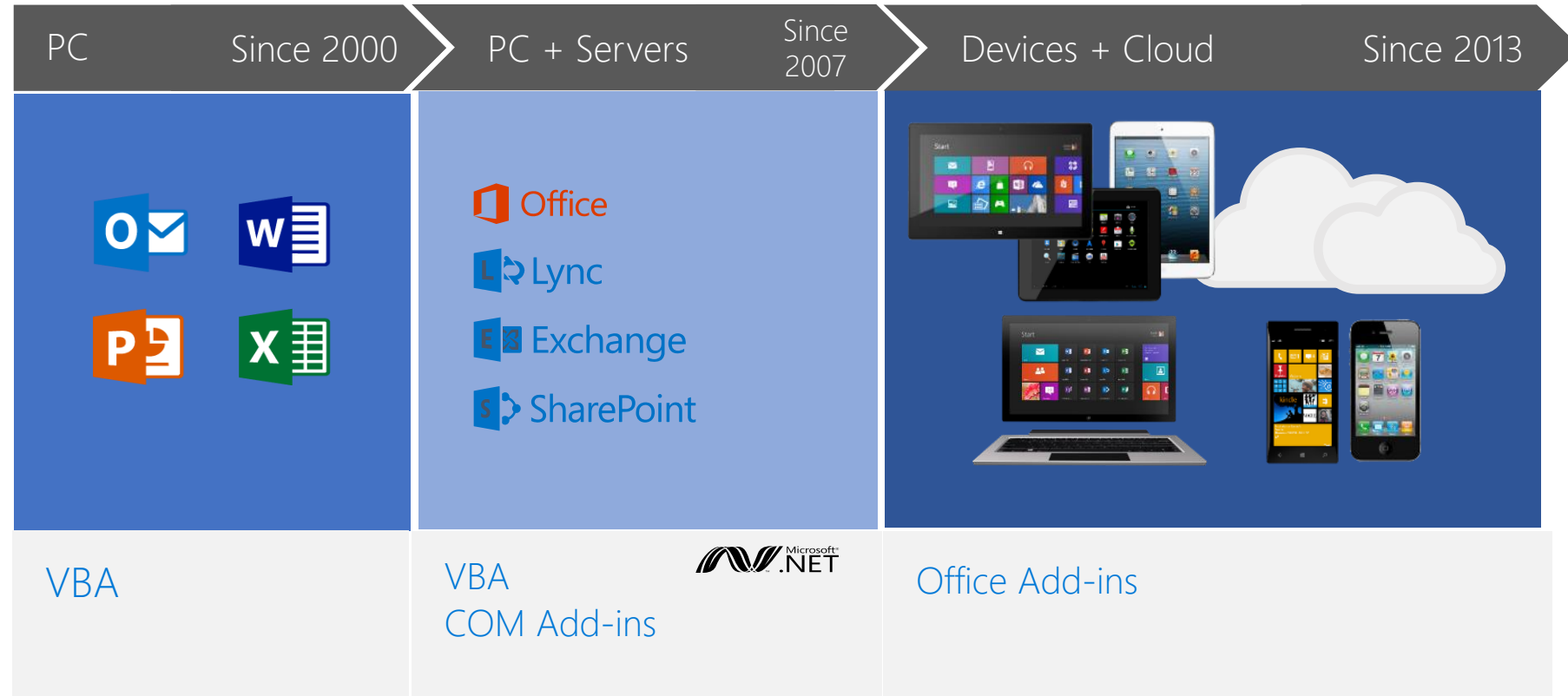


Script Lab Workshop

Office Add-ins History

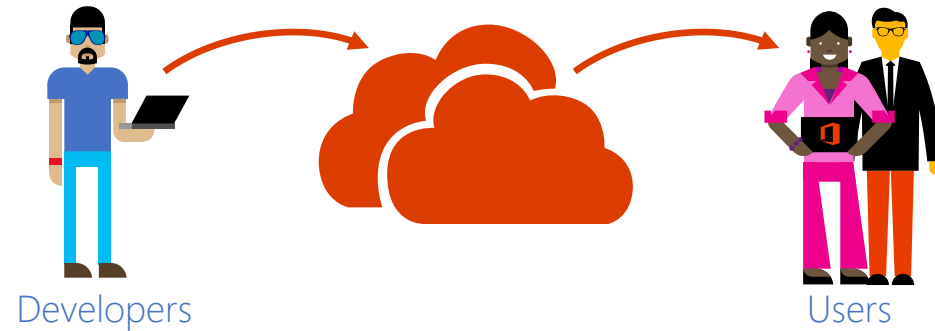


Why Office Add-ins?

Build once, run everywhere



Streamlined lifecycle



Web standards, open platform

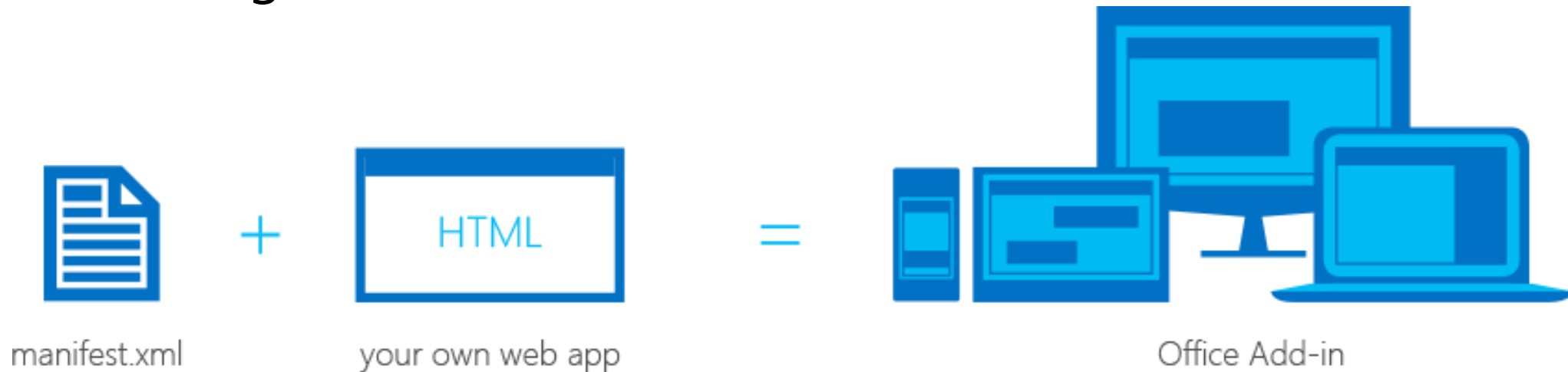


O365 integration



Office Add-ins

Extend Office clients across platforms using web technologies



npm install -g yo generator-office

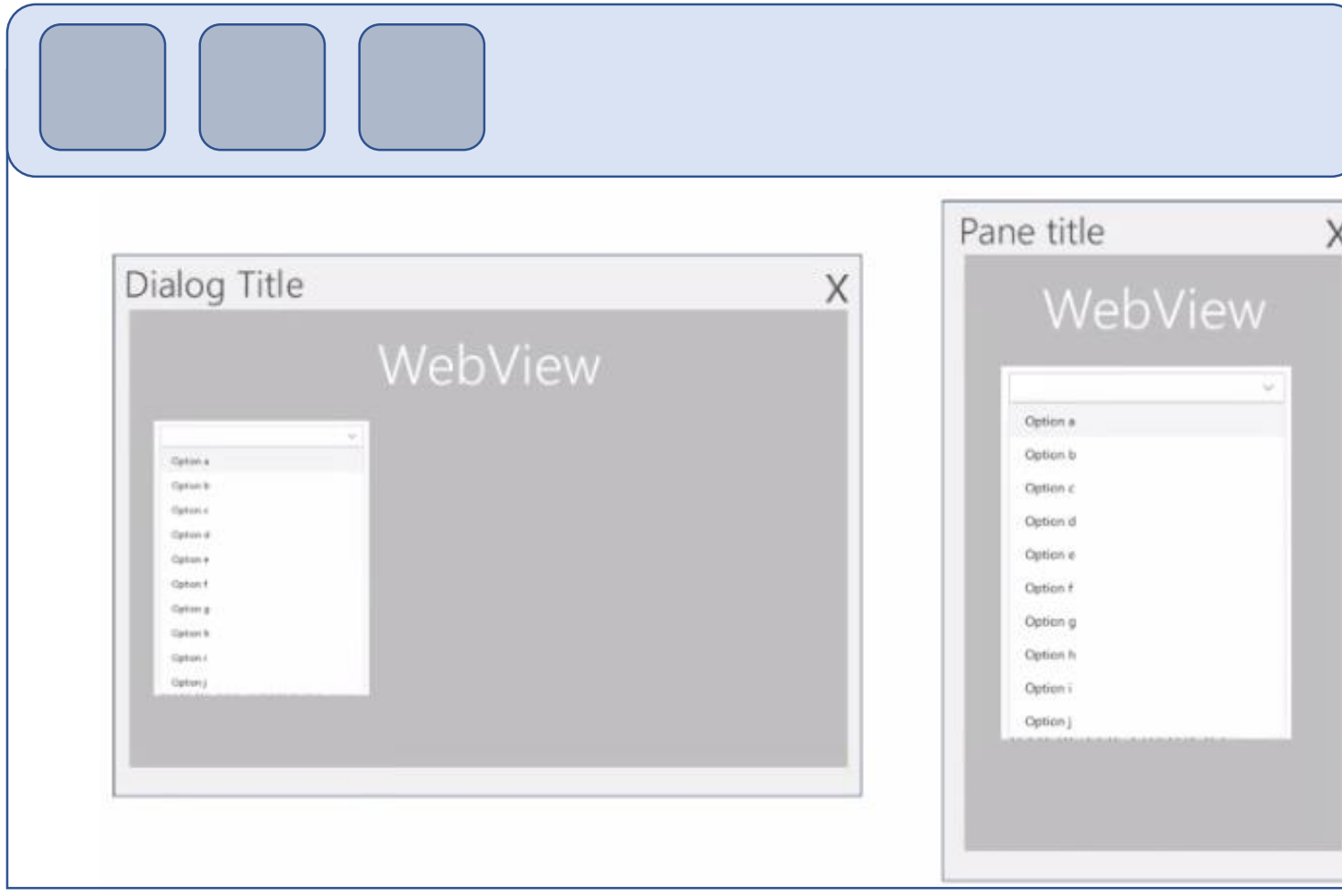
<https://dev.office.com/getting-started/addins>

+ your favorite IDE/Editor

Visual Studio

Free [Visual Studio 2015/2017 Community Edition](#)

Office Add-ins User Interface Elements



Add-in Commands
(e.g. Ribbon Buttons)

HTML Canvases
(e.g. Taskpanes, Dialogs)

Office UI Fabric (optional)
(e.g. dropdown controls)

Add-In Commands

Custom UI hooks into Office clients

Entry points

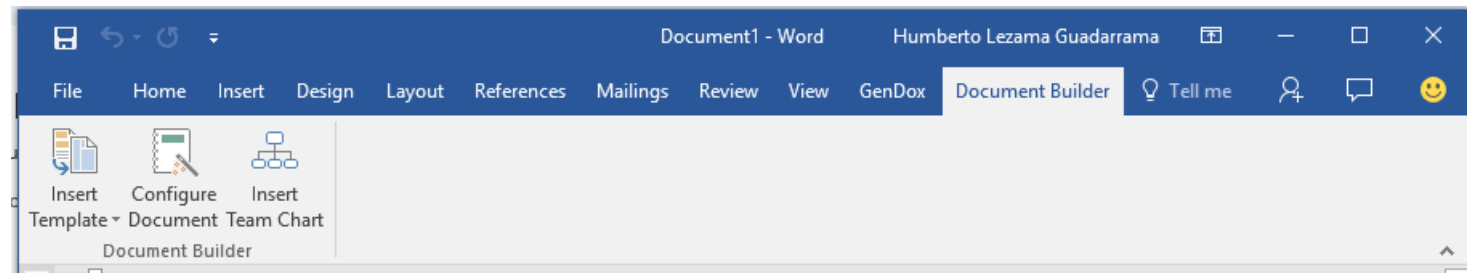
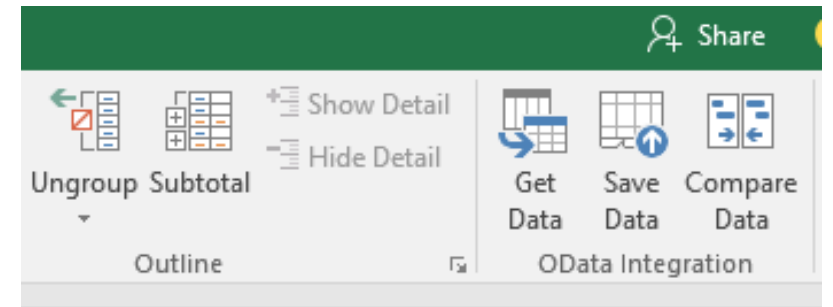
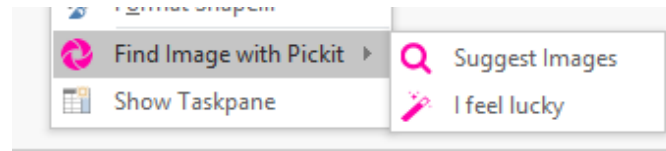
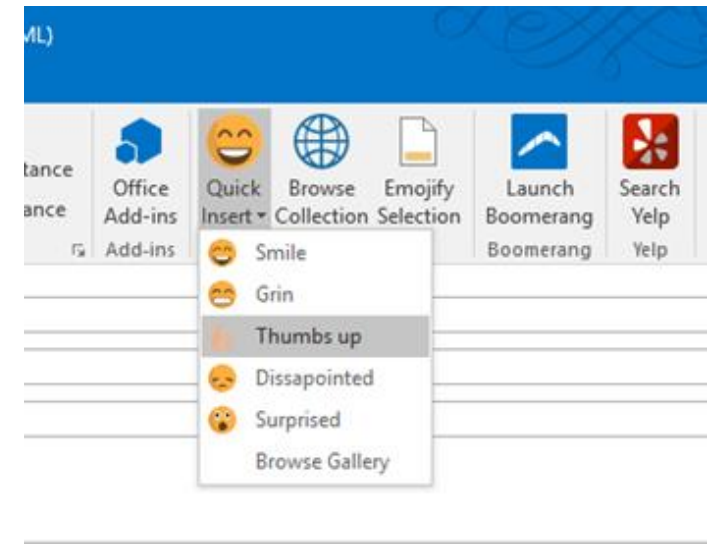
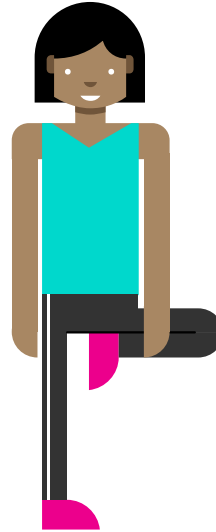
- Buttons on existing tabs
- Buttons on a custom tab
- Contextual Menus

Actions

- ShowTaskpane
- ExecuteFunction

InfoBar

- Show simple text msgs
- Alerts/dialogs not allowed



Add-in commands on the Mac

Ribbon and context menus

Show a pane or execute silent function on command activation

Exactly same manifest and code:

Office for Windows Desktop

Office Online

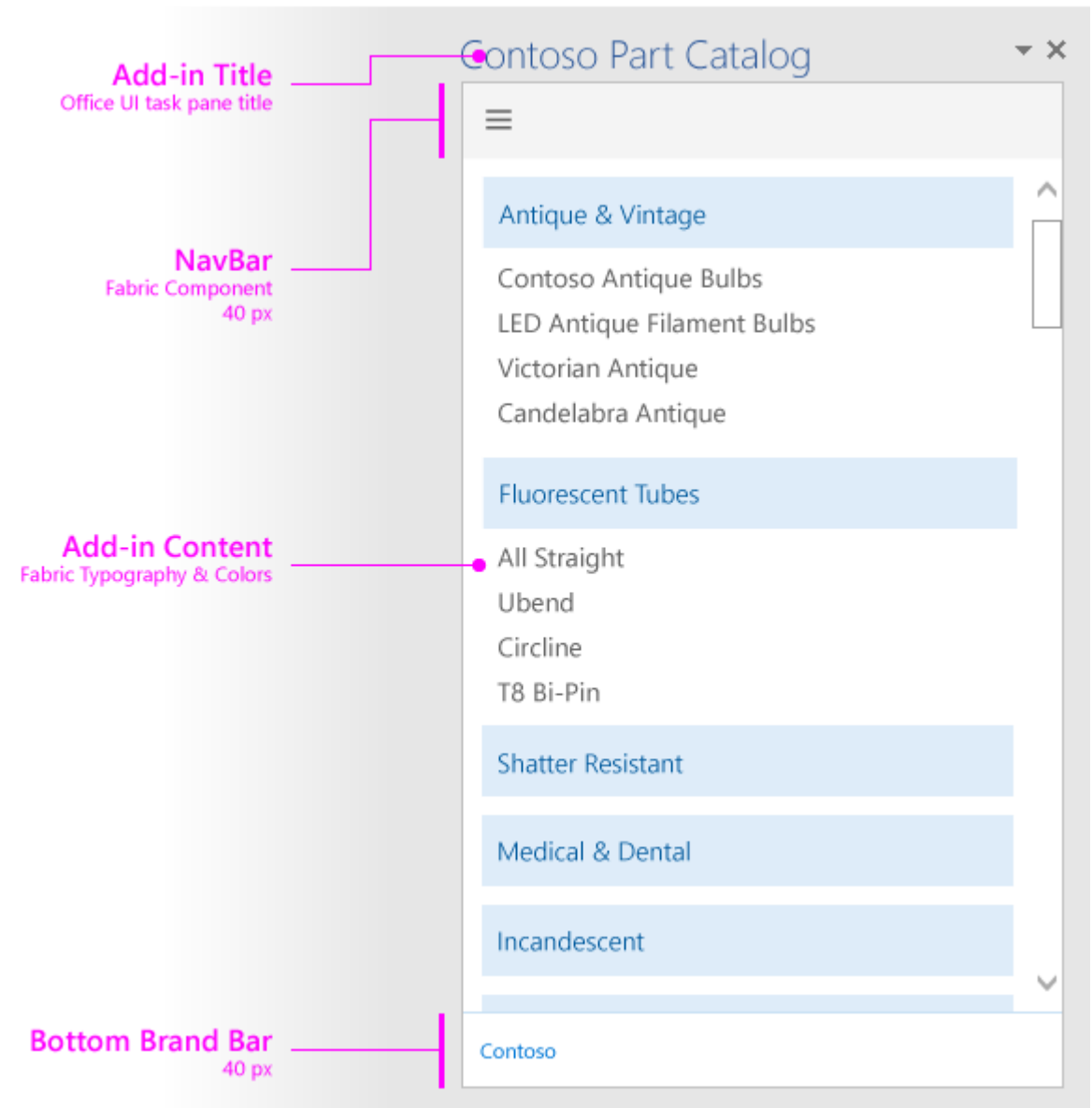
Now for Office Mac

Task Pane Layout Recommendation

Navigation element (optional) ≤ 80 pixels.

Add-in content

Branding element (optional) 40 pixels



Types of Office Add-ins

Task Pane Add-in

The screenshot shows the Microsoft Word interface with the 'Bing Translate' task pane add-in open on the right side. The add-in has a title bar 'Bing Translate' and a close button. It contains a 'From' dropdown set to 'Auto-Detect', a 'To' dropdown set to 'Spanish', and a text input field with the text 'Rojo, verde y azul.'. Below the input field is a blue button labeled 'Add translation to document'. The translation of the text, 'Red, green, and blue.', is displayed in a grey box at the bottom of the pane. Annotations with orange lines point to the 'Bing Translate' title bar (labeled 'App name'), the 'Add translation to document' button (labeled 'Task pane app'), and the translated text (labeled 'Translation based on user's selection').

Annotations:

- Insert Task pane apps from the Insert tab
- App name
- Task pane app
- Translation based on user's selection

Content Add-in

The screenshot shows the Microsoft Excel interface with a content add-in titled 'Sales Heat Map' displayed within the worksheet area. The add-in shows a map of the United States with states colored in green, red, and yellow. Above the map is a table with columns 'State' and 'Sales'. The data in the table is as follows:

State	Sales
California	\$800,000
Pennsylvania	\$600,000
Texas	\$500,000
New York	\$500,000

Annotations with orange lines point to the 'Insert Content apps from the Insert tab' button in the ribbon, the 'Sales Heat Map' title, and the map itself (labeled 'This Content app displays a geographical heat map of the selected data').

Annotations:

- Insert Content apps from the Insert tab
- This Content app displays a geographical heat map of the selected data

Mail Add-in

The screenshot shows the Microsoft Outlook interface with a mail add-in titled 'Bing Maps' displayed on the right side of an email. The add-in shows a map of downtown Kirkland, WA, with a red pin indicating a location. Below the map, there is a text area with the following content:

Hi Wendy,

Shall we go swimming this Saturday at 3pm?

We can go to this private club Adventure Works. This is their address, in case you use a GPS:

123 Fifth Avenue, Kirkland, WA 98033

We should check their hours first. Can you call them at 425-555-0199?

Thanks!

Derek Brown

Annotations with orange lines point to the 'Bing Maps' title bar (labeled 'Bing Maps mail app') and the address '123 Fifth Avenue, Kirkland, WA 98033' (labeled 'Contextual trigger in mail body').

Annotations:

- Bing Maps mail app
- Contextual trigger in mail body

Typescript vs js

Javascript:

PRO: familiar, popular, web language
widely supported in browsers

CON:

ES5, ES6, browser support, shims
lack of strong typing, late syntax/error

Typescript:

PRO: strongly typed, catches errors up front

Ready?

ScriptLab Workshop Goals:

- Learn to use ScriptLab

- Learn about Office JS API and add-ins (Excel)

- Write code

- Test

- Share your code

Get Set...

Getting setup:

1. Load Excel

Preferably Excel desktop but you can use online

2. Install the ScriptLab add-in from Store

3. Load the Office JS docs on

<http://dev.office.com/>

<http://dev.office.com/devprogram> - free 1 year subscription to Office

<https://dev.office.com/docs/add-ins/excel/excel-add-ins-javascript-programming-overview>

4. Get the lab modules:

<https://github.com/tomjebo/addin-workshop>

ScriptLab

The screenshot displays the Microsoft Excel ScriptLab interface. The main window is an Excel spreadsheet with a green header bar. The 'Script Lab' tab is active, showing a code editor on the right and a 'Run' panel on the left. The code editor contains a JavaScript snippet that uses the Excel API to format a range of cells. The 'Run' panel shows a 'Run code' button and a console area.

Code

```
Script Template Style Libraries
1 $("#run").click(run);
2
3 async function run() {
4   try {
5
6     await Excel.run(async
7       (context) => {
8         var range =
9           context.workbook.
10             getSelectedRange();
11         range.format.fill.
12           color = "yellow";
13         range.load(
14           ["address", "values"]);
15         await context.sync
16           ()
17           console.log("The
18             range address was \" +
19             range.address + \"\");
20         await populateRange
21           (context, range);
22       })
23     }
24     catch (error) {
```

Run

← ↻ Basic API call (JavaScript) →

Executes a simple code snippet.

Run code

Clear

Console DOM

Ready

context

```
var ctx = new Excel.RequestContext();
```

Requests to the Excel application

proxy objects

```
var selectedRange = ctx.workbook.getSelectedRange();
```

Trust them, they know what they're doing!

sync()

Synchronize between proxies and Excel.
Batched operations queued up get synced.

Excel.run()

Excel.run() executes a batch script that performs actions on the Excel object model.

This is where your code runs!

load()

```
object.load(string: properties);
```

```
//or
```

```
object.load(array: properties);
```

```
//or
```

```
object.load({loadOption});
```

```
var range =
```

```
ctx.workbook.worksheets.getActiveWorksheet().getRange("A1:A2").load("values");
```

You can chain together these calls to get a proxy and load values.

Async and promises

```
Excel.run(function (ctx) {
```

```
    return ctx.sync().then(function() {  
        console.log("Done");  
    });
```

```
}).catch(function(error) {  
    console.log("Error: " + error);  
    if (error instanceof OfficeExtension.Error) {  
        console.log("Debug info: " + JSON.stringify(error.debugInfo));  
    }  
});
```

Context object returns a promise. JS provides the .then construct.

Go!

Module 1

ScriptLab

Run Basic API call (js)

Populate cells

MySnippets

Async and Typescript

```
async function run() {  
  try {
```

```
    await Excel.run(async (context) => {  
      // some code ...
```

```
      await context.sync()
```

```
      console.log("The range address was \"\" + range.address + "\".");  
      await populateRange(context, range);
```

```
    })
```

```
  }
```

```
  catch (error) {
```

```
    OfficeHelpers.UI.notify(error);
```

```
    OfficeHelpers.Utilities.log(error);
```

```
  }
```

```
}
```

Use async and await
construct.
More readable!

Better, faster, smarter!

Module 2

Typescript exercise

Run Copy and Multiply Values sample

Grand Total

Tax

Charts

Module 3

Add a chart

Hints:

- Use the chart collection add method.
- See <https://dev.office.com/reference/add-ins/excel/chartcollection>

Functions

Module 4

Range.Calculate() and calculate():

See : <https://dev.office.com/reference/add-ins/excel/range>

ConditionalFormat object:

See: <https://github.com/OfficeDev/office-js-docs/blob/ExcelJs\OpenSpec/reference/excel/conditionalformatcollection.md>

Calling Graph

Module 5

- Create a new add-in
- Register the add-in application for use with Graph permission scopes
- Use the Office JS Helpers to authenticate
- Call Graph to send email

<https://dev.office.com/docs/add-ins/excel/excel-add-ins-javascript-api-reference?product=excel>

<https://dev.office.com/docs/add-ins/excel/excel-add-ins-javascript-programming-overview?product=excel#excelrunfunctioncontext--batch->

Thank You!

Questions?