```

Sets a fixed height for the Footer section

Does not show bullet points in the unordered list

No space between the list border and the list items

Displays as an inline-block element to allow padding and margin

Places the list items next to one another from the left

5.8 ADD STYLES TO THE FOOTER HYPERLINKS

Next, you will need to add styles for the hyperlinks that appear within the Footer section. When the mouse hovers over them, the color of the text will change from white to black.

Styles will be applied to all anchor tags that are inside list items within the "footer" div

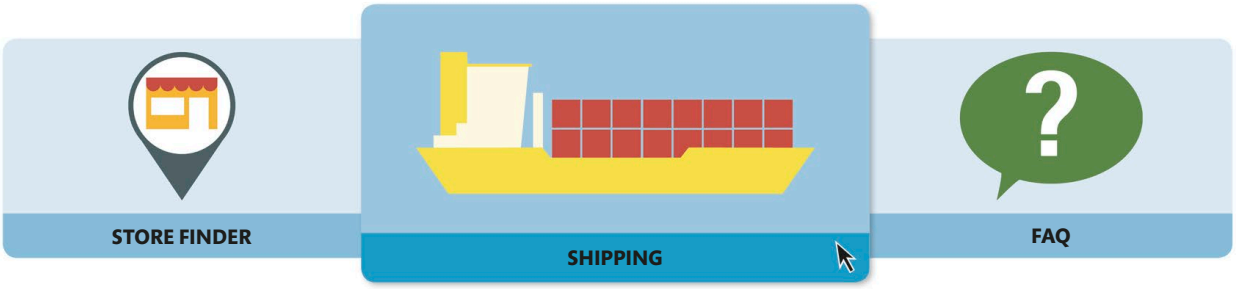
```
#footer li a {
        color: white;
        text-align: center;
        padding: 20px;
        text-decoration: none;
        font-size: 18px;
        -webkit-transition: all 250ms linear;
        -ms-transition: all 250ms linear;
        transition: all 250ms linear;
}

      #footer li a:hover {
            color: #333;
            -webkit-transition: all 250ms linear;
            -ms-transition: all 250ms linear;
            transition: all 250ms linear;
        }
```

The footer hyperlink text will appear in white

The hyperlink text will not be underlined

The text color of the footer hyperlink will change to dark gray



5.9 STYLE THE COPYRIGHT SECTION

The last element to style is the Copyright section. In this step, you will add styles for the "copyright" div and the logo it contains. Add this code and then refresh the browser to check if the Footer and Copyright sections are displaying correctly.

```
#copyright {
    text-align: center;
    background-color: #345995;
    color: white;
    height: 40px;
    padding-top: 18px;
    font-size: 16px;
}

    #copyright .logo {
        font-family: "Open Sans", sans-serif;
        font-weight: bold;
    }
```

Aligns the "copyright" contents to the center of the page

Selects the tag with the id "copyright"

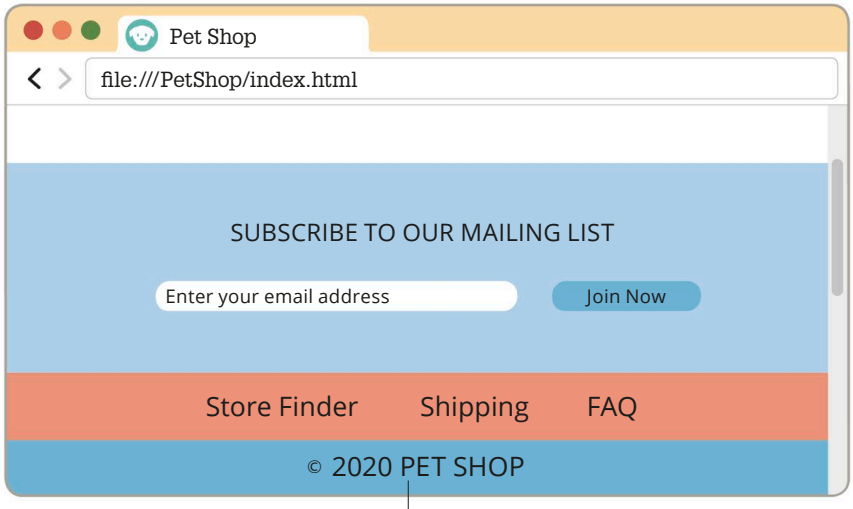
The "copyright" text will appear in white

Space between the top border of the "copyright" section and its text

Selects the tag with class "logo" inside the tag with id "copyright"

Overrides the default style definition for "logo" with a sans-serif font

SAVE



The copyright text appears in the center of the page with an HTML entity for the copyright symbol

What is JavaScript?

JavaScript is one of the most popular modern programming languages for the Web. It is an object-oriented language that is used to enhance HTML pages by adding dynamic and interactive elements to websites. Programs written in JavaScript are called scripts.

Why JavaScript?

JavaScript was invented to implement client-side behavior in web browsers. Today, however, it is used in many kinds of software and server-side web applications. For example, developers can use a cross-platform run time environment like Node.js (see p.284) to run scripts outside of the browser. This allows for a wide variety of server-side applications, such as generating dynamic HTML pages and sending responses from a Node web server.

Using JavaScript online

All modern web browsers can read and run JavaScript when rendering an HTML page. JavaScript code is interpreted and run by the browser in real time and does not need to be compiled before it is executed.

The program within a browser that executes a script is called the JavaScript Engine. This engine is an interpreter that first reads the script and converts it into machine code, then executes the machine code.

On an HTML page

To use JavaScript on an HTML page, simply enclose the script in a <script> tag. This tag can be placed either within the <head> tag or the <body> tag, depending on when the script is run—before, during, or after the HTML.

```
<script type="text/javascript">
    var x = "hello world";
</script>
```

The JavaScript instruction must be inside a <script> tag

The closing <script> tag

On an external file

JavaScript can also be placed in an external file and referenced with an "src" attribute in the <script> tag. The external JavaScript file does not need to include the <script> tag, as this has already been declared in the calling file.

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

The "src" attribute points to an external JavaScript file

CUP OF MOCHA

The language currently known as JavaScript was created by Brendan Eich for the Netscape browser. It was called Mocha during the development stage. When it was released, Netscape changed the name of the scripting language to LiveScript, renaming it JavaScript within the first year.

INTERACTION AND FEEDBACK IMPROVES **USER EXPERIENCE** AND PROMOTES EFFECTIVE NAVIGATION THROUGH A **WEBSITE**

Features of JavaScript

JavaScript allows programmers to perform calculations, validate user input, and manipulate and inject HTML elements on the page. It also has a vast library of advanced features that can be easily imported and employed in customized scripts. Even though JavaScript is a flexible language, there are limits to what the JavaScript Engine can do in the browser. For example, it cannot write files to the hard drive or run programs outside the browser.

Dedicated code editor
JavaScript can be written into any standard text file, but it is much easier to use a dedicated code editor to work with it. There are several code editors (see pp.208–209) that can be used to do this.

Scripting language
JavaScript is a dynamic language that is interpreted each time it is run. When a user requests an HTML page, the HTML page and its JavaScript code is sent to a browser where the JavaScript is processed and executed.

Web browser
While all modern browsers can execute JavaScript, each browser implements the language slightly differently. This is why programmers use libraries such as JQuery (see p.284) to code instructions that will be correctly implemented on every browser.

AJAX
AJAX, or Asynchronous JavaScript and XML, can be used to do partial updates of the content in the browser. This prevents the browser from having to do a full page load, allowing the user to stay in the same document while sending requests and receiving responses from the server.

Document Object Model
The Document Object Model (DOM) is a programming interface for HTML documents. It structures a web page so that programmers can easily access and manipulate elements on the page. JavaScript can add, edit, or delete elements in an HTML document by interacting with the DOM.

Community sharing
Programmers can share their projects through online communities or by adding to JavaScript's existing libraries. There are several code-sharing websites available online, such as Dabblet, JSFiddle, Codeshare, and Github Gist.

Variables and data types

Variables are containers that store data. When JavaScript code runs, these variables can be compared and manipulated. A variable can contain different types of data, and logical operations (see pp.270–271) should only be performed with variables of the same data type.

Primitive data types

A primitive data type is a simple data value that is not an object or a method. There are three main primitive data types in JavaScript—numbers, Booleans, and strings. Data types do not need to be explicitly stated at the time of declaring a variable (see right); JavaScript automatically infers them from the code.

Numbers

Unlike other programming languages, JavaScript does not distinguish between integers (whole numbers without a decimal) and floating point numbers (numbers with a decimal). All numbers in JavaScript are treated as floating point numbers.

```
var price = 250;
```

Number values do not have quotation marks around them

Booleans

Similar to Scratch and Python, Boolean variables in JavaScript also contain only two possible values—true or false. As the result of every logical operation is a Boolean value, these variables determine the flow of a program.

```
var isThisGold = true
```

Boolean values do not have quotation marks around them

Declaring variables

It is important to declare and initialize a variable before it can be used in a script. Initialization means to assign a value to the variable. It allows JavaScript to determine the data type that the variable contains and access its value. A variable should only be declared once in a program.

```
var lastName = "Smith";
var fullName = firstName + " " + lastName;
var firstName = "John";
console.log(fullName);
```

Declares the variable `lastName` and sets its value to `Smith`

The variable `firstName` is used before it is declared

Incorrect declaration
In this example, the variable `firstName` is used before it is declared in the code. Because its value is unclear at the time of use, the output displayed will be "undefined Smith".

string.length
Returns the number of characters in a string, starting from 1. It can be used by adding **.length** after the name of the string.

string.indexOf()
Finds the position of a string segment within another string. It gives the index position of the first character of the segment, starting from 0.

string.slice()
Extracts part of a string. It takes two parameters, the start and end index positions (beginning at 0), and returns the string segment between them.

string.substr()
Returns a string segment where the first input parameter is the start position and the second is the length of the segment.

string.split()
Divides a string into an array of substrings. For example, if "a" is the input parameter, new substrings will be formed at every instance of the letter "a".

Strings

Strings are data types that can store a series of characters or numbers. They have a number of useful properties and methods that are described above.

```
var myString = "Hello world";
```

String values always have quotation marks around them

Concatenating strings
As in other programming languages, strings in JavaScript can be joined together by using the plus (+) symbol. However, a better way to join, or concatenate, strings is by using the template literal notation (`). This format is easier to read and maintain than using the plus symbol.

```
var myBook = {
    title: "Great Expectations",
    format: "paperback",
};
var myBookDetails = `Title: ${myBook.title}
Format: ${myBook.format}`;
console.log(myBookDetails);
```

`title` is a string value

Template literals are enclosed within back-tick characters instead of quotation marks

Template literals can contain placeholders, indicated by the dollar sign and curly brackets

Nonprimitive data types

Primitive data types in JavaScript can be grouped together to form composite data types. These nonprimitive data types help organize variables into meaningful data structures that facilitate effective processing of the data. They are also called "reference variables," because they give the location of where the data is stored.

Arrays

An array is a single variable that contains a list of values. These may be strings, numbers, and even objects. Each array item can be accessed by its index position. Similar to strings, the index of the first item in an array is 0, the second is 1, and so on.

A JavaScript array value is always surrounded by square brackets

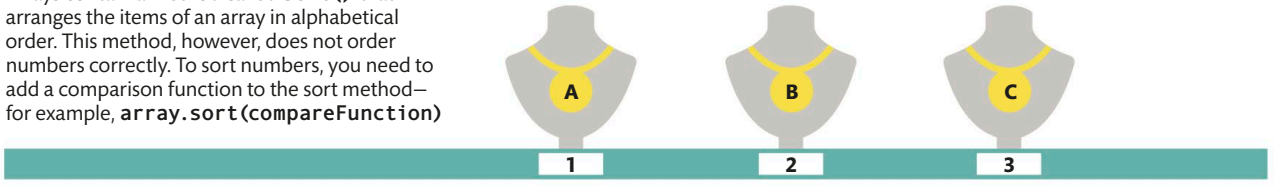
```
var jewelry = ["Locket","Earring","Ring"];
```

This array contains three strings

Index value of `Earring` is 1

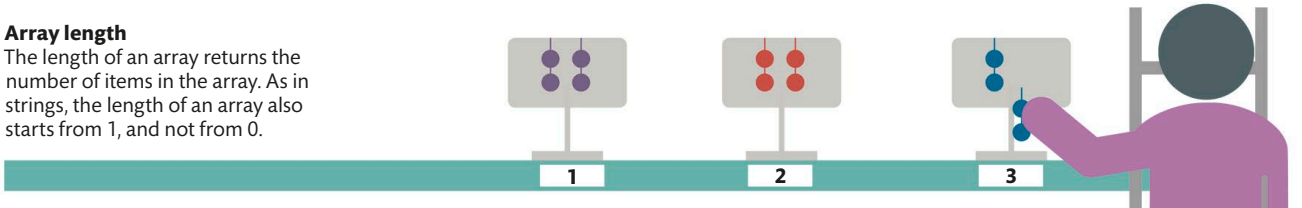
Sorting items in an array

Arrays contain a method called `sort()` that arranges the items of an array in alphabetical order. This method, however, does not order numbers correctly. To sort numbers, you need to add a comparison function to the sort method—for example, `array.sort(compareFunction)`



Array length

The length of an array returns the number of items in the array. As in strings, the length of an array also starts from 1, and not from 0.

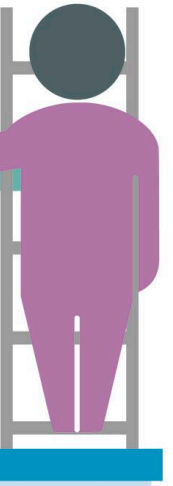


Array index

The value of an item in an array can be obtained by its index, using the syntax `value = array[index]`. To update an array item by its index, use the syntax `array[index] = newValue`

Adding items to an array

Items can be added to an existing array using the `push()` method. Though it is possible to add an item to an array by directly calling the array index, it is easier to use the `push()` method.



SPREAD SYNTAX

The `push()` method adds items to an array one at a time. To add all of the items in one go, use the spread syntax (`...`). Not only does this allow multiple new items to be added all at once, it is also possible to decide whether these should be added before or after the existing array items.

Looping through an array

It is possible to access all of the items in an array by using a **for** loop (see p.274). The loop counter loops over every item in the array, starting from 0 to the number of items in the array.

VARIABLE SCOPE

The scope of a variable describes where in the code the variable can be accessed from. JavaScript has only two kinds of scope—local variables and global variables. Local variables are declared in a function and can only be accessed from within that function. Global variables are declared outside a function and have a global scope. They can be accessed from anywhere in the HTML document.

Declares **firstName** with a global scope

`firstName` and `lastName` are available inside the function because they have global scope

```
var firstName = "John";
var lastName = "Smith";
function getFullName() {
        var result = firstName +
" " + lastName;
        return result;
}
console.log(getFullName());
```

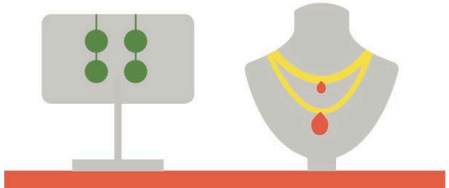
Global variable
In this example, the variable `firstName` is declared before the function and has global scope. It can be accessed from inside of a function, as well as outside of it.

JavaScript objects

A JavaScript object is a variable that has a set of properties made up of primitive data types. This way of packaging data is known as the Json data format. This format has become the primary format for packaging and transmitting data in web applications, as it is easy to use and process.

Each property is separated by a comma

The list of properties is surrounded by curly brackets



```
var person =
{
        firstName: "John",
        lastName: "Smith",
        age: 39,
        hasDriversLicense: true,
        education: [
                "Elementary School",
                "High School",
                "BA Degree"
        ]
}
```

The property key and value are separated by a colon

If the value is a string, it must be surrounded by quotation marks

If the value is an array, it must be surrounded by square brackets

Json object with mixed property types
In this example, the Json object contains properties that have different data types. The data type of each property is set when the value is assigned to the variable.

Logic and branching

Logic is concerned with determining whether a statement is true or false. JavaScript uses logical statements to determine if a variable satisfies a certain condition and then makes decisions based on whether the statement is true or false.

Boolean values

A Boolean data type only has two possible values: true or false. This means that a logical statement will always return one of the two Boolean values. These values allow an algorithm to execute a particular branch of code to produce a desired outcome.

COMPARING VALUES

Comparison operators are used in conditional statements. They compare different values to decide if a statement is true or false.

COMPARISON OPERATORS	
Symbol	**Meaning**
==	Is equal value
===	Is equal value and data type
!=	Is not equal value
!==	Is not equal value or data type
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to

Logical operators
Logical operators combine multiple Boolean values into a single Boolean result. The most common logical operators are "And", "Or", and "Not". The "And" operator (&&) demands that both Boolean values are true. The "Or" operator (||) demands that any of the Boolean values are true. The "Not" operator (!) swaps the Boolean value, so true becomes false and false becomes true. For example, "variable1 && !variable2" means "Is variable1 true and variable2 false? If so, return true."

AND
Burger **AND** fries. Both the statements must be true for the logical statement to return a true value.

OR
Meal1 **OR** Meal2. One of the statements must be true for the logical statement to return a true value.

Branching in JavaScript

The most commonly used conditional statement is the **if-then** branching statement. This statement instructs the program to execute a block of code only if a logical condition is true. A program (or algorithm) is really a sequence of conditional logical steps that are designed to transform a given input into a desired output. More steps can be added to the conditional logic by using **if-then-else** and **else-if** branching statements.

```
if (amount >= 30) {
        payment = "Card";
}
```

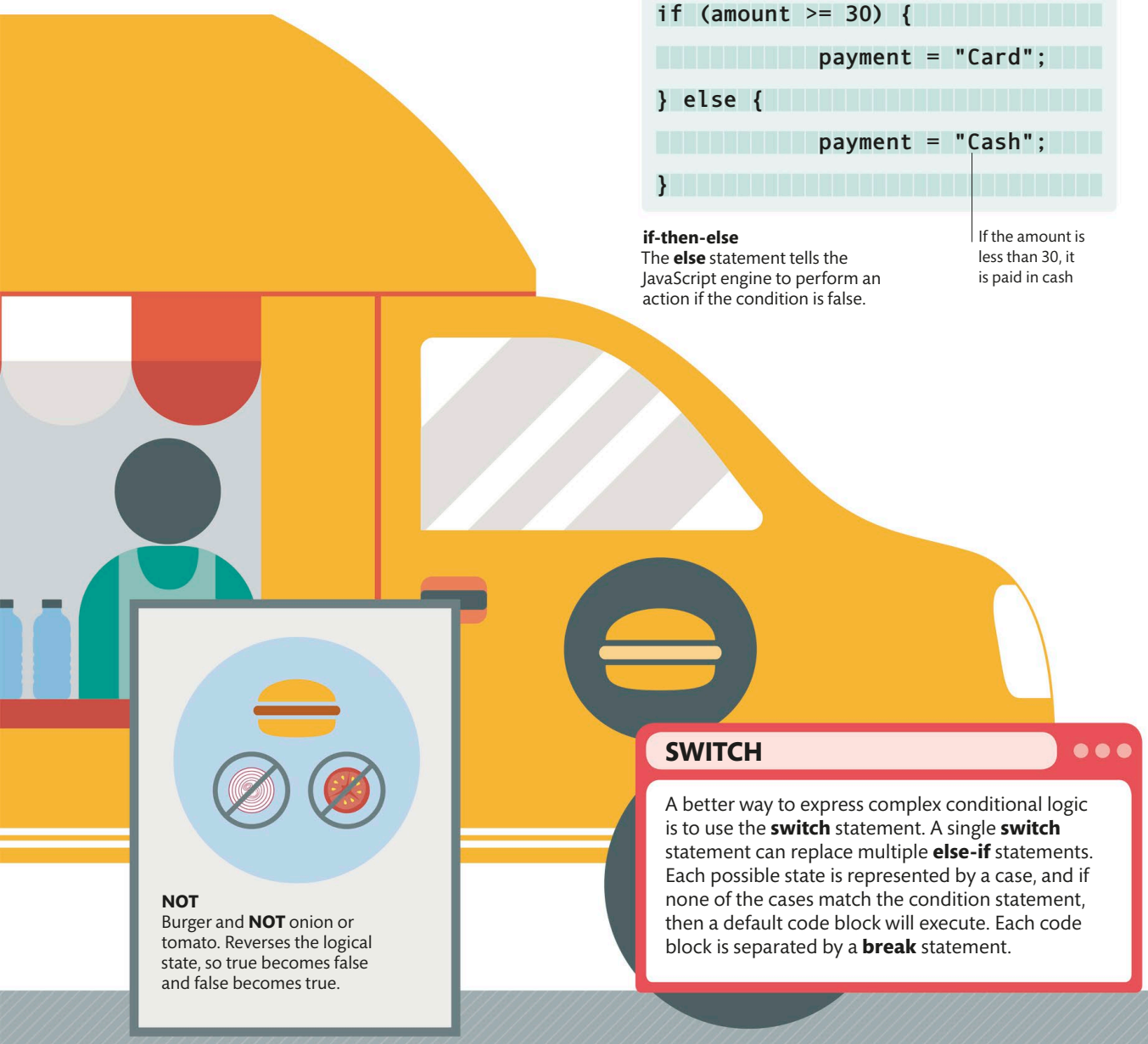
if-then
The **if** statement is used to specify a block of JavaScript code to be executed if a condition is true.

If the amount is greater than or equal to 30, it is paid by card

```
if (amount >= 30) {
        payment = "Card";
} else {
        payment = "Cash";
}
```

if-then-else
The **else** statement tells the JavaScript engine to perform an action if the condition is false.

If the amount is less than 30, it is paid in cash



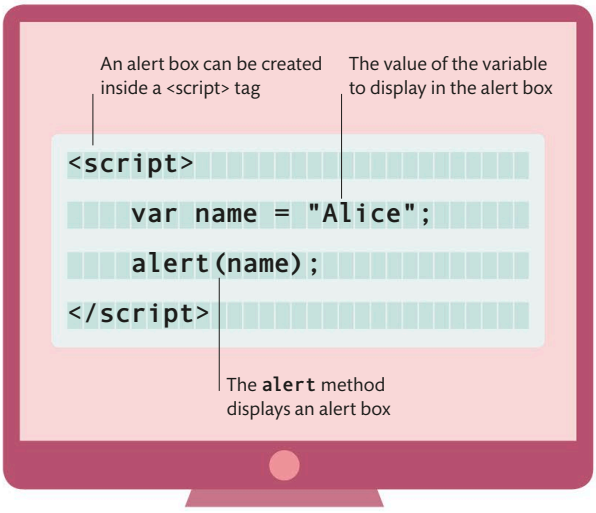
NOT
Burger and **NOT** onion or tomato. Reverses the logical state, so true becomes false and false becomes true.

SWITCH

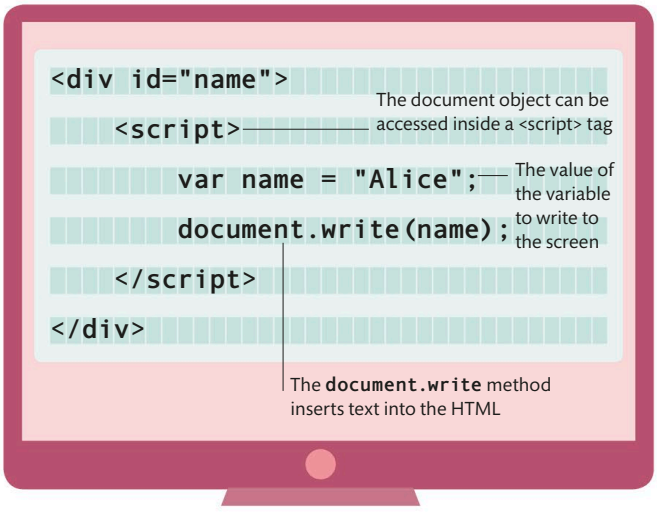
A better way to express complex conditional logic is to use the **switch** statement. A single **switch** statement can replace multiple **else-if** statements. Each possible state is represented by a case, and if none of the cases match the condition statement, then a default code block will execute. Each code block is separated by a **break** statement.

Input and output

One of the best features of the Web is that it is interactive. Using JavaScript, it is possible to program a web page to output information to the user in different forms, as well as to accept input from the user in various ways.



Show a modal alert box
An alert box is a modal window that opens above the normal browser window with a message. Users cannot continue until they dismiss the alert box.

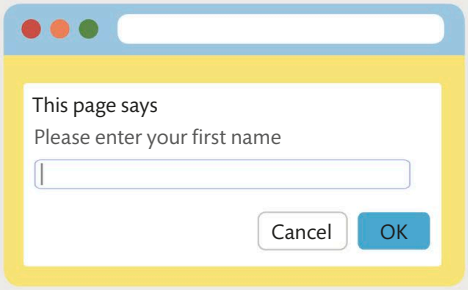


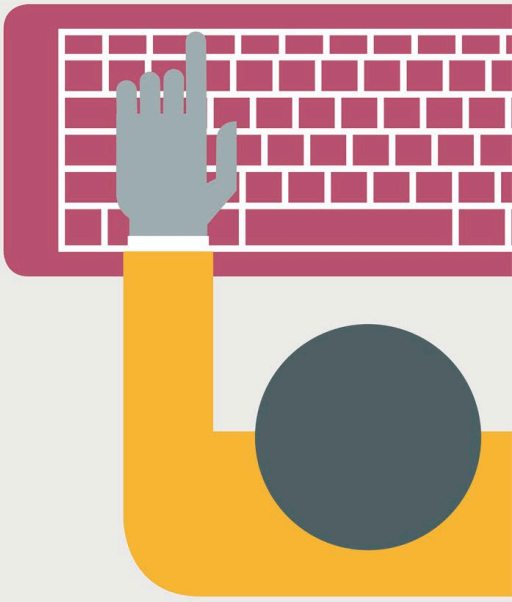
Insert data into the HTML output
This allows programmers to execute JavaScript and output some data into the HTML at the exact location where they want the output to appear onscreen.

User input

There are several ways to capture user input and work with the data in JavaScript. The choice of input method depends on the degree of urgency involved in entering the data, whether the input fields need to conform to the visual style of the page, or whether the user must answer the questions in a specific order.

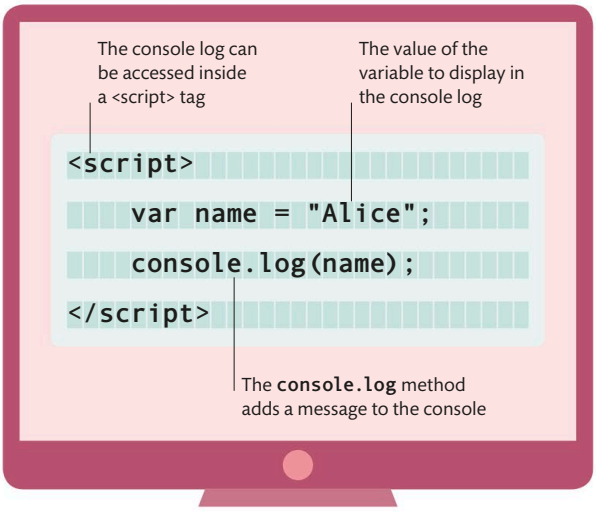
Prompt
A prompt is a modal message box that asks the user for a single line of input. The user must answer the question before doing anything else in the browser. Prompts are helpful in cases where the user must answer questions urgently or in a specific order.



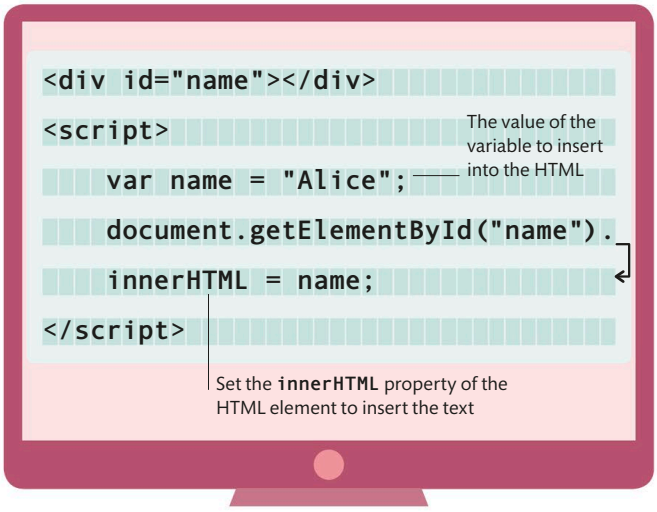


Output data onscreen

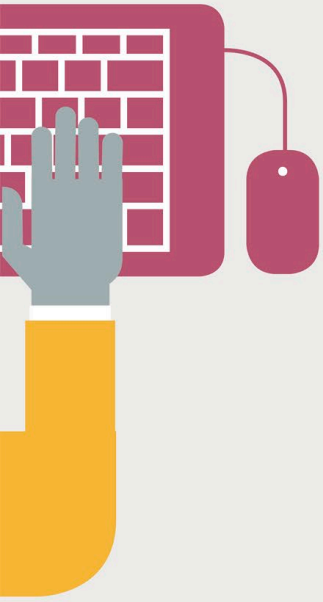
There are four different ways for JavaScript to display data back to the screen. The choice of method to employ is based on the type of information being displayed and whether the output is meant for the developer or the end user. For example, an urgent alert or question should be displayed in a modal window because users must acknowledge it before they can proceed. Debugging information, on the other hand, is intended for the developer and should be displayed in the JavaScript console log.



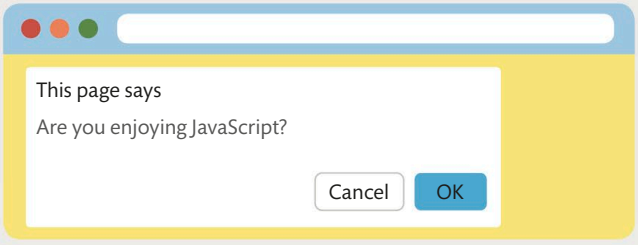
Show data in the console
Information can be output to the JavaScript console log. These log messages are very useful when debugging to see what is happening during the execution of the code.



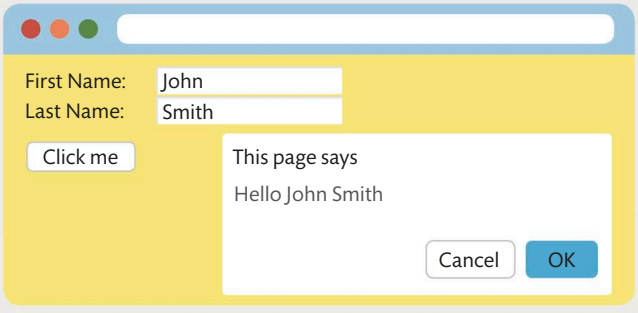
Insert data into an HTML element
Allows output to be calculated during the execution of a script and then inserted into the correct location via a placeholder HTML element.



Confirmation box
A confirmation box is a modal dialog box that is used to verify a user's intention. Users are forced to interact with the confirmation box before they can return to the main page.



HTML Input
An HTML <form> tag (see p.212) is usually used to send information entered into the input fields back to the server. However, it is also possible to use this data in JavaScript code—for example, using HTML input controls to get a user's first name and last name.



For loop

A **for** loop will repeat a block of code and each time increment a counter until the counter no longer satisfies a given condition. With each increment, the counter can increase or decrease, allowing the loop to run from top to bottom, or vice versa.

```
for (let loopCounter = 0; loopCounter < 5;
loopCounter++) {
    console.log(loopCounter);
}
```

The logical condition appears before the loop

Displays the value of the `loopCounter` variable in the console log

For loop with positive increments
The `loopCounter` is increased by `1` each time the loop repeats the block of code. The loop will stop when `loopCounter` equals `5`.

While loop

A **while** loop will repeat a block of code until a condition returns false. This is similar to the **do while** loop, except that the condition runs before the block of code, so it might not run the first time.

Using while loops
This loop is ideal when an instruction must repeat an unknown number of times. However, depending on the condition, the loop may not qualify to execute even once.

```
var numberOfDaysCounter = 0;
var numberOfDays = 3;
var daysOfWeek = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"];
while (numberOfDaysCounter < numberOfDays) {
    console.log(daysOfWeek[numberOfDaysCounter]);
    numberOfDaysCounter++;
}
```

The logical condition defines when the loop is executed. Here, it will run if the counter is smaller than the number of days

Loops in JavaScript

In programming, instructions may often need to be repeated a certain number of times or until a condition has been met. Loops allow us to control how many times a set of instructions should be repeated. There are several different kinds of loops, and each loop has it own way of knowing when to stop.

For in loop

A **for in** loop repeats a block of code for each property of an object. Inside the loop instruction, a variable is declared that will hold the value of the property as it is being processed by the loop.

Looping through arrays
This loop is perfect for processing arrays of data. The code block will process each item in the array and stop when there are no more items.

```
var myBook = {
    name: "Great Expectations",
    numberOfPages: 250,
    format: "paperback"
}
for (let property in myBook) {
    console.log(` ${property} ${myBook[property]}`)
}
```

The **myBook** variable has three properties: **name**, **numberOfPages**, and **format**

The **format** property has a string value "paperback"

The **property** variable represents the current property being processed by the loop

Do while loop

Similar to a **while** loop, a **do while** loop will also repeat a block of code until a condition returns false. However, the condition appears after the block of code and will only be checked after the code block has run the first time.

Using do while loops
This loop is used when the block of code must repeat an unknown number of times, but it must be executed at least once.

```
var numberOfDaysCounter = 0;
var numberOfDays = 3;
var daysOfWeek = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"];
do {
    console.log(daysOfWeek[numberOfDaysCounter]);
    numberOfDaysCounter++;
} while (numberOfDaysCounter < numberOfDays)
```

The condition may depend on the state of variables outside of the loop

The block of code will run before the condition is checked

Nested loops

Loops can be nested, or contained, within other loops. This allows us to iterate sequentially through all the items in a list or multidimensional array (an array containing one or more arrays).

Using nested loops

In this example, arrays represent the days of the week and temperature readings taken during that day. Nested loops are used to find the highest temperature. The outer loop represents the days of the week, while the inner loop represents the data for each day.

```
var daysAndTemperature = [
    ["Monday", 26,21,24],
    ["Tuesday", 24],
    ["Wednesday", 28,21],
];
var maxTemperature = 0;
for (let outerCounter = 0; outerCounter < daysAndTemperature.
length; outerCounter++) {
    for (let innerCounter = 0; innerCounter < daysAndTemperature
    [outerCounter].length; innerCounter++) {
        var innerValue = daysAndTemperature[outerCounter]
        [innerCounter];
        if (isNaN(innerValue)) {
            continue;
        } else {
            if (maxTemperature < innerValue) {
                maxTemperature = innerValue;
            }
        }
    }
}
console.log(`Max Temperature ${maxTemperature}`);
```

Each array has a different number of items

The `outerCounter` loop iterates through each day

`innerValue` will represent each array item inside `daysAndTemperature[outerCounter]`

If "innerValue" is not a number the code jumps to, the next iteration of the "innerCounter" loop

The `innerCounter` loop iterates through the data for each day

This variable will hold the highest value found in the array items

Displays the value of the highest temperature in the console log

Escaping loops

Sometimes the current iteration of the loop is not worth running, or programmers may have already found the answer they were looking for. To avoid wasting time processing loops that are not required, you can use the **continue** command to stop the current iteration of the loop and begin the next iteration. The **break** command can be used to stop running the loop altogether.

Break
The **break** statement tells the JavaScript Engine to stop running the loop and jump to the next instruction after the loop. This is useful, as once the loop has found what it is looking for, it can move on with the rest of the program.

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday"];
var whenIsWednesday = function (days) {
    let result = null;
    for (let i = 0; i < 7; i++) {
        if (days[i] === "Wednesday") {
            result = i + 1;
            break;
        }
    }
    return result;
};
console.log(`Wednesday is day ${whenIsWednesday(days)}`);
```

This function will return the index of the array item that matches the string `Wednesday`

Each time the loop iterates, the `i` counter will increase by 1

Checks if the value of the array item is equivalent to the string "Wednesday"

Array indexes always start from 0, so often you need to add 1 to get a "human friendly" result

Returns the value of the `result` variable after the loop has been completed

Displays the result of the function in the console log; in this case, the result is `Wednesday is day 3`

Continue
This statement tells a loop to stop the current iteration and start running the next iteration. This is useful when you know that the current iteration does not need to be executed, and you can carry on with the next iteration through the loop.

```
for (let i = 0; i < 7; i++) {
        if (days[i] !== "Wednesday") {
            continue;
        }
        result = i + 1;
```

The `whenIsWednesday` function using **continue** rather than **break**

Functions in JavaScript

A function is a block of instructions that performs a task. The code inside the function usually only executes when the function is called. To use a function, it must first be defined somewhere in the scope (local or global) from which it needs to be called.

Declaring functions

A function is declared by providing a name, a list of input parameters, and a block of code enclosed in curly brackets. A value can be returned by the function by using the "return" statement.

```
var firstName = "John";
function getFirstName() {
    return firstName;
}
console.log(getFirstName());
```

Input parameters are declared in parentheses. There are none in this example

Name of the function

Code to be executed

Outputs the result of the `getFirstName()` function to the console log

Simple function definition
Once a function has been defined, it can be called many times from elsewhere in the code.

Function statement vs. function expression

In JavaScript, a function will behave differently depending on how it was declared. Function statements can be called before the function has been declared, while function expressions must be declared before they can be used.

Function statement
A function statement begins with the word "function" followed by the function name, the input parameters, and then the code block in curly brackets.

```
function getFullName(firstName, lastName) {
    return `${firstName} ${lastName}`;
}
console.log(getFullName("John","Smith"));
```

Input parameters for the function `getFullName()`

The template literal notation `` `${variable}` `` returns a string with the variable value embedded in place

Function expression
A function expression begins with a variable declaration and then assigns a function to the variable.

```
var fullName = function getFullName(firstName, lastName) {
    return `${firstName} ${lastName}`;
}
console.log(fullName("John","Smith"));
```

Variable declaration

Nested functions

It is also possible to nest a function within another function. The inner function, however, can only be called by its outer function. The inner function can use variables from the outer function, but the outer function cannot use the variables of the inner function.

Why use nested functions?
Nested functions are only accessible from inside the parent function. This means that the inner function contains the scope of the outer function.

A function expression declaration

A nested function expression declared inside the **car** function

```
var car = function (carName) {
            var getCarName = function () {
                return carName;
            }
            return getCarName();
}
console.log(car("Toyota"));
```

The nested function **getCarName** can access the variable **carName** from the parent **car** function

Self-executing functions

Normally a function needs to be called in order to execute its code. However, a function that is surrounded by a self-executing function will run as soon as it is declared. Self-executing functions are often used to initialize the JavaScript application by declaring a global scope variable counter.

```
(function getFullName() {
        var firstName = "John";
        var lastName = "Smith";
        function fullName() {
            return firstName + " " + lastName;
        }
        console.log(fullName());
})();
```

These variables are only accessible within the self-executing function

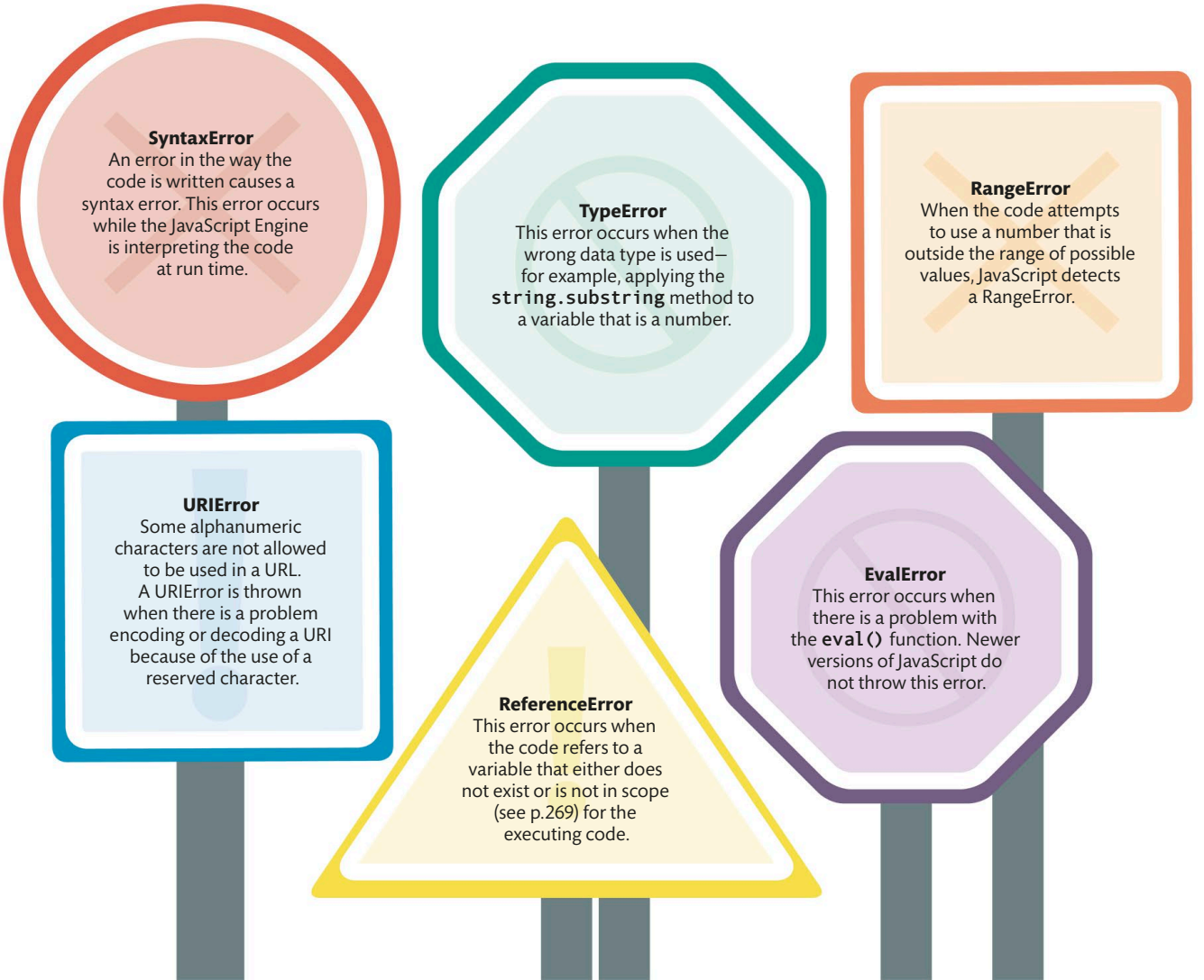
Using self-executing functions
Variables and functions declared in a self-executing function are only available within that function. In this example, the nested function **fullName()** can access the variables **firstName** and **lastName** from the parent function **getFullName()**.

JavaScript debugging

Programmers spend a lot of time diagnosing and remedying errors and omissions in their code. Debugging slows down the JavaScript execution and shows how data is modified line by line. Because JavaScript is interpreted at run time and executed inside the browser, debugging is performed with tools built in to the browser.

Errors in JavaScript

In JavaScript, an error is detected or thrown when a program tries to perform an unexpected or forbidden action. JavaScript uses an Error object to provide information about the unexpected event. JavaScript errors can be viewed in a browser's Developer Tools, inside the Console tab. Every Error object has two properties: a "name" and a "message." The name indicates the type of error, while the message provides further details about the error, such as the location in the JavaScript file where the error was thrown.



Developer tools

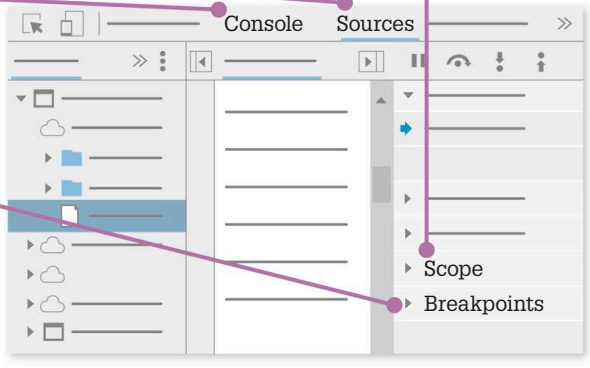
All modern browsers contain a set of Developer Tools to help programmers work with HTML, CSS, and JavaScript. The Developer Tools contain functionality to debug JavaScript and view the state of HTML elements in the browser. To open the Developer Tools for the Google Chrome browser, press Command+Option+I (Mac) or Control+Shift+I (Windows, Linux).

The Console
Web developers can output messages to the console log to make sure their code is executing as expected. The "Console" tab contains two areas:
- **Console Output Log:** Displays system and user messages from the JavaScript execution.
- **Console Command Line Interface:** Accepts any JavaScript instructions and executes them immediately.

JavaScript debugger
The JavaScript debugger can be found under the Sources tab. The debugger makes it possible to step through the code line by line to see what is happening to the variables as the code executes. On the left is a list of all the source files used by the HTML document. Select the file to debug from this list.

Scope
In the "Sources" tab, the window on the right contains the Scope (see p.269). The local and global sections under this show the variables that are defined in the current scope. The Scope pane is only populated with variables when the script is being debugged.

Breakpoints
The JavaScript Engine pauses the execution of code when it hits a breakpoint. This allows programmers to examine it. The execution can proceed in one of the following ways:
- **Resume Script Execution:** Resumes execution until the program hits another breakpoint or the program ends.
- **Step over:** Executes the next line of code in a single step and then pauses on the following line. It steps over a function without debugging the individual steps of the function.
- **Step into:** Executes the next line of code and then pauses on the following line. It will step into a function line by line.
- **Step out:** Executes the remaining code in the current function, and pauses when run time returns to the line of code, after the function was called.



GOOGLE CHROME DEVELOPER TOOLS

Error handling

In JavaScript, the **try...catch** statement allows programmers to handle errors in the code. Normally program execution stops when an error is thrown by the JavaScript Engine. However, if the code is wrapped in a try block, the execution will jump to the catch block if an exception is thrown, and the program will continue as normal. It is also possible to manually raise an error using the "throw" statement.

```
try {
    noSuchCommand();
}
catch (err) {
    console.error(err.message);
}
console.log("Script continues to
run after the exception");
```

This function does not exist

The code jumps to the catch block instead of stopping the program execution

The error message is displayed in the console

TRY...CATCH STATEMENT

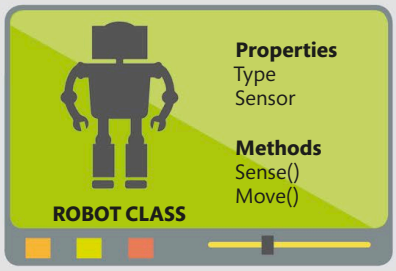
```
throw("Oops there was an error");
```

The throw operator generates an error

THROW STATEMENT

Object-oriented JavaScript

It is common in programming to create many objects of the same type. Object-oriented programming encapsulates properties and methods into classes. Functionality can be reused by creating new child classes.



Class inheritance
In JavaScript, an object can be declared as an instance of the class, and it will inherit all of the properties and methods belonging to that class. Here, the properties and methods for the class Robot can be inherited by each of its child objects.

Prototypes

Every JavaScript object comes with a built-in variable called a prototype. Any properties or functions added to the prototype object will be accessible to a child object. A child object is created as an instance of the parent object using the keyword "new".

```
let parentObject = function() {
    this.title = "123";
}
let childObject = new parentObject();
childObject.title = "ABC";
parentObject.prototype.getTitle = function(){
    return this.title;
}
console.log(childObject.getTitle());
```

Creates a new parent object

Creates a new child object as an instance of the parent object

Sets the child object's title property

Adds a new method to the parent object's prototype

Calls the method in the parent object's prototype from the child object and returns the child object's "title" property, ABC

Functions

Just as in prototypes, an object can be declared as an instance of a function with the "new" command. This command acts as a constructor (a method used for initializing the properties of the class). The child object inherits all of the properties and methods defined in the function.

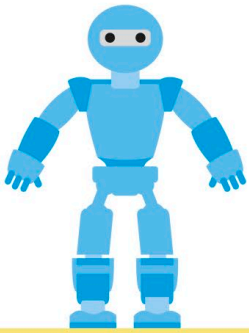
```
function Book(title, numberOfPages) {
        this.title = title;
        this.numberOfPages = numberOfPages;
};
let JaneEyre = new Book("Jane Eyre", 200)
console.log(JaneEyre.title);
```

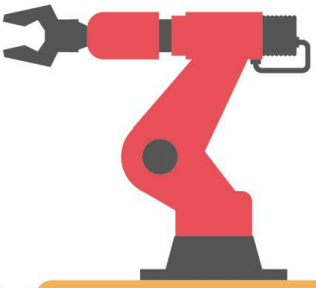
Instantiates the new book

Properties and methods of the function

Defining objects in JavaScript

JavaScript is a prototype-based language, which means that properties and methods can be inherited via the "prototype" property of the object. This differs from the way that other object-oriented languages, such as Python, construct classes (see pp.156–157). There are three ways to define and instantiate a JavaScript object in an object-oriented way: prototypes, functions, and classes.





CHILD OBJECT

Properties	Methods
Type: Humanoid	Sense()
Sensor: Temperature	Move()

CHILD OBJECT

Properties	Methods
Type: Industrial	Sense()
Sensor: Light	Move()

Classes

A JavaScript class is a special kind of function that contains a constructor method and the getter and setter methods. The constructor method runs when the object is instantiated with the "new" command, while getters and setters define how a property should be read and written. Similar to functions, classes can be defined in the ways shown below.

Class declaration
A class can be declared with the "class" keyword. The constructor method takes the input parameters necessary to initialize the object properties.

```
class Book {
    constructor(title, numberOfPages, format) {
        this.title = title;
        this.numberOfPages = numberOfPages;
        this.format = format;
    }
}
let JaneEyre = new Book("Jane Eyre", 200, "Paperback")
console.log(JaneEyre.title);
```

Defines the properties and methods of the class Book

Calls the "title" property of the object

Class assigned to the variable **Book**

Class expression
A class can also be assigned to a variable that can be passed around and returned by a function.

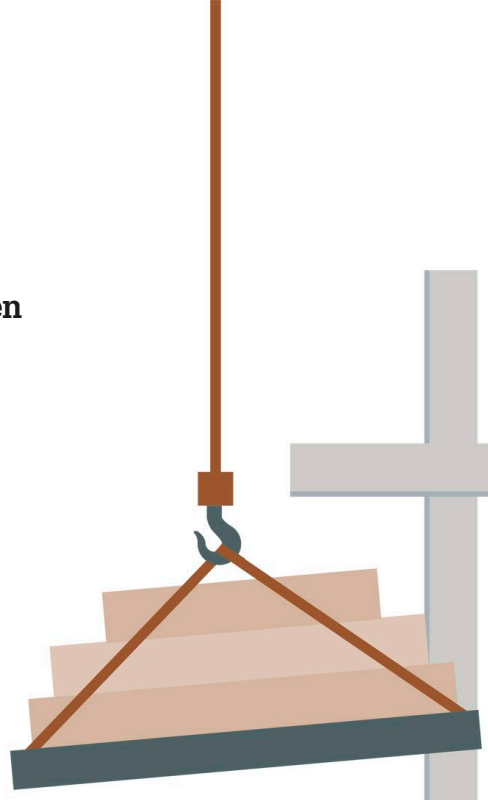
```
let Book = class {
    constructor(title, numberOfPages, format) {
        this.title = title;
```

Libraries and frameworks

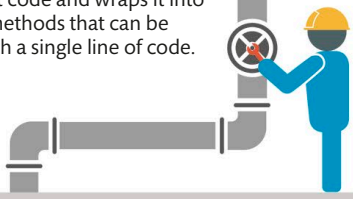
JavaScript makes extensive use of libraries of prewritten functionality that can be called in the code to make programming easier and faster. Frameworks, on the other hand, provide a standard way of programming by calling and using the code as needed.

Types of libraries and frameworks

There are various JavaScript libraries to help with all common programming tasks. For the user interface, there are tools for responsive layouts, manipulating HTML elements, and managing graphics onscreen. For data processing, there are libraries to keep data synchronized; to validate user input; and to work with math, date, time, and currencies. There are even comprehensive testing frameworks to ensure that code runs as expected in the future.



JQuery
JQuery is a framework that contains many useful tools, such as animation, event handling, and AJAX (see p.265). It takes complex JavaScript code and wraps it into simpler methods that can be called with a single line of code.

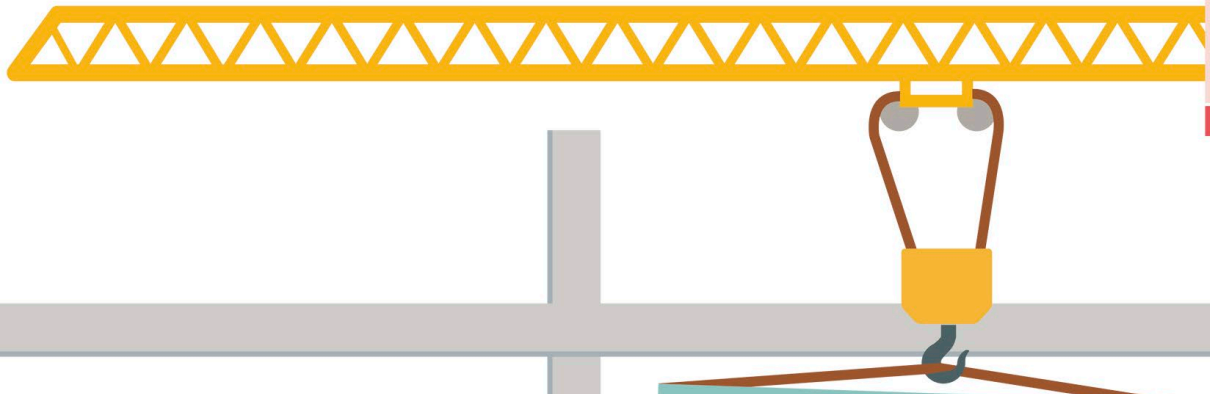


ReactJS
This library is used for building interactive user interfaces (UIs). It allows programmers to create complex UIs from small pieces of code, called "components." ReactJS uses this component model to maintain state and data binding in single-page apps.

NODE.JS AND NPM

Node.js is a run time environment that is used to create web server and API applications in JavaScript. It has a large library of JavaScript files that perform all of the common tasks on a web server, such as sending requests to a computer's file system and returning the content to a client once the file system has read and processed the requests. The JavaScript files that define the Node.js environment are interpreted by the Google JavaScript Engine outside of the browser.

Node Package Manager (NPM) is a package manager for programs written in JavaScript. It contains a database of both free and paid-for applications. You need to install Node.js before using NPM.



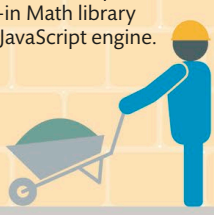
TypeScript
TypeScript is a scripting language that is used to export simple JavaScript files that can be run inside the browser. It offers support for the latest and evolving JavaScript features to help build powerful components.

RequireJS
This library manages the loading of JavaScript files and modules. It ensures that the scripts are loaded in the correct order and are available to other modules that depend on them.

Angular
This framework is used for building dynamic single-page apps. It can implement complex requirements of an app, such as data binding and navigating through "views" and animations. Angular provides specific guidelines on how to structure and build apps.

Moment.js
This library makes it easy to work with dates and times in JavaScript. It helps parse, manipulate, validate, and display date and time onscreen. Moment.js works both in the browser and in Node.js (see opposite).

MathJS
MathJS is a library that features extensive tools for working with math. It supports fractions, matrices, complex numbers, calculus, and so on. It is compatible with JavaScript's built-in Math library and runs on any JavaScript engine.



Bootstrap
This library contains many useful graphical elements and grid layout tools, which are used to create visually appealing websites that can scale to fit screens of any size. Bootstrap is a combination of HTML, CSS, and JavaScript. When applied to a page, it creates an attractive graphic user interface.

Graphic user interfaces

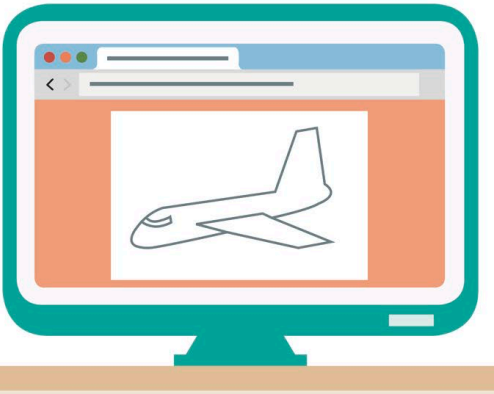
A web page is a graphic user interface (GUI) through which a user navigates a website. HTML and CSS provide the basis for the graphic design, while JavaScript adds custom logic and business rules to the elements on the page to improve the quality of the interaction.

Working with graphics in JavaScript

In an HTML document, <img> tags are used to display image files and <svg> tags are used to display vector images. JavaScript can be used to modify the properties of these graphic elements in response to user interaction. The <canvas> HTML element allows JavaScript to draw graphics directly on the screen. JavaScript also has an extensive library of frameworks (see pp.284–285) that can be imported and employed to produce complex graphic applications.

Scalable Vector Graphics (SVG)

SVG is a format that describes two-dimensional graphics in code. These graphics are then drawn by the browser on the screen. SVGs have a small file size and can be scaled to any size without losing quality. They can be drawn and exported from graphics software, such as Adobe Illustrator or Gimp. Graphics in SVG can also be styled with CSS and indexed by search engines.



Draw a company logo in SVG

In this example, you can draw a rectangle shape for the background using the <rect> tag. The <text> tag can be used to draw the logo text. You can modify the final drawing with the style attributes.

```
<svg width="200" height="100">
        <rect style="stroke:grey;stroke-width:10px;fill:red;"
        x="0" y="0" height="100" width="200" />
        <text fill="white" font-size="30" font-family="Verdana" x="20"
        y="60">SVG LOGO</text>
</svg>
```

Draws a red-colored rectangle with a gray border

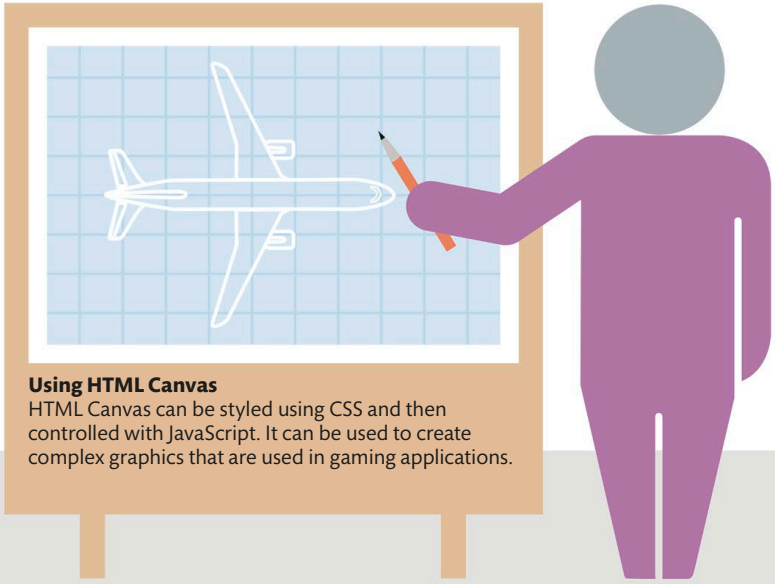
The closing </svg> tag

Draws the logo text in front of the rectangle

Uses CSS style attributes to define SVG elements

HTML Canvas

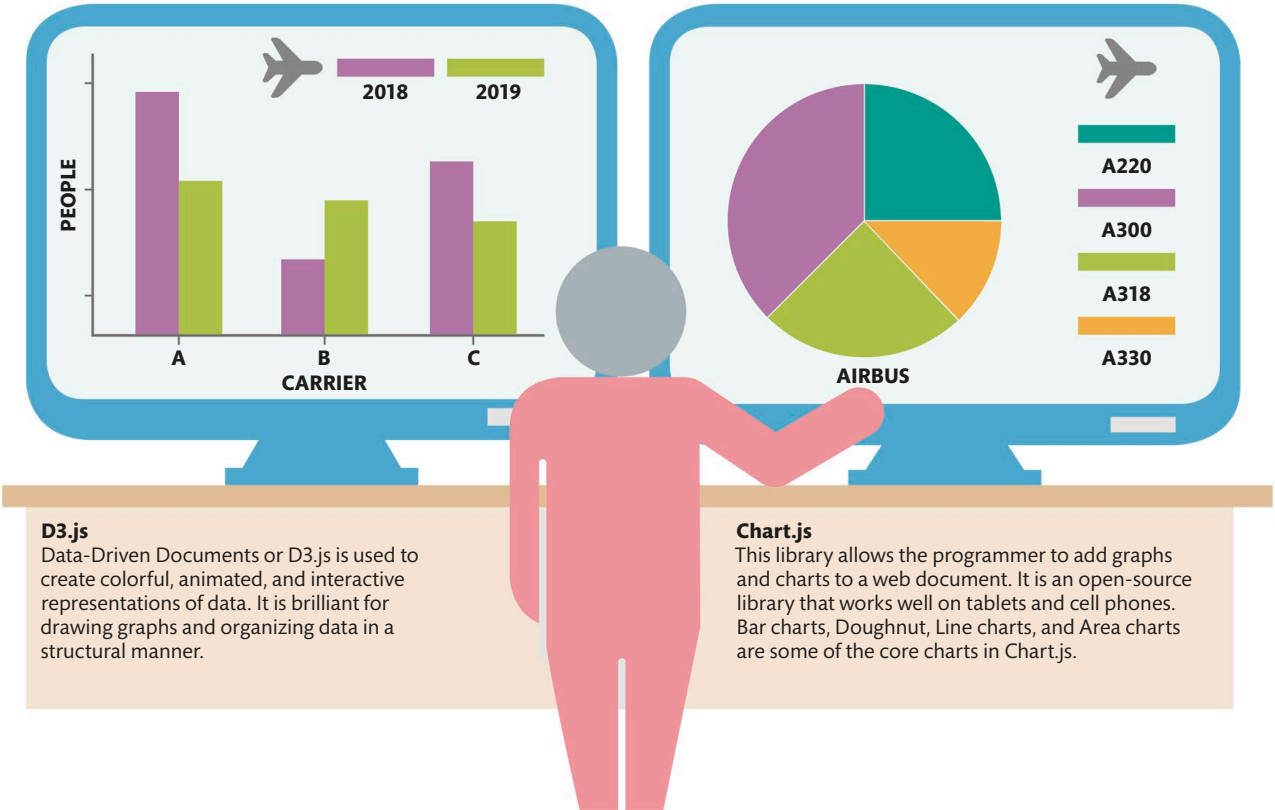
The <canvas> element defines a space on the web page where graphics can be created using JavaScript. This space is a two-dimensional grid onto which JavaScript can draw lines, shapes, and text. The grid coordinates (0, 0) are measured from the upper left hand corner.



Using HTML Canvas
HTML Canvas can be styled using CSS and then controlled with JavaScript. It can be used to create complex graphics that are used in gaming applications.

Graphics libraries

JavaScript has several built-in graphics libraries that make it easier to work with complex graphics on the Web. Each library has a specific purpose, such as converting numeric data into graphs, representing statistical data as infographics, or mapping a virtual world in a computer game.



D3.js
Data-Driven Documents or D3.js is used to create colorful, animated, and interactive representations of data. It is brilliant for drawing graphs and organizing data in a structural manner.

Chart.js
This library allows the programmer to add graphs and charts to a web document. It is an open-source library that works well on tablets and cell phones. Bar charts, Doughnut, Line charts, and Area charts are some of the core charts in Chart.js.