the email factory

An inside look at email development

Meet the email devs

We are **The Email Factory** and we do all things email. Design, coding, deployment, data-processing to name just a few.

Today we'll focus on the technical side of **email development**. And who is responsible for that? Our team of talented email coders, of course! Let's take a look inside their world...

The quirky world of email development

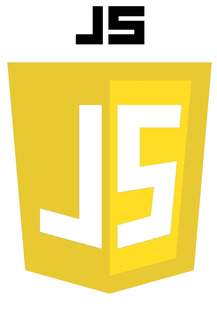
Email can be thought of as **web development's quirky cousin**. They share some fundamentals... but there are also some stark differences. Let's start by looking at the three base technologies that comprise the front-end of a *website*:

HyperText Markup Language

Sets the structure and content of a page – this is a nav bar, this is a paragraph, this is a button.

Cascading Style Sheets

Controls the appearance – colours, font sizes, scaling on mobile and so on.



JavaScript

Enables user interactivity and other advanced behaviour.

Emails on the other hand are built using only **two** of those technologies:

That's right, email is a JavaScript-free environment. But we still have access to everything HTML and CSS have to offer, right? Well, not exactly.

Both HTML and CSS have developed over the years. Web browsers are largely standardised and can use all the latest code. Email applications on the other hand... not so much.

We're **web browsers**, and we (mostly) agree on how a website should look.

We're **email applications**... and we do whatever the heck we want!

What this disparity actually means in practice is significant: **email applications have very different ideas about how your email should look**. We'll illustrate this with an extreme example – Apple Mail versus Microsoft Outlook. Check out this portion of an email in each app:

Apple Mail

VS.

Microsoft Outlook

Clearly the Apple Mail rendition is visually richer in every way. Here's a list of the specific elements that are missing in Outlook:

- ⊗ Rounded container corners
- ⊗ Tilted container
- ⊗ Gradient fill
- ⊗ Drop shadow
- ⊗ Glow effect on button
- ⊗ Web font

Ouch! What went wrong? The answer may surprise you: **nothing went wrong**.

Apple Mail boasts near-web browser levels of CSS support. It's an email developer's dream. Outlook, on the other hand, **uses Microsoft Word as its rendering engine**. That's right, a *word processor* is attempting to render your fancy HTML email.

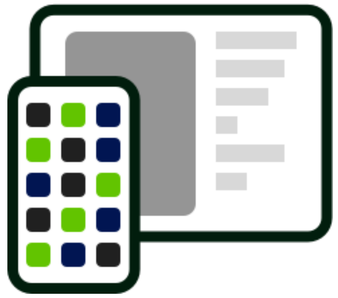
And that's ok. Outlook is not designed to render complex email designs that resemble mini websites. Its purpose is to write, send and receive simple **electronic mail** with some basic formatting. And it performs that role just fine.

That brings us to the concept of **graceful degradation**. This is web dev speak for the idea that a website should still work on old browsers in a simpler form... rather than a *broken* form.

It's easy to see how the same principle applies to email. There's a wide spectrum of email-rendering capabilities.

Outlook, Gmail and Apple Mail may be popular, but they are only three of the wide and varied range of email apps out there. There's also Yahoo! and Outlook.com webmail and Samsung Mail and Thunderbird and Mail.ru and BlueMail and... you get the idea!

Sounds complicated, right? We've only just begun! Not only does an email developer need to consider all of these email apps, they also need to consider **user device**, software versions, display settings like **dark mode** and assistive technologies like **screen readers**.



And all of this must be achieved via a single HTML email. That's a big task – but not an impossible one. The solution: a whole bag of undocumented **email-coding quirks, tricks and workarounds** that would make a web developer say *huuuuh?*

Set the table

Let's take a look at a fundamental aspect of web and email development: **page structure**. A modern website is laid out using **divs** – they're a highly versatile, general-purpose container.

Divs (and other page elements) are positioned using modern CSS layout systems like **grid** and **flexbox**. That gives a developer fine control over page structure at any screen size.

Your heading goes here

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

But it wasn't always like this. Back in the 1990s, **HTML tables** were used for page layout. It's important to note that page structure was *not* tables' original purpose, but rather the presentation of data in a spreadsheet-like matrix. You know, something like this:

Pet	US population	US inhabited households
Dog	77.5 million	45.6 million
Cat	93.6 million	38.2 million
Fish	171.7 million	13.3 million
Birds	15.0 million	6.0 million

Wow, that's a lot of fish! But let's get back on-topic. HTML tables were later **repurposed to arrange the structure of a web page**. It wasn't a perfect system, but it was good enough for the web at that time in history.

Have you guessed where this is going? That's right – **emails to this day are built using HTML tables**. Although modern CSS technologies like divs and grid work in some email applications, *they don't work in all*. Therefore HTML emails must be constructed the old-fashioned way.

```
<table>
  <tr>
    <td>
```

Let's try to put this into perspective. If you asked a web developer these days to make a website using tables, it would be like asking a builder to construct your home using sticks and mud... while still making it look thoroughly modern!

Style-conscious

There's another fundamental to look at: **the implementation of CSS in email**.

CSS is responsible for bringing an HTML page to life visually. It has grown and developed over the years and is now jam-packed with features. Here's a rough idea of what it can do:

Generally speaking, the newer and more advanced a feature of CSS is, the less likely it is to work in *all* email applications. That's doesn't necessarily mean that you can't use it. But you do need to design and code with **graceful degradation** in mind.

The scale of this task should not be underestimated. There are **over 200 CSS properties**. And there are lots of email applications out there. What's more, whenever an app is updated, its CSS abilities can change overnight without warning – for better or worse!

Campaign Monitor's *Ultimate Guide to CSS* is an essential resource that attempts to keep tabs on this ever-changing landscape.

Uneven compatibility however isn't the only CSS challenge unique to email...

We need a quick lesson about the 'C' in CSS. It stands for ***cascading***. This refers to the hierarchical nature of CSS.

A property can be passed down from one page element to another. For example, let's say we apply the text colour *green* to all `<article>` tags on the page:

```
article {
  color: green;
}
```

Within our article, let's say there's a div, followed by a paragraph tag – with some text.

```
<article>
  <div>
    <p>
      This is some text.
    </p>
  </div>
</article>
```

Here's how that would look in a web browser:

This is some text.

We didn't colour paragraphs green. We didn't even colour divs green. But as child elements of our article tag, the green colouring has cascaded all the way down to paragraph level.

The next thing to consider is how CSS code is applied to a document. There are three methods:

1. **External style sheet**: your HTML documents refer to a separate CSS file.
2. **Embedded style sheet**: your HTML documents include internal CSS rules.
3. **Inline CSS**: CSS is applied directly to individual HTML elements on your page.

On a website, external style sheets are the best way to manage your CSS. This makes it possible to apply rules globally to your website. If you'd like to change something, you need only update a single file. Phew!

But email is not a website. **Email is email** – and the technicalities here are different. For maximum email app compatibility, styles are largely applied **inline**. Yes, that means virtually every visible or structural element on the page has CSS applied directly to it. An inline style looks something like this:

```
style="font-family: Arial, Helvetica, sans-serif;
font-size: 16px; line-height: 22px; color: #011d0d;
font-weight: 400;"
```

With dozens of components such as images, buttons, links, containers, nav bars and text blocks making up an email, these inline styles add up to a *lot* of **code repetition**.

But there's more! Inline styles are not the only CSS present in an email. There is in fact also an **embedded style sheet** at the start – or head – of the document.

It serves a couple of purposes. The first, less glamorous function is to apply a few general CSS rules and fixes for known rendering quirks in specific email applications. Don't worry, we won't bore you with the details.

The next function is **responsive design**. That is to say, the essential rules for how your email should be resized, reshaped and rejigged to look its best on mobiles.

We've now covered the basics of table-based structure and the unusual application of CSS. And yet we've only scratched the surface of email development's quirky nature...

Why is email dev like this?

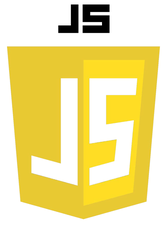
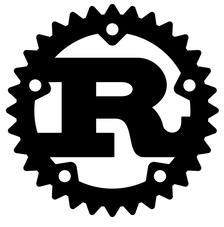
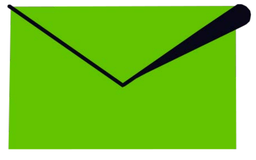
You might be wondering **why email code is so... weird**. The answer is complex and multi-faceted but it can be summarised with this one-liner:

Email code is not a real thing

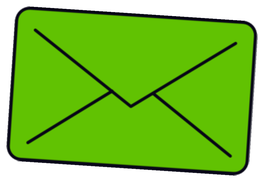
Or to phrase it less dramatically, **email code is not a formally-documented discipline**. By contrast, if you set out to learn a programming language like JavaScript or Python or even C++ if you're feeling brave, there are official docs and tutorials aplenty.

Not so for email. Email coding practices are largely based on experimentation, with tricks and tips shared among a small, specialist community.

Let’s make this visual. While Java and Node and Ruby have their rules written down in black and white, email development is the outlier with no official ways of doing things.

Rules	No rules
        	

This naturally leads on to the question: **why isn't email coding documented?** And although it's uncool to answer a question with a question, that's what we're going to do. That question is:



What *is* an email?

A silly question on the surface, perhaps. We all receive dozens of emails every day. Of course we know what an email is!

But **electronic mail** comes in many forms. It can be a message from Wendy in accounts about that overdue invoice. It can be a simple text-based booking confirmation for that weekend in Barcelona you've been looking forward to. It can be the question you shoot off to customer services to double-check your baggage allowance.

Or it can be an all-singing all-dancing marketing email.

Let's focus on the last one – **marketing emails**. That's what we're all here for.

Email as a medium was not originally envisioned as a graphically-rich, website-esque environment. And it can be argued that it *still* isn't meant as such. Email dev very much involves **pushing email applications beyond their intended functionality**.

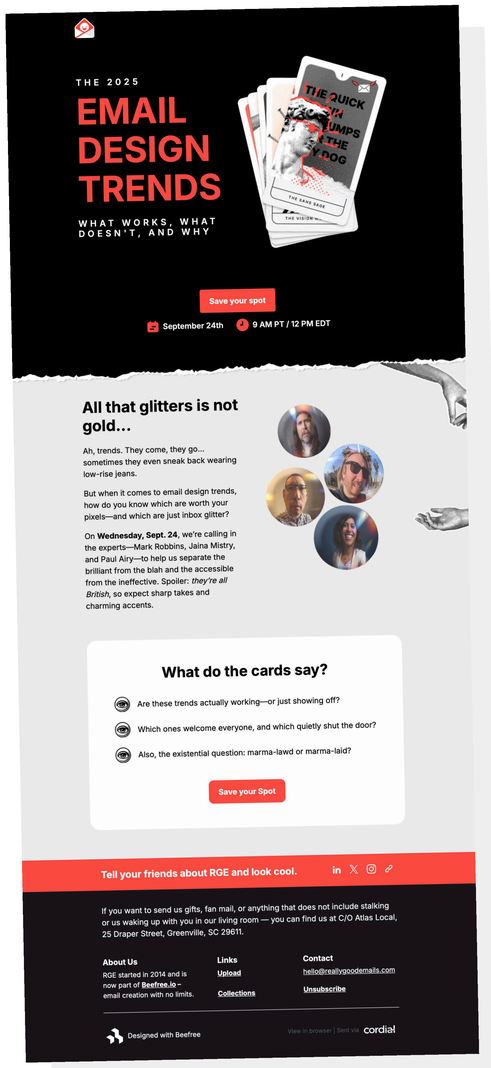
There's often a gap between a brand's vision of what an email should be... and an email application's capabilities. So, to answer the question – *what is an email* – let's go with this:

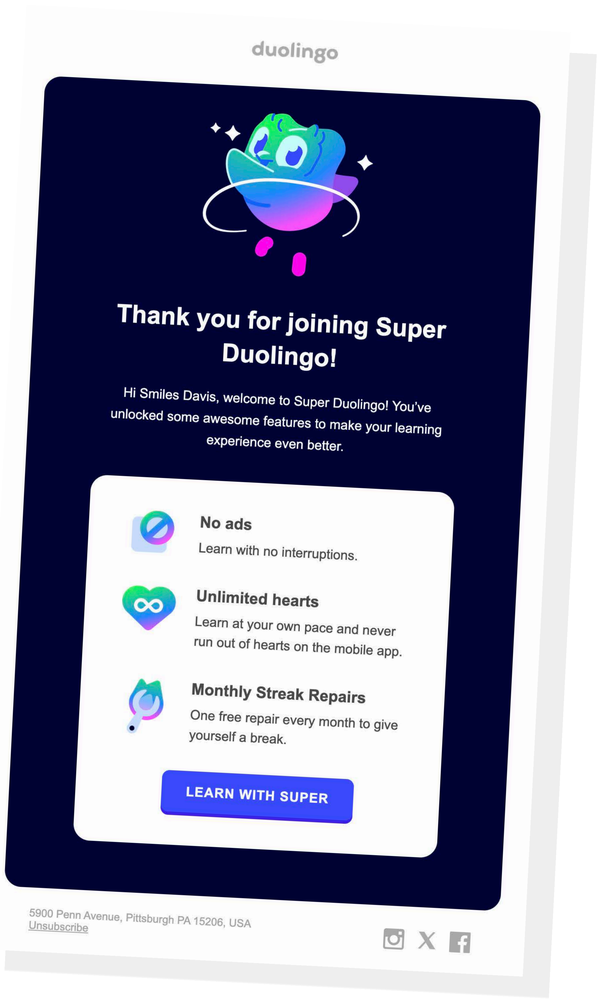
An email is whatever the app allows it to be

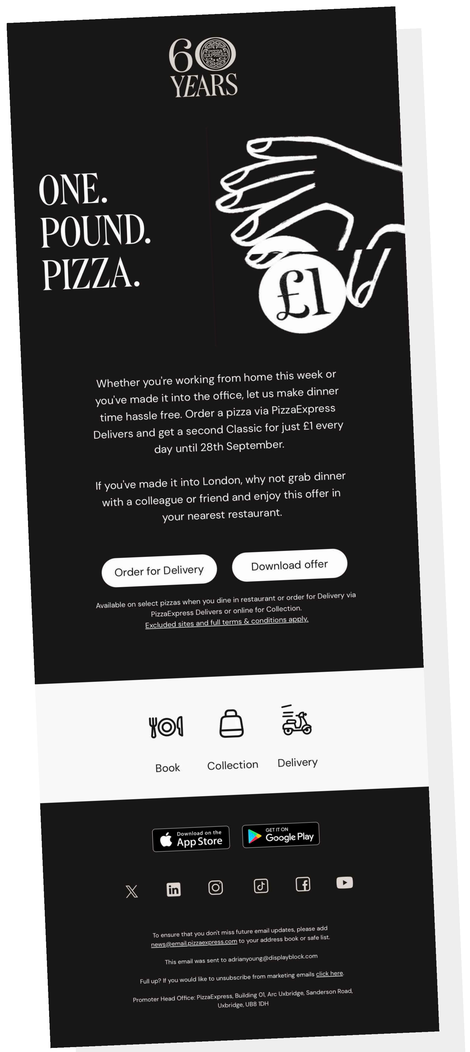
The rendering differences between the market-leading applications suggest that Microsoft, Google, Apple and Samsung have very different interpretations of the medium.

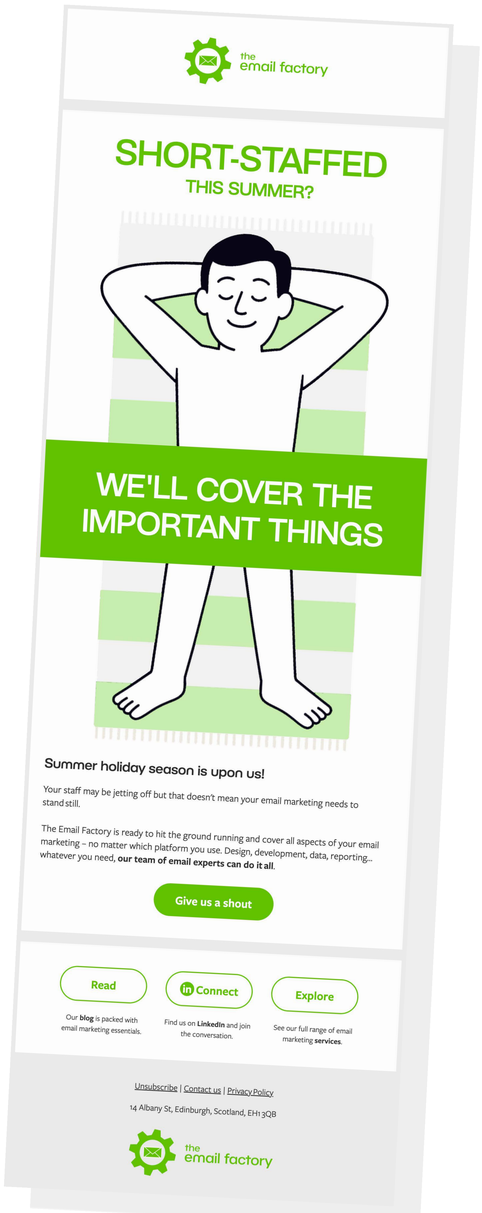
This might all sound a little negative. But we don't see it that way.

The quirks and inconsistencies of email development are what **make it special**. By producing a professional-quality marketing email that looks beautiful *everywhere* – and is **accessible** – you are using niche skills to bend the rules. That's a satisfying thing.









Tools of the trade

Web development is rich with frameworks, plug-ins and – more recently – AI coding assistants.

Email development, on the other hand, is a world of unorthodox code, clever tricks and workarounds. As we mentioned earlier, the field largely involves pushing the medium beyond its original scope.

A side effect of that is a lack of bespoke email development tools. No problem – if the perfect tool does not exist, **we'll make it**. Email teams often find themselves in a situation where they need to modify an existing application... or build one from scratch!

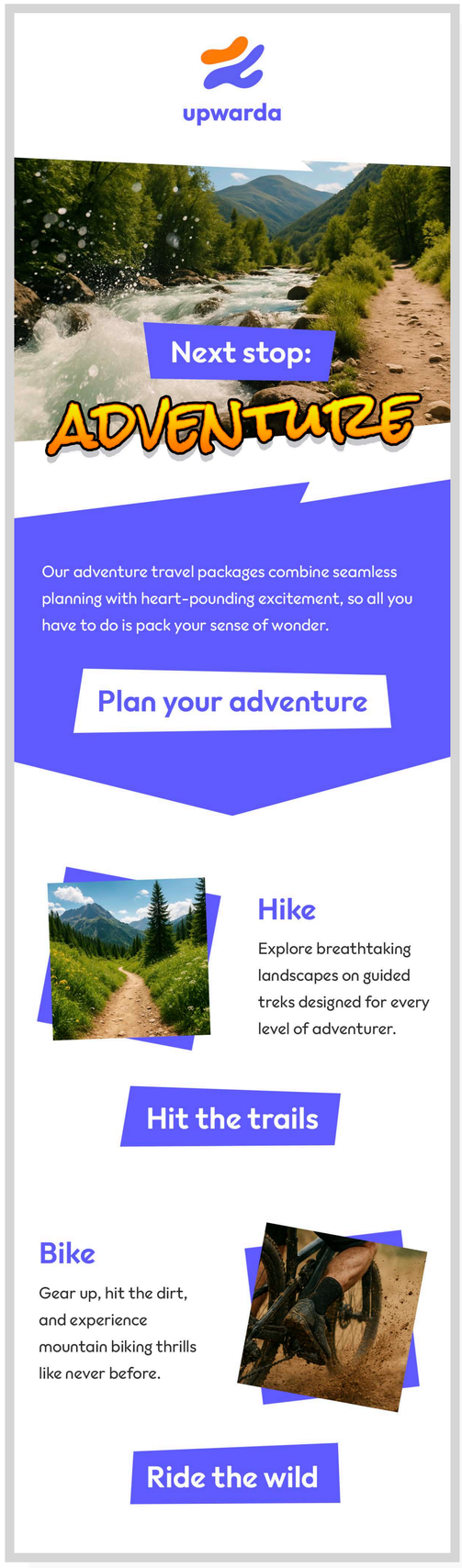
Let's build an email

We've designed a mailing for a make-believe adventure travel brand. Now, we're going to turn it into a real email.

A little bit of foresight and planning never hurts. Two important things to consider off the bat:

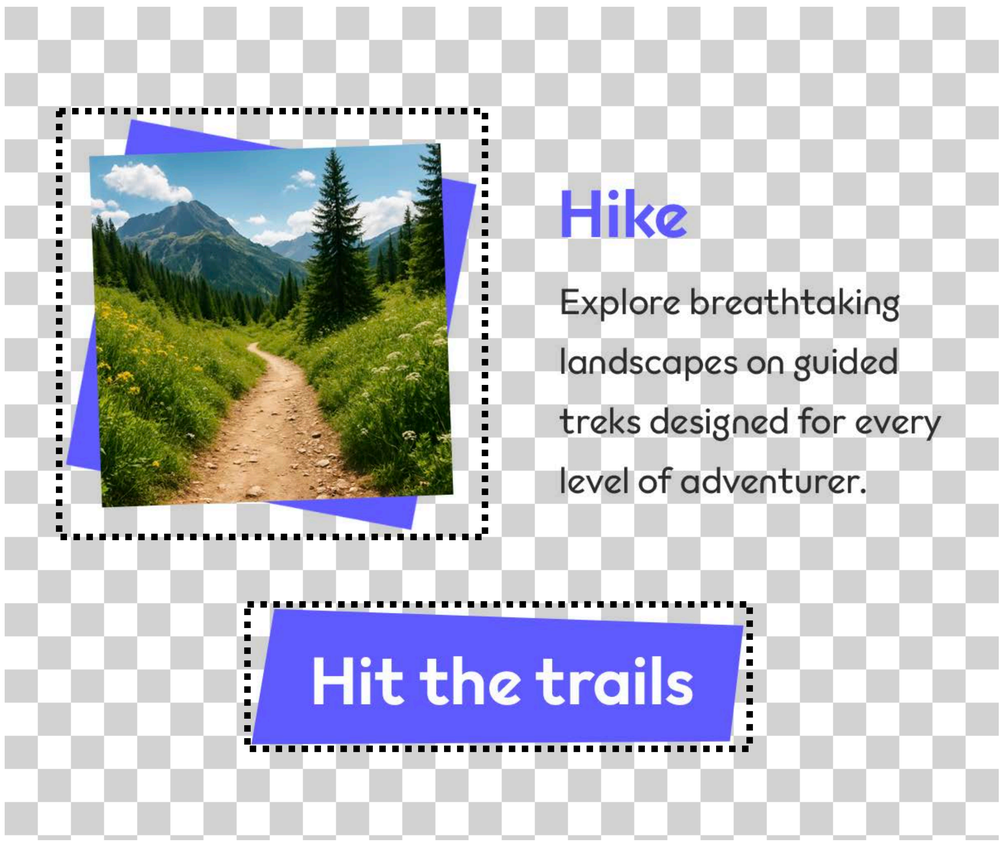
1. **Responsive design:**
how is it going to scale and adapt for mobile?

2. **Accessibility:**
how much of the copy can be real text...
rather than *images* of text.



We will **stack** the secondary *hike* and *bike* features into a single column on mobile. That ensures a nice clean layout on narrower screens, without having to squash anything into place.

We'll use a **web font** for paragraphs of copy, but *not* for the buttons. Their funky shape is part of the fictional brand's identity and we want to preserve it. And the only way to universally achieve that in email is with images.



About that – we will use **PNG format** images with transparency. That allows them to blend correctly with the background in **dark mode**... more on that topic later.

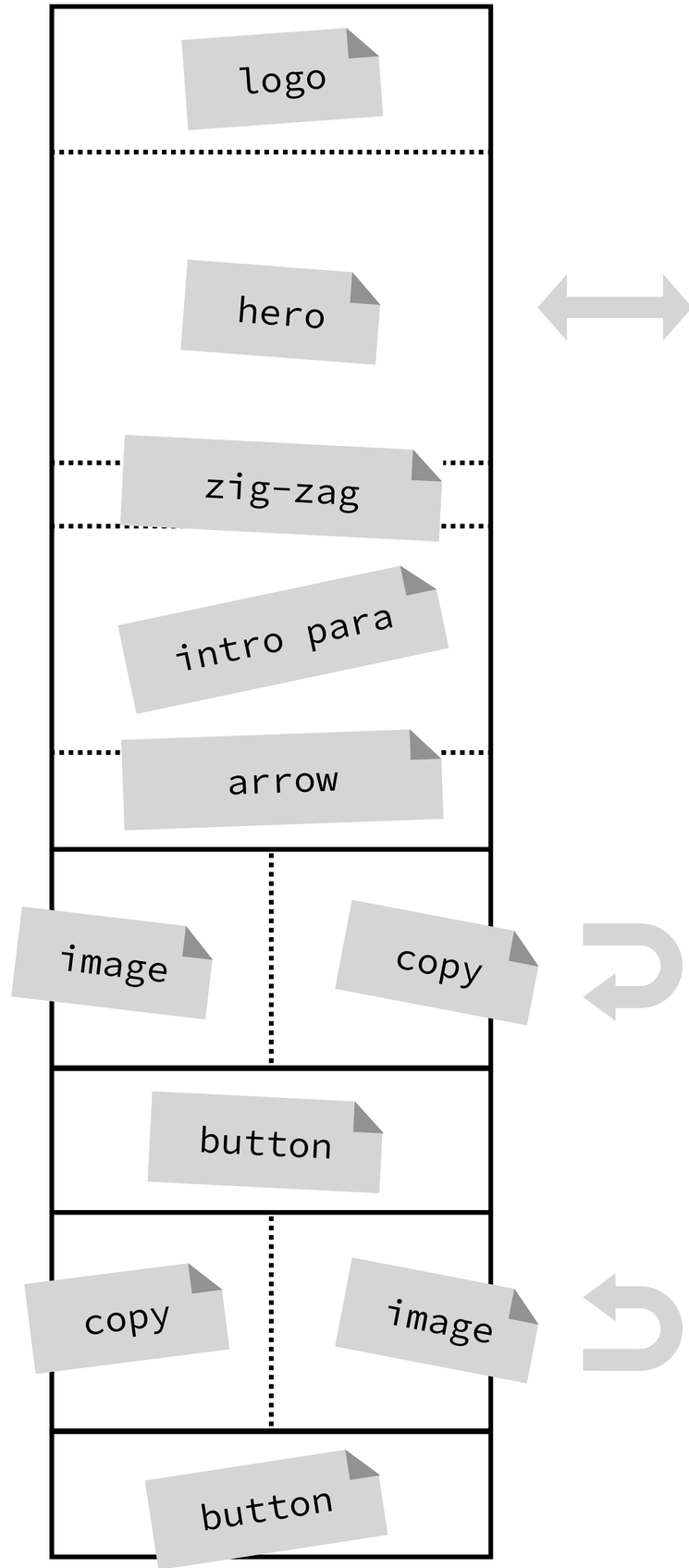
Image optimisation is important. The smaller the file size, the faster the load time. While *some* users have the luxury of superfast fibre broadband, others may be reading your email while connected to slower data networks.

Coding is next. We *could* spend page after page examining the HTML and CSS in detail. But we're not going to do that. Instead, we'll highlight some key aspects of the build process.

Our content is going to be housed in a series of **HTML tables**. Here's a simplified visualisation. In reality, there will be additional spacer rows and columns, and nested tables for copy.

We've added some arrows to indicate how aspects of the design should behave on mobile. The hero image scales fluidly, the first of the secondary features stacks into a single column, and the second feature *reverse* stacks into a single column.

Let's see how that is achieved.



Responsive code works via **media queries**. The specific aspect of that that we're interested in is **screen width**. For viewports below a certain size, we include a set of **mobile-only CSS rules** to kick in.

```
@media only screen and (max-width:639px) {
  /* your CSS here */
}
```

And this is a good example of where email code becomes... less-than-pretty. There's a little code flag that can be applied when using CSS: `!important`

This is the coding equivalent of a **sledgehammer**. It breaks free from the elegant hierarchical nature of CSS, and immediately forces priority for that rule. On a website, this is generally considered terrible practice.

But in email, we rely on inline CSS for desktop email apps. Inline CSS is highly-specific and not easy to override. Therefore email mobile code is peppered with `!important` flags. Messy, but it works!

Our design uses a **web font**. That's a font pulled from a remote host rather than one that is typically found on a user's device. The Verdanas and Helveticas of the world seemed a little prosaic for our kinetic Upwarda brand. We opted for a dynamic-looking Google Font called Outfit.

So, web fonts work in email? The answer, as is so often the case in this medium, is *sort of*. Some apps support them, some don't. For those that don't, standard system fonts can be included in the **font stack** as a **fallback**. That is to say, if Outfit cannot be displayed then it will revert to a standard font in its place.

But guess what – even that doesn't quite work properly in email. A behind-the-scenes fix is needed to prevent some versions of Outlook from defaulting to Times New Roman. Such is the strange and wonderful world of email dev!

Regular 400

Look at this lovely font

Medium 500

Look at this lovely font

SemiBold 600

Look at this lovely font

Bold 700

Look at this lovely font

ExtraBold 800

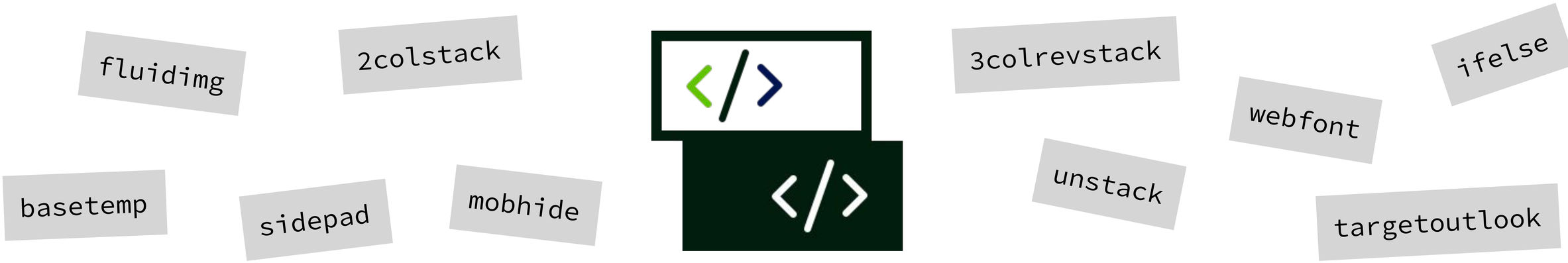
Look at this lovely font

Black 900

Look at this lovely font

Professional email developers **hand-code** their mailings. Wait... does that mean they sit and type every single character?

Thankfully, no. That would be unworkably inefficient. An email marketing department or agency develops a library of **templates** and **code snippets** over time. Some of these are general purpose, others are client-specific.



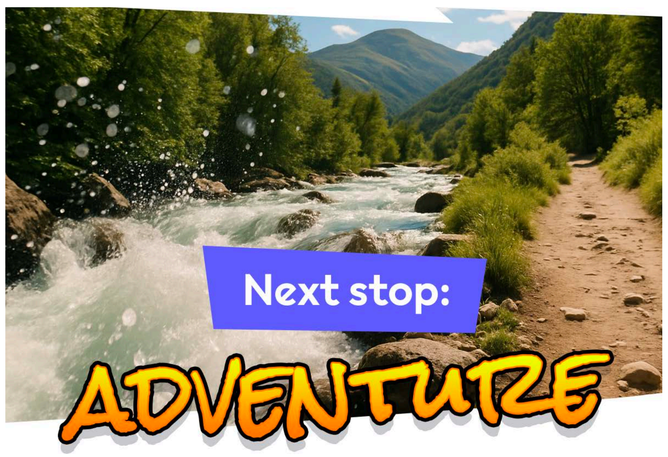
By building these code libraries into their **integrated development environment**, an email dev can work quickly, accurately and consistently... without having to re-invent the wheel for every email.

At all steps of an email build, there's a very important matter to keep in mind: **accessibility**. All users must be able to understand, navigate and operate the email.

Semantic code – often overlooked in email - lets you mark specific components of an email, such as headings and bullet points. This is essential information for people using assistive technologies such as **screen readers**.

```
<h2>
```

Similarly, **alt tags** provide a means of describing images or relaying the copy contained within them.



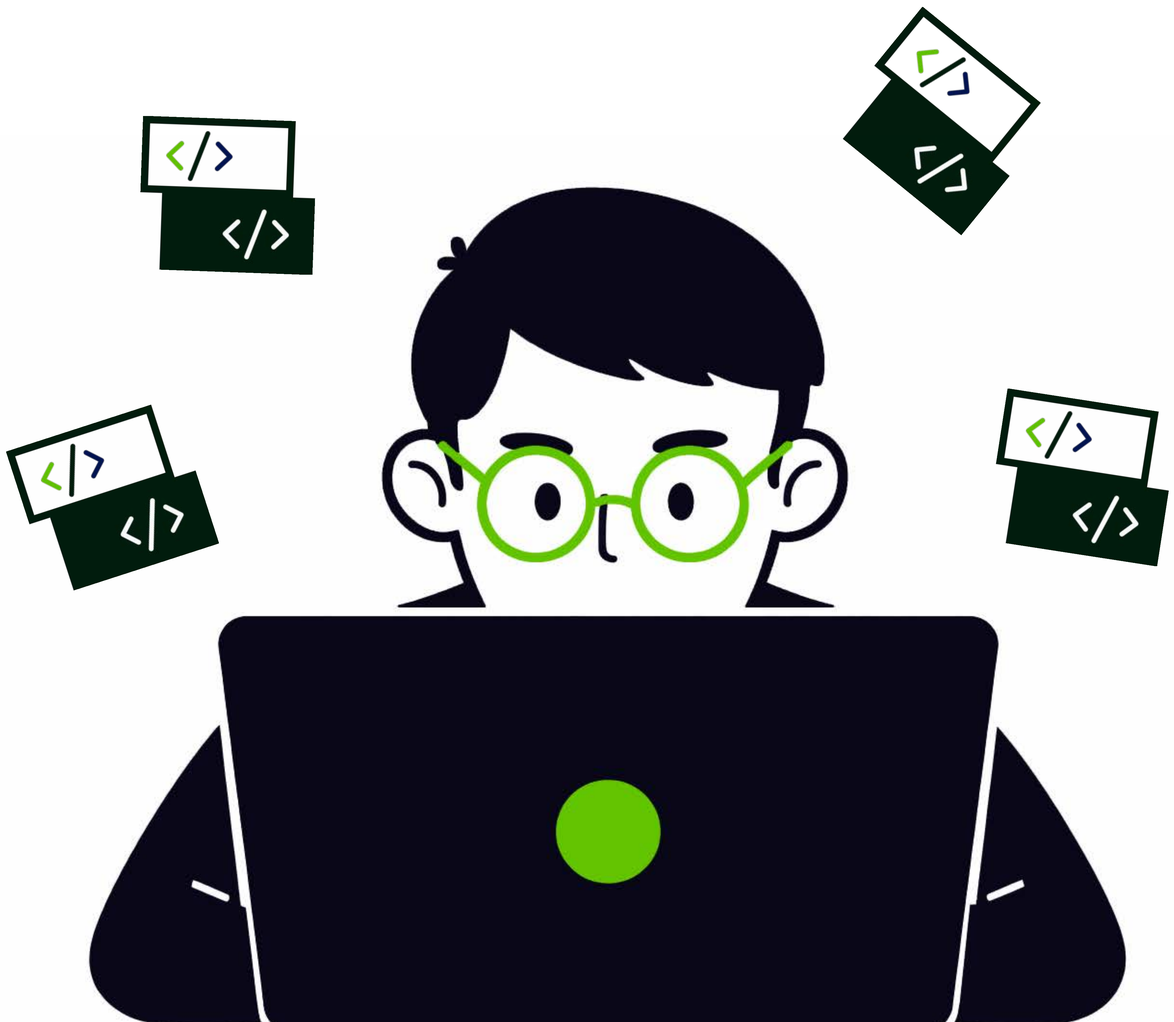

```
alt="Next stop: adventure"
```

```
alt="Summer forest trail with mountains in the background"
```

Good news – while we've been talking, one of our trusty email devs went to work on our project. All of the images are optimised to perfection. The code has been checked for errors and refined for maximum efficiency. Seems like it's good to go.

Great! If it were a real mailing, this is where we'd send it to real customers, right?

Not quite. There's one crucial final step in the email build process...



Test, test and test some more

Testing is as important a stage of the development process as coding is. The email 'ecosystem' is volatile place where even a tried & tested template might suddenly stop working. Unannounced app updates can break emails!

That's why *every* mailing needs to be tested before the live send. Email previewing services such as **Litmus** and **Email on Acid** are a handy way to see snapshots of your email across a range of apps and devices.

It's also highly recommended to always check on some **real devices**. Be sure to include a range of devices – iPhone, Android, PC, Mac – *and* a range of email applications and webmail services.

Oh, and let's not forget about **display mode**...

A mailshot in the dark

Dark mode is an optional display mode on today's mobiles and computers. What does that mean for email? As ever, it depends.

Some applications automatically apply their own dark mode colour scheme, and sometimes invert black or white images to boot. There's no official term for this behaviour, but we like to call it **forced dark mode**. Gmail is one such environment.

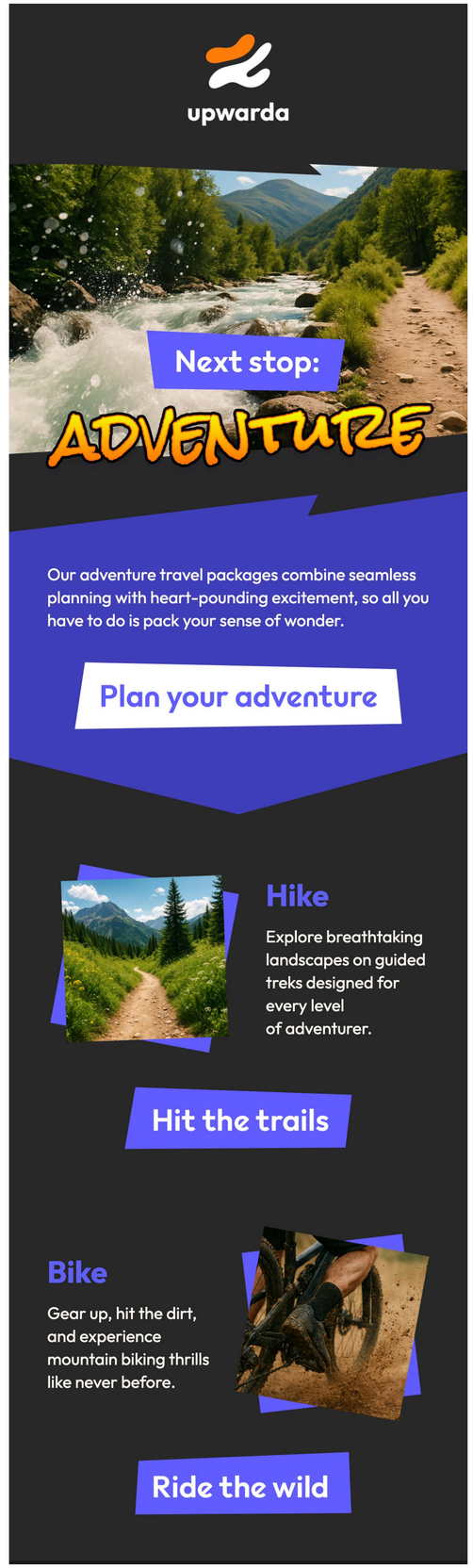
Other applications grant the developer full control over the email's appearance in dark mode. More specifically, they recognise the **CSS media queries** that make it possible. We call it **controlled dark mode**. iPhone's ever-popular Mail app is the most significant example.

In **controlled dark mode environments**, an application will not visually alter your email in any way... unless you want it to. Note: you *do* want that.

By creating a unique dark mode colour palette, you can remain on-brand while respecting the user's wish not to be dazzled.

Here's how our Upwarda email looks in controlled dark mode. The grey background colour and white text are perhaps the most obvious changes, but notice that we've also darkened the blue intro section.

And we've swapped the default orange and blue logo for a contrast-friendly orange and white version. Yep, image swaps are also possible – just use the technique sparingly.



Now, let's talk about **forced dark mode**. In these environments, it is not possible to directly control an email's dark appearance.

In fact, forced dark mode can very easily turn an otherwise well-designed email into an unsightly mess. Logo floating in a white box, anyone?

That was the bad news. The good news is that there are some production techniques to mitigate these problems. PNG images with transparency and hidden border effects are two such examples.

Don’t get the chop

Email developers face a day-to-day battle: **truncation**. Some applications – most notably Gmail – automatically truncate mailings once their code file size reaches a certain level. And by “a certain level”, we mean around 100 kilobytes. That is *tiny*.

This meagre upper limit on code presents a challenge when creating a richly-designed commercial email with lots of content. It could be argued that email service providers and brands have contrasting views on what an email should be.

The solution? **Code efficiently** as possible. Remove code-formatting. Know what to merge and what to simplify behind the scenes. These are techniques that are learned with industry experience – and can save your email from getting unceremoniously chopped!

Stick to the script

There's one little problem with our Upwarda email – not everybody loves mountain biking. Some people prefer white water rafting. Or maybe canyoning. And yet we are showing the same static content to our entire imaginary audience. We need some **targeted content** to tailor our email to the individual.

We're going to handle that with **dynamic content** via the email platform's built-in **scripting language**.

Let's imagine that Upwarda have customer interests recorded in their data. We can build an ***if* statement** around that.

```
{{if customer.primaryInterest == 'mtb'}}
<!-- mountain biking content goes here -->
{{elseif customer.primaryInterest == 'wwr'}}
<!-- white water rafting content goes here -->
{{elseif customer.primaryInterest == 'can'}}
<!-- canyoning content goes here -->
{{elseif customer.primaryInterest == 'mtr'}}
<!-- mountaineering content goes here -->
{{elseif customer.primaryInterest == 'hrs'}}
<!-- horseriding content goes here -->
{{else}}
<!-- fallback content goes here -->
{{endif}}
```

The syntax and capabilities of **scripting languages** vary from email platform to email platform. A few example languages include Salesforce Marketing Cloud's AMPscript, Responsys Personalisation Language and Emarsys Scripting Language.

Our earlier example only scraped the surface of what a good quality scripting language can do. Here are a few more examples of its usage:

- **String operations**

  You don't want to say "Hi, david". A string operation lets you correct imperfect data. "Hi, David" is much better.

- **Arithmetical calculation**

  Maybe you have a customer loyalty points scheme. In-email calculation could let you present that information visually in a chart.

- **Working with variables**

  No programming language would be complete without variables! Temporarily assign data for performance and efficiency.

- **Looping over arrays**

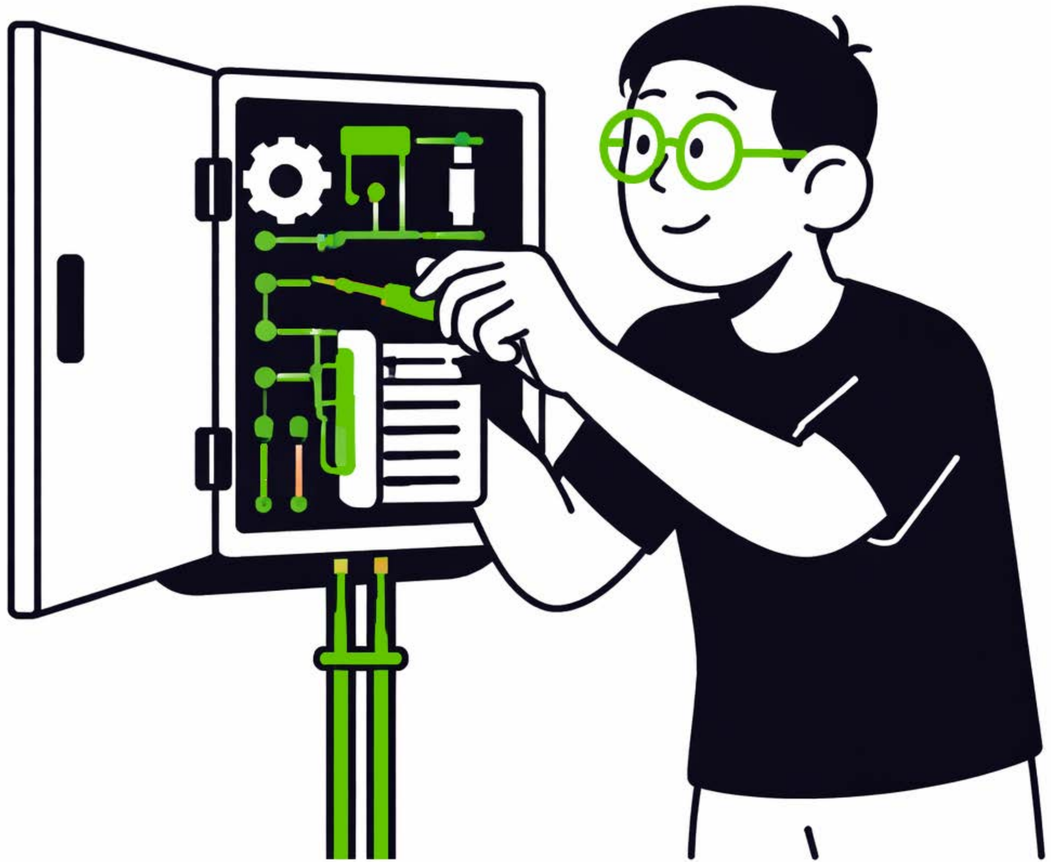
  Uh oh – your customer left three items in their cart. Loop over that array of data and incorporate it into a basket abandonment email.

What would you recommend?

True one-to-one personalisation is the name of the game. Customised **product recommendations** are a great piece of content for many emails.

Modern email platforms, or third party plug-ins, can bridge the gap between website and email. **Machine learning** can then suggest the perfect product based on individual user behaviour. That's great for the brand and the customer alike.

In order to make the most of this technology, it's often necessary to get 'under the hood'. This is where a skilled email developer comes in – adjusting filtering data or coding bespoke adjustments as required.



Looking back

So far, everything we've looked at relates to the part of the email that we can *see*. That's the **front end**.

But just like on the web, there's a whole other side to email development: the **back end**. Here we find essential processes such as data management, auto email triggers, list hygiene procedures, analytics, authentication and lots, lots more. The back end can be thought of as essential **behind-the-scenes infrastructure**.

Back end development typically uses programming languages such as PHP and SQL. It's a big, complex topic... and deserving of a white paper all on its own!

Our thoughts on WYSIWYG

You may be wondering: why go to all this bother when **drag & drop email builders** exist?

That's a fair question. Our view – while trying to remain as impartial as possible – is that drag & drop builders have their place. Some of these products are reasonably flexible and can output emails that display correctly on all major email apps. Drag & drop can be convenient for companies that prefer a DIY approach to email without having access to dedicated email coders.

Here comes the *but*. Even the best WYSIWYG tools are still essentially limited to presets. The code they produce tends to be bloated, as the app can't look for clever shortcuts like a human can.

By contrast, hand-coding means **complete creative and technical control** over every aspect of your emails. And although this may seem counterintuitive, a skilled email dev will always be *faster* than someone working with a WYSIWYG.

The next level

We've learned that email is a medium where Javascript does not exist, and HTML and CSS are limited compared to the web.

It may therefore surprise you that it is in fact possible to build surprisingly in-depth interactivity directly into an email. Think in terms of quizzes, surveys, product browsers and even simple games.

How is any of that possible? It works via a clever piece of **conditional CSS trickery**. It's a complex topic – but perhaps our **Complete Guide to Interactive Email** will help to demystify it.

Prepare to be *amazed* at what you can do in email.

The future of email and the role of AI

Email is a funny old medium. In some regards it's unapologetically stuck in the past, with its table-based structure and lack of standardisation. In other ways, it's state-of-the-art with algorithmically-generated content and a unique level of one-to-one targeting.

Where it goes from here largely depends on how the major email apps evolve. Will Gmail and Outlook one day match Apple Mail's level of CSS support? Can AI perhaps start to *really* customise emails for the individual, perhaps down to unique imagery and copy to suit their personality?

Only time will tell. We'll be there. And **we hope you'll be alongside us** to see it happen.

Let’s make some emails

We don’t know about you, but all this talk of building emails has really put us in the mood for... building some emails!

Speak to our team and **take your brand’s email marketing to the next level**.

Let’s do email

Read our blog

Connect on LinkedIn

the email factory